Oracle® Database

PL/SQL Packages and Types Reference

10*g* Release 2 (10.2)

B14258-01

June 2005

Oracle Database PL/SQL Packages and Types Reference, 10*g* Release 2 (10.2)

B14258-01

Primary Author: Denis Raphaely

Contributing Author: Rhonda Day, Craig Foch, Steve Fogel, Paul Lane, Chuck Murray, Sue Pelski, Kathy Rich, Antonio Romero, Vivian Schupmann, Margaret Taft, Kathy Taylor, Randy Urbano, Rodney Ward

Contents

15 DBMS_APPLY_ADM

16 DBMS_AQ

17 DBMS_AQADM

26 DBMS_DATA_MINING_TRANSFORM

27 DBMS_DATAPUMP

33 DBMS_DEFER_SYS

34 DBMS_DESCRIBE

35 DBMS_DIMENSION

36 DBMS_DISTRIBUTED_TRUST_ADMIN

37 DBMS_EPG

41 DBMS_FILE_GROUP

42 DBMS_FILE_TRANSFER

43 DBMS_FLASHBACK

56 DBMS_LOGSTDBY

57 DBMS_METADATA

58 DBMS_MGWADM

59 DBMS_MGWMSG

62 DBMS_OBFUSCATION_TOOLKIT

63 DBMS_ODCI

64 DBMS_OFFLINE_OG

65 DBMS_OLAP

66 DBMS_OUTLN

67 DBMS_OUTLN_EDIT

68 DBMS_OUTPUT

69 DBMS_PCLXUTIL

70 DBMS_PIPE

71 DBMS_PREPROCESSOR

72 DBMS_PREDICTIVE_ANALYTICS

73 DBMS_PROFILER

97 DBMS_SHARED_POOL

98 DBMS_SPACE

99 DBMS_SPACE_ADMIN

100 DBMS_SQL

101 DBMS_SQLTUNE

104 DBMS_STORAGE_MAP

105 DBMS_STREAMS

106 DBMS_STREAMS_ADM

111 DBMS_TRANSACTION

112 DBMS_TRANSFORM

113 DBMS_TDB

114 DBMS_TTS

117 DBMS_WARNING

118 DBMS_WORKLOAD_REPOSITORY

119 DBMS_WM

120 DBMS_XDB

123 DBMS_XDBZ

124 DBMS_XMLDOM

125 DBMS_XMLGEN

126 DBMS_XMLPARSER

127 DBMS_XMLQUERY

128 DBMS_XMLSAVE

134 HTF

138 HTMLDB_UTIL

139 HTP

140 OWA_CACHE

161 SDO_TUNE

162 SDO_UTIL

163 UTL_COLL

164 UTL_COMPRESS

165 UTL_DBWS

166 UTL_ENCODE

167 UTL_FILE

168 UTL_HTTP

171 UTL_LMS

172 UTL_MAIL

173 UTL_NLA

174 UTL_RAW

177 UTL_SMTP

178 UTL_TCP

183 ANYTYPE TYPE

184 Oracle Streams AQ TYPEs

185 Database URI TYPEs

Send Us Your Comments

Oracle Database PL/SQL Packages and Types Reference, 10*g* Release 2 (10.2)
B14258-01

Oracle welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the title and part number of the documentation and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: infodev_us@oracle.com
- FAX: (650) 506-7227. Attn: Server Technologies Documentation Manager
- Postal service:

  Oracle Corporation
  Server Technologies Documentation
  500 Oracle Parkway, Mailstop 4op11
  Redwood Shores, CA 94065
  USA

If you would like a reply, please give your name, address, telephone number, and electronic mail address (optional).

If you have problems with the software, please contact your local Oracle Support Services.

Preface

This Preface contains these topics:

Audience

Oracle Database PL/SQL Packages and Types Reference is intended for programmers, systems analysts, project managers, and others interested in developing database applications. This manual assumes a working knowledge of application programming and familiarity with SQL to access information in relational database systems. Some sections also assume a knowledge of basic object-oriented programming:

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at

`http://www.oracle.com/accessibility/`

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Structure

See Table 1–1, " Summary of Oracle Supplied PL/SQL Packages" for information about the organization of this reference.

Related Documents

You can find links in specific chapters to related documents. For general information, see these Oracle resources:

- *Oracle Database Application Developer's Guide - Fundamentals*
- *Oracle Database PL/SQL User's Guide and Reference*

Many of the examples in this book use the sample schemas, which are installed by default when you select the Basic Installation option with an Oracle Database installation. Refer to *Oracle Database Sample Schemas* for information on how these schemas were created and how you can use them yourself.

Printed documentation is available for sale in the Oracle Store at

`http://oraclestore.oracle.com/`

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

`http://otn.oracle.com/membership/`

If you already have a username and password for OTN, then you can go directly to the documentation section of the OTN Web site at

`http://otn.oracle.com/documentation/`

Conventions

This section describes the conventions used in the text and code examples of this documentation set. It describes:

- Conventions in Text
- Conventions in Code Examples

Conventions in Text

We use various conventions in text to help you more quickly identify special terms. The following table describes those conventions and provides examples of their use.

Convention	Meaning	Example
Bold	Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.	When you specify this clause, you create an **index-organized table**.
Italics	Italic typeface indicates book titles or emphasis.	*Oracle Database Concepts* Ensure that the recovery catalog and target database do *not* reside on the same disk.
`UPPERCASE monospace (fixed-width) font`	Uppercase monospace typeface indicates elements supplied by the system. Such elements include parameters, privileges, datatypes, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, usernames, and roles.	You can specify this clause only for a `NUMBER` column. You can back up the database by using the `BACKUP` command. Query the `TABLE_NAME` column in the `USER_TABLES` data dictionary view. Use the `DBMS_STATS.GENERATE_STATS` procedure.
`lowercase monospace (fixed-width) font`	Lowercase monospace typeface indicates executable programs, filenames, directory names, and sample user-supplied elements. Such elements include computer and database names, net service names and connect identifiers, user-supplied database objects and structures, column names, packages and classes, usernames and roles, program units, and parameter values. *Note:* Some programmatic elements use a mixture of `UPPERCASE` and `lowercase`. Enter these elements as shown.	Enter `sqlplus` to start SQL*Plus. The password is specified in the `orapwd` file. Back up the datafiles and control files in the `/disk1/oracle/dbs` directory. The `department_id`, `department_name`, and `location_id` columns are in the `hr.departments` table. Set the `QUERY_REWRITE_ENABLED` initialization parameter to `true`. Connect as `oe` user. The `JRepUtil` class implements these methods.
`lowercase italic monospace (fixed-width) font`	Lowercase italic monospace font represents placeholders or variables.	You can specify the *`parallel_clause`*. Run *`old_release`*`.SQL` where *`old_release`* refers to the release you installed prior to upgrading.

Conventions in Code Examples

Code examples illustrate SQL, PL/SQL, SQL*Plus, or other command-line statements. They are displayed in a monospace (fixed-width) font and separated from normal text as shown in this example:

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

The following table describes typographic conventions used in code examples and provides examples of their use.

Convention	Meaning	Example
`[ ]`	Anything enclosed in brackets is optional.	`DECIMAL (`*`digits`*` [ , `*`precision`*` ])`
`{ }`	Braces are used for grouping items.	`{ENABLE | DISABLE}`
`|`	A vertical bar represents a choice of two options.	`{ENABLE | DISABLE}` `[COMPRESS | NOCOMPRESS]`

Convention	Meaning	Example

`...`	Ellipsis points mean repetition in syntax descriptions.
	
	In addition, ellipsis points can mean an omission in code examples or text.
		`CREATE TABLE ... AS subquery;`
		
		`SELECT col1, col2, ... , coln FROM employees;`

Other symbols	You must use symbols other than brackets ([ ]), braces ({ }), vertical bars (|), and ellipsis points (...) exactly as shown.	`acctbal NUMBER(11,2);` `acct    CONSTANT NUMBER(4) := 3;`
Italics	Italicized text indicates placeholders or variables for which you must supply particular values.	`CONNECT SYSTEM/system_password` `DB_NAME = database_name`

`UPPERCASE`	Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. Because these terms are not case sensitive, you can use them in either UPPERCASE or lowercase.	`SELECT last_name, employee_id FROM employees;`
		`SELECT * FROM USER_TABLES;`
		`DROP TABLE hr.employees;`

`lowercase`	Lowercase typeface indicates user-defined programmatic elements, such as names of tables, columns, or files.
	
	Note: Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.
		`SELECT last_name, employee_id FROM employees;`
		`sqlplus hr/hr`
		`CREATE USER mjones IDENTIFIED BY ty3MU9;`

What's New in PL/SQL Packages and Types Reference?

The following sections describe the new features in Oracle PL/SQL Packages and Types Reference:

Oracle Database 10g Release 2 (10.2) New Features

New Packages

- DBMS_AQELM
- DBMS_AQIN
- DBMS_CHANGE_NOTIFICATION
- DBMS_DB_VERSION
- DBMS_EPG
- DBMS_ERRLOG
- DBMS_EXPFIL
- DBMS_FILE_GROUP
- DBMS_PREDICTIVE_ANALYTICS
- DBMS_PREPROCESSOR
- DBMS_RLMGR
- DBMS_TDB

- SDO_NET_MEM
- UTL_NLA

Updated Packages

- DBMS_ADVISOR
- DBMS_APPLY_ADM
- DBMS_AQ
- DBMS_AQADM
- DBMS_CAPTURE_ADM
- DBMS_CDC_PUBLISH
- DBMS_CRYPTO
- DBMS_DATA_MINING
- DBMS_DATA_MINING_TRANSFORM
- DBMS_DATAPUMP
- DBMS_DDL
- DBMS_DESCRIBE
- DBMS_FGA
- DBMS_FILE_TRANSFER
- DBMS_LOB
- DBMS_LOGMNR
- DBMS_LOGMNR_D
- DBMS_LOGSTDBY
- DBMS_METADATA
- DBMS_MGWADM
- DBMS_MGWMSG
- DBMS_MONITOR
- DBMS_OUTLN_EDIT
- DBMS_OUTPUT
- DBMS_PROPAGATION_ADM
- DBMS_REDEFINITION
- DBMS_REPAIR
- DBMS_RESOURCE_MANAGER
- DBMS_RULE_ADM
- DBMS_SCHEDULER
- DBMS_SERVER_ALERT
- DBMS_SERVICE
- DBMS_SESSION
- DBMS_SPACE

- DBMS_SPACE_ADMIN
- DBMS_SQLTUNE
- DBMS_STATS
- DBMS_STREAMS
- DBMS_STREAMS_ADM
- DBMS_STREAMS_TABLESPACE_ADM
- DBMS_TTS
- DBMS_UTILITY
- DBMS_WORKLOAD_REPOSITORY
- DBMS_XDB
- DBMS_XDBZ
- DBMS_XMLDOM
- DBMS_XMLSCHEMA
- DBMS_XPLAN
- DBMS_XSLPROCESSOR
- SDO_CS
- SDO_TOPO_MAP
- SDO_UTIL
- UTL_I18N

Updated Types

- Oracle Streams AQ TYPEs

Oracle Database 10*g* Release 1 (10.1) New Features

New Packages

- DBMS_ADVANCED_REWRITE
- DBMS_ADVISOR
- DBMS_CRYPTO
- DBMS_DATAPUMP
- DBMS_DATA_MINING
- DBMS_DATA_MINING_TRANSFORM
- DBMS_DIMENSION
- DBMS_FILE_TRANSFER
- DBMS_FREQUENT_ITEMSET
- DBMS_JAVA
- DBMS_MONITOR
- DBMS_SCHEDULER

- DBMS_SERVER_ALERT
- DBMS_SERVICE
- DBMS_SQLTUNE
- DBMS_STAT_FUNCS
- DBMS_STREAMS_AUTH
- DBMS_STREAMS_MESSAGING
- DBMS_STREAMS_TABLESPACE_ADM
- DBMS_WARNING
- DBMS_WORKLOAD_REPOSITORY
- DBMS_XDBZ
- DBMS_XMLSTORE
- HTF
- HTMLDB_APPLICATION
- HTMLDB_CUSTOM_AUTH
- HTMLDB_ITEM
- HTMLDB_UTIL
- HTP
- OWA_CACHE
- OWA_COOKIE
- OWA_CUSTOM
- OWA_IMAGE
- OWA_OPT_LOCK
- OWA_PATTERN
- OWA_SEC
- OWA_TEXT
- OWA_UTIL
- SDO_GCDR
- SDO_GEOR
- SDO_GEOR_UTL
- SDO_NET
- SDO_SAM
- SDO_TOPO
- SDO_TOPO_MAP
- UTL_COMPRESS
- UTL_DBWS
- UTL_I18N
- UTL_LMS

- UTL_MAIL
- UTL_RECOMP
- WPG_DOCLOAD

New Types

- Database URI TYPEs
- XMLTYPE

Updated Packages

- DBMS_APPLICATION_INFO
- DBMS_APPLY_ADM
- DBMS_AQ
- DBMS_AQADM
- DBMS_AQELM
- DBMS_CAPTURE_ADM
- DBMS_CDC_PUBLISH
- DBMS_CDC_SUBSCRIBE
- DBMS_DDL
- DBMS_DESCRIBE
- DBMS_DISTRIBUTED_TRUST_ADMIN
- DBMS_FGA
- DBMS_JOB
- DBMS_LIBCACHE
- DBMS_LOB
- DBMS_LOGMNR
- DBMS_LOGMNR_D
- DBMS_METADATA
- DBMS_MGWADM
- DBMS_MGWMSG
- DBMS_MVIEW
- DBMS_OBFUSCATION_TOOLKIT
- DBMS_OLAP
- DBMS_OUTLN
- DBMS_OUTLN_EDIT
- DBMS_OUTPUT
- DBMS_PROPAGATION_ADM
- DBMS_REDEFINITION
- DBMS_RESOURCE_MANAGER

- DBMS_RLS
- DBMS_ROWID
- DBMS_RULE
- DBMS_RULE_ADM
- DBMS_SESSION
- DBMS_SPACE
- DBMS_SQL
- DBMS_STATS
- DBMS_STREAMS
- DBMS_STREAMS_ADM
- DBMS_TTS
- DBMS_TYPES
- DBMS_UTILITY
- DBMS_WM
- DBMS_XDB
- DBMS_XDB_VERSION
- DBMS_XMLGEN
- DBMS_XMLSCHEMA
- DBMS_XPLAN
- DBMS_XSLPROCESSOR
- DBMS_WM
- SDO_CS
- SDO_GEOM
- SDO_LRS
- SDO_MIGRATE
- SDO_TUNE
- SDO_UTIL
- UTL_ENCODE
- UTL_FILE
- UTL_RAW
- UTL_URL

Updated Types

- ANYDATA TYPE
- ANYDATASET TYPE
- Oracle Streams AQ TYPEs
- Logical Change Record TYPEs
- Rule TYPEs

1
Introduction

Oracle supplies many PL/SQL packages with the Oracle server to extend database functionality and provide PL/SQL access to SQL features. You can use the supplied packages when creating your applications or for ideas in creating your own stored procedures.

> **Note:** This manual covers the packages provided with the Oracle database server. Packages supplied with other products, such as Oracle Developer or the Oracle Application Server, are not covered.

This chapter contains the following topics:

- Package Overview
- Summary of Oracle Supplied PL/SQL Packages

> **See Also:** *Oracle Database Application Developer's Guide - Fundamentals* for information on how to create your own packages

Package Overview

A *package* is an encapsulated collection of related program objects stored together in the database. Program objects are procedures, functions, variables, constants, cursors, and exceptions.

Packages have many advantages over standalone procedures and functions. For example, they:

- Let you organize your application development more efficiently.
- Let you grant privileges more efficiently.
- Let you modify package objects without recompiling dependent schema objects.
- Enable Oracle to read multiple package objects into memory at once.
- Let you *overload* procedures or functions. Overloading means creating multiple procedures with the same name in the same package, each taking arguments of different number or datatype.
- Can contain global variables and cursors that are available to all procedures and functions in the package.

Package Components

PL/SQL packages have two parts: the specification and the body, although sometimes the body is unnecessary. The specification is the interface to your application; it declares the types, variables, constants, exceptions, cursors, and subprograms available for use. The body fully defines cursors and subprograms, and so implements the specification.

Unlike subprograms, packages cannot be called, parameterized, or nested. However, the formats of a package and a subprogram are similar:

```
CREATE PACKAGE name AS  -- specification (visible part)
   -- public type and item declarations
   -- subprogram specifications
END [name];

CREATE PACKAGE BODY name AS  -- body (hidden part)
   -- private type and item declarations
   -- subprogram bodies
[BEGIN
   -- initialization statements]
END [name];
```

The specification holds public declarations that are visible to your application. The body holds implementation details and private declarations that are hidden from your application. You can debug, enhance, or replace a package body without changing the specification. You can change a package body without recompiling calling programs because the implementation details in the body are hidden from your application.

Using Oracle Supplied Packages

Most Oracle supplied packages are automatically installed when the database is created and the CATPROC.SQL script is run. For example, to create the DBMS_ALERT package, the DBMSALRT.SQL and PRVTALRT.PLB scripts must be run when connected as the user SYS. These scripts are run automatically by the CATPROC.SQL script.

Certain packages are not installed automatically. Special installation instructions for these packages are documented in the individual chapters.

To call a PL/SQL function from SQL, you must either own the function or have EXECUTE privileges on the function. To select from a view defined with a PL/SQL function, you must have SELECT privileges on the view. No separate EXECUTE privileges are needed to select from the view. Instructions on special requirements for packages are documented in the individual chapters.

Creating New Packages

To create packages and store them permanently in an Oracle database, use the `CREATE PACKAGE` and `CREATE PACKAGE BODY` statements. You can execute these statements interactively from SQL*Plus or Enterprise Manager.

To create a new package, do the following:

1. Create the package specification with the `CREATE PACKAGE` statement.

   You can declare program objects in the package specification. Such objects are called *public* objects. Public objects can be referenced outside the package, as well as by other objects in the package.

   Note: It is often more convenient to add the `OR REPLACE` clause in the `CREATE PACKAGE` statement. But note that `CREATE PACKAGE` warns you if you are about to overwrite an existing package with the same name while `CREATE OR REPLACE just` overwrites it with no warning.

2. Create the package body with the `CREATE PACKAGE BODY` statement.

   You can declare and define program objects in the package body.

   - You must define public objects declared in the package specification.
   - You can declare and define additional package objects, called *private* objects. Private objects are declared in the package body rather than in the package specification, so they can be referenced only by other objects in the package. They cannot be referenced outside the package.

     See Also:

     - *Oracle Database PL/SQL User's Guide and Reference*
     - *Oracle Database Application Developer's Guide - Fundamentals*for more information on creating new packages
     - *Oracle Database Concepts*

     for more information on storing and executing packages

Separating the Specification and Body

The specification of a package declares the public types, variables, constants, and subprograms that are visible outside the immediate scope of the package. The body of a package defines the objects declared in the specification, as well as private objects that are not visible to applications outside the package.

Oracle stores the specification and body of a package separately in the database. Other schema objects that call or reference public program objects depend only on the package specification, not on the package body. Using this distinction, you can change the definition of a program object in the package body without causing Oracle to invalidate other schema objects that call or reference the program object. Oracle invalidates dependent schema objects only if you change the declaration of the program object in the package specification.

Creating a New Package: Example

The following example shows a package specification for a package named EMPLOYEE_MANAGEMENT. The package contains one stored function and two stored procedures.

```
CREATE PACKAGE employee_management AS
   FUNCTION hire_emp (name VARCHAR2, job VARCHAR2,
      mgr NUMBER, hiredate DATE, sal NUMBER, comm NUMBER,
      deptno NUMBER) RETURN NUMBER;
   PROCEDURE fire_emp (emp_id NUMBER);
   PROCEDURE sal_raise (emp_id NUMBER, sal_incr NUMBER);
END employee_management;
```

The body for this package defines the function and the procedures:

```
CREATE PACKAGE BODY employee_management AS
   FUNCTION hire_emp (name VARCHAR2, job VARCHAR2,
      mgr NUMBER, hiredate DATE, sal NUMBER, comm NUMBER,
      deptno NUMBER) RETURN NUMBER IS
```

The function accepts all arguments for the fields in the employee table except for the employee number. A value for this field is supplied by a sequence. The function returns the sequence number generated by the call to this function.

```
       new_empno    NUMBER(10);

   BEGIN
      SELECT emp_sequence.NEXTVAL INTO new_empno FROM dual;
      INSERT INTO emp VALUES (new_empno, name, job, mgr,
         hiredate, sal, comm, deptno);
      RETURN (new_empno);
   END hire_emp;

   PROCEDURE fire_emp(emp_id IN NUMBER) AS
```

The procedure deletes the employee with an employee number that corresponds to the argument emp_id. If no employee is found, then an exception is raised.

```
   BEGIN
      DELETE FROM emp WHERE empno = emp_id;
      IF SQL%NOTFOUND THEN
      raise_application_error(-20011, 'Invalid Employee
         Number: ' || TO_CHAR(emp_id));
   END IF;
END fire_emp;

PROCEDURE sal_raise (emp_id IN NUMBER, sal_incr IN NUMBER) AS
```

The procedure accepts two arguments. Emp_id is a number that corresponds to an employee number. Sal_incr is the amount by which to increase the employee's salary.

```
   BEGIN

   -- If employee exists, then update salary with increase.

      UPDATE emp
         SET sal = sal + sal_incr
         WHERE empno = emp_id;
      IF SQL%NOTFOUND THEN
         raise_application_error(-20011, 'Invalid Employee
```

```
            Number: ' || TO_CHAR(emp_id));
      END IF;
   END sal_raise;
END employee_management;
```

Note: If you want to try this example, then first create the sequence number `emp_sequence`. You can do this using the following SQL*Plus statement:

```
SQL> CREATE SEQUENCE emp_sequence
   > START WITH 8000 INCREMENT BY 10;
```

Referencing Package Contents

To reference the types, items, and subprograms declared in a package specification, use the dot notation. For example:

```
package_name.type_name
package_name.item_name
package_name.subprogram_name
```

Summary of Oracle Supplied PL/SQL Packages

Table 1–1 lists the supplied PL/SQL server packages. These packages run as the invoking user, rather than the package owner. Unless otherwise noted, the packages are callable through public synonyms of the same name.

Caution:

- The procedures and functions provided in these packages and their external interfaces are reserved by Oracle and are subject to change.
- Modifying Oracle supplied packages can cause internal errors and database security violations. Do not modify supplied packages.

Table 1–1 Summary of Oracle Supplied PL/SQL Packages

Package Name	Description
CTX_ADM	Lets you administer servers and the data dictionary.
CTX_CLS	Lets you generate CTXRULE rules for a set of documents.
CTX_DDL	Lets you create and manage the preferences, section lists and stopgroups required for Text indexes.
CTX_DOC	Lets you request document services.
CTX_OUTPUT	Lets you manage the index log.
CTX_QUERY	Lets you generate query feedback, count hits, and create stored query expressions.
CTX_REPORT	Lets you create various index reports.
CTX_THES	Lets you to manage and browse thesauri.
CTX_ULEXER	For use with the user-lexer.
DBMS_ADVANCED_REWRITE	Contains interfaces for advanced query rewrite users to create, drop, and maintain functional equivalence declarations for query rewrite.
DBMS_ADVISOR	Part of the SQLAccess Advisor, an expert system that identifies and helps resolve performance problems relating to the execution of SQL statements.
DBMS_ALERT	Provides support for the asynchronous notification of database events.
DBMS_APPLICATION_INFO	Lets you register an application name with the database for auditing or performance tracking purposes.
DBMS_APPLY_ADM	Provides administrative procedures to start, stop, and configure an apply process.
DBMS_AQ	Lets you add a message (of a predefined object type) onto a queue or to dequeue a message.
DBMS_AQADM	Lets you perform administrative functions on a queue or queue table for messages of a predefined object type.

Table 1–1 (Cont.) Summary of Oracle Supplied PL/SQL Packages

Package Name	Description
DBMS_AQELM	Provides procedures to manage the configuration of Advanced Queuing asynchronous notification by e-mail and HTTP.
DBMS_AQIN	Plays a part in providing secure access to the Oracle JMS interfaces.
DBMS_CAPTURE_ADM	Describes administrative procedures to start, stop, and configure a capture process; used in Streams.
DBMS_CDC_PUBLISH	Identifies new data that has been added to, modified, or removed from, relational tables and publishes the changed data in a form that is usable by an application.
DBMS_CDC_SUBSCRIBE	Lets you view and query the change data that was captured and published with the DBMS_LOGMNR_CDC_PUBLISH package.
DBMS_CHANGE_NOTIFICATION	Is part of a set of features that clients use to receive notifications when result sets of a query have changed. The package contains interfaces that can be used by mid-tier clients to register objects and specify delivery mechanisms.
DBMS_CRYPTO	Lets you encrypt and decrypt stored data, can be used in conjunction with PL/SQL programs running network communications, and supports encryption and hashing algorithms.
DBMS_DATA_MINING	Lets you use data mining to discover hidden patterns and use that knowledge to make predictions.
DBMS_DATA_MINING_TRANSFORM	Provides set of data transformation utilities available for use with the DBMS_DATA_MINING package for preparing mining data.
DBMS_DATAPUMP	Lets you move all, or part of, a database between databases, including both data and metadata.
DBMS_DB_VERSION	Specifies the Oracle version numbers and other information useful for simple conditional compilation selections based on Oracle versions.
DBMS_DDL	Provides access to some SQL DDL statements from stored procedures, and provides special administration operations not available as DDLs.
DBMS_DEBUG	Implements server-side debuggers and provides a way to debug server-side PL/SQL program units.
DBMS_DEFER	Provides the user interface to a replicated transactional deferred remote procedure call facility. Requires the Distributed Option.
DBMS_DEFER_QUERY	Permits querying the deferred remote procedure calls (RPC) queue data that is not exposed through views. Requires the Distributed Option.
DMBS_DEFER_SYS	Provides the system administrator interface to a replicated transactional deferred remote procedure call facility. Requires the Distributed Option.
DBMS_DESCRIBE	Describes the arguments of a stored procedure with full name translation and security checking.
DBMS_DIMENSION	Enables you to verify dimension relationships and provides an alternative to the Enterprise Manager Dimension Wizard for displaying a dimension definition

Table 1–1 (Cont.) Summary of Oracle Supplied PL/SQL Packages

Package Name	Description
DBMS_DISTRIBUTED_TRUST_ ADMIN	Maintains the Trusted Database List, which is used to determine if a privileged database link from a particular server can be accepted.
DBMS_EPG	Implements the embedded PL/SQL gateway that enables a web browser to invoke a PL/SQL stored procedure through an HTTP listener.
DBMS_ERRLOG	Provides a procedure that enables you to create an error logging table so that DML operations can continue after encountering errors rather than abort and roll back.
DBMS_EXPFIL	Contains all the procedures used to manage attribute sets, expression sets, expression indexes, optimizer statistics, and privileges by Expression Filter.
DBMS_FGA	Provides fine-grained security functions.
DBMS_FILE_GROUP	One of a set of Streams packages, provides administrative interfaces for managing file groups, file group versions, files and file group repositories.
DMBS_FILE_TRANSFER	Lets you copy a binary file within a database or to transfer a binary file between databases.
DMBS_FLASHBACK	Lets you flash back to a version of the database at a specified wall-clock time or a specified system change number (SCN).
DBMS_FREQUENT_ITEMSET	Enables frequent itemset counting.
DBMS_HS_PASSTHROUGH	Lets you use Heterogeneous Services to send pass-through SQL statements to non-Oracle systems.
DBMS_IOT	Creates a table into which references to the chained rows for an Index Organized Table can be placed using the ANALYZE command.
DBMS_JAVA	Provides a PL/SQL interface for accessing database functionality from Java
DBMS_JOB	Lets you schedule administrative procedures that you want performed at periodic intervals; it is also the interface for the job queue.
DBMS_LDAP	Provides functions and procedures to access data from LDAP servers.
DBMS_LDAP_UTL	Provides the Oracle Extension utility functions for LDAP.
DBMS_LIBCACHE	Prepares the library cache on an Oracle instance by extracting SQL and PL/SQL from a remote instance and compiling this SQL locally without execution.
DBMS_LOB	Provides general purpose routines for operations on Oracle Large Object (LOBs) datatypes - BLOB, CLOB (read/write), and BFILEs (read-only).
DBMS_LOCK	Lets you request, convert and release locks through Oracle Lock Management services.
DBMS_LOGMNR	Provides functions to initialize and run the log reader.
DBMS_LOGMNR_D	Queries the dictionary tables of the current database, and creates a text based file containing their contents.
DBMS_LOGSTDBY	Describes procedures for configuring and managing the logical standby database environment.

Table 1–1 (Cont.) Summary of Oracle Supplied PL/SQL Packages

Package Name	Description
DBMS_METADATA	Lets callers easily retrieve complete database object definitions (metadata) from the dictionary.
DBMS_MGWADM	Describes the Messaging Gateway administrative interface; used in Advanced Queuing.
DBMS_MGWMSG	Describes object types—used by the canonical message types to convert message bodies—and helper methods, constants, and subprograms for working with the Messaging Gateway message types; used in Advanced Queuing.
DBMS_MONITOR	Let you use PL/SQL for controlling additional tracing and statistics gathering.
DBMS_MVIEW	Lets you refresh snapshots that are not part of the same refresh group and purge logs. DBMS_SNAPSHOT is a synonym.
DBMS_OBFUSCATION_TOOLKIT	Provides procedures for Data Encryption Standards.
DBMS_ODCI	Returns the CPU cost of a user function based on the elapsed time of the function.
DBMS_OFFLINE_OG	Provides public APIs for offline instantiation of master groups.
DBMS_OLAP	Provides procedures for summaries, dimensions, and query rewrites.
DBMS_OUTLN	Provides the interface for procedures and functions associated with management of stored outlines. Synonymous with OUTLN_PKG
DBMS_OUTLN_EDIT	Lets you edit an invoker's rights package.
DBMS_OUTPUT	Accumulates information in a buffer so that it can be retrieved later.
DBMS_PCLXUTIL	Provides intra-partition parallelism for creating partition-wise local indexes.
DBMS_PIPE	Provides a DBMS pipe service which enables messages to be sent between sessions.
DBMS_PREDICTIVE_ANALYTICS	Automates the data mining process from data preprocessing through model building to scoring new data.
DBMS_PREPROCESSOR	Provides an interface to print or retrieve the source text of a PL/SQL unit in its post-processed form.
DBMS_PROFILER	Provides a Probe Profiler API to profile existing PL/SQL applications and identify performance bottlenecks.
DBMS_PROPAGATION_ADM	Provides administrative procedures for configuring propagation from a source queue to a destination queue.
DBMS_RANDOM	Provides a built-in random number generator.
DBMS_RECTIFIER_DIFF	Provides APIs used to detect and resolve data inconsistencies between two replicated sites.
DBMS_REDEFINITION	Lets you perform an online reorganization of tables.
DBMS_REFRESH	Lets you create groups of snapshots that can be refreshed together to a transactionally consistent point in time. Requires the Distributed Option.

Table 1–1 (Cont.) Summary of Oracle Supplied PL/SQL Packages

Package Name	Description
DBMS_REPAIR	Provides data corruption repair procedures.
DBMS_REPCAT	Provides routines to administer and update the replication catalog and environment. Requires the Replication Option.
DBMS_REPCAT_ADMIN	Lets you create users with the privileges needed by the symmetric replication facility. Requires the Replication Option.
DBMS_REPCAT_INSTATIATE	Instantiates deployment templates. Requires the Replication Option.
DBMS_REPCAT_RGT	Controls the maintenance and definition of refresh group templates. Requires the Replication Option.
DBMS_REPUTIL	Provides routines to generate shadow tables, triggers, and packages for table replication.
DBMS_RESOURCE_MANAGER	Maintains plans, consumer groups, and plan directives; it also provides semantics so that you may group together changes to the plan schema.
DBMS_RESOURCE_MANAGER_PRIVS	Maintains privileges associated with resource consumer groups.
DBMS_RESUMABLE	Lets you suspend large operations that run out of space or reach space limits after executing for a long time, fix the problem, and make the statement resume execution.
DBMS_RLMGR	Contains various procedures to create and manage rules and rule sessions by the Rules Manager.
DBMS_RLS	Provides row level security administrative interface.
DBMS_ROWID	Provides procedures to create rowids and to interpret their contents.
DBMS_RULE	Describes the EVALUATE procedure used in Streams.
DBMS_RULE_ADM	Describes the administrative interface for creating and managing rules, rule sets, and rule evaluation contexts; used in Streams.
DBMS_SCHEDULER	Provides a collection of scheduling functions that are callable from any PL/SQL program.
DBMS_SERVER_ALERT	Lets you issue alerts when some threshold has been violated.
DBMS_SERVICE	Lets you create, delete, activate and deactivate services for a single instance.
DBMS_SESSION	Provides access to SQL ALTER SESSION statements, and other session information, from stored procedures.
DBMS_SHARED_POOL	Lets you keep objects in shared memory, so that they will not be aged out with the normal LRU mechanism.
DBMS_SPACE	Provides segment space information not available through standard SQL.
DBMS_SPACE_ADMIN	Provides tablespace and segment space administration not available through the standard SQL.
DBMS_SQL	Lets you use dynamic SQL to access the database.
DBMS_SQLTUNE	Provides the interface to tune SQL statements.

Table 1–1 (Cont.) Summary of Oracle Supplied PL/SQL Packages

Package Name	Description
DBMS_STAT_FUNCS	Provides statistical functions.
DBMS_STATS	Provides a mechanism for users to view and modify optimizer statistics gathered for database objects.
DBMS_STORAGE_MAP	Communicates with FMON to invoke mapping operations.
DBMS_STREAMS	Describes the interface to convert SYS.AnyData objects into LCR objects and an interface to annotate redo entries generated by a session with a binary tag.
DBMS_STREAMS_ADMIN	Describes administrative procedures for adding and removing simple rules, without transformations, for capture, propagation, and apply at the table, schema, and database level.
DBMS_STREAMS_AUTH	Provides interfaces for granting privileges to Streams administrators and revoking privileges from Streams administrators.
DBMS_STREAMS_MESSAGING	Provides interfaces to enqueue messages into and dequeue messages from a SYS.AnyData queue.
DBMS_STREAMS_TABLESPACE_ADM	Provides administrative procedures for copying tablespaces between databases and moving tablespaces from one database to another.
DBMS_TRACE	Provides routines to start and stop PL/SQL tracing.
DBMS_TRANSACTION	Provides access to SQL transaction statements from stored procedures and monitors transaction activities.
DBMS_TRANSFORM	Provides an interface to the message format transformation features of Oracle Advanced Queuing.
DBMS_TDB	Reports whether a database can be transported between platforms using the RMAN CONVERT DATABASE command. It verifies that databases on the current host platform are of the same endian format as the destination platform, and that the state of the current database does not prevent transport of the database.
DBMS_TTS	Checks if the transportable set is self-contained.
DBMS_TYPES	Consists of constants, which represent the built-in and user-defined types.
DBMS_UTILITY	Provides various utility routines.
DBMS_WARNING	Provides the interface to query, modify and delete current system or session settings.
DBMS_WORKLOAD_REPOSITORY	lets you manage the Workload Repository, performing operations such as managing snapshots and baselines.
DBMS_WM	Describes how to use the programming interface to Oracle Database Workspace Manager to work with long transactions.
DBMS_XDB	Describes Resource Management and Access Control APIs for PL/SQL
DBMS_XDB_VERSION	Describes versioning APIs
DBMS_XDBT	Describes how an administrator can create a ConText index on the XML DB hierarchy and configure it for automatic maintenance

Table 1–1 (Cont.) Summary of Oracle Supplied PL/SQL Packages

Package Name	Description
DBMS_XDBZ	Controls the Oracle XML DB repository security, which is based on Access Control Lists (ACLs).
DBMS_XMLDOM	Explains access to XMLType objects
DBMS_XMLGEN	Converts the results of a SQL query to a canonical XML format.
DBMS_XMLPARSER	Explains access to the contents and structure of XML documents.
DMBS_XMLQUERY	Provides database-to-XMLType functionality.
DBMS_XMLSAVE	Provides XML-to-database-type functionality.
DBMS_XMLSCHEMA	Explains procedures to register and delete XML schemas.
DBMS_XMLSTORE	Provides the ability to store XML data in relational tables.
DBMS_XPLAN	Describes how to format the output of the EXPLAIN PLAN command.
DBMS_XSLPROCESSOR	Explains access to the contents and structure of XML documents.
DEBUG_EXTPROC	Lets you debug external procedures on platforms with debuggers that attach to a running process.
HTF	Hypertext functions generate HTML tags.
HTMLDB_APPLICATION	Enables users to take advantage of global variables
HTMLDB_CUSTOM_AUTH	Enables users to create form elements dynamically based on a SQL query instead of creating individual items page by page.
HTMLDB_ITEM	Enables users to create form elements dynamically based on a SQL query instead of creating individual items page by page.
HTMLDB_UTIL	Provides utilities for getting and setting session state, getting files, checking authorizations for users, resetting different states for users, and also getting and setting preferences for users.
HTP	Hypertext procedures generate HTML tags.
OWA_CACHE	Provides an interface that enables the PL/SQL Gateway cache to improve the performance of PL/SQL web applications.
OWA_COOKIE	Provides an interface for sending and retrieving HTTP cookies from the client's browser.
OWA_CUSTOM	Provides a Global PLSQL Agent Authorization callback function
OWA_IMAGE	Provides an interface to access the coordinates where a user clicked on an image.
OWA_OPT_LOCK	Contains subprograms that impose optimistic locking strategies so as to prevent lost updates.
OWA_PATTERN	Provides an interface to locate text patterns within strings and replace the matched string with another string.
OWA_SEC	Provides an interface for custom authentication.

Table 1–1 (Cont.) Summary of Oracle Supplied PL/SQL Packages

Package Name	Description
OWA_TEXT	Contains subprograms used by OWA_PATTERN for manipulating strings. They are externalized so you can use them directly.
OWA_UTIL	Contains utility subprograms for performing operations such as getting the value of CGI environment variables, printing the data that is returned to the client, and printing the results of a query in an HTML table.
SDO_CS	Provides functions for coordinate system transformation.
SDO_GCDR	Contains the Oracle Spatial geocoding subprograms, which let you geocode unformatted postal addresses.
SDO_GEOM	Provides functions implementing geometric operations on spatial objects.
SDO_GEOR	Contains functions and procedures for the Spatial GeoRaster feature, which lets you store, index, query, analyze, and deliver raster image data and its associated Spatial vector geometry data and metadata.
SDO_GEOR_UTL	Contains utility functions and procedures for the Spatial GeoRaster feature, including those related to using triggers with GeoRaster data.
SDO_LRS	Provides functions for linear referencing system support.
SDO_MIGRATE	Provides functions for migrating spatial data from previous releases.
SDO_NET	Provides functions and procedures for working with data modeled as nodes and links in a network.
SDO_NET_MEM	Contains functions and procedures for performing editing and analysis operations on network data using a network memory object
SDO_SAM	Contains functions and procedures for spatial analysis and data mining.
SDO_TOPO	Provides procedures for creating and managing Spatial topologies.
SDO_TOPO_MAP	Contains subprograms for editing Spatial topologies using a cache (TopoMap object).
SDO_TUNE	Provides functions for selecting parameters that determine the behavior of the spatial indexing scheme used in Oracle Spatial.
SDO_UTIL	Provides utility functions and procedures for Oracle Spatial.
UTL_COLL	Enables PL/SQL programs to use collection locators to query and update.
UTL_COMPRESS	Provides a set of data compression utilities.
UTL_DBWS	Provides database web services.
UTL_ENCODE	Provides functions that encode RAW data into a standard encoded format so that the data can be transported between hosts.
UTL_FILE	Enables your PL/SQL programs to read and write operating system text files and provides a restricted version of standard operating system stream file I/O.

Table 1–1 (Cont.) Summary of Oracle Supplied PL/SQL Packages

Package Name	Description
UTL_HTTP	Enables HTTP callouts from PL/SQL and SQL to access data on the Internet or to call Oracle Web Server Cartridges.
UTL_I18N	Provides a set of services (Oracle Globalization Service) that help developers build multilingual applications.
UTL_INADDR	Provides a procedure to support internet addressing.
UTL_LMS	Retrieves and formats error messages in different languages.
UTL_MAIL	A utility for managing email which includes commonly used email features, such as attachments, CC, BCC, and return receipt.
UTL_NLA	Exposes a subset of the BLAS and LAPACK (Version 3.0) operations on vectors and matrices represented as VARRAYs
UTL_RAW	Provides SQL functions for RAW datatypes that concat, substr to and from RAWS.
UTL_RECOMP	Recompiles invalid PL/SQL modules, Java classes, indextypes and operators in a database, either sequentially or in parallel.
UTL_REF	Enables a PL/SQL program to access an object by providing a reference to the object.
UTL_SMTP	Provides PL/SQL functionality to send emails.
UTL_TCP	Provides PL/SQL functionality to support simple TCP/IP-based communications between servers and the outside world.
UTL_URL	Provides escape and unescape mechanisms for URL characters.
WPG_DOCLOAD	Provides an interface to download files, both BLOBs and BFILEs
ANYDATA TYPE	A self-describing data instance type containing an instance of the type plus a description
ANYDATASET TYPE	Contains a description of a given type plus a set of data instances of that type
ANYTYPE TYPE	Contains a type description of any persistent SQL type, named or unnamed, including object types and collection types; or, it can be used to construct new transient type descriptions
Oracle Streams AQ Types	Describes the types used in Advanced Queuing
Database URI Type	Contains URI Support, UriType Super Type, HttpUriType Subtype, DBUriType Subtype, XDBUriType Subtype, UriFactory Package
JMS TYPES	Describes JMS types so that a PL/SQL application can use JMS queues of JMS types
LOGICAL CHANGE RECORD TYPES	Describes LCR types, which are message payloads that contain information about changes to a database, used in Streams
interMedia ORDAudio Type	Supports the storage and management of audio data.

Table 1–1 (Cont.) Summary of Oracle Supplied PL/SQL Packages

Package Name	Description
`interMedia ORDDoc Type`	Supports the storage and management of heterogeneous media data including image, audio, and video.
`interMedia ORDImage Type`	Supports the storage, management, and manipulation of image data.
`interMedia ORDImageSignature Type`	Supports content-based retrieval of images (image matching).
`interMedia SQL/MM Still Image Type`	Provides support for the SQL/MM Still Image Standard, which lets you store, retrieve, and modify images in the database and locate images using visual predicates.
`interMedia ORDVideo Type`	Supports the storage and management of video data.
`RULES TYPES`	Describes the types used with rules, rule sets, and evaluation contexts
`XMLType`	Describes the types and functions used for native XML support in the server.

2

CTX_ADM

This Oracle Text package lets you administer servers and the data dictionary. Note that you must install this package in order to use it.

Documentation of CTX_ADM

For a complete description of this package within the context of Oracle Text, see `CTX_ADM` in the *Oracle Text Reference*.

3

CTX_CLS

This Oracle Text package enables generation of CTXRULE rules for a set of documents.

- Documentation of CTX_CLS

Documentation of CTX_CLS

For a complete description of this package within the context of Oracle Text, see `CTX_CLS` in the *Oracle Text Reference*.

4

CTX_DDL

This Oracle Text package lets you create and manage the preferences, section groups, and stoplists required for Text indexes. Note that you must install this package in order to use it.

Documentation of CTX_DDL

For complete description of this package within the context of Oracle Text, see `CTX_DDL` in the *Oracle Text Reference.*

5

CTX_DOC

This Oracle Text package lets you request document services. Note that you must install this package in order to use it.

Documentation of CTX_DOC

For a complete description of this package within the context of Oracle Text, see `CTX_DOC` in the *Oracle Text Reference*.

6

CTX_OUTPUT

This Oracle Text package lets you manage the index log. Note that you must install this package in order to use it.

Documentation of CTX_OUTPUT

For a complete description of this package within the context of Oracle Text, see `CTX_OUTPUT` in the *Oracle Text Reference*.

7

CTX_QUERY

This Oracle Text package lets you generate query feedback, count hits, and create stored query expressions. Note that you must install this package in order to use it.

Documentation of CTX_QUERY

For a complete description of this package within the context of Oracle Text, see `CTX_QUERY` in the *Oracle Text Reference*.

8

CTX_REPORT

This Oracle Text package lets you create various index reports. Note that you must install this package in order to use it.

Documentation of CTX_REPORT

For a complete description of this package within the context of Oracle Text, see `CTX_REPORT` in the *Oracle Text Reference*.

9

CTX_THES

This Oracle Text package lets you to manage and browse thesauri. Note that you must install this package in order to use it.

Documentation of CTX_THES

For a complete description of this package within the context of Oracle Text, see `CTX_THES` in the *Oracle Text Reference*.

10

CTX_ULEXER

This Oracle Text package is for use with the user-lexer. Note that you must install this package in order to use it.

Documentation of CTX_ULEXER

For a complete description of this package within the context of Oracle Text, see `CTX_ULEXER` in the *Oracle Text Reference.*

11

DBMS_ADVANCED_REWRITE

DBMS_ADVANCED_REWRITE contains interfaces for advanced query rewrite users. Using this package, you can create, drop, and maintain functional equivalence declarations for query rewrite.

> **See Also:** *Oracle Database Data Warehousing Guide* for more information about query rewrite

This chapter contains the following topics:

Using DBMS_ADVANCED_REWRITE

This section contains topics which relate to using the `DBMS_ADVANCED_REWRITE` package.

Security Model

No privileges to access these procedures are granted to anyone by default. To gain access to these procedures, you must connect as SYSDBA and explicitly grant execute access to the desired database administrators.

You can control security on this package by granting the EXECUTE privilege to selected database administrators or roles. For example, the user er can be given access to use this package by the following statement, executed as SYSDBA:

```
GRANT EXECUTE ON DBMS_ADVANCED_REWRITE TO er;
```

You may want to write a separate cover package on top of this package for restricting the alert names used. Instead of granting the EXECUTE privilege on the DBMS_ADVANCED_REWRITE package directly, you can then grant it to the cover package.

In addition, similar to the privilege required for regular materialized views, the user should be granted the privilege to create an equivalence. For example, the user er can be granted this privilege by executing the following statement as SYSDBA:

```
GRANT CREATE MATERIALIZED VIEW TO er;
```

Summary of DBMS_ADVANCED_REWRITE Subprograms

This table list the all the package subprograms in alphabetical order.

Table 11–1 DBMS_ADVANCED_REWRITE Package Subprograms

Subprogram	Description
ALTER_REWRITE_EQUIVALENCE Procedure on page 11-5	Changes the mode of the rewrite equivalence declaration to the mode you specify
BUILD_SAFE_REWRITE_EQUIVALENCE Procedure on page 11-6	Enables the rewrite of top-level materialized views using submaterialized views. Oracle Corporation does not recommend you directly use this procedure
DECLARE_REWRITE_EQUIVALENCE Procedures on page 11-7	Creates a declaration indicating that `source_stmt` is functionally equivalent to `destination_stmt` for as long as the equivalence declaration remains enabled, and that `destination_stmt` is more favorable in terms of performance
DROP_REWRITE_EQUIVALENCE Procedure on page 11-9	Drops the specified rewrite equivalence declaration
VALIDATE_REWRITE_EQUIVALENCE Procedure on page 11-10	Validates the specified rewrite equivalence declaration using the same validation method as described with the `validate` parameter

ALTER_REWRITE_EQUIVALENCE Procedure

This procedure changes the mode of the rewrite equivalence declaration to the mode you specify.

Syntax

```
DBMS_ADVANCED_REWRITE.ALTER_REWRITE_EQUIVALENCE (
   name        VARCHAR2,
   mode        VARCHAR2);
```

Parameters

Table 11–2 ALTER_REWRITE_EQUIVALENCE Procedure Parameters

Parameter	Description
name	A name for the equivalence declaration to alter. The name can be of the form owner.name, where owner complies with the rules for a schema name, and name compiles with the rules for a table name. Alternatively, a simple name that complies with the rules for a table name can be specified. In this case, the rewrite equivalence is altered in the current schema. The invoker must have the appropriate alter materialized view privileges to alter an equivalence declaration outside their own schema.
mode	The following modes are supported, in increasing order of power: disabled: Query rewrite does not use the equivalence declaration. Use this mode to temporarily disable use of the rewrite equivalence declaration. text_match: Query rewrite uses the equivalence declaration only in its text match modes. This mode is useful for simple transformations. general: Query rewrite uses the equivalence declaration in all of its transformation modes against the incoming request queries. However, query rewrite makes no attempt to rewrite the specified destination_query. recursive: Query rewrite uses the equivalence declaration in all of its transformation modes against the incoming request queries. Moreover, query rewrite further attempts to rewrite the specified destination_query for further performance enhancements whenever it uses the equivalence declaration. Oracle Corporation recommends you use the least powerful mode that is sufficient to solve your performance problem.

BUILD_SAFE_REWRITE_EQUIVALENCE Procedure

This procedure enables the rewrite and refresh of top-level materialized views using submaterialized views. It is provided for the exclusive use by scripts generated by the `DBMS_ADVISOR.TUNE_MVIEW` procedure. It is required to enable query rewrite and fast refresh when `DBMS_ADVISOR.TUNE_MVIEW` decomposes a materialized view into a top-level materialized view and one or more submaterialized views.

Oracle does not recommend you directly use the `BUILD_SAFE_REWRITE_EQUIVALENCE` procedure. You should use either the `DBMS_ADVISOR.TUNE_MVIEW` or the `DBMS_ADVANCED_REWRITE.CREATE_REWRITE_EQUIVALENCE` procedure as appropriate.

DECLARE_REWRITE_EQUIVALENCE Procedures

This procedure creates a declaration indicating that source_stmt is functionally equivalent to destination_stmt for as long as the equivalence declaration remains enabled, and that destination_stmt is more favorable in terms of performance. The scope of the declaration is system wide. The query rewrite engine uses such declarations to perform rewrite transformations in QUERY_REWRITE_INTEGRITY = trusted and stale_tolerated modes.

Syntax

```
DBMS_ADVANCED_REWRITE.DECLARE_REWRITE_EQUIVALENCE (
   name                 VARCHAR2,
   source_stmt          VARCHAR2,
   destination_stmt     VARCHAR2,
   validate             BOOLEAN    := TRUE,
   mode                 VARCHAR2   := 'TEXT_MATCH');

DBMS_ADVANCED_REWRITE.DECLARE_REWRITE_EQUIVALENCE (
   name                 VARCHAR2,
   source_stmt          CLOB,
   destination_stmt     CLOB,
   validate             BOOLEAN   := TRUE,
   mode                 VARCHAR2  := 'TEXT_MATCH');
```

Parameters

Table 11–3 DECLARE_REWRITE_EQUIVALENCE Procedure Parameters

Parameter	Description
name	A name for the equivalence declaration. The name can be of the form owner.name, where owner complies with the rules for a schema name, and name compiles with the rules for a table name. Alternatively, a simple name that complies with the rules for a table name can be specified. In this case, the rewrite equivalence is created in the current schema. The invoker must have the appropriate CREATE MATERIALIZED VIEW privileges to alter an equivalence declaration.
source_stmt	A sub-SELECT expression in either VARCHAR2 or CLOB format. This is the query statement that is the target of optimization.
destination_stmt	A sub-SELECT expression in either VARCHAR2 or CLOB format.
validate	A Boolean indicating whether to validate that the specified source_stmt is functionally equivalent to the specified destination_stmt. If validate is specified as TRUE, DECLARE_REWRITE_EQUIVALENCE evaluates the two sub-SELECTs and compares their results. If the results are not the same, DECLARE_REWRITE_EQUIVALENCE does not create the rewrite equivalence and returns an error condition. If FALSE, DECLARE_REWRITE_EQUIVALENCE does not validate the equivalence.

Table 11–3 (Cont.) DECLARE_REWRITE_EQUIVALENCE Procedure Parameters

Parameter	Description
mode	The following modes are supported, in increasing order of power: ■ disabled: Query rewrite does not use the equivalence declaration. Use this mode to temporarily disable use of the rewrite equivalence declaration. ■ text_match: Query rewrite uses the equivalence declaration only in its text match modes. This mode is useful for simple transformations. ■ general: Query rewrite uses the equivalence declaration in all of its transformation modes against the incoming request queries. However, query rewrite makes no attempt to rewrite the specified destination_query. ■ recursive: Query rewrite uses the equivalence declaration in all of its transformation modes against the incoming request queries. Moreover, query rewrite further attempts to rewrite the specified destination_query for further performance enhancements whenever it uses the equivalence declaration. Oracle recommends you use the least powerful mode that is sufficient to solve your performance problem.

Usage Notes

Query rewrite using equivalence declarations occurs simultaneously and in concert with query rewrite using materialized views. The same query rewrite engine is used for both. The query rewrite engine uses the same rewrite rules to rewrite queries using both equivalence declarations and materialized views. Because the rewrite equivalence represents a specific rewrite crafted by a sophisticated user, the query rewrite engine gives priority to rewrite equivalences over materialized views when it is possible to perform a rewrite with either a materialized view or a rewrite equivalence. For this same reason, the cost-based optimizer (specifically, cost-based rewrite) will not choose an unrewritten query plan over a query plan that is rewritten to use a rewrite equivalence even if the cost of the un-rewritten plan appears more favorable. Query rewrite matches properties of the incoming request query against the equivalence declaration's source_stmt or the materialized view's defining statement, respectively, and derives an equivalent relational expression in terms of the equivalence declaration's destination_stmt or the materialized view's container table, respectively.

DROP_REWRITE_EQUIVALENCE Procedure

This procedure drops the specified rewrite equivalence declaration.

Syntax

```
DBMS_ADVANCED_REWRITE.DROP_REWRITE_EQUIVALENCE (
   name         VARCHAR2);
```

Parameters

Table 11–4 DROP_REWRITE_EQUIVALENCE Procedure Parameters

Parameter	Description
`name`	A name for the equivalence declaration to drop. The name can be of the form `owner.name`, where owner complies with the rules for a schema name, and name compiles with the rules for a table name. Alternatively, a simple name that complies with the rules for a table name can be specified. In this case, the rewrite equivalence is dropped in the current schema. The invoker must have the appropriate drop materialized view privilege to drop an equivalence declaration outside their own schema.

VALIDATE_REWRITE_EQUIVALENCE Procedure

This procedure validates the specified rewrite equivalence declaration using the same validation method as described with the `VALIDATE` parameter in "VALIDATE_REWRITE_EQUIVALENCE Procedure" on page 11-10.

Syntax

```
DBMS_ADVANCED_REWRITE.VALIDATE_REWRITE_EQUIVALENCE (
   name         VARCHAR2);
```

Parameters

Table 11–5 VALIDATE_REWRITE_EQUIVALENCE Procedure Parameters

Parameter	Description
`name`	A name for the equivalence declaration to validate. The name can be of the form owner.name, where owner complies with the rules for a schema name, and name compiles with the rules for a table name. Alternatively, a simple name that complies with the rules for a table name can be specified. In this case, the rewrite equivalence is validated in the current schema. The invoker must have sufficient privileges to execute both the `source_stmt` and `destination_stmt` of the specified equivalence declaration.

12

DBMS_ADVISOR

DBMS_ADVISOR is part of the Server Manageability suite of Advisors, a set of expert systems that identifies and helps resolve performance problems relating to the various database server components.

See Also:

- *Oracle Database Administrator's Guide* for information regarding the Segment Advisor
- *Oracle Database Performance Tuning Guide* for information regarding the SQL Tuning Advisor and SQL Access Advisor
- *Oracle Database 2 Day DBA* and *Oracle Database Administrator's Guide* for information regarding the Undo Advisor

This chapter contains the following topics:

Using DBMS_ADVISOR

This section contains topics which relate to using the `DBMS_ADVISOR` package.

Security Model

Security on this package can be controlled by granting EXECUTE on this package to selected users or roles. You might want to write a cover package on top of this one that restricts the alert names used. EXECUTE privilege on this cover package can then be granted rather than on this package. In addition, there is an ADVISOR privilege, which is required by DBMS_ADVISOR procedures.

Summary of DBMS_ADVISOR Subprograms

Table 12–1 DBMS_ADVISOR Package Subprograms

Subprogram	Description	Used in
ADD_SQLWKLD_REF Procedure on page 12-6	Adds a workload reference to an Advisor task	SQL Access Advisor only
ADD_SQLWKLD_ STATEMENT Procedure on page 12-7	Adds a single statement to a workload	SQL Access Advisor only
CANCEL_TASK Procedure on page 12-9	Cancels a currently executing task operation	All Advisors
CREATE_FILE Procedure on page 12-10	Creates an external file from a PL/SQL CLOB variable, which is useful for creating scripts and reports	All Advisors
CREATE_OBJECT Procedure on page 12-11	Creates a new task object	All Advisors
CREATE_SQLWKLD Procedure on page 12-13	Creates a new workload object	SQL Access Advisor only
CREATE_TASK Procedures on page 12-14	Creates a new Advisor task in the repository	All Advisors
DELETE_SQLWKLD Procedure on page 12-16	Deletes an entire workload object	SQL Access Advisor only
DELETE_SQLWKLD_ REF Procedure on page 12-17	Deletes an entire workload object	SQL Access Advisor only
DELETE_SQLWKLD_ STATEMENT Procedure on page 12-18	Deletes one or more statements from a workload	SQL Access Advisor only
DELETE_TASK Procedure on page 12-19	Deletes the specified task from the repository	All Advisors
EXECUTE_TASK Procedure on page 12-20	Executes the specified task	All Advisors
GET_REC_ATTRIBUTES Procedure on page 12-21	Retrieves specific recommendation attributes from a task	All Advisors
GET_TASK_REPORT Function on page 12-22	Creates and returns a report for the specified task	All Advisors
GET_TASK_SCRIPT Function on page 12-23	Creates and returns an executable SQL script of the Advisor task's recommendations in a buffer	All Advisors
IMPLEMENT_TASK Procedure on page 12-25	Implements the tuning recommendations for a task	All Advisors
IMPORT_SQLWKLD_ SCHEMA Procedure on page 12-26	Imports data into a workload from the current SQL cache	SQL Access Advisor only
IMPORT_SQLWKLD_ SQLCACHE Procedure on page 12-28	Imports data into a workload from the current SQL cache	SQL Access Advisor only
IMPORT_SQLWKLD_ STS Procedure on page 12-30	Imports data into a workload from a SQL Tuning Set into a SQL workload data object	SQL Access Advisor only

Table 12–1 (Cont.) DBMS_ADVISOR Package Subprograms

Subprogram	Description	Used in
IMPORT_SQLWKLD_SUMADV Procedure on page 12-32	Imports data into a workload from the current SQL cache	SQL Access Advisor only
IMPORT_SQLWKLD_USER Procedure on page 12-34	Imports data into a workload from the current SQL cache	SQL Access Advisor only
INTERRUPT_TASK Procedure on page 12-36	Stops a currently executing task, ending its operations as it would at a normal exit, so that the recommendations are visible	All Advisors
MARK_RECOMMENDATION Procedure on page 12-37	Sets the `annotation_status` for a particular recommendation	All Advisors
QUICK_TUNE Procedure on page 12-38	Performs an analysis on a single SQL statement	All Advisors
RESET_TASK Procedure on page 12-40	Resets a task to its initial state	All Advisors
SET_DEFAULT_SQLWKLD_PARAMETER Procedure on page 12-41	Imports data into a workload from schema evidence	SQL Access Advisor only
SET_DEFAULT_TASK_PARAMETER Procedures on page 12-42	Modifies a default task parameter	All Advisors
SET_SQLWKLD_PARAMETER Procedure on page 12-43	Sets the value of a workload parameter	SQL Access Advisor only
SET_TASK_PARAMETER Procedure on page 12-49	Sets the specified task parameter value	All Advisors
TUNE_MVIEW Procedure on page 12-60	Shows how to decompose a materialized view into two or more materialized views or to restate the materialized view in a way that is more advantageous for fast refresh and query rewrite	SQL Access Advisor only
UPDATE_OBJECT Procedure on page 12-62	Updates a task object	All Advisors
UPDATE_REC_ATTRIBUTES Procedure on page 12-64	Updates an existing recommendation for the specified task	All Advisors
UPDATE_SQLWKLD_ATTRIBUTES Procedure on page 12-66	Updates a workload object	SQL Access Advisor only
UPDATE_SQLWKLD_STATEMENT Procedure on page 12-67	Updates one or more SQL statements in a workload	SQL Access Advisor only
UPDATE_TASK_ATTRIBUTES Procedure on page 12-69	Updates a task's attributes	All Advisors

ADD_SQLWKLD_REF Procedure

This procedure establishes a link between the current SQL Access Advisor task and a SQL Workload object. The link allows an advisor task to access interesting data for doing an analysis. The link also provides a stable view of the data. Once a connection between a SQL Access Advisor task and a SQL Workload object is made, the workload is protected from removal or modification.

Syntax

```
DBMS_ADVISOR.ADD_SQLWKLD_REF (
   task_name              IN VARCHAR2,
   workload_name          IN VARCHAR2);
```

Parameters

Table 12–2 ADD_SQLWKLD_REF Procedure Parameters

Parameter	Description
task_name	The SQL Access task name that uniquely identifies an existing task.
workload_name	The name of the workload object to be linked. Once a object has been linked to a task, it becomes read-only and cannot be deleted. There is no limit to the number of links to workload objects. To remove the link to the workload object, use the procedure DELETE_REFERENCE.

Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
  workload_name VARCHAR2(30);
BEGIN
  task_name := 'My Task';
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_REF(task_name, workload_name);
END;
/
```

ADD_SQLWKLD_STATEMENT Procedure

This procedure adds a single statement to the specified workload.

Syntax

```
DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT (
   workload_name          IN VARCHAR2,
   module                 IN VARCHAR2,
   action                 IN VARCHAR2,
   cpu_time               IN NUMBER := 0,
   elapsed_time           IN NUMBER := 0,
   disk_reads             IN NUMBER := 0,
   buffer_gets            IN NUMBER := 0,
   rows_processed         IN NUMBER := 0,
   optimizer_cost         IN NUMBER := 0,
   executions             IN NUMBER := 1,
   priority               IN NUMBER := 2,
   last_execution_date    IN DATE := 'SYSDATE',
   stat_period            IN NUMBER := 0,
   username               IN VARCHAR2,
   sql_text               IN CLOB);
```

Parameters

Table 12–3 ADD_SQLWKLD_STATEMENT Procedure Parameters

Parameter	Description
workload_name	The workload name that uniquely identifies an existing workload.
module	An optional business application module that will be associated with the SQL statement.
action	An optional application action that will be associated with the SQL statement.
cpu_time	The total CPU time in seconds that is consumed by the SQL statement.
elapsed_time	The total elapsed time in seconds that is consumed by the SQL statement.
disk_reads	The total disk-read operations that are consumed by the SQL statement.
buffer_gets	The total buffer-get operations that are consumed by the SQL statement.
rows_processed	The average number of rows processed by the SQL statement.
optimizer_cost	The optimizer's calculated cost value.
executions	The total execution count by the SQL statement. This value should be greater than zero.
priority	The relative priority of the SQL statement. The value must be one of the following: 1-HIGH, 2-MEDIUM, or 3-LOW.
last_execution_date	The date and time at which the SQL statement last executed. If the value is NULL, then the current date and time will be used.
stat_period	Time interval in seconds from which statement statistics were calculated.

Table 12–3 (Cont.) ADD_SQLWKLD_STATEMENT Procedure Parameters

Parameter	Description
username	The Oracle user name that executed the SQL statement. Because a username is an Oracle identifier, the username value must be entered exactly as it is stored in the server. For example, if the user SCOTT is the executing user, then you must provide the user identifier SCOTT in all uppercase letters. It will not recognize the user scott as a match for SCOTT.
sql_text	The complete SQL statement. To increase the quality of a recommendation, the SQL statement should not contain bind variables.

Usage Notes

A workload cannot be modified or deleted if it is currently referenced by an active task. A task is considered active if it is not in its initial state. See "RESET_TASK Procedure" on page 12-40 for directions on setting a task to its initial state.

The ADD_SQLWKLD_STATEMENT procedure accepts several parameters that may be ignored by the caller. cpu_time, elapsed_time, disk_reads, buffer_gets, and optimizer_cost are only used to sort workload data when actual analysis occurs, so actual values are only necessary when the order_list task parameter references a particular statistic.

To determine what statistics to provide when adding a new SQL statement to a workload, examine or set the task parameter order_list. The order_list parameter accepts any combination of the keys: buffer_gets, optimizer_cost, cpu_time, disk_reads, elapsed_time, executions, and priority. A typical setting of priority, optimizer_cost would indicate the SQL Access Advisor will sort the workload data by priority and optimizer_cost and process the highest cost statements first. Any statements added to the workload would need to include appropriate priority and optimizer_cost values. All other statistics can be defaulted or set to zero.

For the statistical keys referenced by the order_list task parameter, the actual parameter values should be reasonably accurate since they will be compared to other statements in the workload. If the caller is unable to estimate values, choose values that would determine its importance relative to other statements in the workload. For example, if the current statement is considered the most critical query in your business, then an appropriate value would be anything greater than all other values for the same statistic found in the workload.

Examples

```
DECLARE
  workload_name VARCHAR2(30);
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
                                     100,400,5041,103,640445,680000,2,
                                     1,SYSDATE,1,'SH','SELECT AVG(amount_sold)
FROM sh.sales');
END;
/
```

CANCEL_TASK Procedure

This procedure causes a currently executing operation to terminate. This call does a soft interrupt. It will not break into a low-level database access call like a hard interrupt such as `Ctrl-C`. The SQL Access Advisor periodically checks for soft interrupts and acts appropriately. As a result, this operation may take a few seconds to respond to a call.

Syntax

```
DBMS_ADVISOR.CANCEL_TASK (
   task_name      IN  VARCHAR2);
```

Parameters

Table 12–4 CANCEL_TASK Procedure Parameter

Parameter	Description
`task_name`	A valid Advisor task name that uniquely identifies an existing task.

Usage Notes

A cancel command effective restores the task to its condition prior to the start of the cancelled operation. Therefore, a cancelled task or data object cannot be resumed.

Because all Advisor task procedures are synchronous, to cancel an operation, you must use a separate database session.

Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
  workload_name VARCHAR2(30);
BEGIN
  task_name := 'My Task';
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
  DBMS_ADVISOR.CANCEL_TASK('My Task');
END;
/
```

CREATE_FILE Procedure

This procedure creates an external file from a PL/SQL CLOB variable, which is used for creating scripts and reports. CREATE_FILE accepts a CLOB input parameter and writes the character string contents to the specified file.

Syntax

```
DBMS_ADVISOR.CREATE_FILE (
   buffer       IN  CLOB,
   location     IN  VARCHAR2,
   filename     IN  VARCHAR2);
```

Parameters

Table 12–5 CREATE_FILE Procedure Parameters

Parameter	Description
buffer	A CLOB buffer containing report or script information.
location	Specifies the directory that will contain the new file. You must use the directory alias as defined by the CREATE DIRECTORY statement. The Advisor will translate the alias into the actual directory location.
filename	Specifies the output file to receive the script commands. The filename can only contain the name and an optional file type of the form filename.filetype.

Usage Notes

All formatting must be embedded within the CLOB.

The Oracle server restricts file access within Oracle Stored Procedures. This means that file locations and names must adhere to the known file permissions in the server.

Examples

```
CREATE DIRECTORY MY_DIR as '/homedir/user4/gssmith';
GRANT READ,WRITE ON DIRECTORY MY_DIR TO PUBLIC;

DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
  workload_name VARCHAR2(30);
BEGIN
  task_name := 'My Task';
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_REF(task_name, workload_name);
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
                                     100,400,5041,103,640445,680000,2,
                                     1,SYSDATE,1,'SH','SELECT AVG(amount_sold)
                                     FROM sh.sales');
  DBMS_ADVISOR.EXECUTE_TASK(task_name);
  DBMS_ADVISOR.CREATE_FILE(DBMS_ADVISOR.GET_TASK_SCRIPT(task_name),
    'MY_DIR','script.sql');

END;
/
```

CREATE_OBJECT Procedure

This procedure creates a new task object.

Syntax

```
DBMS_ADVISOR.CREATE_OBJECT (
   task_name         IN VARCHAR2,
   object_type       IN VARCHAR2,
   attr1             IN VARCHAR2 :=  NULL,
   attr2             IN VARCHAR2 :=  NULL,
   attr3             IN VARCHAR2 :=  NULL,
   attr4             IN CLOB     :=  NULL,
   object_id         OUT NUMBER);

DBMS_ADVISOR.CREATE_OBJECT (
   task_name         IN VARCHAR2,
   object_type       IN VARCHAR2,
   attr1             IN VARCHAR2 :=  NULL,
   attr2             IN VARCHAR2 :=  NULL,
   attr3             IN VARCHAR2 :=  NULL,
   attr4             IN CLOB     :=  NULL,
   attr5             IN VARCHAR2 :=  NULL,
   object_id         OUT NUMBER);
```

Parameters

Table 12–6 CREATE_OBJECT Procedure Parameters

Parameter	Description
task_name	A valid Advisor task name that uniquely identifies an existing task.
object_type	Specifies the external object type.
attr1	Advisor-specific data.
attr2	Advisor-specific data.
attr3	Advisor-specific data.
attr4	Advisor-specific data.
attr5	Advisor-specific data.
object_id	The advisor-assigned object identifier.

The attribute parameters have different values depending upon the object type. See *Oracle Database Administrator's Guide* for details regarding these parameters and object types.

Return Values

Returns the new object identifier.

Usage Notes

Task objects are typically used as input data for a particular advisor. Segment advice can be generated at the object, segment, or tablespace level. If for the object level, advice is generated on all partitions of the object (if the object is partitioned). The advice is not cascaded to any dependent objects. If for the segment level, advice can be obtained on a single segment, such as the partition or subpartition of a table, index, or

LOB column. If for a tablespace level, target advice for every segment in the tablespace will be generated.

See *Oracle Database Administrator's Guide* for further information regarding the Segment Advisor.

Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
  obj_id NUMBER;
BEGIN
  task_name := 'My Task';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
  DBMS_ADVISOR.CREATE_OBJECT (task_name,'SQL',NULL,NULL,NULL,
                              'SELECT * FROM SH.SALES',obj_id);
END;
/
```

CREATE_SQLWKLD Procedure

This procedure creates a new private SQL Workload object for the user. A SQL Workload object manages a SQL workload on behalf of the SQL Access Advisor. A SQL Workload object must exist prior to performing any other SQL Workload operations, such as importing or updating SQL statements.

Syntax

```
DBMS_ADVISOR.CREATE_SQLWKLD (
   workload_name            IN OUT VARCHAR2,
   description              IN VARCHAR2 := NULL,
   template                 IN VARCHAR2 := NULL,
   is_template              IN VARCHAR2 := 'FALSE');
```

Parameters

Table 12–7 CREATE_SQLWKLD Procedure Parameters

Parameter	Description
workload_name	A name that uniquely identifies the created workload. If not specified, the system will generate a unique name. Names can be up to 30 characters long.
description	Specifies an optional workload description. Descriptions can be up to 256 characters.
template	An optional SQL Workload name of an existing workload data object or data object template.
is_template	An optional value that enables you to set the newly created workload as a template. Valid values are TRUE and FALSE.

Return Values

The SQL Access Advisor returns a unique workload object identifier number that must be used for subsequent activities within the new SQL Workload object.

Usage Notes

By default, workload objects are created using built-in default settings. To create a workload using the parameter settings of an existing workload or workload template, the user may specify an existing workload name.

Once a SQL Workload object is present, it can then be referenced by one or more SQL Access Advisor tasks using the ADD_SQLWKLD_REF procedure.

Examples

```
DECLARE
  workload_name VARCHAR2(30);
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
END;
/
```

CREATE_TASK Procedures

This procedure creates a new Advisor task in the repository.

Syntax

```
DBMS_ADVISOR.CREATE_TASK (
   advisor_name          IN VARCHAR2,
   task_id               OUT NUMBER,
   task_name             IN OUT VARCHAR2,
   task_desc             IN VARCHAR2 := NULL,
   template              IN VARCHAR2 := NULL,
   is_template           IN VARCHAR2 := 'FALSE',
   how_created           IN VARCHAR2 := NULL);

DBMS_ADVISOR.CREATE_TASK (
   advisor_name          IN VARCHAR2,
   task_name             IN VARCHAR2,
   task_desc             IN VARCHAR2 := NULL,
   template              IN VARCHAR2 := NULL,
   is_template           IN VARCHAR2 := 'FALSE',
   how_created           IN VARCHAR2 := NULL);
```

Parameters

Table 12–8 CREATE_TASK Procedure Parameters

Parameter	Description
advisor_name	Specifies the unique advisor name as defined in the view DBA_ADVISOR_DEFINITIONS.
task_id	A number that uniquely identifies the created task. The number is generated by the procedure and returned to the user.
task_name	Specifies a new task name. Names must be unique among all tasks for the user. When using the second form of the CREATE_TASK syntax listed above (with OUT), a unique name can be generated. Names can be up to 30 characters long.
task_desc	Specifies an optional task description. Descriptions can be up to 256 characters in length.
template	An optional task name of an existing task or task template. To specify built-in SQL Access Advisor templates, use the template name as described earlier.
is_template	An optional value that allows the user to set the newly created task as template. Valid values are: TRUE and FALSE.
how_created	An optional value that identifies how the source was created.

Return Values

Returns a unique task ID number and a unique task name if one is not specified.

Usage Notes

A task must be associated with an advisor, and once the task has been created, it is permanently associated with the original advisor. By default, tasks are created using built-in default settings. To create a task using the parameter settings of an existing task or task template, the user may specify an existing task name.

For the SQL Access Advisor, use the identifier `DBMS_ADVISOR.SQLACCESS_ADVISOR` as the `advisor_name`.

The SQL Access Advisor provides three built-in task templates, using the following constants:

- `DBMS_ADVISOR.SQLACCESS_OLTP`

  Parameters are preset to favor an OLTP application environment.

- `DBMS_ADVISOR.SQLACCESS_WAREHOUSE`

  Parameters are preset to favor a data warehouse application environment.

- `DBMS_ADVISOR.SQLACCESS_GENERAL`

  Parameters are preset to favor a hybrid application environment where both OLTP and data warehouse operations may occur. For the SQL Access Advisor, this is the default template.

Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
BEGIN
  task_name := 'My Task';
  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
END;
/
```

DELETE_SQLWKLD Procedure

This procedure deletes an existing SQL Workload object from the repository.

Syntax

```
DBMS_ADVISOR.DELETE_SQLWKLD (
   workload_name        IN VARCHAR2);
```

Parameters

Table 12–9 DELETE_SQLWKLD Procedure Parameters

Parameter	Description
workload_name	The workload object name that uniquely identifies an existing workload. The wildcard % is supported as a WORKLOAD_NAME. The rules of use are identical to the LIKE operator. For example, to delete all tasks for the current user, use the wildcard % as the WORKLOAD_NAME. If a wildcard is provided, the DELETE_SQLWKLD operation will not delete any workloads marked as READ_ONLY or TEMPLATE.

Usage Notes

A workload cannot be modified or deleted if it is currently referenced by an active task. A task is considered active if it is not in its initial state. See the "RESET_TASK Procedure" on page 12-40 to set a task to its initial state.

Examples

```
DECLARE
  workload_name VARCHAR2(30);
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.DELETE_SQLWKLD(workload_name);
END;
/
```

DELETE_SQLWKLD_REF Procedure

This procedure removes a link between the current SQL Access task and a SQL Workload data object.

Syntax

```
DBMS_ADVISOR.DELETE_SQLWKLD_REF (
   task_name              IN VARCHAR2,
   workload_name          IN NUMBER);
```

Parameters

Table 12–10 DELETE_SQLWKLD_REF Procedure Parameters

Parameter	Description
task_name	The SQL Access task name that uniquely identifies an existing task.
workload_name	The name of the workload object to be unlinked. The wildcard % is supported as a workload_name. The rules of use are identical to the LIKE operator. For example, to remove all links to workload objects, use the wildcard % as the workload_name.

Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
  workload_name VARCHAR2(30);
BEGIN
  task_name := 'My Task';
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_REF(task_name, workload_name);
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
                                     100,400,5041,103,640445,680000,2,
                                     1,SYSDATE,1,'SH','SELECT AVG(amount_sold)
                                     FROM sh.sales');
  DBMS_ADVISOR.DELETE_SQLWKLD_REF(task_name, workload_name);
END;
/
```

DELETE_SQLWKLD_STATEMENT Procedure

This procedure deletes one or more statements from a workload.

Syntax

```
DBMS_ADVISOR.DELETE_SQLWKLD_STATEMENT (
   workload_name        IN VARCHAR2,
   sql_id               IN NUMBER);

DBMS_ADVISOR.DELETE_SQLWKLD_STATEMENT (
   workload_name        IN VARCHAR2,
   search               IN VARCHAR2,
   deleted              OUT NUMBER);
```

Parameters

Table 12–11 DELETE_SQLWKLD_STATEMENT Procedure Parameters

Parameter	Description
workload_name	The workload object name that uniquely identifies an existing workload.
sql_id	The Advisor-generated identifier number that is assigned to the statement. To specify all workload statements, use the constant ADVISOR_ALL.
search	Disabled.
deleted	Returns the number of statements deleted by the searched deleted operation.

Usage Notes

A workload cannot be modified or deleted if it is currently referenced by an active task. A task is considered active if it is not in its initial state. See the "RESET_TASK Procedure" on page 12-40 to set a task to its initial state.

Examples

```
DECLARE
  workload_name VARCHAR2(30);
  deleted NUMBER;
  id NUMBER;
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'YEARLY', 'ROLLUP',
                                     100,400,5041,103,640445,680000,2,
                                     1,SYSDATE,1,'SH','SELECT AVG(amount_sold)
                                     FROM sh.sales');

   SELECT sql_id INTO id FROM USER_ADVISOR_SQLW_STMTS
   WHERE workload_name = 'My Workload';

  DBMS_ADVISOR.DELETE_SQLWKLD_STATEMENT(workload_name, id);
END;
/
```

DELETE_TASK Procedure

This procedure deletes an existing task from the repository.

Syntax

```
DBMS_ADVISOR.DELETE_TASK (
   task_name          IN VARCHAR2);
```

Parameters

Table 12–12 DELETE_TASK Procedure Parameters

Parameter	Description
task_name	A single Advisor task name that will be deleted from the repository. The wildcard % is supported as a TASK_NAME. The rules of use are identical to the LIKE operator. For example, to delete all tasks for the current user, use the wildcard % as the TASK_NAME. If a wildcard is provided, the DELETE_TASK operation will not delete any tasks marked as READ_ONLY or TEMPLATE.

Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
BEGIN
  task_name := 'My Task';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
  DBMS_ADVISOR.DELETE_TASK(task_name);
END;
/
```

EXECUTE_TASK Procedure

This procedure performs the Advisor analysis or evaluation for the specified task.

Syntax

```
DBMS_ADVISOR.EXECUTE_TASK (
   task_name          IN VARCHAR2);
```

Parameters

Table 12–13 EXECUTE_TASK Procedure Parameters

Parameter	Description
task_name	The task name that uniquely identifies an existing task.

Usage Notes

Task execution is a synchronous operation. Control will not be returned to the caller until the operation has completed, or a user-interrupt was detected.

Upon return, you can check the DBA_ADVISOR_LOG table for the execution status.

Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
  workload_name VARCHAR2(30);
BEGIN
  task_name := 'My Task';
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_REF(task_name, workload_name);
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
                                     100,400,5041,103,640445,680000,2,
                                     1,SYSDATE,1,'SH','SELECT AVG(amount_sold)
                                     FROM sh.sales');
  DBMS_ADVISOR.EXECUTE_TASK(task_name);
END;
/
```

GET_REC_ATTRIBUTES Procedure

This procedure retrieves a specified attribute of a new object as recommended by Advisor analysis.

Syntax

```
DBMS_ADVISOR.GET_REC_ATTRIBUTES (
   workload_name         IN VARCHAR2,
   rec_id                IN NUMBER,
   action_id             IN NUMBER,
   attribute_name        IN VARCHAR2,
   value                 OUT VARCHAR2,
   owner_name            IN VARCHAR2 := NULL);
```

Parameters

Table 12–14 GET_REC_ATTRIBUTES Procedure Parameters

Parameter	Description
task_name	The task name that uniquely identifies an existing task.
rec_id	The Advisor-generated identifier number that is assigned to the recommendation.
action_id	The Advisor-generated action identifier that is assigned to the particular command.
attribute_name	Specifies the attribute to change.
value	The buffer to receive the requested attribute value.
owner_name	Optional owner name of the target task. This permits access to task data not owned by the current user.

Return Values

The requested attribute value is returned in the VALUE argument.

Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
  workload_name VARCHAR2(30);
  attribute VARCHAR2(100);
BEGIN
  task_name := 'My Task';
  workload_name := 'My Workload';
  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_REF(task_name, workload_name);
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
                                     100,400,5041,103,640445,680000,2,
                                     1,SYSDATE,1,'SH','SELECT AVG(amount_sold)
                                     FROM sh.sales WHERE promo_id = 10');
  DBMS_ADVISOR.EXECUTE_TASK(task_name);
  DBMS_ADVISOR.GET_REC_ATTRIBUTES(task_name, 1, 1, 'NAME', attribute);
END;
/
```

GET_TASK_REPORT Function

This function creates and returns a report for the specified task.

Syntax

```
DBMS_ADVISOR.GET_TASK_REPORT (
   task_name      IN VARCHAR2,
   type           IN VARCHAR2 := 'TEXT',
   level          IN VARCHAR2 := 'TYPICAL',
   section        IN VARCHAR2 := 'ALL',
   owner_name     IN VARCHAR2 := NULL)
RETURN CLOB;
```

Parameters

Table 12–15 GET_TASK_REPORT Function Parameters

Parameter	Description
task_name	The name of the task from which the script will be created.
type	The only valid value is TEXT.
level	The possible values are BASIC, TYPICAL, and ALL.
section	Advisor-specific report sections.
owner_name	Owner of the task. If specified, the system will check to see if the current user has read privileges to the task data.

Return Values

Returns the buffer receiving the script.

GET_TASK_SCRIPT Function

This function creates a SQL*Plus-compatible SQL script and sends the output to file. The script will contain all of the accepted recommendations from the specified task.

Syntax

```
DBMS_ADVISOR.GET_TASK_SCRIPT (
   task_name          IN VARCHAR2
   type               IN VARCHAR2 := 'IMPLEMENTATION',
   rec_id             IN NUMBER := NULL,
   act_id             IN NUMBER := NULL,
   owner_name         IN VARCHAR2 := NULL)
RETURN CLOB;
```

Parameters

Table 12–16 GET_TASK_SCRIPT Function Parameters

Parameter	Description
task_name	The task name that uniquely identifies an existing task.
type	Specifies the type of script to generate. The possible values are IMPLEMENTATION and UNDO.
rec_id	An optional recommendation identifier number that can be used to extract a subset of the implementation script. A zero or the value DBMS_ADVISOR.ADVISOR_ALL indicates all accepted recommendations would be included. The default is to include all accepted recommendations for the task.
act_id	Optional action identifier number that can be used to extract a single action as a DDL command. A zero or the value DBMS_ADVISOR.ADVISOR_ALL indicates all actions for the recommendation would be included. The default is to include all actions for a recommendation.
owner_name	Optional task owner name.

Return Values

Returns the script as a CLOB buffer.

Usage Notes

Though the script is ready to execute, Oracle recommends that the user review the script for acceptable locations for new materialized views and indexes.

For a recommendation to appear in a generated script, it must be marked as accepted.

Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
  workload_name VARCHAR2(30);
  buf CLOB;
BEGIN
  task_name := 'My Task';
  workload_name := 'My Workload';
```

```
  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_REF(task_name, workload_name);
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
                                     100,400,5041,103,640445,680000,2,
                                     1,SYSDATE,1,'SH','SELECT AVG(amount_sold)
                                     FROM sh.sales');
  DBMS_ADVISOR.EXECUTE_TASK(task_name);
    buf := DBMS_ADVISOR.GET_TASK_SCRIPT(task_name);
END;
/
```

IMPLEMENT_TASK Procedure

Implements the recommendations of the specified task.

Syntax

```
DBMS_ADVISOR.IMPLEMENT_TASK (
   task_name          IN VARCHAR2,
   rec_id             IN NUMBER := NULL,
   exit_on_error      IN BOOLEAN := NULL);
```

Parameters

Table 12–17 IMPLEMENT_TASK Procedure Parameters

Parameter	Description
task_name	The name of the task.
rec_id	An optional recommendation ID.
exit_on_error	An optional boolean to exit on the first error.

IMPORT_SQLWKLD_SCHEMA Procedure

This procedure constructs and loads a SQL workload based on schema evidence. The workload is also referred to as a hypothetical workload.

Syntax

```
DBMS_ADVISOR.IMPORT_SQLWKLD_SCHEMA (
   workload_name         IN VARCHAR2,
   import_mode           IN VARCHAR2 := 'NEW',
   priority              IN NUMBER := 2,
   saved_rows            OUT NUMBER,
   failed_rows           OUT NUMBER);
```

Parameters

Table 12–18 IMPORT_SQLWKLD_SCHEMA Procedure Parameters

Parameter	Description
workload_name	The workload object name that uniquely identifies an existing workload.
import_mode	Specifies the action to be taken when storing the workload. Possible values are: ■ APPEND Indicates that the collected workload will be added to any existing workload in the task. ■ NEW Indicates that the collected workload will be the exclusive workload for the task. If an existing workload is found, an exception will be thrown. ■ REPLACE Indicates the collected workload will be the exclusive workload for the task. If an existing workload is found, it will be deleted prior to saving the new workload. The default value is NEW.
priority	Specifies the application priority for each statement that is saved in the workload object. The value must be one of the following: 1-HIGH, 2-MEDIUM, or 3-LOW.
failed_rows	Returns the number or rows that were not saved due to syntax or validation errors
saved_rows	Returns the number of rows actually saved in the repository.

Return Values

This call returns the number of rows saved and failed as output parameters.

Usage Notes

To successfully import a hypothetical workload, the target schemas must contain dimensions.

If the VALID_TABLE_LIST parameter is not set, the search space may become very large and require a significant amount of time to complete. Oracle Corporation recommends that you limit your search space to specific set of tables.

If a task contains valid recommendations from a prior run, adding or modifying task will mark the task as invalid, preventing the viewing and reporting of potentially valuable recommendation data.

Examples

```
DECLARE
  workload_name VARCHAR2(30);
  saved NUMBER;
  failed NUMBER;
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.SET_SQLWKLD_PARAMETER(workload_name,'VALID_TABLE_LIST','SH.%');
  DBMS_ADVISOR.IMPORT_SQLWKLD_SCHEMA(workload_name, 'REPLACE', 1, saved,
     failed);
END;
/
```

IMPORT_SQLWKLD_SQLCACHE Procedure

This procedure creates a SQL workload from the current contents of the server's SQL cache.

Syntax

```
DBMS_ADVISOR.IMPORT_SQLWKLD_SQLCACHE (
   workload_name         IN VARCHAR2,
   import_mode           IN VARCHAR2 := 'NEW',
   priority              IN NUMBER := 2,
   saved_rows            OUT NUMBER,
   failed_rows           OUT NUMBER);
```

Parameters

Table 12–19 IMPORT_SQLWKLD_SQLCACHE Procedure Parameters

Parameter	Description
workload_ name	The workload object name that uniquely identifies an existing workload.
import_mode	Specifies the action to be taken when storing the workload. Possible values are: ▪ APPEND Indicates that the collected workload will be added to any existing workload in the task. ▪ NEW Indicates that the collected workload will be the exclusive workload for the task. If an existing workload is found, an exception will be thrown. ▪ REPLACE Indicates the collected workload will be the exclusive workload for the task. If an existing workload is found, it will be deleted prior to saving the new workload. The default value is NEW.
priority	Specifies the application priority for each statement that is saved in the workload object. The value must be one of the following 1-HIGH, 2-MEDIUM, or 3-LOW.
saved_rows	Returns the number of rows saved as output parameters.
failed_rows	Returns the number of rows that were not saved due to syntax or validation errors.

Return Values

This call returns the number of rows saved and failed as output parameters.

Usage Notes

A workload cannot be modified or deleted if it is currently referenced by an active task. A task is considered active if it is not in its initial state. See "RESET_TASK Procedure" on page 12-40 to set a task to its initial state.

Examples

```
DECLARE
  workload_name VARCHAR2(30);
  saved NUMBER;
  failed NUMBER;
```

```
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.SET_SQLWKLD_PARAMETER(workload_name,'VALID_TABLE_LIST','SH.%');
  DBMS_ADVISOR.IMPORT_SQLWKLD_SQLCACHE(workload_name, 'REPLACE', 1, saved,
     failed);
END;
/
```

IMPORT_SQLWKLD_STS Procedure

This procedure loads a SQL workload from an existing SQL Tuning Set. A SQL Tuning Set is typically created from the server workload repository using various time and data filters.

Syntax

```
DBMS_ADVISOR.IMPORT_SQLWKLD_STS (
   workload_name         IN VARCHAR2,
   sts_name              IN VARCHAR2,
   import_mode           IN VARCHAR2 := 'NEW',
   priority              IN NUMBER := 2,
   saved_rows            OUT NUMBER,
   failed_rows           OUT NUMBER);

DBMS_ADVISOR.IMPORT_SQLWKLD_STS (
   workload_name         IN VARCHAR2,
   sts_owner             IN VARCHAR2,
   sts_name              IN VARCHAR2,
   import_mode           IN VARCHAR2 := 'NEW',
   priority              IN NUMBER := 2,
   saved_rows            OUT NUMBER,
   failed_rows           OUT NUMBER);
```

Parameters

Table 12–20 IMPORT_SQLWKLD_STS Procedure Parameters

Parameter	Description
workload_name	The workload object name that uniquely identifies an existing workload.
sts_owner	The optional owner of the SQL Tuning Set.
sts_name	The name of an existing SQL Tuning Set workload from which the data will be imported. If the sts_owner value is not provided, the owner will default to the current user.
import_mode	Specifies the action to be taken when storing the workload. Possible values are: ■ APPEND Indicates that the collected workload will be added to any existing workload in the task. ■ NEW Indicates that the collected workload will be the exclusive workload for the task. If an existing workload is found, an exception will be thrown. ■ REPLACE Indicates the collected workload will be the exclusive workload for the task. If an existing workload is found, it will be deleted prior to saving the new workload. The default value is NEW.
priority	Specifies the application priority for each statement that is saved in the workload object. The value must be one of the following: 1-HIGH, 2-MEDIUM, or 3-LOW. The default value is 2.
saved_rows	Returns the number of rows actually saved in the repository.
failed_rows	Returns the number of rows that were not saved due to syntax or validation errors.

Return Values

This call returns the number of rows saved and failed as output parameters.

Usage Notes

A workload cannot be modified or deleted if it is currently referenced by an active task. A task is considered active if it is not in its initial state. See "RESET_TASK Procedure" on page 12-40 to set a task to its initial state.

Examples

```
DECLARE
  workload_name VARCHAR2(30);
  saved NUMBER;
  failed NUMBER;
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.SET_SQLWKLD_PARAMETER(workload_name,'VALID_TABLE_LIST','SH.%');
  DBMS_ADVISOR.IMPORT_SQLWKLD_STS(workload_name, 'MY_SQLSET', 'REPLACE', 1,
     saved, failed);
END;
/
```

IMPORT_SQLWKLD_SUMADV Procedure

This procedure collects a SQL workload from a Summary Advisor workload. This procedure is intended to assist Oracle9*i* Database Summary Advisor users in the migration to SQL Access Advisor.

Syntax

```
DBMS_ADVISOR.IMPORT_SQLWKLD_SUMADV (
   workload_name         IN VARCHAR2,
   import_mode           IN VARCHAR2 := 'NEW',
   priority              IN NUMBER := 2,
   sumadv_id             IN NUMBER,
   saved_rows            OUT NUMBER,
   failed_rows           OUT NUMBER);
```

Parameters

Table 12–21 IMPORT_SQLWKLD_SUMADV Procedure Parameters

Parameter	Description
workload_name	The workload object name that uniquely identifies an existing workload.
import_mode	Specifies the action to be taken when storing the workload. Possible values are: ■ APPEND Indicates that the collected workload will be added to any existing workload in the task. ■ NEW Indicates that the collected workload will be the exclusive workload for the task. If an existing workload is found, an exception will be thrown. ■ REPLACE Indicates the collected workload will be the exclusive workload for the task. If an existing workload is found, it will be deleted prior to saving the new workload. The default value is NEW.
priority	Specifies the default application priority for each statement that is saved in the workload object. If a Summary Advisor workload statement contains a priority of zero, the default priority will be applied. If the workload statement contains a valid priority, then the Summary Advisor priority will be converted to a comparable SQL Access Advisor priority. The value must be one of the following: 1-HIGH, 2-MEDIUM, or 3-LOW.
sumadv_id	Specifies the Summary Advisor workload identifier number.
saved_rows	Returns the number of rows actually saved in the repository.
failed_rows	Returns the number of rows that were not saved due to syntax or validation errors.

Return Values

This call returns the number of rows saved and failed as output parameters.

Usage Notes

A workload cannot be modified or deleted if it is currently referenced by an active task. A task is considered active if it is not in its initial state. See "RESET_TASK Procedure" on page 12-40 to set a task to its initial state.

Examples

```
DECLARE
  workload_name VARCHAR2(30);
  saved NUMBER;
  failed NUMBER;
  sumadv_id NUMBER;
BEGIN
  workload_name := 'My Workload';
  sumadv_id := 394;

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.SET_SQLWKLD_PARAMETER(workload_name,'VALID_TABLE_LIST','SH.%');
  DBMS_ADVISOR.IMPORT_SQLWKLD_SUMADV(workload_name, 'REPLACE', 1, sumadv_id,
    saved, failed);
END;
/
```

IMPORT_SQLWKLD_USER Procedure

This procedure collects a SQL workload from a specified user table.

Syntax

```
DBMS_ADVISOR.IMPORT_SQLWKLD_USER (
   workload_name          IN VARCHAR2,
   import_mode            IN VARCHAR2 := 'NEW',
   owner_name             IN VARCHAR2,
   table_name             IN VARCHAR2,
   saved_rows             OUT NUMBER,
   failed_rows            OUT NUMBER);
```

Parameters

Table 12–22 IMPORT_SQLWKLD_USER Procedure Parameters

Parameter	Description
workload_name	The workload object name that uniquely identifies an existing workload.
import_mode	Specifies the action to be taken when storing the workload. Possible values are: ■ APPEND Indicates that the collected workload will be added to any existing workload in the task. ■ NEW Indicates that the collected workload will be the exclusive workload for the task. If an existing workload is found, an exception will be thrown. ■ REPLACE Indicates the collected workload will be the exclusive workload for the task. If an existing workload is found, it will be deleted prior to saving the new workload. The default value is NEW.
owner_name	Specifies the owner name of the table or view from which workload data will be collected.
table_name	Specifies the name of the table or view from which workload data will be collected.
saved_rows	Returns the number of rows actually saved in the workload object.
failed_rows	Returns the number of rows that were not saved due to syntax or validation errors.

Return Values

This call returns the number of rows saved and failed as output parameters.

Usage Notes

A workload cannot be modified or deleted if it is currently referenced by an active task. A task is considered active if it is not in its initial state. See "RESET_TASK Procedure" on page 12-40 to set a task to its initial state.

Examples

```
DECLARE
  workload_name VARCHAR2(30);
  saved NUMBER;
  failed NUMBER;
```

```
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.SET_SQLWKLD_PARAMETER(workload_name,'VALID_TABLE_LIST','SH.%');
  DBMS_ADVISOR.IMPORT_SQLWKLD_USER(workload_name, 'REPLACE', 'SH',
    'USER_WORKLOAD', saved, failed);
END;
/
```

INTERRUPT_TASK Procedure

This procedure stops a currently executing task. The task will end its operations as it would at a normal exit. The user will be able to access any recommendations that exist to this point.

Syntax

```
DBMS_ADVISOR.INTERRUPT_TASK (
   task_name          IN VARCHAR2);
```

Parameters

Table 12–23 INTERRUPT_TASK Procedure Parameters

Parameter	Description
task_name	A single Advisor task name that will be interrupted.

Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
BEGIN
  task_name := 'My Task';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
  DBMS_ADVISOR.EXECUTE_TASK(task_name);
END;
/
```

While this session is executing its task, you can interrupt the task from a second session using the following statement:

```
BEGIN
  DBMS_ADVISOR.INTERRUPT_TASK('My Task');
END;
/
```

MARK_RECOMMENDATION Procedure

This procedure marks a recommendation for import or implementation.

Syntax

```
DBMS_ADVISOR.MARK_RECOMMENDATION (
   task_name          IN VARCHAR2
   id                 IN NUMBER,
   action             IN VARCHAR2);
```

Parameters

Table 12–24 MARK_RECOMMENDATION Procedure Parameters

Parameter	Description
task_name	Name of the task.
id	The recommendation identifier number assigned by the Advisor.
action	The recommendation action setting. The possible actions are: ■ ACCEPT Marks the recommendation as accepted. With this setting, the recommendation will appear in implementation and undo scripts. ■ IGNORE Marks the recommendation as ignore. With this setting, the recommendation will not appear in an implementation or undo script. ■ REJECT Marks the recommendation as rejected. With this setting, the recommendation will not appear in any implementation or undo scripts.

Usage Notes

For a recommendation to be implemented, it must be marked as accepted. By default, all recommendations are considered accepted and will appear in any generated scripts.

Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
  workload_name VARCHAR2(30);
  attribute VARCHAR2(100);
  rec_id NUMBER;
BEGIN
  task_name := 'My Task';
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_REF(task_name, workload_name);
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
                                     100,400,5041,103,640445,680000,2,
                                     1,SYSDATE,1,'SH','SELECT AVG(amount_sold)
                                     FROM sh.sales WHERE promo_id = 10');
  DBMS_ADVISOR.EXECUTE_TASK(task_name);

  rec_id := 1;
  DBMS_ADVISOR.MARK_RECOMMENDATION(task_name, rec_id, 'REJECT');
END;
/
```

QUICK_TUNE Procedure

This procedure performs an analysis and generates recommendations for a single SQL statement.

This provides a shortcut method of all necessary operations to analyze the specified SQL statement. The operation creates a task using the specified task name. The task will be created using a specified Advisor task template. Finally, the task will be executed and the results will be saved in the repository.

Syntax

```
DBMS_ADVISOR.QUICK_TUNE (
   advisor_name           IN VARCHAR2,
   task_name              IN VARCHAR2,
   attr1                  IN CLOB,
   attr2                  IN VARCHAR2 := NULL,
   attr3                  IN NUMBER := NULL,
   task_or_template       IN VARCHAR2 := NULL);
```

Parameters

Table 12–25 QUICK_TUNE Procedure Parameters

Parameter	Description
advisor_name	Name of the Advisor that will perform the analysis.
task_name	Name of the task.
attr1	Advisor-specific attribute in the form of a CLOB variable.
attr2	Advisor-specific attribute in the form of a VARCHAR2 variable.
attr3	Advisor-specific attribute in the form of a NUMBER.
task_or_template	An optional task name of an existing task or task template.

Usage Notes

If indicated by the user, the final recommendations can be implemented by the procedure.

The task will be created using either a specified SQL Access task template or the built-in default template of SQLACCESS_GENERAL. The workload will only contain the specified statement, and all task parameters will be defaulted.

attr1 must be the single SQL statement to tune. For the SQL Access Advisor, attr2 is the user who would execute the single statement. If omitted, the current user will be used.

Examples

```
DECLARE
  task_name VARCHAR2(30);
BEGIN
  task_name := 'My Task';

  DBMS_ADVISOR.QUICK_TUNE(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_name,
                  'SELECT AVG(amount_sold) FROM sh.sales WHERE promo_id=10');
END;
/
```

RESET_SQLWKLD Procedure

This procedure resets a workload to its initial starting point. This has the effect of removing all journal messages, log messages, and recalculating necessary volatility and usage statistics.

Syntax

```
DBMS_ADVISOR.RESET_SQLWKLD (
   workload_name        IN VARCHAR2);
```

Parameters

Table 12–26 RESET_SQLWKLD Procedure Parameters

Parameter	Description
workload_name	The SQL Workload object name that uniquely identifies an existing workload.

Usage Notes

RESET_SQLWKLD should be executed after any workload adjustments such as adding or removing SQL statements.

Examples

```
DECLARE
  workload_name VARCHAR2(30);
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
                                     100,400,5041,103,640445,680000,2,
                                     1,SYSDATE,1,'SH','SELECT AVG(amount_sold)
                                     FROM sh.sales WHERE promo_id = 10');

  DBMS_ADVISOR.RESET_SQLWKLD(workload_name);
END;
/
```

RESET_TASK Procedure

This procedure resets a task to its initial state. All intermediate and recommendation data will be removed from the task. The task status will be set to `INITIAL`.

Syntax

```
DBMS_ADVISOR.RESET_TASK (
   task_name          IN VARCHAR2);
```

Parameters

Table 12–27 RESET_TASK Procedure Parameters

Parameter	Description
`task_name`	The task name that uniquely identifies an existing task.

Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
  workload_name VARCHAR2(30);
BEGIN
  task_name := 'My Task';
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_REF(task_name, workload_name);
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
                                     100,400,5041,103,640445,680000,2,
                                     1,SYSDATE,1,'SH','SELECT AVG(amount_sold)
                                     FROM sh.sales WHERE promo_id = 10');
  DBMS_ADVISOR.EXECUTE_TASK(task_name);

  DBMS_ADVISOR.RESET_TASK(task_name);
END;
/
```

SET_DEFAULT_SQLWKLD_PARAMETER Procedure

This procedure modifies the default value for a user parameter within a SQL Workload object or SQL Workload object template. A user parameter is a simple variable that stores various attributes that affect workload collection, tuning decisions and reporting. When a default value is changed for a parameter, workload objects will inherit the new value when they are created.

Syntax

```
DBMS_ADVISOR.SET_DEFAULT_SQLWKLD_PARAMETER (
   parameter            IN VARCHAR2,
   value                IN VARCHAR2);

DBMS_ADVISOR.SET_DEFAULT_SQLWKLD_PARAMETER (
   parameter            IN VARCHAR2,
   value                IN NUMBER);
```

Parameters

Table 12–28 SET_DEFAULT_SQLWKLD_PARAMETER Procedure Parameters

Parameter	Description
parameter	The name of the data parameter to be modified. Parameter names are not case sensitive. Parameter names are unique to the workload object type, but not necessarily unique to all workload object types. Various object types may use the same parameter name for different purposes.
value	The value of the specified parameter. The value can be specified as a string or a number. If the value is DBMS_ADVISOR.DEFAULT, the value will be reset to the default value.

Usage Notes

A parameter will only affect operations that modify the workload collection. Therefore, parameters should be set prior to importing or adding new SQL statements to a workload. If a parameter is set after data has been placed in a workload object, it will have no effect on the existing data.

Examples

```
BEGIN
  DBMS_ADVISOR.SET_DEFAULT_SQLWKLD_PARAMETER('VALID_TABLE_LIST','SH.%');
END;
/
```

SET_DEFAULT_TASK_PARAMETER Procedures

This procedure modifies the default value for a user parameter within a task or a template. A user parameter is a simple variable that stores various attributes that affect various Advisor operations. When a default value is changed for a parameter, tasks will inherit the new value when they are created.

A default task is different from a regular task. The default value is the initial value that will be inserted into a newly created task, while setting a task parameter with SET_TASK_PARAMETER sets the local value only. Thus, SET_DEFAULT_TASK_PARAMETER has no effect on an existing task.

Syntax

```
DBMS_ADVISOR.SET_DEFAULT_TASK_PARAMETER (
   advisor_name        IN VARCHAR2
   parameter           IN VARCHAR2,
   value               IN VARCHAR2);

DBMS_ADVISOR.SET_DEFAULT_TASK_PARAMETER (
   advisor_name        IN VARCHAR2
   parameter           IN VARCHAR2,
   value               IN NUMBER);
```

Parameters

Table 12–29 SET_DEFAULT_TASK_PARAMETER Procedure Parameters

Parameter	Description
advisor_name	Specifies the unique advisor name as defined in the view DBA_ADVISOR_DEFINITIONS.
parameter	The name of the task parameter to be modified. Parameter names are not case sensitive. Parameter names are unique to the task type, but not necessarily unique to all task types. Various task types may use the same parameter name for different purposes.
value	The value of the specified task parameter. The value can be specified as a string or a number.

Examples

```
BEGIN
  DBMS_ADVISOR.SET_DEFAULT_TASK_PARAMETER(DBMS_ADVISOR.SQLACCESS_ADVISOR,
   'VALID_TABLE_LIST', 'SH.%');
END;
/
```

SET_SQLWKLD_PARAMETER Procedure

This procedure modifies a user parameter within a SQL Workload object or SQL Workload object template. A user parameter is a simple variable that stores various attributes that affect workload collection, tuning decisions and reporting.

Syntax

```
DBMS_ADVISOR.SET_SQLWKLD_PARAMETER (
   workload_name        IN VARCHAR2,
   parameter            IN VARCHAR2,
   value                IN VARCHAR2);

DBMS_ADVISOR.SET_SQLWKLD_PARAMETER (
   workload_name        IN VARCHAR2,
   parameter            IN VARCHAR2,
   value                IN NUMBER);
```

Parameters

Table 12–30 SET_SQLWKLD_PARAMETER Procedure Parameters

Parameter	Description
`workload_name`	The SQL Workload object name that uniquely identifies an existing workload.
`parameter`	The name of the data parameter to be modified. Parameter names are not case sensitive.
`value`	The value of the specified parameter. The value can be specified as a string or a number. If the value is `DBMS_ADVISOR.DEFAULT`, the value will be reset to the default value.

Usage Notes

A parameter will only affect operations that modify the workload collection. Therefore, parameters should be set prior to importing or adding new SQL statements to a workload. If a parameter is set after data has been placed in a workload object, it will have no effect on the existing data.

SQL Workload Object Parameters

Table 12–31 lists SQL Access Advisor object parameters.

Table 12–31 SQL Workload Object Parameters

Name	Datatype	Description
ACTION_LIST	STRINGLIST	Use VALID_ACTION_LIST instead. Contains a fully qualified list of actions that are eligible for saving in a workload. An action can be any string. If an action is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. An action string is not scanned for correctness. During a workload import operation, if a SQL statements action does not match a name in the action list, it will not be stored in the workload object. An action name is case sensitive. The possible values are: ■ single action ■ comma-delimited action list ■ ADVISOR_UNUSED The default value is ADVISOR_UNUSED.
COMMENTED_ FILTER_LIST	NUMBER	Use INVALID_SQLSTRING_LIST instead. Comma-delimited list of strings. When set, SQL Access Advisor will filter out any SQL statement that contain any of the specified strings in the first 20 characters of its text.
DAYS_TO_EXPIRE	NUMBER	Specifies the expiration time in days for the current SQL Workload object. The value is relative to the last modification date. Once the data expires, it will become a candidate for removal by an automatic purge operation. The possible values are: ■ an integer in the range of 0 to 2147483647 ■ ADVISOR_UNLIMITED ■ ADVISOR_UNUSED The default value is 30.
END_TIME	STRING	Specifies an end time for selecting SQL statements. If the statement did not execute on or before the specified time, it will not be processed. Each date must be in the standard Oracle form of MM-DD-YYY HH24:MI:SS, where: ■ DD is the numeric date ■ MM is the numeric month ■ YYYY is the numeric year ■ HH is the hour in 24 hour format ■ MI is the minute ■ SS is the second
INVALID_ ACTION_LIST	STRINGLIST	Contains a fully qualified list of actions that are not eligible for saving in a workload. An action can be any string. If an action is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. An action string is not scanned for correctness. During workload collection, if a SQL statement's action matches a name in the action list, it will not be processed by the operation. An action name is case sensitive. Possible values are: ■ single action ■ comma-delimited action list ■ ADVISOR_UNUSED The default value is ADVISOR_UNUSED.

Table 12–31 (Cont.) SQL Workload Object Parameters

Name	Datatype	Description
INVALID_MODULE_LIST	STRINGLIST	Contains a fully qualified list of application modules that are not eligible when populating a SQL workload object. The list elements are comma-delimited, and quoted names are supported. A module can be any string. If a module is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. A module string is not scanned for correctness. During workload collection, if a SQL statement's module matches a name in the list, it will not be processed by the operation. A module name is case sensitive. The possible values are: ■ single application ■ comma-delimited module list ■ ADVISOR_UNUSED The default value is ADVISOR_UNUSED.
INVALID_SQLSTRING_LIST		Contains a fully qualified list of text strings that are not eligible when populating a SQL workload object. The list of elements is comma-delimited, and quoted values are supported. A SQL string can be any string. If a string is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. A SQL string is not scanned for correctness. During workload collection, if a SQL statement contains a string in the SQL string list, it will not be processed by the operation. The possible values are: ■ single string ■ comma-delimited string list ■ ADVISOR_UNUSED The default value is ADVISOR_UNUSED.
INVALID_TABLE_LIST	TABLELIST	Contains a fully qualified list of tables that are not eligible for tuning. The list elements are comma-delimited, and quoted identifiers are supported. Wildcard specifications are supported for both schemas and tables. The default value is all tables within the users scope are eligible for tuning. The supported wildcard character is %. A % wildcard matches any set of consecutive characters. When a SQL statement is processed, it will not be accepted if any referenced table matches an entry in the invalid table list. The valid syntax for a table reference is: ■ schema.table ■ schema ■ schema.% (equivalent to schema) The possible values are: ■ single table reference ■ comma-delimited table reference list ■ ADVISOR_UNUSED The default value is ADVISOR_UNUSED. Note that SQL Access Advisor maintains an internal list of non-tunable tables regardless of the contents of the INVALID_TABLE_LIST parameter. No table that is owned by SYS, SYSTEM or any other pre-defined Oracle schema can be tuned.
INVALID_USERNAME_LIST	STRINGLIST	Contains a fully qualified list of usernames that are not eligible when populating a SQL workload object. The list elements are comma-delimited, and quoted names are supported. During workload collection, if a SQL statements' username matches a name in the username list, it will not be processed by the operation. A username is not case sensitive unless it is quoted. The possible values are: ■ single username ■ comma-delimited username list ■ ADVISOR_UNUSED The default value is ADVISOR_UNUSED.

Table 12–31 (Cont.) SQL Workload Object Parameters

Name	Datatype	Description
JOURNALING	NUMBER	Controls the logging of messages to the journal (USER_ADVISOR_JOURNAL view). The higher the setting, the more information is logged to the journal. The valid settings are: ■ 0: no journal messages ■ 1: fatal error messages ■ 2: simple error messages ■ 3: non-fatal warning messages ■ 4-9: information messages (4 is the default)
MODULE_LIST	STRINGLIST	Use VALID_MODULE_LIST instead. Contains a fully qualified list of application modules that are eligible for saving in a SQL Workload object. The list elements are comma-delimited, and quoted names are supported. A module can be any string. If a module is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. A module string is not scanned for correctness. During a workload import operation, if a SQL statements application module does not match a name in the module list, it will not be stored in the workload object. The possible values are: ■ single module ■ comma-delimited module list ■ ADVISOR_UNUSED The default value is ADVISOR_UNUSED.
ORDER_LIST	STRING	Contains the primary natural order in which the SQL Access Advisor processes workload elements during the import operation. Possible values are: BUFFER_GETS, OPTIMIZER_COST, CPU_TIME, DISK_READS, ELAPSED_TIME, EXECUTIONS, and PRIORITY.
REPORT_DATE_ FORMAT		This parameter is not used.
SQL_LIMIT	NUMBER	Specifies the maximum number of SQL statements to be saved during a workload import operation. The SQL_LIMIT filter is applied after all other filters have been applied. For example, if only statements referencing the table foo.bar are to be accepted, the SQL_ LIMIT value will be only apply to those statements. When used in conjunction with the parameter ORDER_LIST, Access Advisor will process and save the most interesting SQL statements by ordering the statements according to the specified sort keys. The possible values are: ■ an integer in the range of 1 to 2147483647 ■ ADVISOR_UNLIMITED ■ ADVISOR_UNUSED The default value is ADVISOR_UNUSED.
START_TIME	STRING	Specifies a start time for selecting SQL statements. If the statement did not execute on or before the specified time, it will not be processed. Each date must be in the standard Oracle form of MM-DD-YYY HH24:MI:SS, where: ■ DD is the numeric date ■ MM is the numeric month ■ YYYY is the numeric year ■ HH is the hour in 24 hour format ■ MI is the minute ■ SS is the second

Table 12–31 (Cont.) SQL Workload Object Parameters

Name	Datatype	Description
USERNAME_LIST	STRINGLIST	Use VALID_USERNAME_LIST instead. Contains a fully qualified list of usernames that are eligible for processing in a SQL Workload object. The list elements are comma-delimited, and quoted names are supported. During a workload import operation, if a SQL statements username does not match a name in the username list, it will not be stored in the workload object. A Username is not case sensitive unless it is quoted. The possible values are: ■ single username ■ comma-delimited username list ■ ADVISOR_UNUSED The default value is ADVISOR_UNUSED.
VALID_ACTION_LIST	STRINGLIST	Contains a fully qualified list of actions that are eligible when populating a SQL workload object. The list elements are comma-delimited, and quoted names are supported. An action can be any string. If an action is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. An action string is not scanned for correctness. During workload collection, if a SQL statement's action does not match a name in the action list, it will not be processed by the operation. An action name is case sensitive. The possible values are: ■ single action ■ comma-delimited action list ■ ADVISOR_UNUSED The default value is ADVISOR_UNUSED.
VALID_MODULE_LIST	STRINGLIST	Contains a fully qualified list of application modules that are eligible when populating a SQL workload object. The list elements are comma-delimited, and quoted names are supported. A module can be any string. If a module is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. A module string is not scanned for correctness. During workload collection, if a SQL statement's module does not match a name in the module list, it will not be processed by the operation. A module name is case sensitive. The possible values are: ■ single application ■ comma-delimited module list ■ ADVISOR_UNUSED The default value is ADVISOR_UNUSED.

Table 12–31 (Cont.) SQL Workload Object Parameters

Name	Datatype	Description
VALID_ SQLSTRING_LIST	STRINGLIST	Contains a fully qualified list of text strings that are eligible when populating a SQL workload object. The list elements are comma-delimited, and quoted values are supported. A SQL string can be any string. If a string is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. A SQL string is not scanned for correctness. During workload collection, if a SQL statement does not contain a string in the SQL string list, it will not be processed by the operation. The possible values are: ■ single string ■ comma-delimited string list ■ ADVISOR_UNUSED The default value is ADVISOR_UNUSED.
VALID_TABLE_ LIST	TABLELIST	Contains a fully qualified list of tables that are eligible for tuning. The list elements are comma-delimited, and quoted identifiers are supported. Wildcard specifications are supported for tables. The default value is all tables within the user's scope are eligible for tuning. The supported wildcard character is %. A % wildcard matches any set of consecutive characters. When a SQL statement is processed, it will not be accepted unless at least one referenced table is specified in the valid table list. If the list is unused, then all table references within a SQL statement are considered valid. When using the IMPORT_SQLWKLD_SCHEMA procedure, the valid_table_list parameter cannot contain wildcards such as SCO% or SCOTT.EMP%. The only form of wildcards supported is SCOTT.%, which specifies all tables in a given schema. The valid syntax for a table reference is: ■ schema.table ■ schema ■ schema.% (equivalent to schema) The possible values are: ■ single table reference ■ comma-delimited table reference list ■ ADVISOR_UNUSED The default value is ADVISOR_UNUSED.
VALID_ USERNAME_LIST	STRINGLIST	Contains a fully qualified list of usernames that are eligible when populating a SQL workload object. The list of elements is comma-delimited, and quoted names are supported. During workload collection, if a SQL statement's username does not match a name in the username list, it will not be processed by the operation. A username is not case sensitive unless it is quoted. The possible values are: ■ single username ■ comma-delimited username list ■ ADVISOR_UNUSED The default value is ADVISOR_UNUSED.

Examples

```
DECLARE
  workload_name VARCHAR2(30);
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.SET_SQLWKLD_PARAMETER(workload_name, 'VALID_TABLE_LIST','SH.%');
END;
/
```

SET_TASK_PARAMETER Procedure

This procedure modifies a user parameter within an Advisor task or a template. A user parameter is a simple variable that stores various attributes that affect workload collection, tuning decisions and reporting.

Syntax

```
DBMS_ADVISOR.SET_TASK_PARAMETER (
   task_name           IN VARCHAR2
   parameter           IN VARCHAR2,
   value               IN VARCHAR2);

DBMS_ADVISOR.SET_TASK_PARAMETER (
   task_name           IN VARCHAR2
   parameter           IN VARCHAR2,
   value               IN NUMBER);
```

Parameters

Table 12–32 SET_TASK_PARAMETER Procedure Parameters

Parameter	Description
task_name	The Advisor task name that uniquely identifies an existing task.
parameter	The name of the task parameter to be modified. Parameter names are not case sensitive. Parameter names are unique to the task type, but not necessarily unique to all task types. Various task types may use the same parameter name for different purposes.
value	The value of the specified task parameter. The value can be specified as a string or a number. If the value is DEFAULT, the value will be reset to the default value.

Usage Notes

A task cannot be modified unless it is in its initial state. See "RESET_TASK Procedure" on page 12-40 to set a task to its initial state. See your Advisor-specific documentation for further information on using this procedure.

SQL Access Advisor Task Parameters

Table 12–33 lists SQL Access Advisor task parameters.

Table 12–33 SQL Access Advisor Task Parameters

Parameter	Datatype	Description
ACTION_LIST	STRINGLIST	Use VALID_ACTION_LIST instead. Contains a fully qualified list of actions that are eligible for processing in a SQL Workload object. The list elements are comma-delimited, and quoted names are supported. An action can be any string. If an action is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. An action string is not scanned for correctness. During a task execution, if a SQL statement's action does not match a name in the action list, it will not be processed by the task. An action name is case sensitive. The possible values are: ■ single action ■ comma-delimited action list ■ ADVISOR_UNUSED The default value is ADVISOR_UNUSED.
COMMENTED_ FILTER_LIST	NUMBER	Use INVALID_SQLSTRING_LIST instead. Comma-delimited list of strings. When set, the SQL Access Advisor will filter out any SQL statement that contain any of the specified strings in the first 20 characters of its text.
CREATION_COST	STRING	When set to true (default), the SQL Access Advisor will weigh the cost of creation of the access structure (index or materialized view) against the frequency of the query and potential improvement in the query execution time. When set to false, the cost of creation is ignored.
DAYS_TO_ EXPIRE	NUMBER	Specifies the expiration time in days for the current SQL Access Advisor task. The value is relative to the last modification date. Once the task expires, it will become a candidate for removal by an automatic purge operation. Specifies the expiration time in days for the current Access Advisor task. The value is relative to the last modification date. Once the task expires, it becomes a candidate for removal by an automatic purge operation. The possible values are: ■ an integer in the range of 0 to 2147483647 ■ ADVISOR_UNLIMITED ■ ADVISOR_UNUSED The default value is 30.
DEF_INDEX_ OWNER	STRING	Specifies the default owner for new index recommendations. When a script is created, this value will be used to qualify the index name. Possible values are: ■ Existing schema name. Quoted identifiers are supported. ■ ADVISOR_UNUSED The default value is ADVISOR_UNUSED.
DEF_INDEX_ TABLESPACE	STRING	Specifies the default tablespace for new index recommendations. When a script is created, this value will be used to specify a tablespace clause. Possible values are: ■ Existing tablespace name. Quoted identifiers are supported. ■ ADVISOR_UNUSED No tablespace clause will be present in the script for indexes. The default value is ADVISOR_UNUSED.
DEF_MVIEW_ OWNER	STRING	Specifies the default owner for new materialized view recommendations. When a script is created, this value will be used to qualify the materialized view name. Possible values are: ■ Existing schema name. Quoted identifiers are supported. ■ ADVISOR_UNUSED The default value is ADVISOR_UNUSED.

Table 12–33 (Cont.) SQL Access Advisor Task Parameters

Parameter	Datatype	Description
DEF_MVIEW_ TABLESPACE	STRING	Specifies the default tablespace for new materialized view recommendations. When a script is created, this value will be used to specify a tablespace clause. Possible values are ■ Existing tablespace name. Quoted identifiers are supported. ■ ADVISOR_UNUSED. No tablespace clause will be present in the script for materialized view logs. The default value is ADVISOR_UNUSED.
DEF_MVLOG_ TABLSPACE	STRING	Specifies the default tablespace for new materialized view log recommendations. When a script is created, this value will be used to specify a tablespace clause. Possible values are: ■ Existing tablespace name. Quoted identifiers are supported. ■ ADVISOR_UNUSED. No tablespace clause will be present in the script for materialized view logs. The default value is ADVISOR_UNUSED.
DML_ VOLATILITY	STRING	When set to TRUE, the SQL Access Advisor will consider the impact of index maintenance and materialized view refresh in determining the recommendations. It will limit the access structure recommendations involving columns or tables that are frequently updated. For example, if there are too many DMLs on a column, it may favor a Btree index over a bitmap index on that column. For this process to be effective, the workload must include DML (insert/update/delete/merge/direct path inserts) statements that represent the update behavior of the application. See the related parameter refresh_mode.
END_TIME	STRING	Specifies an end time for selecting SQL statements. If the statement did not execute on or before the specified time, it will not be processed. Each date must be in the standard Oracle form of MM-DD-YYY HH24:MI:SS, where: ■ DD is the numeric date ■ MM is the numeric month ■ YYYY is the numeric year ■ HH is the hour in 24 hour format ■ MI is the minute ■ SS is the second
EVALUATION_ ONLY	STRING	If set to TRUE, causes SQL Access Advisor to analyze the workload, but only comment on how well the current configuration is supporting it. No tuning recommendations will be generated. Possible values are: ■ FALSE ■ TRUE The default value is FALSE.
EXECUTION_ TYPE	STRINGLIST	The type of recommendations that is desired. Possible values: ■ FULL All supported recommendation types will be considered. ■ INDEX_ONLY The SQL Access Advisor will only consider index solutions as recommendations. ■ MVIEW_ONLY The SQL Access Advisor will consider materialized view and materialized view log solutions as recommendations. ■ MVIEW_LOG_ONLY The SQL Access Advisor will only consider materialized view log solutions as recommendations. The default value is FULL.

Table 12–33 (Cont.) SQL Access Advisor Task Parameters

Parameter	Datatype	Description
IMPLEMENT_ EXIT_ON_ERROR	STRING	When performing an IMPLEMENT_TASK operation, this parameter will control behavior when an action fails to implement. If set to TRUE, IMPLEMENT_TASK will stop on the first unexpected error. The possible values are: ■ TRUE ■ FALSE The default value is TRUE.
INDEX_NAME_ TEMPLATE	STRING	Specifies the method by which new index names are formed. If the *TASK_ID* is omitted from the template, names generated by two concurrently executing SQL Access Advisor tasks may conflict and cause undesirable effects. So it is recommended that you include the *TASK_ID* in the template. Once formatted, the maximum size of a name is 30 characters. Valid keywords are: ■ Any literal value up to 22 characters. ■ TABLE Causes the parent table name to be substituted into the index name. If the name is too long, it will be trimmed to fit. ■ TASK_ID Causes the current task identifier number to be inserted in hexadecimal form. ■ SEQ Causes a sequence number to be inserted in hexadecimal form. Because this number is used to guarantee uniqueness, it is a required token. The default template is <TABLE>_IDX$$_<TASK_ID><SEQ>.
INVALID_ ACTION_LIST	STRINGLIST	Contains a fully qualified list of actions that are not eligible for processing in a SQL workload object. The list elements are comma-delimited, and quoted names are supported. An action can be any string. If an action is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. An action string is not scanned for correctness. During a task execution, if a SQL statement's action matches a name in the action list, it will not be processed by the task. An action name is case sensitive. The possible values are: ■ single action ■ comma-delimited action list ■ ADVISOR_UNUSED The default value is ADVISOR_UNUSED.
INVALID_ MODULE_LIST	STRINGLIST	Contains a fully qualified list of modules that are not eligible for processing in a SQL workload object. The list elements are comma-delimited, and quoted names are supported. A module can be any string. If a module is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. A module string is not scanned for correctness. During a task execution, if a SQL statement's module matches a name in the list, it will not be processed by the task. A module name is case sensitive. The possible values are: ■ single application ■ comma-delimited module list ■ ADVISOR_UNUSED The default value is ADVISOR_UNUSED.

Table 12–33 (Cont.) SQL Access Advisor Task Parameters

Parameter	Datatype	Description
INVALID_ SQLSTRING_ LIST	STRINGLIST	Contains a fully qualified list of text strings that are not eligible for processing in a SQL workload object. The list elements are comma-delimited, and quoted values are supported. A SQL string can be any string. If a string is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. A SQL string is not scanned for correctness. During a task execution, if a SQL statement contains a string in the SQL string list, it will not be processed by the task. The possible values are: ■ single string ■ comma-delimited string list ■ ADVISOR_UNUSED The default value is ADVISOR_UNUSED.
INVALID_ USERNAME_LIST	STRINGLIST	Contains a fully qualified list of usernames that are not eligible for processing in a SQL workload object. The list elements are comma-delimited, and quoted names are supported. During a task execution, if a SQL statement's username matches a name in the username list, it will not be processed by the task. A username is not case sensitive unless it is quoted. The possible values are: ■ single username ■ comma-delimited username list ■ ADVISOR_UNUSED The default value is ADVISOR_UNUSED.
JOURNALING	NUMBER	Controls the logging of messages to the journal (USER_ADVISOR_JOURNAL view). The higher the setting, the more information is logged to the journal. Valid settings are: ■ 0: no journal messages ■ 1: fatal error messages ■ 2: simple error messages ■ 3: non-fatal warning messages ■ 4-9: information messages (4 is the default)
MODE	STRING	Specifies the mode by which Access Advisor will operate during an analysis. Valid values are: ■ LIMITED Indicates the Advisor will attempt to a quick job by limiting the search-space of candidate recommendations, and correspondingly, the results may be of a low quality. ■ COMPREHENSIVE Indicates the Advisor will search a large pool of candidates that may take long to run, but the resulting recommendations will be of the highest quality. The default value is COMPREHENSIVE.
MODULE_LIST	STRINGLIST	Use VALID_MODULE_LIST instead. Contains a fully qualified list of application modules that are eligible for processing in a SQL Workload object. The list elements are comma-delimited, and quoted names are supported. A module can be any string. If a module is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. A module string is not scanned for correctness. During a workload import operation, if a SQL statement's application module does not match a name in the module list, it will not be stored in the workload object. Possible values are: ■ single application ■ comma-delimited module list ■ ADVISOR_UNUSED The default value is ADVISOR_UNUSED.

Table 12–33 (Cont.) SQL Access Advisor Task Parameters

Parameter	Datatype	Description
MVIEW_NAME_ TEMPLATE	STRING	Specifies the method by which new materialized view names are formed. If the *TASK_ID* is omitted from the template, names generated by two concurrently executing SQL Access Advisor tasks may conflict and cause undesirable effects. So it is recommended that you include the *TASK_ID* in the template. The format is any combination of keyword tokens and literals. However, once formatted, the maximum size of a name is 30 characters. Valid tokens are: ■ Any literal value up to 22 characters. ■ *TASK_ID* Causes the current task identifier number to be inserted in hexadecimal form. ■ *SEQ* Causes a sequence number to be inserted in hexadecimal form. Because this number is used to guarantee uniqueness, it is a required token. The default template is: MV$$_<TASK_ID><SEQ>.
ORDER_LIST	STRINGLIST	Contains the primary natural order in which the Access Advisor processes workload elements during the analysis operation. To determine absolute natural order, Access Advisor sorts the workload using ORDER_LIST values. A comma must separate multiple order keys. Possible values are: ■ BUFFER_GETS Sets the order using the SQL statement's buffer-get count value. ■ CPU_TIME Sets the order using the SQL statement's CPU time value. ■ DISK_READS Sets the order using the SQL statement's disk-read count value. ■ ELAPSED_TIME Sets the order using the SQL statement's elapsed time value. ■ EXECUTIONS Sets the order using the SQL statement's execution frequency value. ■ OPTIMIZER_COST Sets the order using the SQL statement's optimizer cost value. ■ I/O Sets the order using the SQL statement's I/O count value. ■ PRIORITY Sets the order using the user-supplied business priority value. All values are accessed in descending order, where a high value is considered more interesting than a low value. The default value is PRIORITY, OPTIMIZER_COST.
RECOMMEND_MV_ EXACT_TEXT_ MATCH	STRING	When considering candidate materialized views, exact text match solutions will only be included if this parameter contains TRUE. The possible values are: ■ TRUE ■ FALSE The default value is TRUE.
REFRESH_MODE	STRING	Specifies whether materialized views are refreshed ON_DEMAND or ON_COMMIT. This will be used to weigh the impact of materialized view refresh when the parameter dml_ volatility is set to TRUE. Possible values are: ■ ON_DEMAND ■ ON_COMMIT The default value is ON_DEMAND.
REPORT_DATE_ FORMAT	STRING	This is the default date and time formatting template. The default format is DD/MM/YYYYHH24:MI.
SHOW_RETAINS	STRING	Controls the display of RETAIN actions within an implementation script and the SQL Access Advisor wizard. The possible values are: ■ TRUE ■ FALSE The default value is TRUE.

Table 12–33 (Cont.) SQL Access Advisor Task Parameters

Parameter	Datatype	Description
SQL_LIMIT	NUMBER	Specifies the number of SQL statements to be analyzed. The SQL_LIMIT filter is applied after all other filters have been applied. For example, if only statements referencing the table foo.bar are to be accepted, the SQL_LIMIT value will be only apply to those statements. When used in conjunction with the parameter ORDER_LIST, SQL Access Advisor will process the most interesting SQL statements by ordering the statements according to the specified sort keys. The possible values are: ■ an integer in the range of 1 to 2147483647 ■ ADVISOR_UNLIMITED ■ ADVISOR_UNUSED The default value is ADVISOR_UNUSED.
START_TIME	STRING	Specifies a start time for selecting SQL statements. If the statement did not execute on or before the specified time, it will not be processed. Each date must be in the standard Oracle form of MM-DD-YYY HH24:MI:SS, where: ■ DD is the numeric date ■ MM is the numeric month ■ YYYY is the numeric year ■ HH is the hour in 24 hour format ■ MI is the minute ■ SS is the second
STORAGE_ CHANGE	NUMBER	Contains the amount of space adjustment that can be consumed by SQL Access Advisor recommendations. Zero or negative values are only permitted if the workload scope is marked as FULL. When the SQL Access Advisor produces a set of recommendations, the resultant physical structures must be able to fit into the budgeted space. A space budget is computed by adding the STORAGE_CHANGE value to the space quantity currently used by existing access structures. A negative STORAGE_CHANGE value may force SQL Access Advisor to remove existing structures in order to shrink space demand. Possible values: ■ Any valid integer including negative values, zero and positive values. The default value is ADVISOR_UNLIMITED.
USERNAME_LIST	STRINGLIST	Use VALID_USERNAME_LIST instead. Contains a fully qualified list of usernames that are eligible for processing in a workload object. The list elements are comma-delimited, and quoted names are supported. During a task execution, if a SQL statement's username does not match a name in the username list, it will not be processed by the task. A username is not case sensitive unless it is quoted. Possible values: ■ single username ■ comma-delimited username list ■ ADVISOR_UNUSED The default value is ADVISOR_UNUSED.

Table 12–33 (Cont.) SQL Access Advisor Task Parameters

Parameter	Datatype	Description
VALID_ACTION_LIST	STRINGLIST	Contains a fully qualified list of actions that are eligible for processing in a SQL workload object. The list elements are comma-delimited, and quoted names are supported. An action can be any string. If an action is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. An action string is not scanned for correctness. During a task execution, if a SQL statement's action does not match a name in the action list, it will not be processed by the task. An action name is case sensitive. The possible values are: ■ single action ■ comma-delimited action list ■ ADVISOR_UNUSED The default value is ADVISOR_UNUSED.
VALID_MODULE_LIST	STRINGLIST	Contains a fully qualified list of application modules that are eligible for processing in a SQL workload object. The list elements are comma-delimited, and quoted names are supported. A module can be any string. If a module is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. A module string is not scanned for correctness. During a task execution, if a SQL statement's module does not match a name in the module list, it will not be processed by the task. A module name is case sensitive. The possible values are: ■ single application ■ comma-delimited module list ■ ADVISOR_UNUSED The default value is ADVISOR_UNUSED.
VALID_SQLSTRING_LIST	STRINGLIST	Contains a fully qualified list of text strings that are eligible for processing in a SQL workload object. The list elements are comma-delimited, and quoted names are supported. A SQL string can be any string. If a string is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. A SQL string is not scanned for correctness. During a task execution, if a SQL statement does not contain string in the SQL string list, it will not be processed by the task. The possible values are: ■ single string ■ comma-delimited string list ■ ADVISOR_UNUSED The default value is ADVISOR_UNUSED.

Table 12–33 (Cont.) SQL Access Advisor Task Parameters

Parameter	Datatype	Description
VALID_TABLE_LIST	TABLELIST	Contains a fully qualified list of tables that are eligible for tuning. The list elements are comma-delimited, and quoted identifiers are supported. Wildcard specifications are supported for tables. The default value is all tables within the user's scope are eligible for tuning. Supported wildcard character is %. A % wildcard matches any set of consecutive characters. When a SQL statement is processed, it will not be accepted unless at least one referenced table is specified in the valid table list. If the list is unused, then all table references within a SQL statement are considered valid. The valid syntax for a table reference is: ■ schema.table ■ schema ■ schema.% (equivalent to schema) ■ comma-delimited action list ■ ADVISOR_UNUSED The possible values are: ■ single table reference ■ comma-delimited reference list ■ ADVISOR_UNUSED The default value is ADVISOR_UNUSED.
VALID_USERNAME_LIST	STRINGLIST	Contains a fully qualified list of usernames that are eligible for processing in a SQL workload object. The list elements are comma-delimited, and quoted names are supported. During a task execution, if a SQL statement's username does not match a name in the username list, it will not be processed by the task. A username is not case sensitive unless it is quoted. The possible values are: ■ single username ■ comma-delimited username list ■ ADVISOR_UNUSED The default value is ADVISOR_UNUSED.
WORKLOAD_SCOPE	STRING	Describes the level of application coverage the workload represents. Possible values are FULL and PARTIAL. FULL Should be used if the workload contains all interesting application SQL statements for the targeted tables. PARTIAL (default) Should be used if the workload contains anything less than a full representation of the interesting application SQL statements for the targeted tables.

Segment Advisor Parameters

Table 12–35 lists the input task parameters that can be set in the Segment Advisor using the SET_TASK_PARAMETER procedure.

Table 12–34 Segment Advisor Task Parameters

Parameter	Default Value	Possible Values	Description
MODE	COMPREHENSIVE	LIMITED: Analysis restricted to statistics available in Automatic Workload Repository. COMPREHENSIVE: Comprehensive analysis based on sampling and Automatic Workload Repository statistics.	The data to use for analysis.
TIME_LIST	UNLIMITED	UNLIMITED	The time limit for which the Advisor should run. Specified in seconds.
RECOMMEND_ALL	TRUE	If set to TRUE, generate recommendations on all segments specified by the user. If set to FALSE, recommendations for only those objects that are eligible for shrink.	Whether to generate recommendations for all segments.

Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
BEGIN
  task_name := 'My Task';

   DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
   DBMS_ADVISOR.SET_TASK_PARAMETER(task_name, 'VALID_TABLELIST',
      'SH.%,SCOTT.EMP');
END;
/
```

Undo Advisor Task Parameters

Table 12–35 lists the input task parameters that can be set in the Undo Advisor using the SET_TASK_PARAMETER procedure.

Table 12–35 Undo Advisor Task Parameters

Parameter	Default Value	Possible Values	Description
TARGET_OBJECTS	none	UNDO_TBS	The target object is the undo tablespace of the system.
START_SNAPSHOT	none	The valid snapshot numbers in the AWR repository.	The starting time for the system to perform analysis using the snapshot numbers in the AWR repository.
END_SNAPSHOT	none	The valid snapshot numbers in the AWR repository.	The ending time for the system to perform analysis using the snapshot numbers in the AWR repository.
BEGIN_TIME_SEC	none	Any positive integer.	The number of seconds between the beginning time of the period and now. Describes a period of time for the system to perform analysis. BEGIN_TIME_SEC should be greater than END_TIME_SEC.
END_TIME_SEC	none	Any positive integer.	The number of seconds between the ending time of the period and now. END_TIME_SEC should be less than BEGIN_TIME_SEC.

Examples

```
DECLARE
   tname  VARCHAR2(30);
   oid    NUMBER;
   BEGIN
     DBMS_ADVISOR.CREATE_TASK('Undo Advisor', tid, tname, 'Undo Advisor Task');
     DBMS_ADVISOR.CREATE_OBJECT(tname, 'UNDO_TBS', null, null, null, 'null', oid);
     DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'TARGET_OBJECTS', oid);
     DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'START_SNAPSHOT', 1);
     DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'END_SNAPSHOT', 2);
     DBMS_ADVISOR.EXECUTE_TASK(tname);
   END;
/
```

Automatic Database Diagnostic Monitor (ADDM) Task Parameters

Table 12–36 lists the input task parameters that can be set in ADDM using the SET_TASK_PARAMETER procedure.

Table 12–36 ADDM Task Parameters

Parameter	Default	Possible Values	Description
START_SNAPSHOT	none	The valid snapshot numbers in the AWR repository.	The starting time for the system to perform analysis using the snapshot numbers in the AWR repository.
END_SNAPSHOT	none	The valid snapshot numbers in the AWR repository.	The ending time for the system to perform analysis using the snapshot numbers in the AWR repository.
DB_ID	The current database ID.	Not applicable.	The database for START_SNAPSHOT and END_SNAPSHOT.
INSTANCE	The current instance ID.	Not applicable.	The instance for START_SNAPSHOT and END_SNAPSHOT.
DBIO_EXPECTED	10 milliseconds	System dependent.	The average time to read the database block in microseconds.

Examples

```
DECLARE
   tid     NUMBER;
BEGIN
   DBMS_ADVISOR.CREATE_TASK('ADDM', tid, 'ADDM_TEST', 'my test');
   DBMS_ADVISOR.SET_TASK_PARAMETER('ADDM_TEST', 'START_SNAPSHOT', '19',
                                                -  'END_SNAPSHOT',   '26',
                                                -  'DB_ID', '155789304',
                                                -  'INSTANCE', '1');
   DBMS_ADVISOR.EXECUTE_TASK('ADDM_TEST');
END;
/
```

SQL Tuning Advisor Task Parameters

See the DBMS_SQLTUNE package on page 101-1 and *Oracle Database Performance Tuning Guide* for more information.

TUNE_MVIEW Procedure

This procedure shows how to decompose a materialized view into two or more materialized views and to restate the materialized view in a way that is more advantageous for fast refresh and query rewrite. It also shows how to fix materialized view logs and to enable query rewrite.

Syntax

```
DBMS_ADVISOR.TUNE_MVIEW (
   task_name IN OUT VARCHAR2,
   mv_create_stmt IN [CLOB | VARCHAR2]);
```

Parameters

Table 12–37 TUNE_MVIEW Procedure Parameters

Parameter	Description
task_name	The task name for looking up the results in a catalog view. If not specified, the system will generate a name and return.
mv_create_stmt	The original materialized view creation statement.

See Also: *Oracle Database Performance Tuning Guide* for more information about using the TUNE_MVIEW procedure

Usage Notes

Executing TUNE_MVIEW generates two sets of output results: one is for CREATE implementation and the other is for undoing the CREATE MATERIALIZED VIEW implementation. The output results are accessible through USER_TUNE_MVIEW and DBA_TUNE_MVIEW views. You can also use DBMS_ADVISOR.GET_TASK_SCRIPT and DBMS_ADVISOR.CREATE_FILE to output the TUNE_MVIEW results into a script file for later execution.

USER_TUNE_MVIEW and DBA_TUNE_MVIEW Views

These views are to get the result after executing the TUNE_MVIEW procedure.

Table 12–38 USER_TUNE_MVIEW and DBA_TUNE_MVIEW Views

Column Name	Column Description
OWNER	The materialized view owner's name.
TASK_NAME	The task name as a key to access the set of recommendations
SCRIPT_TYPE	Recommendation ID used to indicate the row is for IMPLEMENTATION or UNDO script.
ACTION_ID	Action ID used as the command order number.
STATEMENT	For TUNE_MVIEW output, this column represents the following statements, and includes statement properties such as REFRESH and REWRITE options: ■ CREATE MATERIALIZED VIEW LOG ■ ALTER MATERIALIZED VIEW LOG FORCE ■ [CREATE | DROP] MATERIALIZED VIEW

Examples

```
name VARCHAR2(30);
DBMS_ADVISOR.TUNE_MVIEW.(name, 'SELECT AVG(C1) FROM my_fact_table WHERE c10 = 7');
```

The following is an example to show how to use TUNE_MVIEW to optimize a CREATE MATERIALIZED VIEW statement:

```
NAME VARCHAR2(30) := 'my_tune_mview_task';
EXECUTE DBMS_ADVISOR.TUNE_MVIEW (name, 'CREATE MATERIALIZED VIEW MY_MV
REFRESH FAST AS SELECT C2, AVG(C1) FROM MY_FACT_TABLE WHERE C10 = 7
GROUP BY C2');
```

You can view the CREATE output results by querying USER_TUNE_MVIEW or DBA_TUNE_MVIEW as the following example:

```
SELECT * FROM USER_TUNE_MVIEW WHERE TASK_NAME='my_tune_mview_task' AND
SCRIPT_TYPE='CREATE';
```

Alternatively, you can save the output results in an external script file as in the following example:

```
CREATE DIRECTORY TUNE_RESULTS AS  ''/myscript_dir'' ;
GRANT READ, WRITE ON DIRECTORY TUNE_RESULTS TO PUBLIC;
EXECUTE DBMS_ADVISOR.CREATE_FILE(DBMS_ADVISOR.GET_TASK_SCRIPT('my_tune_mview_
task'), -
'/homes/tune','my_tune_mview_create.sql');
```

The preceding statement will save the CREATE output results in /myscript_dir/my_tune_mview_create.sql.

UPDATE_OBJECT Procedure

This procedure updates an existing task object. Task objects are typically used as input data for a particular advisor. Segment advice can be generated at the object, segment, or tablespace level.

Syntax

```
DBMS_ADVISOR.UPDATE_OBJECT (
   task_name       IN VARCHAR2
   object_id       IN NUMBER,
   attr1           IN VARCHAR2 := NULL,
   attr2           IN VARCHAR2 := NULL,
   attr3           IN VARCHAR2 := NULL,
   attr4           IN CLOB := NULL,
   attr5           IN VARCHAR2 := NULL);
```

Parameters

Table 12–39 UPDATE_OBJECT Procedure Parameters

Parameter	Description
task_name	A valid advisor task name that uniquely identifies an existing task.
object_id	The advisor-assigned object identifier.
attr1	Advisor-specific data. If set to NULL, there will be no effect on the target object.
attr2	Advisor-specific data. If set to NULL, there will be no effect on the target object.
attr3	Advisor-specific data. If set to NULL, there will be no effect on the target object.
attr4	Advisor-specific data. If set to NULL, there will be no effect on the target object.
attr5	Advisor-specific data. If set to null, there will be no effect on the target object.

The attribute parameters have different values depending upon the object type. See *Oracle Database Administrator's Guide* for details regarding these parameters and object types.

Usage Notes

If for the object level, advice is generated on all partitions of the object (if the object is partitioned). The advice is not cascaded to any dependent objects. If for the segment level, advice can be obtained on a single segment, such as the partition or subpartition of a table, index, or lob column. If for a tablespace level, target advice for every segment in the tablespace will be generated.

See *Oracle Database Administrator's Guide* for further information regarding the Segment Advisor.

Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
  obj_id NUMBER;
```

```
BEGIN
  task_name := 'My Task';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
  DBMS_ADVISOR.CREATE_OBJECT (task_name,'SQL',NULL,NULL,NULL,
                              'SELECT * FROM SH.SALES',obj_id);
  DBMS_ADVISOR.UPDATE_OBJECT (task_name, obj_id,NULL,NULL,NULL,
                              'SELECT count(*) FROM SH.SALES');
END;
/
```

UPDATE_REC_ATTRIBUTES Procedure

This procedure updates the owner, name, and tablespace for a recommendation.

Syntax

```
DBMS_ADVISOR.UPDATE_REC_ATTRIBUTES (
   task_name            IN VARCHAR2
   rec_id               IN NUMBER,
   action_id            IN NUMBER,
   attribute_name       IN VARCHAR2,
   value                IN VARCHAR2);
```

Parameters

Table 12–40 UPDATE_REC_ATTRIBUTES Procedure Parameters

Parameter	Description
`task_name`	The task name that uniquely identifies an existing task.
`rec_id`	The Advisor-generated identifier number that is assigned to the recommendation.
`action_id`	The Advisor-generated action identifier that is assigned to the particular command.
`attribute_name`	Name of the attribute to be changed. The valid values are: ■ `owner` The new owner of the object. ■ `name` The new name of the object. ■ `tablespace` The new tablespace for the object.
`value`	Specifies the new value for the recommendation attribute.

Usage Notes

Recommendation attributes cannot be modified unless the task has successfully executed.

Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
  workload_name VARCHAR2(30);
  attribute VARCHAR2(100);
BEGIN
  task_name := 'My Task';
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_REF(task_name, workload_name);
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
                                     100,400,5041,103,640445,680000,2,
                                     1,SYSDATE,1,'SH','SELECT AVG(amount_sold)
                                     FROM sh.sales WHERE promo_id = 10');
  DBMS_ADVISOR.EXECUTE_TASK(task_name);

attribute := 'SH';
```

```
  DBMS_ADVISOR.UPDATE_REC_ATTRIBUTES(task_name, 1, 3, 'OWNER', attribute);
END;
/
```

UPDATE_SQLWKLD_ATTRIBUTES Procedure

This procedure changes various attributes of a SQL Workload object or template.

Syntax

```
DBMS_ADVISOR.UPDATE_SQLWKLD_ATTRIBUTES (
   workload_name        IN VARCHAR2,
   new_name             IN VARCHAR2 := NULL,
   description          IN VARCHAR2 := NULL,
   read_only            IN VARCHAR2 := NULL,
   is_template          IN VARCHAR2 := NULL,
   how_created          IN VARCHAR2 := NULL);
```

Parameters

Table 12–41 UPDATE_SQLWKLD_ATTRIBUTES Procedure Parameters

Parameter	Description
workload_name	The workload object name that uniquely identifies an existing workload.
new_name	The new workload object name. If the value is NULL or contains the value ADVISOR_UNUSED, the workload will not be renamed. A task name can be up to 30 characters long.
description	A new workload description. If the value is NULL or contains the value ADVISOR_UNUSED, the description will not be changed. Names can be up to 256 characters long.
read_only	Set to TRUE so it cannot be changed.
is_template	TRUE if workload is to be used as a template.
how_created	Indicates a source application name that initiated the workload creation. If the value is NULL or contains the value ADVISOR_UNUSED, the source will not be changed.

Examples

```
DECLARE
  workload_name VARCHAR2(30);
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
                                     100,400,5041,103,640445,680000,2,
                                     1,SYSDATE,1,'SH','SELECT AVG(amount_sold)
                                     FROM sh.sales WHERE promo_id = 10');
  DBMS_ADVISOR.UPDATE_SQLWKLD_ATTRIBUTES(workload_name,'New workload name');
END;
/
```

UPDATE_SQLWKLD_STATEMENT Procedure

This procedure updates an existing SQL statement in a specified SQL workload.

Syntax

```
DBMS_ADVISOR.UPDATE_SQLWKLD_STATEMENT (
   workload_name     IN VARCHAR2,
   sql_id            IN NUMBER,
   application       IN VARCHAR2 := NULL,
   action            IN VARCHAR2 := NULL,
   priority          IN NUMBER := NULL,
   username          IN VARCHAR2 := NULL);

DBMS_ADVISOR.UPDATE_SQLWKLD_STATEMENT (
   workload_name     IN VARCHAR2,
   search            IN VARCHAR2,
   updated           OUT NUMBER,
   application       IN VARCHAR2 := NULL,
   action            IN VARCHAR2 := NULL,
   priority          IN NUMBER := NULL,
   username          IN VARCHAR2 := NULL);
```

Parameters

Table 12–42 UPDATE_SQLWKLD_STATEMENT Procedure Parameters

Parameter	Description
workload_name	The SQL Workload object name that uniquely identifies an existing workload.
sql_id	The Advisor-generated identifier number that is assigned to the statement. To specify all workload statements, use the constant DBMS_ADVISOR.ADVISOR_ALL.
updated	Returns the number of statements changed by a searched update.
application	Specifies a business application name that will be associated with the SQL statement. If the value is NULL or contains the value ADVISOR_UNUSED, then the column will not be updated in the repository.
action	Specifies the application action for the statement. If the value is NULL or contains the value ADVISOR_UNUSED, then the column will not be updated in the repository.
priority	The relative priority of the SQL statement. The value must be one of the following: 1 - HIGH, 2 - MEDIUM, or 3 - LOW. If the value is NULL or contains the value ADVISOR_UNUSED, then the column will not be updated in the repository.
username	The Oracle user name that executed the SQL statement. If the value is NULL or contains the value ADVISOR_UNUSED, then the column will not be updated in the repository. Because a username is an Oracle identifier, the username value must be entered exactly like it is stored in the server. For example, if the user SCOTT is the executing user, then you must provide the user identifier SCOTT in all uppercase letters. It will not recognize the user scott as a match for SCOTT.
search	Disabled.

Usage Notes

A workload cannot be modified or deleted if it is currently referenced by an active task. A task is considered active if it is not in its initial state. See "RESET_TASK Procedure" on page 12-40 to set a task to its initial state.

Examples

```
DECLARE
  workload_name VARCHAR2(30);
  updated NUMBER;
  id NUMBER;
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
                                     100,400,5041,103,640445,680000,2,
                                     1,SYSDATE,1,'SH','SELECT AVG(amount_sold)
                                     FROM sh.sales WHERE promo_id = 10');

   SELECT sql_id INTO id FROM USER_ADVISOR_SQLW_STMTS
   WHERE workload_name = 'My Workload';

  DBMS_ADVISOR.UPDATE_SQLWKLD_STATEMENT(workload_name, id);
END;
/
```

UPDATE_TASK_ATTRIBUTES Procedure

This procedure changes various attributes of a task or a task template.

Syntax

```
DBMS_ADVISOR.UPDATE_TASK_ATTRIBUTES (
   task_name          IN VARCHAR2
   new_name           IN VARCHAR2 := NULL,
   description        IN VARCHAR2 := NULL,
   read_only          IN VARCHAR2 := NULL,
   is_template        IN VARCHAR2 := NULL,
   how_created        IN VARCHAR2 := NULL);
```

Parameters

Table 12–43 UPDATE_TASK_ATTRIBUTES Procedure Parameters

Parameter	Description
task_name	The Advisor task name that uniquely identifies an existing task.
new_name	The new Advisor task name. If the value is NULL or contains the value ADVISOR_UNUSED, the task will not be renamed. A task name can be up to 30 characters long.
description	A new task description. If the value is NULL or contains the value ADVISOR_UNUSED, the description will not be changed. Names can be up to 256 characters long.
read_only	Sets the task to read-only. Possible values are: TRUE and FALSE. If the value is NULL or contains the value ADVISOR_UNUSED, the setting will not be changed.
is_template	Marks the task as a template. Physically, there is no difference between a task and a template; however, a template cannot be executed. Possible values are: TRUE and FALSE. If the value is NULL or contains the value ADVISOR_UNUSED, the setting will not be changed.
how_created	Indicates a source application name that initiated the task creation. If the value is NULL or contains the value ADVISOR_UNUSED, the source will not be changed.

Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
BEGIN
  task_name := 'My Task';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
  DBMS_ADVISOR.UPDATE_TASK_ATTRIBUTES(task_name,'New Task Name');
  DBMS_ADVISOR.UPDATE_TASK_ATTRIBUTES('New Task Name',NULL,'New description');
END;
/
```

13

DBMS_ALERT

DBMS_ALERT supports asynchronous notification of database events (alerts). By appropriate use of this package and database triggers, an application can notify itself whenever values of interest in the database are changed.

This chapter contains the following topics:

Using DBMS_ALERT

Overview

Suppose a graphics tool is displaying a graph of some data from a database table. The graphics tool can, after reading and graphing the data, wait on a database alert (`WAITONE`) covering the data just read. The tool automatically wakes up when the data is changed by any other user. All that is required is that a trigger be placed on the database table, which performs a signal (`SIGNAL`) whenever the trigger is fired.

Security Model

Security on this package can be controlled by granting `EXECUTE` on this package to selected users or roles. You might want to write a cover package on top of this one that restricts the alert names used. `EXECUTE` privilege on this cover package can then be granted rather than on this package.

Constants

The DBMS_ALERT package uses the constants shown in Table 13–1:

Table 13–1 DBMS_ALERT Constants

Name	Type	Value	Description
MAXWAIT	INTEGER	86400000	The maximum time to wait for an alert (1000 days which is essentially forever).

Restrictions

Because database alerters issue commits, they cannot be used with Oracle Forms. For more information on restrictions on calling stored procedures while Oracle Forms is active, refer to your Oracle Forms documentation.

Exceptions

DBMS_ALERT raises the application error -20000 on error conditions. Table 13–2 shows the messages and the procedures that can raise them.

Operational Notes

The following notes relate to general and specific applications:

- Alerts are transaction-based. This means that the waiting session is not alerted until the transaction signalling the alert commits. There can be any number of concurrent signalers of a given alert, and there can be any number of concurrent waiters on a given alert.
- A waiting application is blocked in the database and cannot do any other work.
- An application can register for multiple events and can then wait for any of them to occur using the `WAITANY` procedure.
- An application can also supply an optional `timeout` parameter to the `WAITONE` or `WAITANY` procedures. A `timeout` of 0 returns immediately if there is no pending alert.
- The signalling session can optionally pass a message that is received by the waiting session.
- Alerts can be signalled more often than the corresponding application wait calls. In such cases, the older alerts are discarded. The application always gets the latest alert (based on transaction commit times).
- If the application does not require transaction-based alerts, the `DBMS_PIPE` package may provide a useful alternative.

  See Also: Chapter 70, "DBMS_PIPE"

- If the transaction is rolled back after the call to `SIGNAL`, no alert occurs.
- It is possible to receive an alert, read the data, and find that no data has changed. This is because the data changed after the *prior* alert, but before the data was read for that *prior* alert.
- Usually, Oracle is event-driven; this means that there are no polling loops. There are two cases where polling loops can occur:
  - Shared mode. If your database is running in shared mode, a polling loop is required to check for alerts from another instance. The polling loop defaults to one second and can be set by the `SET_DEFAULTS` procedure.
  - `WAITANY` procedure. If you use the `WAITANY` procedure, and if a signalling session does a signal but does not commit within one second of the signal, a polling loop is required so that this uncommitted alert does not camouflage other alerts. The polling loop begins at a one second interval and exponentially backs off to 30-second intervals.

Table 13–2 DBMS_ALERT Error Messages

Error Message	Procedure
`ORU-10001 lock request error, status: N`	`SIGNAL`
`ORU-10015 error: N waiting for pipe status`	`WAITANY`
`ORU-10016 error: N sending on pipe 'X'`	`SIGNAL`
`ORU-10017 error: N receiving on pipe 'X'`	`SIGNAL`
`ORU-10019 error: N on lock request`	`WAIT`
`ORU-10020 error: N on lock request`	`WAITANY`

Table 13–2 (Cont.) DBMS_ALERT Error Messages

Error Message	Procedure
ORU-10021 lock request error; status: N	REGISTER
ORU-10022 lock request error, status: N	SIGNAL
ORU-10023 lock request error; status N	WAITONE
ORU-10024 there are no alerts registered	WAITANY
ORU-10025 lock request error; status N	REGISTER
ORU-10037 attempting to wait on uncommitted signal from same session	WAITONE

Examples

Suppose you want to graph average salaries by department, for all employees. Your application needs to know whenever EMP is changed. Your application would look similar to this code:

```
DBMS_ALERT.REGISTER('emp_table_alert');
    <<readagain>>:
   /* ... read the emp table and graph it */
      DBMS_ALERT.WAITONE('emp_table_alert', :message, :status);
      if status = 0 then goto <<readagain>>; else
      /* ... error condition */
```

The EMP table would have a trigger similar to this:

```
CREATE TRIGGER emptrig AFTER INSERT OR UPDATE OR DELETE ON emp
    BEGIN
      DBMS_ALERT.SIGNAL('emp_table_alert', 'message_text');
   END;
```

When the application is no longer interested in the alert, it makes this request:

```
DBMS_ALERT.REMOVE('emp_table_alert');
```

This reduces the amount of work required by the alert signaller. If a session exits (or dies) while registered alerts exist, the alerts are eventually cleaned up by future users of this package.

The example guarantees that the application always sees the latest data, although it may not see every intermediate value.

Summary of DBMS_ALERT Subprograms

Table 13–3 DBMS_ALERT Package Subprograms

Subprogram	Description
REGISTER Procedure on page 13-12	Receives messages from an alert
REMOVE Procedure on page 13-13	Disables notification from an alert
REMOVEALL Procedure on page 13-14	Removes all alerts for this session from the registration list
SET_DEFAULTS Procedure on page 13-15	Sets the polling interval
SIGNAL Procedure on page 13-16	Signals an alert (send message to registered sessions)
WAITANY Procedure on page 13-17	Waits `timeout` seconds to receive alert message from an alert registered for session
WAITONE Procedure on page 13-18	Waits `timeout` seconds to receive message from named alert

REGISTER Procedure

This procedure lets a session register interest in an alert.

Syntax

```
DBMS_ALERT.REGISTER (
   name  IN  VARCHAR2);
```

Parameters

Table 13–4 REGISTER Procedure Parameters

Parameter	Description
name	Name of the alert in which this session is interested.

Caution: Alert names beginning with 'ORA$' are reserved for use for products provided by Oracle. Names must be 30 bytes or less. The name is case insensitive.

Usage Notes

A session can register interest in an unlimited number of alerts. Alerts should be deregistered when the session no longer has any interest, by calling REMOVE.

REMOVE Procedure

This procedure enables a session that is no longer interested in an alert to remove that alert from its registration list. Removing an alert reduces the amount of work done by signalers of the alert.

Syntax

```
DBMS_ALERT.REMOVE (
   name  IN  VARCHAR2);
```

Parameters

Table 13–5 REMOVE Procedure Parameters

Parameter	Description
name	Name of the alert (case-insensitive) to be removed from registration list.

Usage Notes

Removing alerts is important because it reduces the amount of work done by signalers of the alert. If a session dies without removing the alert, that alert is eventually (but not immediately) cleaned up.

REMOVEALL Procedure

This procedure removes all alerts for this session from the registration list. You should do this when the session is no longer interested in any alerts.

This procedure is called automatically upon first reference to this package during a session. Therefore, no alerts from prior sessions which may have terminated abnormally can affect this session.

This procedure always performs a commit.

Syntax

```
DBMS_ALERT.REMOVEALL;
```

SET_DEFAULTS Procedure

In case a polling loop is required, use the SET_DEFAULTS procedure to set the polling interval.

Syntax

```
DBMS_ALERT.SET_DEFAULTS (
   sensitivity  IN  NUMBER);
```

Parameters

Table 13–6 SET_DEFAULTS Procedure Parameters

Parameter	Description
sensitivity	Polling interval, in seconds, to sleep between polls. The default interval is five seconds.

SIGNAL Procedure

This procedure signals an alert. The effect of the `SIGNAL` call only occurs when the transaction in which it is made commits. If the transaction rolls back, `SIGNAL` has no effect.

All sessions that have registered interest in this alert are notified. If the interested sessions are currently waiting, they are awakened. If the interested sessions are not currently waiting, they are notified the next time they do a wait call.

Multiple sessions can concurrently perform signals on the same alert. Each session, as it signals the alert, blocks all other concurrent sessions until it commits. This has the effect of serializing the transactions.

Syntax

```
DBMS_ALERT.SIGNAL (
   name     IN  VARCHAR2,
   message  IN  VARCHAR2);
```

Parameters

Table 13–7 SIGNAL Procedure Parameters

Parameter	Description
`name`	Name of the alert to signal.
`message`	Message, of 1800 bytes or less, to associate with this alert. This message is passed to the waiting session. The waiting session might be able to avoid reading the database after the alert occurs by using the information in the message.

WAITANY Procedure

Call this procedure to wait for an alert to occur for any of the alerts for which the current session is registered.

Syntax

```
DBMS_ALERT.WAITANY (
   name      OUT  VARCHAR2,
   message   OUT  VARCHAR2,
   status    OUT  INTEGER,
   timeout   IN   NUMBER DEFAULT MAXWAIT);
```

Parameters

Table 13–8 WAITANY Procedure Parameters

Parameter	Description
name	Returns the name of the alert that occurred.
message	Returns the message associated with the alert. This is the message provided by the SIGNAL call. If multiple signals on this alert occurred before WAITANY, the message corresponds to the most recent SIGNAL call. Messages from prior SIGNAL calls are discarded.
status	Values returned: 0 - alert occurred 1 - timeout occurred
timeout	Maximum time to wait for an alert. If no alert occurs before timeout seconds, this returns a status of 1.

Usage Notes

An implicit COMMIT is issued before this procedure is executed. The same session that waits for the alert may also first signal the alert. In this case remember to commit after the signal and before the wait; otherwise, DBMS_LOCK.REQUEST (which is called by DBMS_ALERT) returns status 4.

Exceptions

```
-20000, ORU-10024: there are no alerts registered.
```

WAITONE Procedure

This procedure waits for a specific alert to occur. An implicit COMMIT is issued before this procedure is executed. A session that is the first to signal an alert can also wait for the alert in a subsequent transaction. In this case, remember to commit after the signal and before the wait; otherwise, DBMS_LOCK.REQUEST (which is called by DBMS_ALERT) returns status 4.

Syntax

```
DBMS_ALERT.WAITONE (
   name      IN   VARCHAR2,
   message   OUT  VARCHAR2,
   status    OUT  INTEGER,
   timeout   IN   NUMBER DEFAULT MAXWAIT);
```

Parameters

Table 13–9 WAITONE Procedure Parameters

Parameter	Description
name	Name of the alert to wait for.
message	Returns the message associated with the alert. This is the message provided by the SIGNAL call. If multiple signals on this alert occurred before WAITONE, the message corresponds to the most recent SIGNAL call. Messages from prior SIGNAL calls are discarded.
status	Values returned: 0 - alert occurred 1 - timeout occurred
timeout	Maximum time to wait for an alert. If the named alert does not occurs before timeout seconds, this returns a status of 1.

14

DBMS_APPLICATION_INFO

Application developers can use the `DBMS_APPLICATION_INFO` package with Oracle Trace and the SQL trace facility to record names of executing modules or transactions in the database for later use when tracking the performance of various modules and debugging.

This chapter contains the following topics:

Using DBMS_APPLICATION_INFO

Overview

Registering the application allows system administrators and performance tuning specialists to track performance by module. System administrators can also use this information to track resource use by module. When an application registers with the database, its name and actions are recorded in the `V$SESSION` and `V$SQLAREA` views.

Security Model

> **Note:** The public synonym for `DBMS_APPLICATION_INFO` is not dropped before creation so that you can redirect the public synonym to point to your own package.

No further privileges are required. The `DBMSAPIN.SQL` script is already run by `catproc`.

Operational Notes

Your applications should set the name of the module and name of the action automatically each time a user enters that module. The module name could be the name of a form in an Oracle Forms application, or the name of the code segment in an Oracle Precompilers application. The action name should usually be the name or description of the current transaction within a module.

If you want to gather your own statistics based on module, you can implement a wrapper around this package by writing a version of this package in another schema that first gathers statistics and then calls the `SYS` version of the package. The public synonym for `DBMS_APPLICATION_INFO` can then be changed to point to the DBA's version of the package.

Summary of DBMS_APPLICATION_INFO Subprograms

Table 14–1 DBMS_APPLICATION_INFO Package Subprograms

Subprogram	Description
READ_CLIENT_INFO Procedure on page 14-7	Reads the value of the `client_info` field of the current session
READ_MODULE Procedure on page 14-8	Reads the values of the module and action fields of the current session
SET_ACTION Procedure on page 14-9	Sets the name of the current action within the current module
SET_CLIENT_INFO Procedure on page 14-10	Sets the `client_info` field of the session
SET_MODULE Procedure on page 14-11	Sets the name of the module that is currently running to a new module
SET_SESSION_LONGOPS Procedure on page 14-12	Sets a row in the `V$SESSION_LONGOPS` table

READ_CLIENT_INFO Procedure

This procedure reads the value of the `client_info` field of the current session.

Syntax

```
DBMS_APPLICATION_INFO.READ_CLIENT_INFO (
   client_info OUT VARCHAR2);
```

Parameters

Table 14–2 READ_CLIENT_INFO Procedure Parameters

Parameter	Description
`client_info`	Last client information value supplied to the `SET_CLIENT_INFO` procedure.

READ_MODULE Procedure

This procedure reads the values of the module and action fields of the current session.

Syntax

```
DBMS_APPLICATION_INFO.READ_MODULE (
   module_name OUT VARCHAR2,
   action_name OUT VARCHAR2);
```

Parameters

Table 14–3 READ_MODULE Procedure Parameters

Parameter	Description
module_name	Last value that the module name was set to by calling SET_MODULE.
action_name	Last value that the action name was set to by calling SET_ACTION or SET_MODULE.

Usage Notes

Module and action names for a registered application can be retrieved by querying V$SQLAREA or by calling the READ_MODULE procedure. Client information can be retrieved by querying the V$SESSION view, or by calling the READ_CLIENT_INFO Procedure.

Examples

The following sample query illustrates the use of the MODULE and ACTION column of the V$SQLAREA.

```
SELECT sql_text, disk_reads, module, action
FROM v$sqlarea
WHERE module = 'add_employee';

SQL_TEXT DISK_READS MODULE ACTION
------------------- ---------- ----------------- ----------------
INSERT INTO emp 1 add_employee insert into emp
(ename, empno, sal, mgr, job, hiredate, comm, deptno)
VALUES
(name, next.emp_seq, manager, title, SYSDATE, commission, department)

1 row selected.
```

SET_ACTION Procedure

This procedure sets the name of the current action within the current module.

Syntax

```
DBMS_APPLICATION_INFO.SET_ACTION (
   action_name IN VARCHAR2);
```

Parameters

Table 14–4 SET_ACTION Procedure Parameters

Parameter	Description
action_name	The name of the current action within the current module. When the current action terminates, call this procedure with the name of the next action if there is one, or NULL if there is not. Names longer than 32 bytes are truncated.

Usage Notes

The action name should be descriptive text about the current action being performed. You should probably set the action name before the start of every transaction.

Set the transaction name to NULL after the transaction completes, so that subsequent transactions are logged correctly. If you do not set the transaction name to NULL, subsequent transactions may be logged with the previous transaction's name.

Example

The following is an example of a transaction that uses the registration procedure:

```
CREATE OR REPLACE PROCEDURE bal_tran (amt IN NUMBER(7,2)) AS
BEGIN

-- balance transfer transaction

   DBMS_APPLICATION_INFO.SET_ACTION(
      action_name => 'transfer from chk to sav');
   UPDATE chk SET bal = bal + :amt
      WHERE acct# = :acct;
   UPDATE sav SET bal = bal - :amt
      WHERE acct# = :acct;
   COMMIT;
   DBMS_APPLICATION_INFO.SET_ACTION(null);

END;
```

SET_CLIENT_INFO Procedure

This procedure supplies additional information about the client application.

Syntax

```
DBMS_APPLICATION_INFO.SET_CLIENT_INFO (
   client_info IN VARCHAR2);
```

Parameters

Table 14–5 SET_CLIENT_INFO Procedure Parameters

Parameter	Description
`client_info`	Supplies any additional information about the client application. This information is stored in the `V$SESSION` view. Information exceeding 64 bytes is truncated.

Note: `CLIENT_INFO` is readable and writable by any user. For storing secured application attributes, you can use the application context feature.

SET_MODULE Procedure

This procedure sets the name of the current application or module.

Syntax

```
DBMS_APPLICATION_INFO.SET_MODULE (
   module_name IN VARCHAR2,
   action_name IN VARCHAR2);
```

Parameters

Table 14–6 SET_MODULE Procedure Parameters

Parameter	Description
module_name	Name of module that is currently running. When the current module terminates, call this procedure with the name of the new module if there is one, or NULL if there is not. Names longer than 48 bytes are truncated.
action_name	Name of current action within the current module. If you do not want to specify an action, this value should be NULL. Names longer than 32 bytes are truncated.

Usage Notes

Example

```
CREATE or replace PROCEDURE add_employee(
  name VARCHAR2,
  salary NUMBER,
  manager NUMBER,
  title VARCHAR2,
  commission NUMBER,
  department NUMBER) AS
BEGIN
  DBMS_APPLICATION_INFO.SET_MODULE(
    module_name => 'add_employee',
    action_name => 'insert into emp');
  INSERT INTO emp
    (ename, empno, sal, mgr, job, hiredate, comm, deptno)
    VALUES (name, emp_seq.nextval, salary, manager, title, SYSDATE,
            commission, department);
  DBMS_APPLICATION_INFO.SET_MODULE(null,null);
END;
```

SET_SESSION_LONGOPS Procedure

This procedure sets a row in the V$SESSION_LONGOPS view. This is a view that is used to indicate the on-going progress of a long running operation. Some Oracle functions, such as parallel execution and Server Managed Recovery, use rows in this view to indicate the status of, for example, a database backup.

Applications may use the SET_SESSION_LONGOPS procedure to advertise information on the progress of application specific long running tasks so that the progress can be monitored by way of the V$SESSION_LONGOPS view.

Syntax

```
DBMS_APPLICATION_INFO.SET_SESSION_LONGOPS (
   rindex      IN OUT BINARY_INTEGER,
   slno        IN OUT BINARY_INTEGER,
   op_name     IN     VARCHAR2       DEFAULT NULL,
   target      IN     BINARY_INTEGER DEFAULT 0,
   context     IN     BINARY_INTEGER DEFAULT 0,
   sofar       IN     NUMBER         DEFAULT 0,
   totalwork   IN     NUMBER         DEFAULT 0,
   target_desc IN     VARCHAR2       DEFAULT 'unknown target',
   units       IN     VARCHAR2       DEFAULT NULL)

set_session_longops_nohint constant BINARY_INTEGER := -1;
```

Parameters

Table 14–7 SET_SESSION_LONGOPS Procedure Parameters

Parameter	Description
rindex	A token which represents the v$session_longops row to update. Set this to set_session_longops_nohint to start a new row. Use the returned value from the prior call to reuse a row.
slno	Saves information across calls to set_session_longops: It is for internal use and should not be modified by the caller.
op_name	Specifies the name of the long running task. It appears as the OPNAME column of v$session_longops. The maximum length is 64 bytes.
target	Specifies the object that is being worked on during the long running operation. For example, it could be a table ID that is being sorted. It appears as the TARGET column of v$session_longops.
context	Any number the client wants to store. It appears in the CONTEXT column of v$session_longops.
sofar	Any number the client wants to store. It appears in the SOFAR column of v$session_longops. This is typically the amount of work which has been done so far.
totalwork	Any number the client wants to store. It appears in the TOTALWORK column of v$session_longops. This is typically an estimate of the total amount of work needed to be done in this long running operation.

Table 14–7 (Cont.) SET_SESSION_LONGOPS Procedure Parameters

Parameter	Description
target_desc	Specifies the description of the object being manipulated in this long operation. This provides a caption for the target parameter. This value appears in the TARGET_DESC field of v$session_longops. The maximum length is 32 bytes.
units	Specifies the units in which sofar and totalwork are being represented. It appears as the UNITS field of v$session_longops. The maximum length is 32 bytes.

Example

This example performs a task on 10 objects in a loop. As the example completes each object, Oracle updates V$SESSION_LONGOPS on the procedure's progress.

```
DECLARE
        rindex    BINARY_INTEGER;
        slno      BINARY_INTEGER;
        totalwork number;
        sofar     number;
        obj       BINARY_INTEGER;

      BEGIN
        rindex := dbms_application_info.set_session_longops_nohint;
        sofar := 0;
        totalwork := 10;

        WHILE sofar < 10 LOOP
          -- update obj based on sofar
          -- perform task on object target

          sofar := sofar + 1;
          dbms_application_info.set_session_longops(rindex, slno,
            "Operation X", obj, 0, sofar, totalwork, "table", "tables");
        END LOOP;
      END;
```

15

DBMS_APPLY_ADM

The DBMS_APPLY_ADM package, one of a set of Streams packages, provides subprograms to start, stop, and configure an apply process. This package includes subprograms for configuring apply handlers, setting enqueue destinations for messages, and specifying execution directives for messages. This package also provides administrative subprograms that set the instantiation SCN for objects at a destination database. This package also includes subprograms for managing apply errors.

> **See Also:** *Oracle Streams Concepts and Administration* and *Oracle Streams Replication Administrator's Guide* for more information about this package and apply processes

This chapter contains the following topic:

Summary of DBMS_APPLY_ADM Subprograms

Table 15–1 DBMS_APPLY_ADM Package Subprograms

Subprogram	Description
ALTER_APPLY Procedure on page 15-4	Alters an apply process
COMPARE_OLD_VALUES Procedure on page 15-9	Specifies whether to compare the old value of one or more columns in a row logical change record (row LCR) with the current value of the corresponding columns at the destination site during apply
CREATE_APPLY Procedure on page 15-11	Creates an apply process
CREATE_OBJECT_DEPENDENCY Procedure on page 15-18	Creates an object dependency
DELETE_ALL_ERRORS Procedure on page 15-19	Deletes all the error transactions for the specified apply process
DELETE_ERROR Procedure on page 15-20	Deletes the specified error transaction
DROP_APPLY Procedure on page 15-21	Drops an apply process
DROP_OBJECT_DEPENDENCY on page 15-22	Drops an object dependency
EXECUTE_ALL_ERRORS Procedure on page 15-23	Reexecutes the error transactions for the specified apply process.
EXECUTE_ERROR Procedure on page 15-24	Reexecutes the specified error transaction
GET_ERROR_MESSAGE Function on page 15-27	Returns the message payload from the error queue for the specified message number and transaction identifier
SET_DML_HANDLER Procedure on page 15-28	Alters operation options for a specified object with a specified apply process
SET_ENQUEUE_DESTINATION Procedure on page 15-33	Sets the queue where the apply process automatically enqueues a message that satisfies the specified rule
SET_EXECUTE Procedure on page 15-35	Specifies whether a message that satisfies the specified rule is executed by an apply process
SET_GLOBAL_INSTANTIATION_SCN Procedure on page 15-37	Records the specified instantiation SCN for the specified source database and, optionally, for the schemas at the source database and the tables owned by these schemas
SET_KEY_COLUMNS Procedures on page 15-39	Records the set of columns to be used as the substitute primary key for local apply purposes and removes existing substitute primary key columns for the specified object if they exist
SET_PARAMETER Procedure on page 15-41	Sets an apply parameter to the specified value
SET_SCHEMA_INSTANTIATION_SCN Procedure on page 15-46	Records the specified instantiation SCN for the specified schema in the specified source database and, optionally, for the tables owned by the schema at the source database

Table 15–1 (Cont.) DBMS_APPLY_ADM Package Subprograms

Subprogram	Description
SET_TABLE_INSTANTIATION_SCN Procedure on page 15-48	Records the specified instantiation SCN for the specified table in the specified source database
SET_UPDATE_CONFLICT_HANDLER Procedure on page 15-50	Adds, updates, or drops an update conflict handler for the specified object
SET_VALUE_DEPENDENCY Procedure on page 15-53	Sets or removes a value dependency
START_APPLY Procedure on page 15-54	Directs the apply process to start applying messages
STOP_APPLY Procedure on page 15-55	Stops the apply process from applying any messages and rolls back any unfinished transactions being applied

Note: All procedures commit unless specified otherwise. However, the `GET_ERROR_MESSAGE` function does not commit.

ALTER_APPLY Procedure

This procedure alters an apply process.

Syntax

```
DBMS_APPLY_ADM.ALTER_APPLY(
   apply_name                IN  VARCHAR2,
   rule_set_name             IN  VARCHAR2  DEFAULT NULL,
   remove_rule_set           IN  BOOLEAN   DEFAULT FALSE,
   message_handler           IN  VARCHAR2  DEFAULT NULL
   remove_message_handler    IN  BOOLEAN   DEFAULT FALSE,
   ddl_handler               IN  VARCHAR2  DEFAULT NULL,
   remove_ddl_handler        IN  BOOLEAN   DEFAULT FALSE,
   apply_user                IN  VARCHAR2  DEFAULT NULL,
   apply_tag                 IN  RAW       DEFAULT NULL,
   remove_apply_tag          IN  BOOLEAN   DEFAULT FALSE,
   precommit_handler         IN  VARCHAR2  DEFAULT NULL,
   remove_precommit_handler  IN  BOOLEAN   DEFAULT FALSE,
   negative_rule_set_name    IN  VARCHAR2  DEFAULT NULL,
   remove_negative_rule_set  IN  BOOLEAN   DEFAULT FALSE);
```

Parameters

Table 15–2 ALTER_APPLY Procedure Parameters

Parameter	Description
apply_name	The name of the apply process being altered. You must specify an existing apply process name. Do not specify an owner.
rule_set_name	The name of the positive rule set for the apply process. The positive rule set contains the rules that instruct the apply process to apply messages. If you want to use a positive rule set for the apply process, then you must specify an existing rule set in the form [*schema_name.*]*rule_set_name*. For example, to specify a positive rule set in the hr schema named job_apply_rules, enter hr.job_apply_rules. If the schema is not specified, then the current user is the default. An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the DBMS_STREAMS_ADM package or the DBMS_RULE_ADM package. If you specify NULL and the remove_rule_set parameter is set to FALSE, then this procedure retains any existing positive rule set. If you specify NULL and the remove_rule_set parameter is set to TRUE, then this procedure removes any existing positive rule set.

Table 15–2 (Cont.) ALTER_APPLY Procedure Parameters

Parameter	Description
remove_rule_set	If TRUE, then the procedure removes the positive rule set for the specified apply process. If you remove the positive rule set for an apply process, and the apply process does not have a negative rule set, then the apply process dequeues all messages in its queue. If you remove the positive rule set for an apply process, and a negative rule set exists for the apply process, then the apply process dequeues all messages in its queue that are not discarded by the negative rule set. If FALSE, then the procedure retains the positive rule set for the specified apply process. If the rule_set_name parameter is non-NULL, then this parameter should be set to FALSE.
message_handler	A user-defined procedure that processes non-LCR messages in the queue for the apply process. See "Usage Notes" on page 15-16 in the CREATE_APPLY Procedure for more information about a message handler procedure.
remove_message_handler	If TRUE, then the procedure removes the message handler for the specified apply process. If FALSE, then the procedure retains any message handler for the specified apply process. If the message_handler parameter is non-NULL, then this parameter should be set to FALSE.
ddl_handler	A user-defined procedure that processes DDL logical change records (DDL LCRs) in the queue for the apply process. All applied DDL LCRs commit automatically. Therefore, if a DDL handler calls the EXECUTE member procedure of a DDL LCR, then a commit is performed automatically. See "Usage Notes" on page 15-16 in the CREATE_APPLY Procedure for more information about a DDL handler procedure.
remove_ddl_handler	If TRUE, then the procedure removes the DDL handler for the specified apply process. If FALSE, then the procedure retains any DDL handler for the specified apply process. If the ddl_handler parameter is non-NULL, then this parameter should be set to FALSE.

Table 15–2 (Cont.) ALTER_APPLY Procedure Parameters

Parameter	Description
apply_user	The user in whose security domain an apply process dequeues messages that satisfy its rule sets, applies messages directly to database objects, runs custom rule-based transformations configured for apply process rules, and runs apply handlers configured for the apply process. If NULL, then the apply user is not changed. If a non-NULL value is specified to change the apply user, then the user who invokes the ALTER_APPLY procedure must be granted DBA role. Only the SYS user can set the apply_user to SYS. If you change the apply user, then this procedure grants the new apply user dequeue privilege on the queue used by the apply process and configures the user as a secure queue user of the queue. In addition to the privileges granted by this procedure, you also should grant the following privileges to the apply user: ■ The necessary privileges to perform DML and DDL changes on the apply objects ■ EXECUTE privilege on the rule sets used by the apply process ■ EXECUTE privilege on all rule-based transformation functions used in the rule set ■ EXECUTE privilege on all apply handler procedures ■ EXECUTE privilege on all packages, including Oracle-supplied packages, that are invoked in subprograms run by the apply process These privileges must be granted directly to the apply user. They cannot be granted through roles. By default, this parameter is set to the user who created the apply process by running either the CREATE_APPLY procedure in this package or a procedure in the DBMS_STREAMS_ADM package. **Note:** If the specified user is dropped using DROP USER... CASCADE, then the apply_user for the apply process is set to NULL automatically. You must specify an apply user before the apply process can run.
apply_tag	A binary tag that is added to redo entries generated by the specified apply process. The tag is a binary value that can be used to track LCRs. The tag is relevant only if a capture process at the database where the apply process is running will capture changes made by the apply process. If so, then the captured changes will include the tag specified by this parameter. If NULL, the default, then the apply tag for the apply process is not changed. The following is an example of a tag with a hexadecimal value of 17: HEXTORAW('17') **See Also:** *Oracle Streams Replication Administrator's Guide* for more information about tags

Table 15–2 (Cont.) ALTER_APPLY Procedure Parameters

Parameter	Description
remove_apply_tag	If TRUE, then the procedure sets the apply tag for the specified apply process to NULL, and the apply process generates redo entries with NULL tags. If FALSE, then the procedure retains any apply tag for the specified apply process. If the apply_tag parameter is non-NULL, then this parameter should be set to FALSE.
precommit_handler	A user-defined procedure that can receive internal commit directives in the queue for the apply process before they are processed by the apply process. Typically, precommit handlers are used for auditing commit information for transactions processed by an apply process. An internal commit directive is enqueued in the following ways: ▪ When a capture process captures row LCRs, the capture process enqueues the commit directive for the transaction that contains the row LCRs. ▪ When a user or application enqueues messages and then issues a COMMIT statement, the commit directive is enqueued automatically. For a captured row LCR, a commit directive contains the commit SCN of the transaction from the source database. For a user-enqueued message, the commit SCN is generated by the apply process. The precommit handler procedure must conform to the following restrictions: ▪ Any work that commits must be an autonomous transaction. ▪ Any rollback must be to a named savepoint created in the procedure. If a precommit handler raises an exception, then the entire apply transaction is rolled back, and all of the messages in the transaction are moved to the error queue. See "Usage Notes" on page 15-16 in the CREATE_APPLY Procedure for more information about a precommit handler procedure.
remove_precommit_handler	If TRUE, then the procedure removes the precommit handler for the specified apply process. If FALSE, then the procedure retains any precommit handler for the specified apply process. If the precommit_handler parameter is non-NULL, then this parameter should be set to FALSE.

Table 15–2 (Cont.) ALTER_APPLY Procedure Parameters

Parameter	Description
negative_rule_set_name	The name of the negative rule set for the apply process. The negative rule set contains the rules that instruct the apply process to discard messages. If you want to use a negative rule set for the apply process, then you must specify an existing rule set in the form [*schema_name*.]*rule_set_name*. For example, to specify a negative rule set in the hr schema named neg_apply_rules, enter hr.neg_apply_rules. If the schema is not specified, then the current user is the default. An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the DBMS_STREAMS_ADM package or the DBMS_RULE_ADM package. If you specify NULL and the remove_negative_rule_set parameter is set to FALSE, then the procedure retains any existing negative rule set. If you specify NULL and the remove_negative_rule_set parameter is set to TRUE, then the procedure removes any existing negative rule set. If you specify both a positive and a negative rule set for an apply process, then the negative rule set is always evaluated first.
remove_negative_rule_set	If TRUE, then the procedure removes the negative rule set for the specified apply process. If you remove the negative rule set for an apply process, and the apply process does not have a positive rule set, then the apply process dequeues all messages in its queue. If you remove the negative rule set for an apply process, and a positive rule set exists for the apply process, then the apply process dequeues all messages in its queue that are not discarded by the positive rule set. If FALSE, then the procedure retains the negative rule set for the specified apply process. If the negative_rule_set_name parameter is non-NULL, then this parameter should be set to FALSE.

Usage Notes

An apply process is stopped and restarted automatically when you change the value of one or more of the following ALTER_APPLY procedure parameters:

- message_handler
- ddl_handler
- apply_user
- apply_tag
- precommit_handler

COMPARE_OLD_VALUES Procedure

This procedure specifies whether to compare the old value of one or more columns in a row logical change record (row LCR) with the current value of the corresponding columns at the destination site during apply. This procedure is relevant only for `UPDATE` and `DELETE` operations because only these operations result in old column values in row LCRs. The default is to compare old values for all columns.

This procedure is overloaded. The `column_list` and `column_table` parameters are mutually exclusive.

> **See Also:** *Oracle Streams Replication Administrator's Guide* for more information about conflict detection and resolution in a Streams environment

Syntax

```
DBMS_APPLY_ADM.COMPARE_OLD_VALUES(
   object_name         IN VARCHAR2,
   column_list         IN VARCHAR2,
   operation           IN VARCHAR2 DEFAULT 'UPDATE',
   compare             IN BOOLEAN  DEFAULT TRUE,
   apply_database_link IN VARCHAR2 DEFAULT NULL);

DBMS_APPLY_ADM.COMPARE_OLD_VALUES(
   object_name         IN VARCHAR2,
   column_table        IN DBMS_UTILITY.LNAME_ARRAY,
   operation           IN VARCHAR2 DEFAULT 'UPDATE',
   compare             IN BOOLEAN  DEFAULT TRUE,
   apply_database_link IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 15–3 COMPARE_OLD_VALUES Procedure Parameters

Parameter	Description
`object_name`	The name of the source table specified as `[`*`schema_name`*`.]`*`object_name`*. For example, `hr.employees`. If the schema is not specified, then the current user is the default.
`column_list`	A comma-delimited list of column names in the table. There must be no spaces between entries. Specify `*` to include all nonkey columns.
`column_table`	A PL/SQL index-by table of type `DBMS_UTILITY.LNAME_ARRAY` that contains names of columns in the table. The first column name should be at position 1, the second at position 2, and so on. The table does not need to be `NULL` terminated.
`operation`	The name of the operation, which can be specified as: ■ `UPDATE` for `UPDATE` operations ■ `DELETE` for `DELETE` operations ■ `*` for both `UPDATE` and `DELETE` operations
`compare`	If `compare` is `TRUE`, the old values of the specified columns are compared during apply. If `compare` is `FALSE`, the old values of the specified columns are not compared during apply.

Table 15–3 (Cont.) COMPARE_OLD_VALUES Procedure Parameters

Parameter	Description
`apply_database_link`	The name of the database link to a non-Oracle database. This parameter should be set only when the destination database is a non-Oracle database.

Usage Notes

By default, an apply process uses the old column values in a row LCR to detect conflicts. You can choose not to compare old column values to avoid conflict detection for specific tables. For example, if you use a time column for conflict detection, then an apply process does not need to check old values for non-key and nontime columns.

Note: An apply process always compares old values for key columns when they are present in a row LCR. This procedure raises an error if a key column is specified in `column_list` or `column_table` and the `compare` parameter is set to `FALSE`.

CREATE_APPLY Procedure

This procedure creates an apply process.

> **Note:** The user who invokes this procedure must be granted `DBA` role.

Syntax

```
DBMS_APPLY_ADM.CREATE_APPLY(
   queue_name              IN  VARCHAR2,
   apply_name              IN  VARCHAR2,
   rule_set_name           IN  VARCHAR2  DEFAULT NULL,
   message_handler         IN  VARCHAR2  DEFAULT NULL,
   ddl_handler             IN  VARCHAR2  DEFAULT NULL,
   apply_user              IN  VARCHAR2  DEFAULT NULL,
   apply_database_link     IN  VARCHAR2  DEFAULT NULL,
   apply_tag               IN  RAW       DEFAULT '00',
   apply_captured          IN  BOOLEAN   DEFAULT FALSE,
   precommit_handler       IN  VARCHAR2  DEFAULT NULL,
   negative_rule_set_name  IN  VARCHAR2  DEFAULT NULL,
   source_database         IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 15–4 CREATE_APPLY Procedure Parameters

Parameter	Description
`queue_name`	The name of the queue from which the apply process dequeues messages. You must specify an existing queue in the form `[`*`schema_name`*`.]`*`queue_name`*. For example, to specify a queue in the `hr` schema named `streams_queue`, enter `hr.streams_queue`. If the schema is not specified, then the current user is the default. **Note:** The `queue_name` setting cannot be altered after the apply process is created.
`apply_name`	The name of the apply process being created. A `NULL` specification is not allowed. Do not specify an owner. The specified name must not match the name of an existing apply process or messaging client. **Note:** The `apply_name` setting cannot be altered after the apply process is created.

Table 15–4 (Cont.) CREATE_APPLY Procedure Parameters

Parameter	Description
rule_set_name	The name of the positive rule set for the apply process. The positive rule set contains the rules that instruct the apply process to apply messages. If you want to use a positive rule set for the apply process, then you must specify an existing rule set in the form [*schema_name*.]*rule_set_name*. For example, to specify a positive rule set in the hr schema named job_apply_rules, enter hr.job_apply_rules. If the schema is not specified, then the current user is the default. If you specify NULL, and no negative rule set is specified, then the apply process applies either all captured messages or all user-enqueued messages in the queue, depending on the setting of the apply_captured parameter. An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the DBMS_STREAMS_ADM package or the DBMS_RULE_ADM package.
message_handler	A user-defined procedure that processes non-LCR messages in the queue for the apply process. See "Usage Notes" on page 15-16 for more information about a message handler procedure.
ddl_handler	A user-defined procedure that processes DDL logical change record (DDL LCRs) in the queue for the apply process. All applied DDL LCRs commit automatically. Therefore, if a DDL handler calls the EXECUTE member procedure of a DDL LCR, then a commit is performed automatically. See "Usage Notes" on page 15-16 for more information about a DDL handler procedure.

Table 15–4 (Cont.) CREATE_APPLY Procedure Parameters

Parameter	Description
apply_user	The user who applies all DML and DDL changes that satisfy the apply process rule sets and who runs user-defined apply handlers. If NULL, then the user who runs the CREATE_APPLY procedure is used. Only a user who is granted DBA role can set an apply user. Only the SYS user can set the apply_user to SYS. The apply user is the user in whose security domain an apply process dequeues messages that satisfy its rule sets, applies messages directly to database objects, runs custom rule-based transformations configured for apply process rules, and runs apply handlers configured for the apply process. This user must have the necessary privileges to apply changes. This procedure grants the apply user dequeue privilege on the queue used by the apply process and configures the user as a secure queue user of the queue. In addition to the privileges granted by this procedure, you also should grant the following privileges to the apply user: ■ The necessary privileges to perform DML and DDL changes on the apply objects ■ EXECUTE privilege on the rule sets used by the apply process ■ EXECUTE privilege on all rule-based transformation functions used in the rule set ■ EXECUTE privilege on all apply handler procedures ■ EXECUTE privilege on all packages, including Oracle-supplied packages, that are invoked in subprograms run by the apply process These privileges must be granted directly to the apply user. They cannot be granted through roles. **Note:** If the specified user is dropped using DROP USER... CASCADE, then the apply_user setting for the apply process is set to NULL automatically. You must specify an apply user before the apply process can run.
apply_database_link	The database at which the apply process applies messages. This parameter is used by an apply process when applying changes from Oracle to non-Oracle systems, such as Sybase. Set this parameter to NULL to specify that the apply process applies messages at the local database. **Note:** The apply_database_link setting cannot be altered after the apply process is created.

Table 15–4 (Cont.) CREATE_APPLY Procedure Parameters

Parameter	Description
apply_tag	A binary tag that is added to redo entries generated by the specified apply process. The tag is a binary value that can be used to track LCRs. The tag is relevant only if a capture process at the database where the apply process is running will capture changes made by the apply process. If so, then the captured changes will include the tag specified by this parameter. By default, the tag for an apply process is the hexadecimal equivalent of '00' (double zero). The following is an example of a tag with a hexadecimal value of 17: HEXTORAW('17') If NULL, then the apply process generates redo entries with NULL tags. **See Also:** *Oracle Streams Replication Administrator's Guide* for more information about tags
apply_captured	Either TRUE or FALSE. If TRUE, then the apply process applies only the messages in a queue that were captured by a Streams capture process. If FALSE, then the apply process applies only the user-enqueued messages in a queue. These messages are user messages that were not captured by a Streams capture process. These messages might or might not contain a user-created LCR. To apply both captured and user-enqueued messages in a queue, you must create at least two apply processes. **Note:** The apply_captured setting cannot be altered after the apply process is created. See Also: *Oracle Streams Concepts and Administration* for more information about processing captured or user-enqueued messages with an apply process

Table 15–4 (Cont.) CREATE_APPLY Procedure Parameters

Parameter	Description
precommit_handler	A user-defined procedure that can receive internal commit directives in the queue for the apply process before they are processed by the apply process. Typically, precommit handlers are used for auditing commit information for transactions processed by an apply process. An internal commit directive is enqueued in the following ways: ▪ When a capture process captures row LCRs, the capture process enqueues the commit directive for the transaction that contains the row LCRs. ▪ When a user or application enqueues messages and then issues a COMMIT statement, the commit directive is enqueued automatically. For a captured row LCR, a commit directive contains the commit SCN of the transaction from the source database. For a user-enqueued message, the commit SCN is generated by the apply process. The precommit handler procedure must conform to the following restrictions: ▪ Any work that commits must be an autonomous transaction. ▪ Any rollback must be to a named savepoint created in the procedure. If a precommit handler raises an exception, then the entire apply transaction is rolled back, and all of the messages in the transaction are moved to the error queue. See "Usage Notes" on page 15-16 for more information about a precommit handler procedure.
negative_rule_set_name	The name of the negative rule set for the apply process. The negative rule set contains the rules that instruct the apply process to discard messages. If you want to use a negative rule set for the apply process, then you must specify an existing rule set in the form [*schema_name*.]*rule_set_name*. For example, to specify a negative rule set in the hr schema named neg_apply_rules, enter hr.neg_apply_rules. If the schema is not specified, then the current user is the default. If you specify NULL, and no positive rule set is specified, then the apply process applies either all captured messages or all user-enqueued messages in the queue, depending on the setting of the apply_captured parameter. An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the DBMS_STREAMS_ADM package or the DBMS_RULE_ADM package. If you specify both a positive and a negative rule set for an apply process, then the negative rule set is always evaluated first.

Table 15–4 (Cont.) CREATE_APPLY Procedure Parameters

Parameter	Description
source_database	The global name of the source database of the changes that will be applied by the apply process. The source database is the database where the changes originated. If an apply process applies captured messages, then the apply process can apply messages from only one capture process at one source database. If NULL, then the source database name of the first LCR received by the apply process is used for the source database. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically. The rules in the apply process rule sets determine which messages are dequeued by the apply process. If the apply process dequeues an LCR with a source database that is different than the source database for the apply process, then an error is raised. You can determine the source database for an apply process by querying the DBA_APPLY_PROGRESS data dictionary view.

Usage Notes

The following sections describe usage notes for this procedure:

Handler Procedure Names

For the message_handler, ddl_handler, and precommit_handler parameters, specify an existing procedure in one of the following forms:

- *[schema_name.]procedure_name*
- *[schema_name.]package_name.procedure_name*

If the procedure is in a package, then the package_name must be specified. For example, to specify a procedure in the apply_pkg package in the hr schema named process_ddls, enter hr.apply_pkg.process_ddls. An error is returned if the specified procedure does not exist.

The user who invokes the CREATE_APPLY procedure must have EXECUTE privilege on a specified handler procedure. Also, if the *schema_name* is not specified, then the user who invokes the CREATE_APPLY procedure is the default.

Message Handler and DDL Handler Procedure

The procedure specified in both the message_handler parameter and the ddl_handler parameter must have the following signature:

```
PROCEDURE handler_procedure (
   parameter_name  IN  ANYDATA);
```

Here, *handler_procedure* stands for the name of the procedure and *parameter_name* stands for the name of the parameter passed to the procedure. For the message handler, the parameter passed to the procedure is a ANYDATA encapsulation of a user message. For the DDL handler procedure, the parameter passed to the procedure is a ANYDATA encapsulation of a DDL LCR.

See Also: Chapter 188, "Logical Change Record TYPEs" for information about DDL LCRs

Precommit Handler Procedure

The procedure specified in the `precommit_handler` parameter must have the following signature:

```
PROCEDURE handler_procedure (
   parameter_name   IN  NUMBER);
```

Here, *`handler_procedure`* stands for the name of the procedure and *`parameter_name`* stands for the name of the parameter passed to the procedure. The parameter passed to the procedure is the commit SCN of a commit directive.

CREATE_OBJECT_DEPENDENCY Procedure

This procedure creates an object dependency. An object dependency is a virtual dependency definition that defines a parent-child relationship between two objects at a destination database.

An apply process schedules execution of transactions that involve the child object after all transactions with a lower commit system change number (commit SCN) that involve the parent object have been committed. An apply process uses the object identifier of the objects in the logical change records (LCRs) to detect dependencies. The apply process does not use column values in the LCRs to detect dependencies.

Note: An error is raised if `NULL` is specified for either of the procedure parameters.

See Also:

- "DROP_OBJECT_DEPENDENCY" on page 15-22
- *Oracle Streams Replication Administrator's Guide*

Syntax

```
DBMS_APPLY_ADM.CREATE_OBJECT_DEPENDENCY(
   object_name         IN  VARCHAR2,
   parent_object_name  IN  VARCHAR2);
```

Parameters

Table 15–5 CREATE_OBJECT_DEPENDENCY Procedure Parameters

Parameter	Description
`object_name`	The name of the child database object, specified as `[`*`schema_name`*`.]`*`object_name`*. For example, `hr.employees`. If the schema is not specified, then the current user is the default.
`parent_object_name`	The name of the parent database object, specified as `[`*`schema_name`*`.]`*`object_name`*. For example, `hr.departments`. If the schema is not specified, then the current user is the default.

DELETE_ALL_ERRORS Procedure

This procedure deletes all the error transactions for the specified apply process.

Syntax

```
DBMS_APPLY_ADM.DELETE_ALL_ERRORS(
   apply_name  IN  VARCHAR2 DEFAULT NULL);
```

Parameter

Table 15–6 DELETE_ALL_ERRORS Procedure Parameter

Parameter	Description
apply_name	The name of the apply process that raised the errors while processing the transactions. Do not specify an owner. If NULL, then all error transactions for all apply processes are deleted.

DELETE_ERROR Procedure

This procedure deletes the specified error transaction.

Syntax

```
DBMS_APPLY_ADM.DELETE_ERROR(
   local_transaction_id  IN  VARCHAR2);
```

Parameter

Table 15–7 DELETE_ERROR Procedure Parameter

Parameter	Description
`local_transaction_id`	The identification number of the error transaction to delete. If the specified transaction does not exist in the error queue, then an error is raised.

DROP_APPLY Procedure

This procedure drops an apply process.

Syntax

```
DBMS_APPLY_ADM.DROP_APPLY(
   apply_name             IN  VARCHAR2,
   drop_unused_rule_sets  IN  BOOLEAN  DEFAULT FALSE);
```

Parameters

Table 15–8 DROP_APPLY Procedure Parameters

Parameter	Description
apply_name	The name of the apply process being dropped. You must specify an existing apply process name. Do not specify an owner.
drop_unused_rule_sets	If TRUE, then the procedure drops any rule sets, positive and negative, used by the specified apply process if these rule sets are not used by any other Streams client. Streams clients include capture processes, propagations, apply processes, and messaging clients. If this procedure drops a rule set, then this procedure also drops any rules in the rule set that are not in another rule set. If FALSE, then the procedure does not drop the rule sets used by the specified apply process, and the rule sets retain their rules.

Usage Notes

When you use this procedure to drop an apply process, information about rules created for the apply process using the DBMS_STREAMS_ADM package is removed from the data dictionary views for Streams rules. Information about such a rule is removed even if the rule is not in either the positive or negative rule set for the apply process. The following are the data dictionary views for Streams rules:

- ALL_STREAMS_GLOBAL_RULES
- DBA_STREAMS_GLOBAL_RULES
- ALL_STREAMS_MESSAGE_RULES
- DBA_STREAMS_MESSAGE_RULES
- ALL_STREAMS_SCHEMA_RULES
- DBA_STREAMS_SCHEMA_RULES
- ALL_STREAMS_TABLE_RULES
- DBA_STREAMS_TABLE_RULES

> **See Also:** *Oracle Streams Concepts and Administration* for more information about Streams data dictionary views

DROP_OBJECT_DEPENDENCY

This procedure drops an object dependency. An object dependency is a virtual dependency definition that defines a parent-child relationship between two objects at a destination database.

Note:

- An error is raised if an object dependency does not exist for the specified database objects.
- An error is raised if `NULL` is specified for either of the procedure parameters.

See Also:

- "CREATE_OBJECT_DEPENDENCY Procedure" on page 15-18
- *Oracle Streams Replication Administrator's Guide*

Syntax

```
DBMS_APPLY_ADM.DROP_OBJECT_DEPENDENCY(
   object_name         IN  VARCHAR2,
   parent_object_name  IN  VARCHAR2);
```

Parameters

Table 15–9 CREATE_OBJECT_DEPENDENCY Procedure Parameters

Parameter	Description
`object_name`	The name of the child database object, specified as `[`*`schema_name`*`.]`*`object_name`*. For example, `hr.employees`. If the schema is not specified, then the current user is the default.
`parent_object_name`	The name of the parent database object, specified as `[`*`schema_name`*`.]`*`object_name`*. For example, `hr.departments`. If the schema is not specified, then the current user is the default.

EXECUTE_ALL_ERRORS Procedure

This procedure reexecutes the error transactions in the error queue for the specified apply process.

The transactions are reexecuted in commit SCN order. Error reexecution stops if an error is raised.

> **See Also:** *Oracle Streams Concepts and Administration* for more information about the error queue

Syntax

```
DBMS_APPLY_ADM.EXECUTE_ALL_ERRORS(
   apply_name       IN  VARCHAR2  DEFAULT NULL,
   execute_as_user  IN  BOOLEAN   DEFAULT FALSE);
```

Parameters

Table 15–10 EXECUTE_ALL_ERRORS Procedure Parameters

Parameter	Description
apply_name	The name of the apply process that raised the errors while processing the transactions. Do not specify an owner. If NULL, then all error transactions for all apply processes are reexecuted.
execute_as_user	If TRUE, then the procedure reexecutes the transactions in the security context of the current user. If FALSE, then the procedure reexecutes each transaction in the security context of the original receiver of the transaction. The original receiver is the user who was processing the transaction when the error was raised. The DBA_APPLY_ERROR data dictionary view lists the original receiver for each error transaction. The user who executes the transactions must have privileges to perform DML and DDL changes on the apply objects and to run any apply handlers. This user must also have dequeue privileges on the queue used by the apply process.

EXECUTE_ERROR Procedure

This procedure reexecutes the specified error transaction in the error queue.

> **See Also:** *Oracle Streams Concepts and Administration* for more information about the error queue

Syntax

```
DBMS_APPLY_ADM.EXECUTE_ERROR(
   local_transaction_id  IN  VARCHAR2,
   execute_as_user       IN  BOOLEAN   DEFAULT FALSE,
   user_procedure        IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 15–11 EXECUTE_ERROR Procedure Parameters

Parameter	Description
`local_transaction_id`	The identification number of the error transaction to execute. If the specified transaction does not exist in the error queue, then an error is raised.
`execute_as_user`	If `TRUE`, then the procedure reexecutes the transaction in the security context of the current user. If `FALSE`, then the procedure reexecutes the transaction in the security context of the original receiver of the transaction. The original receiver is the user who was processing the transaction when the error was raised. The `DBA_APPLY_ERROR` data dictionary view lists the original receiver for each error transaction. The user who executes the transaction must have privileges to perform DML and DDL changes on the apply objects and to run any apply handlers. This user must also have dequeue privileges on the queue used by the apply process.
`user_procedure`	A user-defined procedure that modifies the error transaction so that it can be successfully executed. Specify `NULL` to execute the error transaction without running a user procedure. **See Also:** "Usage Notes" on page 15-24 for more information about the user procedure

Usage Notes

You must specify the full procedure name for the `user_procedure` parameter in one of the following forms:

- `[`*`schema_name`*`.]`*`package_name.procedure_name`*
- `[`*`schema_name`*`.]`*`procedure_name`*

If the procedure is in a package, then the *`package_name`* must be specified. The user who invokes the `EXECUTE_ERROR` procedure must have `EXECUTE` privilege on the specified procedure. Also, if the *`schema_name`* is not specified, then the user who invokes the `EXECUTE_ERROR` procedure is the default.

For example, suppose the `procedure_name` has the following properties:

- `strmadmin` is the *`schema_name`*.

- fix_errors is the *package_name*.
- fix_hr_errors is the *procedure_name*.

In this case, specify the following:

```
strmadmin.fix_errors.fix_hr_errors
```

The procedure you create for error handling must have the following signature:

```
PROCEDURE user_procedure (
     in_anydata                    IN      ANYDATA,
     error_record                  IN      DBA_APPLY_ERROR%ROWTYPE,
     error_message_number          IN      NUMBER,
     messaging_default_processing  IN OUT  BOOLEAN,
     out_anydata                   OUT     ANYDATA);
```

The user procedure has the following parameters:

- in_anydata: The ANYDATA encapsulation of a message that the apply process passes to the procedure. A single transaction can include multiple messages. A message can be a row logical change record (row LCR), a DDL logical change record (DDL LCR), or a user message.
- error_record: The row in the DBA_APPLY_ERROR data dictionary view that identifies the transaction
- error_message_number: The message number of the ANYDATA object in the in_anydata parameter, starting at 1
- messaging_default_processing: If TRUE, then the apply process continues processing the message in the in_anydata parameter, which can include executing DML or DDL statements and invoking apply handlers.

  If FALSE, then the apply process skips processing the message in the in_anydata parameter and moves on to the next message in the in_anydata parameter.
- out_anydata: The ANYDATA object processed by the user procedure and used by the apply process if messaging_default_processing is TRUE.

If an LCR is executed using the EXECUTE LCR member procedure in the user procedure, then the LCR is executed directly, and the messaging_default_processing parameter should be set to FALSE. In this case, the LCR is not passed to any apply handlers.

Processing an error transaction with a user procedure results in one of the following outcomes:

- The user procedure modifies the transaction so that it can be executed successfully.
- The user procedure fails to make the necessary modifications, and an error is raised when transaction execution is attempted. In this case, the transaction is rolled back and remains in the error queue.

The following restrictions apply to the user procedure:

- Do not execute COMMIT or ROLLBACK statements. Doing so can endanger the consistency of the transaction.
- Do not modify LONG, LONG RAW or LOB column data in an LCR.
- If the ANYDATA object in the in_anydata parameter is a row LCR, then the out_anydata parameter must be row LCR if the messaging_default_processing parameter is set to TRUE.

- If the ANYDATA object in the in_anydata parameter is a DDL LCR, then the out_anydata parameter must be DDL LCR if the messaging_default_processing parameter is set to TRUE.
- The user who runs the user procedure must have SELECT privilege on the DBA_APPLY_ERROR data dictionary view.

Note: LCRs containing transactional directives, such as COMMIT and ROLLBACK, are not passed to the user procedure.

GET_ERROR_MESSAGE Function

This function returns the message payload from the error queue for the specified message number and transaction identifier. The message can be a logical change record (LCR) or a non-LCR message.

This function is overloaded. One version of this function contains two `OUT` parameters. These `OUT` parameters contain the destination queue into which the message should be enqueued, if one exists, and whether or not the message should be executed. The destination queue is specified using the `SET_ENQUEUE_DESTINATION` procedure, and the execution directive is specified using the `SET_EXECUTE` procedure.

See Also:

- "SET_ENQUEUE_DESTINATION Procedure" on page 15-33
- "SET_EXECUTE Procedure" on page 15-35

Syntax

```
DBMS_APPLY_ADM.GET_ERROR_MESSAGE(
   message_number          IN   NUMBER,
   local_transaction_id    IN   VARCHAR2,
   destination_queue_name  OUT  VARCHAR2,
   execute                 OUT  BOOLEAN)
RETURN ANYDATA;

DBMS_APPLY_ADM.GET_ERROR_MESSAGE(
   message_number          IN   NUMBER,
   local_transaction_id    IN   VARCHAR2)
RETURN ANYDATA;
```

Parameters

Table 15–12 GET_ERROR_MESSAGE Function Parameters

Parameter	Description
`message_number`	The identification number of the message. This number identifies the position of the message in the transaction. Query the `DBA_APPLY_ERROR` data dictionary view to view the message number of each apply error.
`local_transaction_id`	Identifier of the error transaction for which to return a message
`destination_queue_name`	Contains the name of the queue into which the message should be enqueued. If the message should not be enqueued into a queue, then this parameter contains `NULL`.
`execute`	Contains `TRUE` if the message should be executed Contains `FALSE` if the message should not be executed

SET_DML_HANDLER Procedure

This procedure sets a user procedure as a DML handler for a specified operation on a specified object. The user procedure alters the apply behavior for the specified operation on the specified object.

Syntax

```
DBMS_APPLY_ADM.SET_DML_HANDLER(
   object_name          IN  VARCHAR2,
   object_type          IN  VARCHAR2,
   operation_name       IN  VARCHAR2,
   error_handler        IN  BOOLEAN   DEFAULT FALSE,
   user_procedure       IN  VARCHAR2,
   apply_database_link  IN  VARCHAR2  DEFAULT NULL,
   apply_name           IN  VARCHAR2  DEFAULT NULL,
   assemble_lobs        IN  BOOLEAN   DEFAULT FALSE);
```

Parameters

Table 15–13 SET_DML_HANDLER Procedure Parameters

Parameter	Description
object_name	The name of the source object specified as [*schema_name.*]*object_name*. For example, hr.employees. If the schema is not specified, then the current user is the default. The specified object does not need to exist when you run this procedure.
object_type	The type of the source object. Currently, TABLE is the only possible source object type.
operation_name	The name of the operation, which can be specified as: ■ INSERT ■ UPDATE ■ DELETE ■ LOB_UPDATE For example, suppose you run this procedure twice for the hr.employees table. In one call, you set operation_name to UPDATE and user_procedure to employees_update. In another call, you set operation_name to INSERT and user_procedure to employees_insert. Both times, you set error_handler to FALSE. In this case, the employees_update procedure is run for UPDATE operations on the hr.employees table, and the employees_insert procedure is run for INSERT operations on the hr.employees table.
error_handler	If TRUE, then the specified user procedure is run when a row logical change record (row LCR) involving the specified operation on the specified object raises an apply process error. You can code the user procedure to resolve possible error conditions, notify administrators of the error, log the error, or any combination of these actions. If FALSE, then the handler being set is run for all row LCRs involving the specified operation on the specified object.

Table 15–13 (Cont.) SET_DML_HANDLER Procedure Parameters

Parameter	Description
user_procedure	A user-defined procedure that is invoked during apply for the specified operation on the specified object. If the procedure is a DML handler, then it is invoked instead of the default apply performed by Oracle. If the procedure is an error handler, then it is invoked when the apply process encounters an error. Specify NULL to unset a DML handler that is set for the specified operation on the specified object.
apply_database_link	The name of the database link to a non-Oracle database. This parameter should be set only when the destination database is a non-Oracle database.
apply_name	The name of the apply process that uses the DML handler or error handler. If NULL, then the procedure sets the DML handler or error handler as a general handler for all apply processes in the database. If the user_procedure parameter is set to NULL to unset a handler, and the handler being unset is set for a specific apply process, then use the apply_name parameter to specify the apply process to unset the handler.
assemble_lobs	If TRUE, then LOB assembly is used for LOB columns in LCRs processed by the handler. LOB assembly combines multiple LCRs for a LOB column resulting from a single row change into one row LCR before passing the LCR to the handler. Database compatibility must be 10.2.0 or higher to use LOB assembly. If FALSE, then LOB assembly is not used for LOB columns in LCRs processed by the handler.

Usage Notes

Run this procedure at the destination database. The SET_DML_HANDLER procedure provides a way for users to apply logical change records containing DML changes (row LCRs) using a customized apply.

If the error_handler parameter is set to TRUE, then it specifies that the user procedure is an error handler. An error handler is invoked only when a row LCR raises an apply process error. Such an error can result from a data conflict if no conflict handler is specified or if the update conflict handler cannot resolve the conflict. If the error_handler parameter is set to FALSE, then the user procedure is a DML handler, not an error handler, and a DML handler is always run instead of performing the specified operation on the specified object.

This procedure either sets a DML handler or an error handler for a particular operation on an object. It cannot set both a DML handler and an error handler for the same object and operation.

If the apply_name parameter is non-NULL, then the DML handler or error handler is set for the specified apply process. In this case, this handler is not invoked for other apply processes at the local destination database. If the apply_name parameter is NULL, the default, then the handler is set as a general handler for all apply processes at the destination database. When a handler is set for a specific apply process, then this handler takes precedence over any general handlers. For example, consider the following scenario:

- A DML handler named handler_hr is specified for an apply process named apply_hr for UPDATE operations on the hr.employees table.
- A general DML handler named handler_gen also exists for UPDATE operations on the hr.employees table.

In this case, the apply_hr apply process uses the handler_hr DML handler for UPDATE operations on the hr.employees table.

At the source database, you must specify an unconditional supplemental log group for the columns needed by a DML or error handler.

Attention: Do not modify LONG, LONG RAW, or nonassembled LOB column data in an LCR with DML handlers, error handlers, or custom rule-based transformation functions. DML handlers and error handlers can modify LOB columns in row LCRs that have been constructed by LOB assembly.

Note: Currently, setting an error handler for an apply process that is applying changes to a non-Oracle database is not supported.

The SET_DML_HANDLER procedure can be used to set either a DML handler or an error handler for row LCRs that perform a specified operation on a specified object. The signatures of a DML handler procedure and of an error handler procedure are described following this section.

In either case, you must specify the full procedure name for the user_procedure parameter in one of the following forms:

- [*schema_name*.]*package_name*.*procedure_name*
- [*schema_name*.]*procedure_name*

If the procedure is in a package, then the *package_name* must be specified. The user who invokes the SET_DML_HANDLER procedure must have EXECUTE privilege on the specified procedure. Also, if the *schema_name* is not specified, then the user who invokes the SET_DML_HANDLER procedure is the default.

For example, suppose the procedure_name has the following properties:

- hr is the *schema_name*.
- apply_pkg is the *package_name*.
- employees_default is the *procedure_name*.

In this case, specify the following:

```
hr.apply_pkg.employees_default
```

The following restrictions apply to the user procedure:

- Do not execute COMMIT or ROLLBACK statements. Doing so can endanger the consistency of the transaction that contains the LCR.
- If you are manipulating a row using the EXECUTE member procedure for the row LCR, then do not attempt to manipulate more than one row in a row operation. You must construct and execute manually any DML statements that manipulate more than one row.

- If the command type is UPDATE or DELETE, then row operations resubmitted using the EXECUTE member procedure for the LCR must include the entire key in the list of old values. The key is the primary key or the smallest unique index that has at least one NOT NULL column, unless a substitute key has been specified by the SET_KEY_COLUMNS procedure. If there is no specified key, then the key consists of all non LOB, non LONG, and non LONG RAW columns.
- If the command type is INSERT, then row operations resubmitted using the EXECUTE member procedure for the LCR should include the entire key in the list of new values. Otherwise, duplicate rows are possible. The key is the primary key or the smallest unique index that has at least one NOT NULL column, unless a substitute key has been specified by the SET_KEY_COLUMNS procedure. If there is no specified key, then the key consists of all non LOB, non LONG, and non LONG RAW columns.

See Also: *Oracle Streams Replication Administrator's Guide* for information about and restrictions regarding DML handlers and LOB, LONG, and LONG RAW datatypes

Signature of a DML Handler Procedure

The procedure specified in the user_procedure parameter must have the following signature:

```
PROCEDURE user_procedure (
   parameter_name  IN  ANYDATA);
```

Here, *user_procedure* stands for the name of the procedure and *parameter_name* stands for the name of the parameter passed to the procedure. The parameter passed to the procedure is a ANYDATA encapsulation of a row LCR.

See Also: Chapter 188, "Logical Change Record TYPEs" for more information about LCRs

Signature of an Error Handler Procedure

The procedure you create for error handling must have the following signature:

```
PROCEDURE user_procedure (
     message            IN  ANYDATA,
     error_stack_depth  IN  NUMBER,
     error_numbers      IN  DBMS_UTILITY.NUMBER_ARRAY,
     error_messages     IN  emsg_array);
```

If you want to retry the DML operation within the error handler, then have the error handler procedure run the EXECUTE member procedure for the LCR. The last error raised is on top of the error stack. To specify the error message at the top of the error stack, use error_numbers(1) and error_messages(1).

Note:

- Each parameter is required and must have the specified datatype. However, you can change the names of the parameters.
- The emsg_array value must be a user-defined array that is a table of type VARCHAR2 with at least 76 characters.

Running an error handler results in one of the following outcomes:

- The error handler successfully resolves the error and returns control to the apply process.
- The error handler fails to resolve the error, and the error is raised. The raised error causes the transaction to be rolled back and placed in the error queue.

SET_ENQUEUE_DESTINATION Procedure

This procedure sets the queue where the apply process automatically enqueues a message that satisfies the specified rule.

This procedure modifies the specified rule's action context to specify the queue. A rule action context is optional information associated with a rule that is interpreted by the client of the rules engine after the rule evaluates to TRUE for a message. In this case, the client of the rules engine is a Streams apply process. The information in an action context is an object of type SYS.RE$NV_LIST, which consists of a list of name-value pairs.

A queue destination specified by this procedure always consists of the following name-value pair in an action context:

- The name is APPLY$_ENQUEUE.
- The value is a ANYDATA instance containing the queue name specified as a VARCHAR2.

Syntax

```
DBMS_APPLY_ADM.SET_ENQUEUE_DESTINATION(
  rule_name               IN  VARCHAR2,
  destination_queue_name  IN  VARCHAR2);
```

Parameters

Table 15–14 SET_ENQUEUE_DESTINATION Procedure Parameters

Parameter	Description
rule_name	The name of the rule, specified as [*schema_name.*]*rule_name*. For example, to specify a rule named hr5 in the hr schema, enter hr.hr5 for this parameter. If the schema is not specified, then the current user is the default.
destination_queue_name	The name of the queue into which the apply process should enqueue the message. Specify the queue in the form [*schema_name.*]*queue_name*. Only local queues can be specified. For example, to specify a queue in the hr schema named streams_queue, enter hr.streams_queue. If the schema is not specified, then the current user is the default. If NULL, then an existing name-value pair with the name APPLY$_ENQUEUE is removed. If no name-value pair exists with the name APPLY$_ENQUEUE for the rule, then no action is taken. If non-NULL and a name-value pair already exists for the rule with the name APPLY$_ENQUEUE, then it is removed, and a new name-value pair with the value specified by this parameter is added.

Usage Notes

If an apply handler, such as a DML handler, DDL handler, or message handler, processes a message that also is enqueued into a destination queue, then the apply handler processes the message before it is enqueued.

The following are considerations for using this procedure:

- This procedure does not verify that the specified queue exists. If the queue does not exist, then an error is raised when an apply process tries to enqueue a message into it.
- Streams capture processes, propagations, and messaging clients ignore the action context created by this procedure.
- The apply user of the apply processes using the specified rule must have the necessary privileges to enqueue messages into the specified queue. If the queue is a secure queue, then the apply user must be a secure queue user of the queue.
- The specified rule must be in the positive rule set for an apply process. If the rule is in the negative rule set for an apply process, then the apply process does not enqueue the message into the destination queue.
- If the commit SCN for a message is less than or equal to the relevant instantiation SCN for the message, then the message is not enqueued into the destination queue, even if the message satisfies the apply process rule sets.

SET_EXECUTE Procedure

This procedure specifies whether a message that satisfies the specified rule is executed by an apply process.

This procedure modifies the specified rule's action context to specify message execution. A rule action context is optional information associated with a rule that is interpreted by the client of the rules engine after the rule evaluates to TRUE for a message. In this case, the client of the rules engine is a Streams apply process. The information in an action context is an object of type SYS.RE$NV_LIST, which consists of a list of name-value pairs.

A message execution directive specified by this procedure always consists of the following name-value pair in an action context:

- The name is APPLY$_EXECUTE.
- The value is a ANYDATA instance that contains NO as a VARCHAR2. When the value is NO, then an apply process does not execute the message and does not send the message to any apply handler.

Syntax

```
DBMS_APPLY_ADM.SET_EXECUTE(
  rule_name  IN  VARCHAR2,
  execute    IN  BOOLEAN);
```

Parameters

Table 15–15 SET_EXECUTE Procedure Parameters

Parameter	Description
rule_name	The name of the rule, specified as [*schema_name*.]*rule_name*. For example, to specify a rule named hr5 in the hr schema, enter hr.hr5 for this parameter. If the schema is not specified, then the current user is the default.
execute	If TRUE, then the procedure removes the name-value pair with the name APPLY$_EXECUTE for the specified rule. Removing the name-value pair means that the apply process executes messages that satisfy the rule. If no name-value pair with name APPLY$_EXECUTE exists for the rule, then no action is taken. If FALSE, then the procedure adds a name-value pair to the rule's action context. The name is APPLY$_EXECUTE and the value is NO. An apply process does not execute a message that satisfies the rule and does not send the message to any apply handler. If a name-value pair already exists for the rule with the name APPLY$_EXECUTE, then it is removed, and a new one with the value NO is added. If NULL, then the procedure raises an error.

Usage Notes

If the message is a logical change record (LCR) and the message is not executed, then the change encapsulated in the LCR is not made to the relevant local database object. Also, if the message is not executed, then it is not sent to any apply handler.

Note:

- Streams capture processes, propagations, and messaging clients ignore the action context created by this procedure.
- The specified rule must be in the positive rule set for an apply process for the apply process to follow the execution directive. If the rule is in the negative rule set for an apply process, then the apply process ignores the execution directive for the rule.

SET_GLOBAL_INSTANTIATION_SCN Procedure

This procedure records the specified instantiation SCN for the specified source database and, optionally, for the schemas at the source database and the tables owned by these schemas. This procedure overwrites any existing instantiation SCN for the database, and, if it sets the instantiation SCN for a schema or a table, then it overwrites any existing instantiation SCN for the schema or table.

This procedure gives you precise control over which DDL logical change records (DDL LCRs) from a source database are ignored and which DDL LCRs are applied by an apply process.

Syntax

```
DBMS_APPLY_ADM.SET_GLOBAL_INSTANTIATION_SCN(
  source_database_name  IN  VARCHAR2,
  instantiation_scn     IN  NUMBER,
  apply_database_link   IN  VARCHAR2  DEFAULT NULL,
  recursive             IN  BOOLEAN   DEFAULT FALSE);
```

Parameters

Table 15–16 SET_GLOBAL_INSTANTIATION_SCN Procedure Parameters

Parameter	Description
source_database_name	The global name of the source database. For example, DBS1.NET. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically.
instantiation_scn	The instantiation SCN. Specify NULL to remove the instantiation SCN metadata for the source database from the data dictionary.
apply_database_link	The name of the database link to a non-Oracle database. This parameter should be set only when the destination database of a local apply process is a non-Oracle database.
recursive	If TRUE, then the procedure sets the instantiation SCN for the source database, all schemas in the source database, and all tables owned by the schemas in the source database. This procedure selects the schemas and tables from the ALL_USERS and ALL_TABLES data dictionary views, respectively, at the source database under the security context of the current user. If FALSE, then the procedure sets the global instantiation SCN for the source database, but does not set the instantiation SCN for any schemas or tables **Note:** If recursive is set to TRUE, then a database link from the destination database to the source database is required. This database link must have the same name as the global name of the source database and must be accessible to the current user. Also, a table must be accessible to the current user in either the ALL_TABLES or DBA_TABLES data dictionary view at the source database for this procedure to set the instantiation SCN for the table at the destination database.

Usage Notes

If the commit SCN of a DDL LCR for a database object from a source database is less than or equal to the instantiation SCN for that source database at a destination database, then the apply process at the destination database disregards the DDL LCR. Otherwise, the apply process applies the DDL LCR.

The global instantiation SCN specified by this procedure is used for a DDL LCR only if the DDL LCR does not have `object_owner`, `base_table_owner`, `and base_table_name` specified. For example, the global instantiation SCN set by this procedure is used for DDL LCRs with a `command_type` of `CREATE USER`.

If the `recursive` parameter is set to `TRUE`, then this procedure sets the instantiation SCN for each schema at a source database and for the tables owned by these schemas. This procedure uses the `SET_SCHEMA_INSTANTIATION_SCN` procedure to set the instantiation SCN for each schema, and it uses the `SET_TABLE_INSTANTIATION_SCN` procedure to set the instantiation SCN for each table. Each schema instantiation SCN is used for DDL LCRs on the schema, and each table instantiation SCN is used for DDL LCRs and row LCRs on the table.

If the `recursive` parameter is set to `FALSE`, then this procedure does not set the instantiation SCN for any schemas or tables.

Note:

- Any instantiation SCN specified by this procedure is used only for LCRs captured by a capture process. It is not used for user-created LCRs.
- The instantiation SCN is not set for the `SYS` or `SYSTEM` schemas.

See Also:

- "SET_SCHEMA_INSTANTIATION_SCN Procedure" on page 15-46
- "SET_TABLE_INSTANTIATION_SCN Procedure" on page 15-48
- "LCR$_DDL_RECORD Type" on page 188-3 for more information about DDL LCRs
- *Oracle Streams Replication Administrator's Guide*

SET_KEY_COLUMNS Procedures

This procedure records the set of columns to be used as the substitute primary key for apply purposes and removes existing substitute primary key columns for the specified object if they exist.

This procedure is overloaded. The `column_list` and `column_table` parameters are mutually exclusive.

Syntax

```
DBMS_APPLY_ADM.SET_KEY_COLUMNS(
   object_name          IN  VARCHAR2,
   column_list          IN  VARCHAR2,
   apply_database_link  IN  VARCHAR2  DEFAULT NULL);

DBMS_APPLY_ADM.SET_KEY_COLUMNS(
   object_name          IN  VARCHAR2,
   column_table         IN  DBMS_UTILITY.NAME_ARRAY,
   apply_database_link  IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 15–17 SET_KEY_COLUMNS Procedure Parameters

Parameter	Description
`object_name`	The name of the table specified as `[`*`schema_name`*`.]`*`object_name`*. For example, `hr.employees`. If the schema is not specified, then the current user is the default. If the apply process is applying changes to a non-Oracle database in a heterogeneous environment, then the object name is not verified.
`column_list`	A comma-delimited list of the columns in the table that you want to use as the substitute primary key, with no spaces between the column names. If the `column_list` parameter is empty or `NULL`, then the current set of key columns is removed.
`column_table`	A PL/SQL index-by table of type `DBMS_UTILITY.NAME_ARRAY` of the columns in the table that you want to use as the substitute primary key. The index for `column_table` must be 1-based, increasing, dense, and terminated by a `NULL`. If the `column_table` parameter is empty or `NULL`, then the current set of key columns is removed.
`apply_database_link`	The name of the database link to a non-Oracle database. This parameter should be set only when the destination database is a non-Oracle database.

Usage Notes

When not empty, this set of columns takes precedence over any primary key for the specified object. Do not specify substitute key columns if the object already has primary key columns and you want to use those primary key columns as the key.

Run this procedure at the destination database. At the source database, you must specify an unconditional supplemental log group for the substitute key columns.

Note:

- Unlike true primary keys, columns specified as substitute key column columns can contain NULLs. However, Oracle recommends that each column you specify as a substitute key column be a NOT NULL column. You also should create a single index that includes all of the columns in a substitute key. Following these guidelines improves performance for updates, deletes, and piecewise updates to LOBs because Oracle can locate the relevant row more efficiently.
- You should not permit applications to update the primary key or substitute key columns of a table. This ensures that Oracle can identify rows and preserve the integrity of the data.
- If there is neither a primary key, nor a unique index that has at least one NOT NULL column, nor a substitute key for a table, then the key consists of all non LOB, non LONG, and non LONG RAW columns.

SET_PARAMETER Procedure

This procedure sets an apply parameter to the specified value.

Syntax

```
DBMS_APPLY_ADM.SET_PARAMETER (
   apply_name  IN  VARCHAR2,
   parameter   IN  VARCHAR2,
   value       IN  VARCHAR2);
```

Parameters

Table 15–18 SET_PARAMETER Procedure Parameters

Parameter	Description
apply_name	The apply process name. Do not specify an owner.
parameter	The name of the parameter you are setting. See "Apply Process Parameters" on page 15-41 for a list of these parameters.
value	The value to which the parameter is set

Apply Process Parameters

The following table lists the parameters for the apply process.

Table 15–19 Apply Process Parameters

Parameter Name	Possible Values	Default	Description
allow_duplicate_rows	Y or N	N	If Y and more than one row is changed by a single row logical change record (row LCR) with an UPDATE or DELETE command type, then the apply process only updates or deletes one of the rows. If N, then the apply process raises an error when it encounters a single row LCR with an UPDATE or DELETE command type that changes more than one row in a table. **Note:** Regardless of the setting for this parameter, apply processes do not allow changes to duplicate rows for tables with LOB, LONG, or LONG RAW columns. **See Also:** "Duplicate Rows and Substitute Primary Key Columns" on page 15-44
commit_serialization	full or none	full	The order in which applied transactions are committed. If full, then the apply process commits applied transactions in the order in which they were committed at the source database. If none, then the apply process can commit transactions in any order. Performance is best if you specify none. Regardless of the specification, applied transactions can execute in parallel subject to data dependencies and constraint dependencies. Logical standby environments typically specify full.
disable_on_error	Y or N	Y	If Y, then the apply process is disabled on the first unresolved error, even if the error is not fatal. If N, then the apply process continues regardless of unresolved errors.

Table 15–19 (Cont.) Apply Process Parameters

Parameter Name	Possible Values	Default	Description
disable_on_limit	Y or N	N	If Y, then the apply process is disabled if the apply process terminates because it reached a value specified by the time_limit parameter or transaction_limit parameter. If N, then the apply process is restarted immediately after stopping because it reached a limit.
maximum_scn	A valid SCN or infinite	infinite	The apply process is disabled before applying a transaction with a commit SCN greater than or equal to the value specified. If infinite, then the apply process runs regardless of the SCN value.
parallelism	A positive integer	1	The number of transactions that can be concurrently applied. Setting the parallelism parameter to a number higher than the number of available parallel execution servers can disable the apply process. Make sure the PROCESSES and PARALLEL_MAX_SERVERS initialization parameters are set appropriately when you set the parallelism apply process parameter. **Note:** When the value of this parameter is changed for a running apply process, the apply process is stopped and restarted automatically. This can take some time depending on the size of the transactions currently being applied.
startup_seconds	0, a positive integer, or infinite	0	The maximum number of seconds to wait for another instantiation of the same apply process to finish. If the other instantiation of the same apply process does not finish within this time, then the apply process does not start. If infinite, then an apply process does not start until another instantiation of the same apply process finishes.
time_limit	A positive integer or infinite	infinite	The apply process stops as soon as possible after the specified number of seconds since it started. If infinite, then the apply process continues to run until it is stopped explicitly.
trace_level	0 or a positive integer	0	Set this parameter only under the guidance of Oracle Support Services.
transaction_limit	A positive integer or infinite	infinite	The apply process stops after applying the specified number of transactions. If infinite, then the apply process continues to run regardless of the number of transactions applied.

Table 15–19 (Cont.) Apply Process Parameters

Parameter Name	Possible Values	Default	Description
txn_lcr_spill_threshold	A positive integer or infinite	10000	The apply process begins to spill messages from memory to hard disk for a particular transaction when the number of messages in memory for the transaction exceeds the specified number. The number of messages in first chunk of messages spilled from memory equals the number specified for this parameter, and the number of messages spilled in future chunks is either 100 or the number specified for this parameter, whichever is less. If the reader server of an apply process has the specified number of messages in memory for a particular transaction, then when it detects the next message for this transaction, it spills the messages that are in memory to the hard disk. For example, if this parameter is set to 10000, and a transaction has 10,200 messages, then the reader server handles the transaction in the following way: 1. Reads the first 10,000 messages in the transaction into memory 2. Spills messages 1 - 10,000 to hard disk when it detects message 10,000 3. Reads the next 100 messages in the transaction into memory 4. Spills messages 10,001 - 10,100 to hard disk when it detects message 10,100 5. Reads the next 100 messages in the transaction into memory The apply process applies the first 10,100 messages from the hard disk and the last 100 messages from memory. When the reader server spills messages from memory, the messages are stored in a database table on the hard disk. These messages are not spilled from memory to a queue table. Message spilling occurs at the transaction level. For example, if this parameter is set to 10000, and the reader server of an apply process is assembling two transactions, one with 7,500 messages and another with 8,000 messages, then it does not spill any messages. If infinite, then the apply process does not spill messages to the hard disk. Query the DBA_APPLY_SPILL_TXN data dictionary view for information about transactions spilled by an apply process. **Note:** When the value of this parameter is changed for a running apply process, the new setting does not take effect until the apply process is restarted.
write_alert_log	Y or N	Y	If Y, then the apply process writes a message to the alert log on exit. If N, then the apply process does not write a message to the alert log on exit. The message specifies the reason why the apply process stopped.

Usage Notes

When you alter a parameter value, a short amount of time might pass before the new value for the parameter takes effect.

Note:

- For all parameters that are interpreted as positive integers, the maximum possible value is `4,294,967,295`. Where applicable, specify `infinite` for larger values.
- For parameters that require an SCN setting, any valid SCN value can be specified.

Duplicate Rows and Substitute Primary Key Columns

A table has duplicate rows when the all of the column values are identical for two or more rows in the table, excluding LOB, `LONG`, and `LONG RAW` columns. You can specify substitute primary key columns for a table at a destination database using by the `SET_KEY_COLUMNS` procedure. When substitute primary key columns are specified for a table with duplicate rows at a destination database, and the `allow_duplicate_rows` apply process parameter is set to `Y`, meet the following requirements to keep the table data synchronized at the source and destination databases:

- Ensure that updates at the source database always update at least one of the columns specified as a substitute key column at the destination database.
- Ensure that the substitute key columns uniquely identify each row in the table at the destination database.

The rest of this section provides more details about these requirements.

If a table does not have a primary key, a unique index that has at least one `NOT NULL` column, or a substitute key, then the key consists of all non LOB, non `LONG`, and non `LONG RAW` columns. When there is no key for a table and the `allow_duplicate_rows` apply process parameter is set to `Y`, a single row LCR with an `UPDATE` or `DELETE` command type only is applied to one of the duplicate rows. In this case, if the table at the source database and the table at the destination database have corresponding duplicate rows, then a change that changes all of the duplicate rows at the source database also changes all the duplicate rows at the destination database when the row LCRs resulting from the change are applied.

For example, suppose a table at a source database has two duplicate rows. An update is performed on the duplicate rows, resulting in two row LCRs. At the destination database, one row LCR is applied to one of the duplicate rows. At this point, the rows are no longer duplicate at the destination database because one of the rows has changed. When the second row LCR is applied at the destination database, the rows are duplicate again. Similarly, if a delete is performed on these duplicate rows at the source database, then both rows are deleted at the destination database when the row LCRs resulting from the change are applied.

When substitute primary key columns are specified for a table, row LCRs are identified with rows in the table during apply using the substitute primary key columns. If substitute primary key columns are specified for a table with duplicate rows at a destination database, and the `allow_duplicate_rows` apply process parameter is set to `Y`, then an update performed on duplicate rows at the source database can result in different changes when the row LCRs are applied at the destination database. Specifically, if the update does not change one of the columns specified as a substitute primary key column, then the same duplicate row can be updated multiple times at the destination database, while other duplicate rows might not be updated.

Also, if the substitute key columns do not identify each row in the table at the destination database uniquely, then a row LCR identified with multiple rows can

update any one of the rows. In this case, the update in the row LCR might not be applied to the correct row in the table at the destination database.

An apply process ignores substitute primary key columns when it determines whether rows in a table are duplicates. An apply process determines that rows are duplicates only if all of the column values in the rows are identical (excluding LOB, `LONG`, and `LONG RAW` columns). Therefore, an apply process always raises an error if a single update or delete changes two or more nonduplicate rows in a table.

For example, consider a table with columns `c1`, `c2`, and `c3` on which the `SET_KEY_COLUMNS` procedure is used to designate column `c1` as the substitute primary key. If two rows have the same key value for the `c1` column, but different value for the `c2` or `c3` columns, then an apply process does not treat the rows as duplicates. If an update or delete modifies more than one row because the `c1` values in the rows are the same, then the apply process raises an error regardless of the setting for the `allow_duplicate_rows` apply process parameter.

See Also: "SET_KEY_COLUMNS Procedures" on page 15-39

SET_SCHEMA_INSTANTIATION_SCN Procedure

This procedure records the specified instantiation SCN for the specified schema in the specified source database and, optionally, for the tables owned by the schema at the source database. This procedure overwrites any existing instantiation SCN for the schema, and, if it sets the instantiation SCN for a table, it overwrites any existing instantiation SCN for the table.

This procedure gives you precise control over which DDL logical change records (LCRs) for a schema are ignored and which DDL LCRs are applied by an apply process.

Syntax

```
DBMS_APPLY_ADM.SET_SCHEMA_INSTANTIATION_SCN(
  source_schema_name    IN  VARCHAR2,
  source_database_name  IN  VARCHAR2,
  instantiation_scn     IN  NUMBER,
  apply_database_link   IN  VARCHAR2  DEFAULT NULL,
  recursive             IN  BOOLEAN   DEFAULT FALSE);
```

Parameters

Table 15–20 SET_SCHEMA_INSTANTIATION_SCN Procedure Parameters

Parameter	Description
source_schema_name	The name of the source schema. For example, hr.
source_database_name	The global name of the source database. For example, DBS1.NET.
	If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically.
instantiation_scn	The instantiation SCN. Specify NULL to remove the instantiation SCN metadata for the source schema from the data dictionary.
apply_database_link	The name of the database link to a non-Oracle database. This parameter should be set only when the destination database of a local apply process is a non-Oracle database.
recursive	If TRUE, then the procedure sets the instantiation SCN for the specified schema and all tables owned by the schema in the source database. This procedure selects the tables owned by the specified schema from the ALL_TABLES data dictionary view at the source database under the security context of the current user.
	If FALSE, then the procedure sets the instantiation SCN for specified schema, but does not set the instantiation SCN for any tables
	Note: If recursive is set to TRUE, then a database link from the destination database to the source database is required. This database link must have the same name as the global name of the source database and must be accessible to the current user. Also, a table must be accessible to the current user in either the ALL_TABLES or DBA_TABLES data dictionary view at the source database for this procedure to set the instantiation SCN for the table at the destination database.

Usage Notes

If the commit SCN of a DDL LCR for a database object in a schema from a source database is less than or equal to the instantiation SCN for that database object at a destination database, then the apply process at the destination database disregards the DDL LCR. Otherwise, the apply process applies the DDL LCR.

The schema instantiation SCN specified by this procedure is used on the following types of DDL LCRs:

- DDL LCRs with a `command_type` of `CREATE TABLE`
- DDL LCRs with a non-`NULL` `object_owner` specified and neither `base_table_owner` nor `base_table_name` specified.

For example, the schema instantiation SCN set by this procedure is used for a DDL LCR with a `command_type` of `CREATE TABLE` and `ALTER USER`.

The schema instantiation SCN specified by this procedure is not used for DDL LCRs with a `command_type` of `CREATE USER`. A global instantiation SCN is needed for such DDL LCRs.

If the `recursive` parameter is set to `TRUE`, then this procedure sets the table instantiation SCN for each table at the source database owned by the schema. This procedure uses the `SET_TABLE_INSTANTIATION_SCN` procedure to set the instantiation SCN for each table. Each table instantiation SCN is used for DDL LCRs and row LCRs on the table.

If the `recursive` parameter is set to `FALSE`, then this procedure does not set the instantiation SCN for any tables.

Note: Any instantiation SCN specified by this procedure is used only for LCRs captured by a capture process. It is not used for user-created LCRs.

See Also:

- "SET_GLOBAL_INSTANTIATION_SCN Procedure" on page 15-37
- "SET_TABLE_INSTANTIATION_SCN Procedure" on page 15-48
- "LCR$_DDL_RECORD Type" on page 188-3 for more information about DDL LCRs
- *Oracle Streams Replication Administrator's Guide*

SET_TABLE_INSTANTIATION_SCN Procedure

This procedure records the specified instantiation SCN for the specified table in the specified source database. This procedure overwrites any existing instantiation SCN for the particular table.

This procedure gives you precise control over which logical change records (LCRs) for a table are ignored and which LCRs are applied by an apply process.

Syntax

```
DBMS_APPLY_ADM.SET_TABLE_INSTANTIATION_SCN(
  source_object_name    IN  VARCHAR2,
  source_database_name  IN  VARCHAR2,
  instantiation_scn     IN  NUMBER,
  apply_database_link   IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 15–21 SET_TABLE_INSTANTIATION_SCN Procedure Parameters

Parameter	Description
source_object_name	The name of the source object specified as [*schema_name.*]*object_name*. For example, hr.employees. If the schema is not specified, then the current user is the default.
source_database_name	The global name of the source database. For example, DBS1.NET. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically.
instantiation_scn	The instantiation SCN. Specify NULL to remove the instantiation SCN metadata for the source table from the data dictionary.
apply_database_link	The name of the database link to a non-Oracle database. This parameter should be set only when the destination database of a local apply process is a non-Oracle database.

Usage Notes

If the commit SCN of an LCR for a table from a source database is less than or equal to the instantiation SCN for that table at some destination database, then the apply process at the destination database disregards the LCR. Otherwise, the apply process applies the LCR.

The table instantiation SCN specified by this procedure is used on the following types of LCRs:

- Row LCRs for the table
- DDL LCRs that have a non-NULL base_table_owner and base_table_name specified, except for DDL LCRs with a command_type of CREATE TABLE

For example, the table instantiation SCN set by this procedure is used for DDL LCRs with a command_type of ALTER TABLE or CREATE TRIGGER.

Note: The instantiation SCN specified by this procedure is used only for LCRs captured by a capture process. It is not used for user-created LCRs.

See Also:

- "SET_GLOBAL_INSTANTIATION_SCN Procedure" on page 15-37
- "SET_SCHEMA_INSTANTIATION_SCN Procedure" on page 15-46
- "LCR$_ROW_RECORD Type" on page 188-11 for more information about row LCRs
- "LCR$_DDL_RECORD Type" on page 188-3 for more information about DDL LCRs
- *Oracle Streams Replication Administrator's Guide*

SET_UPDATE_CONFLICT_HANDLER Procedure

This procedure adds, modifies, or removes a prebuilt update conflict handler for the specified object.

Syntax

```
DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
   object_name          IN  VARCHAR2,
   method_name          IN  VARCHAR2,
   resolution_column    IN  VARCHAR2,
   column_list          IN  DBMS_UTILITY.NAME_ARRAY,
   apply_database_link  IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 15–22 SET_UPDATE_CONFLICT_HANDLER Procedure Parameters

Parameter	Description
object_name	The schema and name of the table, specified as [*schema_name.*]*object_name*, for which an update conflict handler is being added, modified, or removed. For example, if an update conflict handler is being added for table employees owned by user hr, then specify hr.employees. If the schema is not specified, then the current user is the default.
method_name	Type of update conflict handler to create. You can specify one of the prebuilt handlers, which determine whether the column list from the source database is applied for the row or whether the values in the row at the destination database are retained: ■ MAXIMUM: Applies the column list from the source database if it has the greater value for the resolution column. Otherwise, retains the values at the destination database. ■ MINIMUM: Applies the column list from the source database if it has the lesser value for the resolution column. Otherwise, retains the values at the destination database. ■ OVERWRITE: Applies the column list from the source database, overwriting the column values at the destination database. ■ DISCARD: Retains the column list from the destination database, discarding the column list from the source database. If NULL, then the procedure removes any existing update conflict handler with the same object_name, resolution_column, and column_list. If non-NULL, then the procedure replaces any existing update conflict handler with the same object_name and resolution_column.

Table 15–22 (Cont.) SET_UPDATE_CONFLICT_HANDLER Procedure Parameters

Parameter	Description
resolution_column	Name of the column used to uniquely identify an update conflict handler. For the MAXIMUM and MINIMUM prebuilt methods, the resolution column is also used to resolve the conflict. The resolution column must be one of the columns listed in the column_list parameter. NULL is not allowed for this parameter. For the OVERWRITE and DISCARD prebuilt methods, you can specify any column in the column list.
column_list	List of columns for which the conflict handler is called. If a conflict occurs for one or more of the columns in the list when an apply process tries to apply a row logical change record (row LCR), then the conflict handler is called to resolve the conflict. The conflict handler is not called if a conflict occurs only for columns that are not in the list. **Note:** Prebuilt update conflict handlers do not support LOB, LONG, LONG RAW, and user-defined type columns. Therefore, you should not include these types of columns in the column_list parameter.
apply_database_link	The name of the database link to a non-Oracle database. This parameter should be set only when the destination database is a non-Oracle database. **Note:** Currently, conflict handlers are not supported when applying changes to a non-Oracle database.

Usage Notes

If you want to modify an existing update conflict handler, then you specify the table and resolution column of an the existing update conflict handler. You can modify the prebuilt method or the column list.

If you want to remove an existing update conflict handler, then specify NULL for the prebuilt method and specify the table, column list, and resolution column of the existing update conflict handler.

If an update conflict occurs, then Oracle completes the following series of actions:

1. Calls the appropriate update conflict handler to resolve the conflict
2. If no update conflict handler is specified or if the update conflict handler cannot resolve the conflict, then calls the appropriate error handler for the apply process, table, and operation to handle the error
3. If no error handler is specified or if the error handler cannot resolve the error, then raises an error and moves the transaction containing the row LCR that caused the error to the error queue

If you cannot use a prebuilt update conflict handler to meet your requirements, then you can create a PL/SQL procedure to use as a custom conflict handler. You use the SET_DML_HANDLER procedure to designate one or more custom conflict handlers for a particular table. In addition, a custom conflict handler can process LOB columns and use LOB assembly.

Note: Currently, setting an update conflict handler for an apply process that is applying to a non-Oracle database is not supported.

See Also:

- "Signature of an Error Handler Procedure" on page 15-31 for information about setting an error handler
- "SET_DML_HANDLER Procedure" on page 15-28
- *Oracle Streams Replication Administrator's Guide* for more information about prebuilt and custom update conflict handlers

Examples

The following is an example for setting an update conflict handler for the `employees` table in the `hr` schema:

```
DECLARE
  cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1) := 'salary';
  cols(2) := 'commission_pct';
  DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name           =>  'hr.employees',
    method_name           =>  'MAXIMUM',
    resolution_column     =>  'salary',
    column_list           =>  cols);
END;
/
```

This example sets a conflict handler that is called if a conflict occurs for the `salary` or `commission_pct` column in the `hr.employees` table. If such a conflict occurs, then the `salary` column is evaluated to resolve the conflict. If a conflict occurs only for a column that is not in the column list, such as the `job_id` column, then this conflict handler is not called.

SET_VALUE_DEPENDENCY Procedure

This procedure sets or removes a value dependency. A value dependency is a virtual dependency definition that defines a relationship between the columns of two or more tables.

An apply process uses the name of a value dependencies to detect dependencies between row logical change records (row LCRs) that contain the columns defined in the value dependency. Value dependencies can define virtual foreign key relationships between tables, but, unlike foreign key relationships, value dependencies can involve more than two database objects.

This procedure is overloaded. The `attribute_list` and `attribute_table` parameters are mutually exclusive.

See Also: *Oracle Streams Replication Administrator's Guide*

Syntax

```
DBMS_APPLY_ADM.SET_VALUE_DEPENDENCY(
   dependency_name IN VARCHAR2,
   object_name     IN VARCHAR2,
   attribute_list  IN VARCHAR2);

DBMS_APPLY_ADM.SET_VALUE_DEPENDENCY(
   dependency_name IN VARCHAR2,
   object_name     IN VARCHAR2,
   attribute_table IN DBMS_UTILITY.NAME_ARRAY);
```

Parameters

Table 15–23 SET_VALUE_DEPENDENCY Procedure Parameters

Parameter	Description
`dependency_name`	The name of the value dependency. If a dependency with the specified name does not exist, then it is created. If a dependency with the specified name exists, then the specified object and attributes are added to the dependency. If `NULL`, an error is raised.
`object_name`	The name of the table, specified as `[`*`schema_name`*`.]`*`table_name`*. For example, `hr.employees`. If the schema is not specified, then the current user is the default. If `NULL` and the specified dependency exists, then the dependency is removed. If `NULL` and the specified dependency does not exist, then an error is raised. If `NULL`, then `attribute_list` and `attribute_table` also must be `NULL`.
`attribute_list`	A comma-delimited list of column names in the table. There must be no spaces between entries.
`attribute_table`	A PL/SQL index-by table of type `DBMS_UTILITY.NAME_ARRAY` that contains names of columns in the table. The first column name should be at position 1, the second at position 2, and so on. The table does not need to be `NULL` terminated.

START_APPLY Procedure

This procedure directs the apply process to start applying messages.

Syntax

```
DBMS_APPLY_ADM.START_APPLY(
   apply_name  IN  VARCHAR2);
```

Parameter

Table 15–24 START_APPLY Procedure Parameter

Parameter	Description
apply_name	The apply process name. A NULL setting is not allowed. Do not specify an owner.

Usage Notes

The apply process status is persistently recorded. Hence, if the status is ENABLED, then the apply process is started upon database instance startup. An apply process (a*nnn*) is an Oracle background process. The enqueue and dequeue state of DBMS_AQADM.START_QUEUE and DBMS_AQADM.STOP_QUEUE have no effect on the start status of an apply process.

STOP_APPLY Procedure

This procedure stops the apply process from applying messages and rolls back any unfinished transactions being applied.

Syntax

```
DBMS_APPLY_ADM.STOP_APPLY(
   apply_name  IN  VARCHAR2,
   force       IN  BOOLEAN   DEFAULT FALSE);
```

Parameters

Table 15–25 STOP_APPLY Procedure Parameters

Parameter	Description
apply_name	The apply process name. A NULL setting is not allowed. Do not specify an owner.
force	If TRUE, then the procedure stops the apply process as soon as possible. If FALSE, then the procedure stops the apply process after ensuring that there are no gaps in the set of applied transactions. The behavior of the apply process depends on the setting specified for the force parameter and the setting specified for the commit_serialization apply process parameter. See "Usage Notes" for more information.

Usage Notes

The apply process status is persistently recorded. Hence, if the status is DISABLED or ABORTED, then the apply process is not started upon database instance startup.

The enqueue and dequeue state of DBMS_AQADM.START_QUEUE and DBMS_AQADM.STOP_QUEUE have no effect on the STOP status of an apply process.

The following table describes apply process behavior for each setting of the force parameter in the STOP_APPLY procedure and the commit_serialization apply process parameter. In all cases, the apply process rolls back any unfinished transactions when it stops.

force	commit_serialization	Apply Process Behavior
TRUE	full	The apply process stops immediately and does not apply any unfinished transactions.
TRUE	none	When the apply process stops, some transactions that have been applied locally might have committed at the source database at a later point in time than some transactions that have not been applied locally.
FALSE	full	The apply process stops after applying the next uncommitted transaction in the commit order, if any such transaction is in progress.
FALSE	none	Before stopping, the apply process applies all of the transactions that have a commit time that is earlier than the applied transaction with the most recent commit time.

For example, assume that the commit_serialization apply process parameter is set to none and there are three transactions: transaction 1 has the earliest commit time, transaction 2 is committed after transaction 1, and transaction 3 has the latest commit time. Also assume that an apply process has applied transaction 1 and transaction 3 and is in the process of applying transaction 2 when the STOP_APPLY procedure is run. Given this scenario, if the force parameter is set to TRUE, then transaction 2 is not applied, and the apply process stops (transaction 2 is rolled back). If, however, the force parameter is set to FALSE, then transaction 2 is applied before the apply process stops.

A different scenario would result if the commit_serialization apply process parameter is set to full. For example, assume that the commit_serialization apply process parameter is set to full and there are three transactions: transaction A has the earliest commit time, transaction B is committed after transaction A, and transaction C has the latest commit time. In this case, the apply process has applied transaction A and is in the process of applying transactions B and C when the STOP_APPLY procedure is run. Given this scenario, if the force parameter is set to TRUE, then transactions B and C are not applied, and the apply process stops (transactions B and C are rolled back). If, however, the force parameter is set to FALSE, then transaction B is applied before the apply process stops, and transaction C is rolled back.

See Also: "SET_PARAMETER Procedure" on page 15-41 for more information about the commit_serialization apply process parameter

16

DBMS_AQ

The DBMS_AQ package provides an interface to Oracle Streams Advanced Queuing (AQ).

See Also:

- *Oracle Streams Advanced Queuing User's Guide and Reference*
- Oracle Streams AQ TYPEs for information about TYPEs to use with DBMS_AQ.

This chapter contains the following topics:

Using DBMS_AQ

Constants

When using enumerated constants such as BROWSE, LOCKED, or REMOVE, the PL/SQL constants must be specified with the scope of the packages defining it. All types associated with the operational interfaces have to be prepended with DBMS_AQ. For example: DBMS_AQ.BROWSE. Table 16–1 lists the PL/SQL enumerated constants that require the prefix, DBMS_AQ.

Table 16–1 Enumerated Constants

Parameter	Options
visibility	IMMEDIATE, ON_COMMIT
dequeue mode	BROWSE, LOCKED, REMOVE, REMOVE_NODATA
navigation	FIRST_MESSAGE, NEXT_MESSAGE, NEXT_TRANSACTION
state	WAITING, READY, PROCESSED, EXPIRED
sequence_deviation	BEFORE, TOP . Note: The sequence_deviation attribute has no effect in releases prior to Oracle Streams AQ 10g Release 1 (10.1) if message_grouping is set to TRANSACTIONAL. The sequence deviation feature is deprecated in Oracle Streams AQ 10g Release 2 (10.2).
wait	FOREVER, NO_WAIT
delay	NO_DELAY
expiration	NEVER
namespace	NAMESPACE_AQ, NAMESPACE_ANONYMOUS

Data Structures

Table 16–2 DBMS_AQ Data Structures

Data Structures	Description
Object Name on page 16-4	Names database objects
Type Name on page 16-4	Defines queue types
Oracle Streams AQ PL/SQL Callback on page 16-5	Specifies the user-defined PL/SQL procedure, defined in the database to be invoked on message notification

Object Name

The `object_name` data structure names database objects. It applies to queues, queue tables, agent names, and object types.

Syntax

```
object_name := VARCHAR2;
object_name := [schema_name.]name;
```

Usage Notes

Names for objects are specified by an optional schema name and a name. If the schema name is not specified, the current schema is assumed. The name must follow object name guidelines in *Oracle Database SQL Reference* with regard to reserved characters. Schema names, agent names, and object type names can be up to 30 bytes long. Queue names and queue table names can be up to 24 bytes long.

Type Name

The `type_name` data structure defines queue types.

Syntax

```
type_name := VARCHAR2;
type_name := object_type | "RAW";
```

Attributes

Table 16–3 Type Name Attributes

Attribute	Description
object_type	Maximum number of attributes in the object type is limited to 900.

Table 16–3 (Cont.) Type Name Attributes

Attribute	Description
"RAW"	To store payload of type RAW, Oracle Streams AQ creates a queue table with a LOB column as the payload repository. The theoretical maximum size of the message payload is the maximum amount of data that can be stored in a LOB column. However, the maximum size of the payload is determined by which programmatic environment you use to access Oracle Streams AQ. For PL/SQL, Java and precompilers the limit is 32K; for the OCI the limit is 4G. Because the PL/SQL enqueue and dequeue interfaces accept RAW buffers as the payload parameters you will be limited to 32K bytes. In OCI, the maximum size of your RAW data will be limited to the maximum amount of contiguous memory (as an OCIRaw is simply an array of bytes) that the OCI Object Cache can allocate. Typically, this will be at least 32K bytes and much larger in many cases. Because LOB columns are used for storing RAW payload, the Oracle Streams AQ administrator can choose the LOB tablespace and configure the LOB storage by constructing a LOB storage string in the storage_clause parameter during queue table creation time.

Oracle Streams AQ PL/SQL Callback

The plsqlcallback data structure specifies the user-defined PL/SQL procedure, defined in the database to be invoked on message notification.

Syntax

If a notification message is expected for a RAW payload enqueue, then the PL/SQL callback must have the following signature:

```
procedure plsqlcallback(
  context  IN  RAW,
  reginfo  IN  SYS.AQ$_REG_INFO,
  descr    IN  SYS.AQ$_DESCRIPTOR,
  payload  IN  RAW,
  payloadl IN  NUMBER);
```

Attributes

Table 16–4 Oracle Streams AQ PL/SQL Callback Attributes

Attribute	Description
context	Specifies the context for the callback function that was passed by dbms_aq.register. See AQ$_REG_INFO Type on page 184-11.
reginfo	See AQ$_REG_INFO Type on page 184-11.
descr	See AQ$_DESCRIPTOR Type on page 184-5
payload	If a notification message is expected for a raw payload enqueue then this contains the raw payload that was enqueued into a non persistent queue. In case of a persistent queue with raw payload this parameter will be null.
payloadl	Specifies the length of payload. If payload is null, payload1 = 0.

If the notification message is expected for an ADT payload enqueue, the PL/SQL callback must have the following signature:

```
procedure plsqlcallback(
  context  IN RAW,
  reginfo  IN SYS.AQ$_REG_INFO,
  descr    IN SYS.AQ$_DESCRIPTOR,
```

```
payload  IN VARCHAR2,
payloadl IN NUMBER);
```

Operational Notes

DBMS_AQ and DBMS_AQADM Java Classes

Java interfaces are available for `DBMS_AQ` and `DBMS_AQADM`. The Java interfaces are provided in the `$ORACLE_HOME/rdbms/jlib/aqapi.jar`. Users are required to have `EXECUTE` privileges on the `DBMS_AQIN` package to use these interfaces.

Summary of DBMS_AQ Subprograms

Table 16–5 DBMS_AQ Package Subprograms

Subprograms	Description
BIND_AGENT Procedure on page 16-9	Creates an entry for an Oracle Streams AQ agent in the LDAP directory
DEQUEUE Procedure on page 16-10	Dequeues a message from the specified queue
DEQUEUE_ARRAY Function on page 16-13	Dequeues an array of messages from the specified queue
ENQUEUE Procedure on page 16-15	Adds a message to the specified queue
ENQUEUE_ARRAY Function on page 16-17	Adds an array of messages to the specified queue
LISTEN Procedures on page 16-18	Listen to one or more queues on behalf of a list of agents
POST Procedure on page 16-20	Posts to a anonymous subscription which allows all clients who are registered for the subscription to get notifications
REGISTER Procedure on page 16-21	Registers for message notifications
UNBIND_AGENT Procedure on page 16-22	Removes an entry for an Oracle Streams AQ agent from the LDAP directory
UNREGISTER Procedure on page 16-23	Unregisters a subscription which turns off notification

Note: `DBMS_AQ` does not have a purity level defined; therefore, you cannot call any procedure in this package from other procedures that have `RNDS`, `WNDS`, `RNPS` or `WNPS` constraints defined.

BIND_AGENT Procedure

This procedure creates an entry for an Oracle Streams AQ agent in the LDAP server.

Syntax

```
DBMS_AQ.BIND_AGENT(
   agent        IN SYS.AQ$_AGENT,
   certificate  IN VARCHAR2 default NULL);
```

Parameters

Table 16–6 BIND_AGENT Procedure Parameters

Parameter	Description
`agent`	Agent that is to be registered in LDAP server.
`certificate`	Location (LDAP distinguished name) of the "organizationalperson" entry in LDAP whose digital certificate (attribute `usercertificate`) is to be used for this agent. Example: "cn=OE, cn=ACME, cn=com" is a distinguished name for a OrganizationalPerson OE whose certificate will be used with the specified agent.

Usage Notes

In the LDAP server, digital certificates are stored as an attribute (`usercertificate`) of the `OrganizationalPerson` entity. The distinguished name for this `OrganizationalPerson` must be specified when binding the agent.

DEQUEUE Procedure

This procedure dequeues a message from the specified queue.

Syntax

```
DBMS_AQ.DEQUEUE (
   queue_name           IN      VARCHAR2,
   dequeue_options      IN      dequeue_options_t,
   message_properties   OUT     message_properties_t,
   payload              OUT     "<ADT_1>"
   msgid                OUT     RAW);
```

Parameters

Table 16–7 DEQUEUE Procedure Parameters

Parameter	Description
`queue_name`	Specifies the name of the queue.
`dequeue_options`	See DEQUEUE_OPTIONS_T Type on page 184-15.
`message_properties`	See MESSAGE_PROPERTIES_T Type on page 184-22.
`payload`	Not interpreted by Oracle Streams AQ. The payload must be specified according to the specification in the associated queue table. For the definition of *`type_name`* refer to Type Name on page 16-4.
`msgid`	System generated identification of the message.

Usage Notes

The search criteria for messages to be dequeued is determined by the following parameters in `dequeue_options`:

- `consumer_name`
- `msgid`

  `Msgid` uniquely identifies the message to be dequeued. Only messages in the `READY` state are dequeued unless `msgid` is specified.

- `correlation`

  Correlation identifiers are application-defined identifiers that are not interpreted by Oracle Streams AQ.

- `deq_condition`

  Dequeue condition is an expression based on the message properties, the message data properties and PL/SQL functions. A `deq_condition` is specified as a Boolean expression using syntax similar to the `WHERE` clause of a SQL query. This Boolean expression can include conditions on message properties, user data properties (object payloads only), and PL/SQL or SQL functions (as specified in the where clause of a SQL query). Message properties include `priority`, `corrid` and other columns in the queue table.

  To specify dequeue conditions on a message payload (object payload), use attributes of the object type in clauses. You must prefix each attribute with `tab.user_data` as a qualifier to indicate the specific column of the queue table that stores the payload.

Example: `tab.user_data.orderstatus='EXPRESS'`

The dequeue order is determined by the values specified at the time the queue table is created unless overridden by the `msgid` and correlation ID in `dequeue_options`.

The database-consistent read mechanism is applicable for queue operations. For example, a `BROWSE` call may not see a message that is enqueued after the beginning of the browsing transaction.

The default `NAVIGATION` parameter during dequeue is `NEXT_MESSAGE`. This means that subsequent dequeues will retrieve the messages from the queue based on the snapshot obtained in the first dequeue. In particular, a message that is enqueued after the first dequeue command will be processed only after processing all the remaining messages in the queue. This is usually sufficient when all the messages have already been enqueued into the queue, or when the queue does not have a priority-based ordering. However, applications must use the `FIRST_MESSAGE` navigation option when the first message in the queue needs to be processed by every dequeue command. This usually becomes necessary when a higher priority message arrives in the queue while messages already-enqueued are being processed.

Note: It may be more efficient to use the `FIRST_MESSAGE` navigation option when messages are concurrently enqueued. If the `FIRST_MESSAGE` option is not specified, Oracle Streams AQ continually generates the snapshot as of the first dequeue command, leading to poor performance. If the `FIRST_MESSAGE` option is specified, then Oracle Streams AQ uses a new snapshot for every dequeue command.

Messages enqueued in the same transaction into a queue that has been enabled for message grouping will form a group. If only one message is enqueued in the transaction, then this will effectively form a group of one message. There is no upper limit to the number of messages that can be grouped in a single transaction.

In queues that have not been enabled for message grouping, a dequeue in `LOCKED` or `REMOVE` mode locks only a single message. By contrast, a dequeue operation that seeks to dequeue a message that is part of a group will lock the entire group. This is useful when all the messages in a group need to be processed as an atomic unit.

When all the messages in a group have been dequeued, the dequeue returns an error indicating that all messages in the group have been processed. The application can then use the `NEXT_TRANSACTION` to start dequeuing messages from the next available group. In the event that no groups are available, the dequeue will time out after the specified `WAIT` period.

Using Secure Queues

For secure queues, you must specify `consumer_name` in the `dequeue_options` parameter. See DEQUEUE_OPTIONS_T Type on page 184-15 for more information about `consumer_name`.

When you use secure queues, the following are required:

- You must have created a valid Oracle Streams AQ agent using `DBMS_AQADM.CREATE_AQ_AGENT`. See CREATE_AQ_AGENT Procedure on page 17-22.
- You must map the Oracle Streams AQ agent to a database user with dequeue privileges on the secure queue. Use `DBMS_AQADM.ENABLE_DB_ACCESS` to do this. See ENABLE_DB_ACCESS Procedure on page 17-35.

See Also: *Oracle Streams Concepts and Administration* for information about secure queues

DEQUEUE_ARRAY Function

This function dequeues an array of messages and returns them in the form of an array of payloads, an array of message properties and an array of message IDs. This function returns the number of messages successfully dequeued.

Syntax

```
DBMS_AQ.DEQUEUE_ARRAY (
   queue_name                IN   VARCHAR2,
   dequeue_options           IN   dequeue_options_t,
   array_size                IN   pls_integer,
   message_properties_array  OUT  message_properties_array_t,
   payload_array             OUT  "<COLLECTION_1>",
   msgid_array               OUT  msgid_array_t,
   error_array               OUT  error_array_t)
RETURN pls_integer;
```

Parameters

Table 16–8 DEQUEUE_ARRAY Function Parameters

Parameter	Description
queue_name	The queue name from which messages are dequeued (same as single-row dequeue).
dequeue_options	The set of options which will be applied to all messages in the array (same as single-row dequeue).
array_size	The number of elements to dequeue.
message_properties_array	A record containing an array corresponding to each message property. Each payload element has a corresponding set of message properties. See MESSAGE_PROPERTIES_ARRAY_T Type on page 184-26.
payload_array	An array of dequeued payload data. "<COLLECTION_1>" can be an associative array, varray or nested table in its PL/SQL representation.
msgid_array	An array of message IDs of the dequeued messages. See MSGID_ARRAY_T Type on page 184-27.
error_array	Currently not implemented

Usage Notes

A nonzero wait time, as specified in dequeue_options, is recognized only when there are no messages in the queue. If the queue contains messages that are eligible for dequeue, then the DEQUEUE_ARRAY function will dequeue up to array_size messages and return immediately.

Dequeue by message_id is not supported. See DEQUEUE Procedure on page 16-10 for more information on the navigation parameter. Existing NAVIGATION modes are supported. In addition, two new NAVIGATION modes are supported for queues enabled for message grouping:

- FIRST_MESSAGE_MULTI_GROUP
- NEXT_MESSAGE_MULTI_GROUP

See Also: ENQUEUE_OPTIONS_T Type on page 184-18

For transaction grouped queues and ONE_GROUP navigation, messages are dequeued from a single transaction group only, subject to the array_size limit. In MULTI_GROUP navigation, messages are dequeued across multiple transaction groups, still subject to the array_size limit. ORA-25235 is returned to indicate the end of a transaction group.

DEQUEUE_ARRAY is not supported for buffered messages, but you can still use this procedure on individual buffered messages by setting array_size to one message.

ENQUEUE Procedure

This procedure adds a message to the specified queue.

Syntax

```
DBMS_AQ.ENQUEUE (
   queue_name           IN      VARCHAR2,
   enqueue_options      IN      enqueue_options_t,
   message_properties   IN      message_properties_t,
   payload              IN       "<ADT_1>",
   msgid                OUT     RAW);
```

Parameters

Table 16–9 ENQUEUE Procedure Parameters

Parameter	Description
queue_name	Specifies the name of the queue to which this message should be enqueued. The queue cannot be an exception queue.
enqueue_options	See ENQUEUE_OPTIONS_T Type on page 184-18.
message_properties	See MESSAGE_PROPERTIES_T Type on page 184-22.
payload	Not interpreted by Oracle Streams AQ. The payload must be specified according to the specification in the associated queue table. NULL is an acceptable parameter. For the definition of *type_name* refer to Type Name on page 16-4.
msgid	System generated identification of the message. This is a globally unique identifier that can be used to identify the message at dequeue time.

Usage Notes

The sequence_deviation parameter in enqueue_options can be used to change the order of processing between two messages. The identity of the other message, if any, is specified by the enqueue_options parameter relative_msgid. The relationship is identified by the sequence_deviation parameter.

Specifying sequence_deviation for a message introduces some restrictions for the delay and priority values that can be specified for this message. The delay of this message must be less than or equal to the delay of the message before which this message is to be enqueued. The priority of this message must be greater than or equal to the priority of the message before which this message is to be enqueued.

> **Note:** The sequence_deviation attribute has no effect in releases prior to Oracle Streams AQ 10g Release 1 (10.1) if message_grouping is set to TRANSACTIONAL. The sequence deviation feature is deprecated in Oracle Streams AQ 10g Release 2 (10.2).

If a message is enqueued to a multiconsumer queue with no recipient, and if the queue has no subscribers (or rule-based subscribers that match this message), then Oracle error ORA_24033 is raised. This is a warning that the message will be discarded because there are no recipients or subscribers to whom it can be delivered.

Using Secure Queues

For secure queues, you must specify the `sender_id` in the `messages_properties` parameter. See MESSAGE_PROPERTIES_T Type on page 184-22 for more information about `sender_id`.

When you use secure queues, the following are required:

- You must have created a valid Oracle Streams AQ agent using `DBMS_AQADM.CREATE_AQ_AGENT`. See CREATE_AQ_AGENT Procedure on page 17-22.
- You must map `sender_id` to a database user with enqueue privileges on the secure queue. Use `DBMS_AQADM.ENABLE_DB_ACCESS` to do this. See ENABLE_DB_ACCESS Procedure on page 17-35.

> **See Also:** *Oracle Streams Concepts and Administration* for information about secure queues

ENQUEUE_ARRAY Function

This function enqueues an array of payloads using a corresponding array of message properties. The output will be an array of message IDs of the enqueued messages.

Syntax

```
DBMS_AQ.ENQUEUE_ARRAY (
   queue_name                IN   VARCHAR2,
   enqueue_options           IN   enqueue_options_t,
   array_size                IN   pls_integer,
   message_properties_array  IN   message_properties_array_t,
   payload_array             IN   "<COLLECTION_1>",
   msgid_array               OUT  msgid_array_t,
   error_array               OUT  error_array_t)
RETURN pls_integer;
```

Parameters

Table 16–10 ENQUEUE_ARRAY Function Parameters

Parameter	Description
queue_name	The queue name in which messages are enqueued (same as single-row enqueue).
enqueue_options	See ENQUEUE_OPTIONS_T Type on page 184-18.
array_size	The number of elements to enqueue.
message_properties_array	A record containing an array corresponding to each message property. For each property, the user must allocate array_size elements. See MESSAGE_PROPERTIES_ARRAY_T Type on page 184-26.
payload_array	An array of payload data. "<COLLECTION_1>" can be an associative array, VARRAY, or nested table in its PL/SQL representation.
msgid_array	An array of message IDs for the enqueued messages. If an error occurs for a particular message, then its corresponding message ID is null. See MSGID_ARRAY_T Type on page 184-27.
error_array	Currently not implemented

Usage Notes

ENQUEUE_ARRAY is not supported for buffered messages, but you can still use this procedure on individual buffered messages by setting array_size to one message.

LISTEN Procedures

This procedure listens on one or more queues on behalf of a list of agents. The address field of the agent indicates the queue the agent wants to monitor. Only local queues are supported as addresses. Protocol is reserved for future use.

Syntax

```
DBMS_AQ.LISTEN (
   agent_list            IN    AQ$_AGENT_LIST_T,
   wait                  IN    BINARY_INTEGER DEFAULT DBMS_AQ.FOREVER,
   agent                 OUT   SYS.AQ$_AGENT);

DBMS_AQ.LISTEN (
   agent_list             IN   AQ$_AGENT_LIST_T,
   wait                   IN   BINARY_INTEGER DEFAULT FOREVER,
   listen_delivery_mode   IN   PLS_INTEGER DEFAULT DBMS_AQ.PERSISTENT,
   agent                  OUT  SYS.AQ$_AGENT,
   message_delivery_mode  OUT  PLS_INTEGER);

TYPE aq$_agent_list_t IS TABLE of aq$_agent INDEXED BY BINARY_INTEGER;
TYPE aq$_agent_list_t IS TABLE of aq$_agent INDEXED BY BINARY_INTEGER;
```

Parameters

Table 16–11 LISTEN Procedure Parameters

Parameter	Description
`agent_list`	List of agents to listen for
`wait`	Time out for the listen call in seconds. By default, the call will block forever.
`listen_delivery_mode`	The caller specifies whether it is interested in persistent, buffered messages or both types of messages, specifying a delivery mode of `DBMS_AQ.PERSISTENT` or `DBMS_AQ.BUFFERED` or `DBMS_AQ.PERSISTENT_OR_BUFFERED`
`agent`	Agent with a message available for consumption
`message_delivery_mode`	Returns the message type along with the queue and consumer for which there is a message

Usage Notes

If agent-address is a multiconsumer queue, then agent-name is mandatory. For single-consumer queues, agent-name must not be specified.

This procedure takes a list of agents as an argument. You specify the queue to be monitored in the address field of each agent listed. You also must specify the name of the agent when monitoring multiconsumer queues. For single-consumer queues, an agent name must not be specified. Only local queues are supported as addresses. Protocol is reserved for future use.

This is a blocking call that returns when there is a message ready for consumption for an agent in the list. If there are messages for more than one agent, only the first agent listed is returned. If there are no messages found when the wait time expires, an error is raised.

A successful return from the LISTEN call is only an indication that there is a message for one of the listed agents in one the specified queues. The interested agent must still dequeue the relevant message.

Note: You cannot call LISTEN on nonpersistent queues.

POST Procedure

This procedure posts to a list of anonymous subscriptions that allows all clients who are registered for the subscriptions to get notifications.

Syntax

```
DBMS_AQ.POST (
  post_list       IN  SYS.AQ$_POST_INFO_LIST,
  post_count      IN  NUMBER);
```

Parameters

Table 16–12 POST Procedure Parameters

Parameter	Description
`post_list`	Specifies the list of anonymous subscriptions to which you want to post. It is a list of AQ$_POST_INFO_LIST Type.
`post_count`	Specifies the number of entries in the `post_list`.

Usage Notes

This procedure is used to post to anonymous subscriptions which allows all clients who are registered for the subscriptions to get notifications. Several subscriptions can be posted to at one time.

REGISTER Procedure

This procedure registers an e-mail address, user-defined PL/SQL procedure, or HTTP URL for message notification.

Syntax

```
DBMS_AQ.REGISTER (
   reg_list     IN  SYS.AQ$_REG_INFO_LIST,
   count        IN  NUMBER);
```

Parameters

Table 16–13 REGISTER Procedure Parameters

Parameter	Description
`reg_list`	Specifies the list of subscriptions to which you want to register for message notifications. It is a list of AQ$_REG_INFO Type.
`count`	Specifies the number of entries in the `reg_list`.

Usage Notes

This procedure is used to register for notifications. You can specify an e-mail address to which message notifications are sent, register a procedure to be invoked on a notification, or register an HTTP URL to which the notification is posted. Interest in several subscriptions can be registered at one time.

If you register for e-mail notifications, you should set the host name and port name for the SMTP server that will be used by the database to send e-mail notifications. If required, you should set the send-from e-mail address, which is set by the database as the `sent from` field. You need a Java-enabled database to use this feature.

If you register for HTTP notifications, you may want to set the host name and port number for the proxy server and a list of no-proxy domains that will be used by the database to post HTTP notifications.

> **See Also:** Chapter 18, "DBMS_AQELM" for more information on e-mail and HTTP notifications

UNBIND_AGENT Procedure

This procedure removes the entry for an Oracle Streams AQ agent from the LDAP server.

Syntax

```
DBMS_AQ.UNBIND_AGENT(
   agent     IN SYS.AQ$_AGENT);
```

Parameters

Table 16–14 BIND_AGENT Procedure Parameters

Parameter	Description
agent	Agent that is to be removed from the LDAP server

UNREGISTER Procedure

This procedure unregisters a subscription which turns off notifications.

Syntax

```
DBMS_AQ.UNREGISTER (
   reg_list     IN  SYS.AQ$_REG_INFO_LIST,
   reg_count    IN  NUMBER);
```

Parameters

Table 16–15 UNREGISTER Procedure Parameters

Parameter	Description
`reg_list`	Specifies the list of subscriptions to which you want to register for message notifications. It is a list of AQ$_REG_INFO Type.
`reg_count`	Specifies the number of entries in the `reg_list`.

Usage Notes

This procedure is used to unregister a subscription which turns off notifications. Several subscriptions can be unregistered from at one time.

17

DBMS_AQADM

The DBMS_AQADM package provides procedures to manage Oracle Streams Advanced Queuing (AQ) configuration and administration information.

See Also:

- *Oracle Streams Advanced Queuing User's Guide and Reference*
- Chapter 184, "Oracle Streams AQ TYPEs" for information about the TYPEs to use with DBMS_AQADM

This chapter contains the following topics:

Using DBMS_AQADM

This section contains the following topics.

Constants

When using enumerated constants, such as INFINITE, TRANSACTIONAL, or NORMAL_QUEUE, the symbol must be specified with the scope of the packages defining it. All types associated with the administrative interfaces must be prepended with DBMS_AQADM. For example: DBMS_AQADM.NORMAL_QUEUE.

Table 17–1 Enumerated Types in the Administrative Interface

Parameter	Options
retention	0, 1, 2...INFINITE
message_grouping	TRANSACTIONAL, NONE
queue_type	NORMAL_QUEUE, EXCEPTION_QUEUE, NON_PERSISTENT_QUEUE

See Also: For more information on the Java classes and data structures used in both DBMS_AQ and DBMS_AQADM, see the DBMS_AQ package.

Subprogram Groups

This DBMS_AQADM package is made up of the following subprogram groups:

Queue Table Subprograms

Table 17–2 Queue Table Subprograms

Subprograms	Description
ALTER_QUEUE_TABLE Procedure on page 17-20	Alters the existing properties of a queue table
CREATE_QUEUE_TABLE Procedure on page 17-26	Creates a queue table for messages of a predefined type
DROP_QUEUE_TABLE Procedure on page 17-34	Drops an existing queue table
ENABLE_JMS_TYPES Procedure on page 17-36	A precondition for the enqueue of JMS types and XML types
MIGRATE_QUEUE_TABLE Procedure on page 17-41	Upgrades an 8.0-compatible queue table to an 8.1-compatible or higher queue table, or downgrades an 8.1-compatible or higher queue table to an 8.0-compatible queue table
PURGE_QUEUE_TABLE Procedure on page 17-42	Purges messages from queue tables

Privilege Subprograms

Table 17–3 Privilege Subprograms

Subprograms	Description
GRANT_QUEUE_PRIVILEGE Procedure on page 17-39	Grants privileges on a queue to users and roles
GRANT_SYSTEM_ PRIVILEGE Procedure on page 17-40	Grants Oracle Streams AQ system privileges to users and roles
REVOKE_QUEUE_ PRIVILEGE Procedure on page 17-46	Revokes privileges on a queue from users and roles
REVOKE_SYSTEM_ PRIVILEGE Procedure on page 17-47	Revokes Oracle Streams AQ system privileges from users and roles

Queue Subprograms

Table 17–4 Queue Subprograms

Subprograms	Description
ALTER_QUEUE Procedure on page 17-19	Alters existing properties of a queue
CREATE_NP_QUEUE Procedure on page 17-23	Creates a nonpersistent `RAW` queue
CREATE_QUEUE Procedure on page 17-24	Creates a queue in the specified queue table
DROP_QUEUE Procedure on page 17-33	Drops an existing queue
QUEUE_SUBSCRIBERS Function on page 17-44	Returns the subscribers to an 8.0-compatible multiconsumer queue in the PL/SQL index by table collection type `DBMS_AQADM.AQ$_subscriber_list_t`
START_QUEUE Procedure on page 17-51	Enables the specified queue for enqueuing or dequeuing
STOP_QUEUE Procedure on page 17-52	Disables enqueuing or dequeuing on the specified queue

Subscriber Subprograms

Table 17–5 Subscriber Subprograms

Subprograms	Description
ADD_SUBSCRIBER Procedure on page 17-16	Adds a default subscriber to a queue
ALTER_SUBSCRIBER Procedure on page 17-21	Alters existing properties of a subscriber to a specified queue
REMOVE_SUBSCRIBER Procedure on page 17-45	Removes a default subscriber from a queue

Notification Subprograms

Table 17–6 Notification Subprograms

Subprograms	Description
GET_WATERMARK Procedure on page 17-38	Retrieves the value of watermark set by the `SET_WATERMARK Procedure`
SET_WATERMARK Procedure on page 17-50	Used for Oracle Streams AQ notification to specify and limit memory use

Propagation Subprograms

Table 17–7 Propagation Subprograms

Subprograms	Description
ALTER_PROPAGATION_SCHEDULE Procedure on page 17-18	Alters parameters for a propagation schedule
DISABLE_PROPAGATION_SCHEDULE Procedure on page 17-31	Disables a propagation schedule
ENABLE_PROPAGATION_SCHEDULE Procedure on page 17-37	Enables a previously disabled propagation schedule
SCHEDULE_PROPAGATION Procedure on page 17-48	Schedules propagation of messages from a queue to a destination identified by a specific database link
UNSCHEDULE_PROPAGATION Procedure on page 17-53	Unschedules previously scheduled propagation of messages from a queue to a destination identified by a specific database link
VERIFY_QUEUE_TYPES Procedure on page 17-54	Verifies that the source and destination queues have identical types

Oracle Streams AQ Agent Subprograms

Table 17–8 Oracle Streams AQ Agent Subprograms

Subprograms	Description
ALTER_AQ_AGENT Procedure on page 17-17	Alters an agent registered for Oracle Streams AQ Internet access, and an Oracle Streams AQ agent that accesses secure queues
CREATE_AQ_AGENT Procedure on page 17-22	Registers an agent for Oracle Streams AQ Internet access using HTTP/SMTP protocols, and creates an Oracle Streams AQ agent to access secure queues
DISABLE_DB_ACCESS Procedure on page 17-30	Revokes the privileges of a specific database user from an Oracle Streams AQ Internet agent
DROP_AQ_AGENT Procedure on page 17-32	Drops an agent that was previously registered for Oracle Streams AQ Internet access
ENABLE_DB_ACCESS Procedure on page 17-35	Grants an Oracle Streams AQ Internet agent the privileges of a specific database user

Alias Subprograms

Table 17–9 Alias Subprograms

Subprograms	Description
ADD_ALIAS_TO_LDAP Procedure on page 17-15	Creates an alias for a queue, agent, or a JMS ConnectionFactory in LDAP
DEL_ALIAS_FROM_LDAP Procedure on page 17-29	Drops an alias for a queue, agent, or JMS ConnectionFactory in LDAP

Summary of DBMS_AQADM Subprograms

Table 17–10 DBMS_AQADM Package Subprograms

Subprograms	Description
ADD_ALIAS_TO_LDAP Procedure on page 17-15	Creates an alias for a queue, agent, or a JMS ConnectionFactory in LDAP
ADD_SUBSCRIBER Procedure on page 17-16	Adds a default subscriber to a queue
ALTER_AQ_AGENT Procedure on page 17-17	Alters an agent registered for Oracle Streams AQ Internet access, and an Oracle Streams AQ agent that accesses secure queues
ALTER_PROPAGATION_SCHEDULE Procedure on page 17-18	Alters parameters for a propagation schedule
ALTER_QUEUE Procedure on page 17-19	Alters existing properties of a queue
ALTER_QUEUE_TABLE Procedure on page 17-20	Alters the existing properties of a queue table
ALTER_SUBSCRIBER Procedure on page 17-21	Alters existing properties of a subscriber to a specified queue
CREATE_AQ_AGENT Procedure on page 17-22	Registers an agent for Oracle Streams AQ Internet access using HTTP/SMTP protocols, and creates an Oracle Streams AQ agent to access secure queues
CREATE_NP_QUEUE Procedure on page 17-23	Creates a nonpersistent `RAW` queue
CREATE_QUEUE Procedure on page 17-24	Creates a queue in the specified queue table
CREATE_QUEUE_TABLE Procedure on page 17-26	Creates a queue table for messages of a predefined type
DEL_ALIAS_FROM_LDAP Procedure on page 17-29	Drops an alias for a queue, agent, or JMS ConnectionFactory in LDAP
DISABLE_DB_ACCESS Procedure on page 17-30	Revokes the privileges of a specific database user from an Oracle Streams AQ Internet agent
DISABLE_PROPAGATION_SCHEDULE Procedure on page 17-31	Disables a propagation schedule
DROP_AQ_AGENT Procedure on page 17-32	Drops an agent that was previously registered for Oracle Streams AQ Internet access
DROP_QUEUE Procedure on page 17-33	Drops an existing queue
DROP_QUEUE_TABLE Procedure on page 17-34	Drops an existing queue table
ENABLE_DB_ACCESS Procedure on page 17-35	Grants an Oracle Streams AQ Internet agent the privileges of a specific database user
ENABLE_JMS_TYPES Procedure on page 17-36	A precondition for the enqueue of JMS types and XML types

Table 17–10 (Cont.) DBMS_AQADM Package Subprograms

Subprograms	Description
ENABLE_PROPAGATION_SCHEDULE Procedure on page 17-37	Enables a previously disabled propagation schedule
GET_WATERMARK Procedure on page 17-38	Retrieves the value of watermark set by the `SET_WATERMARK Procedure`
GRANT_QUEUE_PRIVILEGE Procedure on page 17-39	Grants privileges on a queue to users and roles
GRANT_SYSTEM_PRIVILEGE Procedure on page 17-40	Grants Oracle Streams AQ system privileges to users and roles
MIGRATE_QUEUE_TABLE Procedure on page 17-41	Upgrades an 8.0-compatible queue table to an 8.1-compatible or higher queue table, or downgrades an 8.1-compatible or higher queue table to an 8.0-compatible queue table
PURGE_QUEUE_TABLE Procedure on page 17-42	Purges messages from queue tables
QUEUE_SUBSCRIBERS Function on page 17-44	Returns the subscribers to an 8.0-compatible multiconsumer queue in the PL/SQL index by table collection type `DBMS_AQADM.AQ$_subscriber_list_t`
REMOVE_SUBSCRIBER Procedure on page 17-45	Removes a default subscriber from a queue
REVOKE_QUEUE_PRIVILEGE Procedure on page 17-46	Revokes privileges on a queue from users and roles
REVOKE_SYSTEM_PRIVILEGE Procedure on page 17-47	Revokes Oracle Streams AQ system privileges from users and roles
SCHEDULE_PROPAGATION Procedure on page 17-48	Schedules propagation of messages from a queue to a destination identified by a specific database link
SET_WATERMARK Procedure on page 17-50	Used for Oracle Streams AQ notification to specify and limit memory use
START_QUEUE Procedure on page 17-51	Enables the specified queue for enqueuing or dequeuing
STOP_QUEUE Procedure on page 17-52	Disables enqueuing or dequeuing on the specified queue
UNSCHEDULE_PROPAGATION Procedure on page 17-53	Unschedules previously scheduled propagation of messages from a queue to a destination identified by a specific database link
VERIFY_QUEUE_TYPES Procedure on page 17-54	Verifies that the source and destination queues have identical types

ADD_ALIAS_TO_LDAP Procedure

This procedure creates an alias for a queue, agent, or a JMS ConnectionFactory in LDAP. The alias will be placed directly under the database server's distinguished name in LDAP hierarchy.

Syntax

```
DBMS_AQADM.ADD_ALIAS_TO_LDAP(
   alias          IN VARCHAR2,
   obj_location   IN VARCHAR2);
```

Parameters

Table 17–11 ADD_ALIAS_TO_LDAP Procedure Parameters

Parameter	Description
`alias`	Name of the alias. Example: `west_shipping`.
`obj_location`	The distinguished name of the object (queue, agent or connection factory) to which `alias` refers.

Usage Notes

This method can be used to create aliases for queues, agents, and JMS ConnectionFactory objects. These object must exist before the alias is created. These aliases can be used for JNDI lookup in JMS and Oracle Streams AQ Internet access.

ADD_SUBSCRIBER Procedure

This procedure adds a default subscriber to a queue.

Syntax

```
DBMS_AQADM.ADD_SUBSCRIBER (
   queue_name      IN    VARCHAR2,
   subscriber      IN    sys.aq$_agent,
   rule            IN    VARCHAR2 DEFAULT NULL,
   transformation  IN    VARCHAR2 DEFAULT NULL
   queue_to_queue  IN    BOOLEAN DEFAULT FALSE,
   delivery_mode   IN    PLS_INTEGER DEFAULT DBMS_AQADM.PERSISTENT);
```

Parameters

Table 17–12 ADD_SUBSCRIBER Procedure Parameters

Parameter	Description
queue_name	Name of the queue.
subscriber	Agent on whose behalf the subscription is being defined.
rule	A conditional expression based on the message properties, the message data properties and PL/SQL functions. A rule is specified as a Boolean expression using syntax similar to the WHERE clause of a SQL query. This Boolean expression can include conditions on message properties, user data properties (object payloads only), and PL/SQL or SQL functions (as specified in the where clause of a SQL query). Currently supported message properties are priority and corrid. To specify rules on a message payload (object payload), use attributes of the object type in clauses. You must prefix each attribute with tab.user_data as a qualifier to indicate the specific column of the queue table that stores the payload. The rule parameter cannot exceed 4000 characters.
transformation	Specifies a transformation that will be applied when this subscriber dequeues the message. The source type of the transformation must match the type of the queue. If the subscriber is remote, then the transformation is applied before propagation to the remote queue.
queue_to_queue	If TRUE, propagation is from queue-to-queue.
delivery_mode	The administrator may specify one of DBMS_AQADM.PERSISTENT, DBMS_AQADM.BUFFERED, or DBMS_AQADM.PERSISTENT_OR_BUFFERED for the delivery mode of the messages the subscriber is interested in. This parameter will not be modifiable by ALTER_SUBSCRIBER.

Usage Notes

A program can enqueue messages to a specific list of recipients or to the default list of subscribers. This operation only succeeds on queues that allow multiple consumers. This operation takes effect immediately, and the containing transaction is committed. Enqueue requests that are executed after the completion of this call will reflect the new behavior.

Any string within the rule must be quoted:

```
rule   => 'PRIORITY <= 3 AND CORRID =  ''FROM JAPAN'''
```

Note that these are all single quotation marks.

ALTER_AQ_AGENT Procedure

This procedure alters an agent registered for Oracle Streams AQ Internet access. It is also used to alter an Oracle Streams AQ agent that accesses secure queues.

> **See Also:** *Oracle Streams Concepts and Administration* for information about secure queues

Syntax

```
DBMS_AQADM.ALTER_AQ_AGENT (
  agent_name                IN VARCHAR2,
  certificate_location      IN VARCHAR2 DEFAULT NULL,
  enable_http               IN BOOLEAN DEFAULT FALSE,
  enable_smtp               IN BOOLEAN DEFAULT FALSE,
  enable_anyp               IN BOOLEAN DEFAULT FALSE )
```

Parameters

Table 17–13 ALTER_AQ_AGENT Procedure Parameters

Parameter	Description
`agent_name`	Specifies the username of the Oracle Streams AQ Internet agent.
`certification_location`	Agent's certificate location in LDAP (default is `NULL`). If the agent is allowed to access Oracle Streams AQ through SMTP, then its certificate must be registered in LDAP. For access through HTTP, the certificate location is not required.
`enable_http`	`TRUE` means the agent can access Oracle Streams AQ through HTTP. `FALSE` means the agent cannot access Oracle Streams AQ through HTTP.
`enable_smtp`	`TRUE` means the agent can access Oracle Streams AQ through SMTP (e-mail). `FALSE` means the agent cannot access Oracle Streams AQ through SMTP.
`enable_anyp`	`TRUE` means the agent can access Oracle Streams AQ through any protocol (HTTP or SMTP).

ALTER_PROPAGATION_SCHEDULE Procedure

This procedure alters parameters for a propagation schedule.

Syntax

```
DBMS_AQADM.ALTER_PROPAGATION_SCHEDULE (
   queue_name           IN    VARCHAR2,
   destination          IN    VARCHAR2 DEFAULT NULL,
   duration             IN    NUMBER   DEFAULT NULL,
   next_time            IN    VARCHAR2 DEFAULT NULL,
   latency              IN    NUMBER   DEFAULT 60,
   destination_queue    IN    VARCHAR2 DEFAULT NULL);
```

Parameters

Table 17–14 ALTER_PROPAGATION_SCHEDULE Procedure Parameters

Parameter	Description
queue_name	Name of the source queue whose messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the user.
destination	Destination database link. Messages in the source queue for recipients at this destination are propagated. If it is NULL, then the destination is the local database and messages are propagated to other queues in the local database. The length of this field is currently limited to 128 bytes, and if the name is not fully qualified, then the default domain name is used.
duration	Duration of the propagation window in seconds. A NULL value means the propagation window is forever or until the propagation is unscheduled.
next_time	Date function to compute the start of the next propagation window from the end of the current window. If this value is NULL, then propagation is stopped at the end of the current window. For example, to start the window at the same time every day, next_time should be specified as SYSDATE + 1 - duration/86400.
latency	Maximum wait, in seconds, in the propagation window for a message to be propagated after it is enqueued. The default value is 60. Caution: if latency is not specified for this call, then latency will over-write any existing value with the default value. For example, if the latency is 60 seconds and there are no messages to be propagated during the propagation window, then messages from that queue for the destination are not propagated for at least 60 more seconds. It will be at least 60 seconds before the queue will be checked again for messages to be propagated for the specified destination. If the latency is 600, then the queue will not be checked for 10 minutes and if the latency is 0, then a job queue process will be waiting for messages to be enqueued for the destination and as soon as a message is enqueued it will be propagated.
destination_ queue	Name of the target queue to which messages are to be propagated in the form of a dblink

ALTER_QUEUE Procedure

This procedure alters existing properties of a queue. The parameters max_retries, retention_time, and retry_delay are not supported for nonpersistent queues.

Syntax

```
DBMS_AQADM.ALTER_QUEUE (
   queue_name        IN    VARCHAR2,
   max_retries       IN    NUMBER   DEFAULT NULL,
   retry_delay       IN    NUMBER   DEFAULT NULL,
   retention_time    IN    NUMBER   DEFAULT NULL,
   auto_commit       IN    BOOLEAN  DEFAULT TRUE,
   comment           IN    VARCHAR2 DEFAULT NULL);
```

Parameters

Table 17–15 ALTER_QUEUE Procedure Parameters

Parameter	Description
queue_name	Name of the queue that is to be altered
max_retries	Limits the number of times a dequeue with REMOVE mode can be attempted on a message. The maximum value of max_retries is 2**31 -1. A message is moved to an exception queue if RETRY_COUNT is greater than MAX_RETRIES. RETRY_COUNT is incremented when the application issues a rollback after executing the dequeue. If a dequeue transaction fails because the server process dies (including ALTER SYSTEM KILL SESSION) or SHUTDOWN ABORT on the instance, then RETRY_COUNT is not incremented. Note that max_retries is supported for all single consumer queues and 8.1-compatible or higher multiconsumer queues but not for 8.0-compatible multiconsumer queues.
retry_delay	Delay time in seconds before this message is scheduled for processing again after an application rollback. The default is NULL, which means that the value will not be altered. Note that retry_delay is supported for single consumer queues and 8.1-compatible or higher multiconsumer queues but not for 8.0-compatible multiconsumer queues.
retention_time	Retention time in seconds for which a message is retained in the queue table after being dequeued. The default is NULL, which means that the value will not be altered.
auto_commit	TRUE causes the current transaction, if any, to commit before the ALTER_QUEUE operation is carried out. The ALTER_QUEUE operation become persistent when the call returns. This is the default. FALSE means the operation is part of the current transaction and becomes persistent only when the caller enters a commit. Caution: This parameter has been deprecated.
comment	User-specified description of the queue. This user comment is added to the queue catalog. The default value is NULL, which means that the value will not be changed.

ALTER_QUEUE_TABLE Procedure

This procedure alters the existing properties of a queue table.

Syntax

```
DBMS_AQADM.ALTER_QUEUE_TABLE (
   queue_table          IN   VARCHAR2,
   comment              IN   VARCHAR2       DEFAULT NULL,
   primary_instance     IN   BINARY_INTEGER DEFAULT NULL,
   secondary_instance   IN   BINARY_INTEGER DEFAULT NULL);
```

Parameters

Table 17–16 ALTER_QUEUE_TABLE Procedure Parameters

Parameter	Description
queue_table	Name of a queue table to be created.
comment	Modifies the user-specified description of the queue table. This user comment is added to the queue catalog. The default value is NULL which means that the value will not be changed.
primary_instance	This is the primary owner of the queue table. Queue monitor scheduling and propagation for the queues in the queue table will be done in this instance. The default value is NULL, which means that the current value will not be changed.
secondary_instance	The queue table fails over to the secondary instance if the primary instance is not available. The default value is NULL, which means that the current value will not be changed.

ALTER_SUBSCRIBER Procedure

This procedure alters existing properties of a subscriber to a specified queue. Only the rule can be altered.

Syntax

```
DBMS_AQADM.ALTER_SUBSCRIBER (
   queue_name     IN    VARCHAR2,
   subscriber     IN    sys.aq$_agent,
   rule           IN    VARCHAR2
   transformation IN    VARCHAR2);
```

Parameters

Table 17–17 ALTER_SUBSCRIBER Procedure Parameters

Parameter	Description
queue_name	Name of the queue.
subscriber	Agent on whose behalf the subscription is being altered. See "AQ$_ AGENT Type" on page 184-3.
rule	A conditional expression based on the message properties, the message data properties and PL/SQL functions. The rule parameter cannot exceed 4000 characters. To eliminate the rule, set the rule parameter to NULL.
transformation	Specifies a transformation that will be applied when this subscriber dequeues the message. The source type of the transformation must match the type of the queue. If the subscriber is remote, then the transformation is applied before propagation to the remote queue.

Usage Notes

This procedure alters both the rule and the transformation for the subscriber. If you want to retain the existing value for either of them, you must specify its old value. The current values for rule and transformation for a subscriber can be obtained from the *schema*.AQ$*queue_table*_R and *schema*.AQ$*queue_table*_S views.

CREATE_AQ_AGENT Procedure

This procedure registers an agent for Oracle Streams AQ Internet access using HTTP/SMTP protocols. It is also used to create an Oracle Streams AQ agent to access secure queues.

> **See Also:** *Oracle Streams Concepts and Administration* for information about secure queues

Syntax

```
DBMS_AQADM.CREATE_AQ_AGENT (
  agent_name                IN VARCHAR2,
  certificate_location      IN VARCHAR2 DEFAULT NULL,
  enable_http               IN BOOLEAN DEFAULT FALSE,
  enable_smtp               IN BOOLEAN DEFAULT FALSE,
  enable_anyp               IN BOOLEAN DEFAULT FALSE )
```

Parameters

Table 17–18 CREATE_AQ_AGENT Procedure Parameters

Parameter	Description
agent_name	Specifies the username of the Oracle Streams AQ Internet agent.
certification_location	Agent's certificate location in LDAP (default is NULL). If the agent is allowed to access Oracle Streams AQ through SMTP, then its certificate must be registered in LDAP. For access through HTTP, the certificate location is not required.
enable_http	TRUE means the agent can access Oracle Streams AQ through HTTP. FALSE means the agent cannot access Oracle Streams AQ through HTTP.
enable_smtp	TRUE means the agent can access Oracle Streams AQ through SMTP (e-mail). FALSE means the agent cannot access Oracle Streams AQ through SMTP.
enable_anyp	TRUE means the agent can access Oracle Streams AQ through any protocol (HTTP or SMTP).

Usage Notes

The SYS.AQ$INTERNET_USERS view has a list of all Oracle Streams AQ Internet agents.

CREATE_NP_QUEUE Procedure

> **Note:** nonpersistent queues are deprecated in Release 10gR2. Oracle recommends using buffered messaging.

This procedure creates a nonpersistent RAW queue.

Syntax

```
DBMS_AQADM.CREATE_NP_QUEUE (
   queue_name              IN        VARCHAR2,
   multiple_consumers      IN        BOOLEAN  DEFAULT FALSE,
   comment                 IN        VARCHAR2 DEFAULT NULL);
```

Parameters

Table 17–19 CREATE_NP_QUEUE Procedure Parameters

Parameter	Description
queue_name	Name of the nonpersistent queue that is to be created. The name must be unique within a schema and must follow object name guidelines in *Oracle Database SQL Reference.*
multiple_consumers	FALSE means queues created in the table can only have one consumer for each message. This is the default. TRUE means queues created in the table can have multiple consumers for each message. Note that this parameter is distinguished at the queue level, because a nonpersistent queue does not inherit this characteristic from any user-created queue table.
comment	User-specified description of the queue. This user comment is added to the queue catalog.

Usage Notes

The queue may be either single-consumer or multiconsumer queue. All queue names must be unique within a schema. The queues are created in a 8.1-compatible or higher system-created queue table (AQ$_MEM_SC or AQ$_MEM_MC) in the same schema as that specified by the queue name.

If the queue name does not specify a schema name, the queue is created in the login user's schema. After a queue is created with CREATE_NP_QUEUE, it can be enabled by calling START_QUEUE. By default, the queue is created with both enqueue and dequeue disabled.

You cannot dequeue from a nonpersistent queue. The only way to retrieve a message from a nonpersistent queue is by using the OCI notification mechanism. You cannot invoke the LISTEN call on a nonpersistent queue.

CREATE_QUEUE Procedure

This procedure creates a queue in the specified queue table.

Syntax

```
DBMS_AQADM.CREATE_QUEUE (
   queue_name          IN       VARCHAR2,
   queue_table         IN       VARCHAR2,
   queue_type          IN       BINARY_INTEGER DEFAULT NORMAL_QUEUE,
   max_retries         IN       NUMBER         DEFAULT NULL,
   retry_delay         IN       NUMBER         DEFAULT 0,
   retention_time      IN       NUMBER         DEFAULT 0,
   dependency_tracking IN       BOOLEAN        DEFAULT FALSE,
   comment             IN       VARCHAR2       DEFAULT NULL,
   auto_commit         IN       BOOLEAN        DEFAULT TRUE);
```

Parameters

Table 17–20 CREATE_QUEUE Procedure Parameters

Parameter	Description
queue_name	Name of the queue that is to be created. The name must be unique within a schema and must follow object name guidelines in *Oracle Database SQL Reference* with regard to reserved characters.
queue_table	Name of the queue table that will contain the queue.
queue_type	Specifies whether the queue being created is an exception queue or a normal queue. NORMAL_QUEUE means the queue is a normal queue. This is the default. EXCEPTION_QUEUE means it is an exception queue. Only the dequeue operation is allowed on the exception queue.
max_retries	Limits the number of times a dequeue with the REMOVE mode can be attempted on a message. The maximum value of max_retries is 2**31 -1. A message is moved to an exception queue if RETRY_COUNT is greater than MAX_RETRIES. RETRY_COUNT is incremented when the application issues a rollback after executing the dequeue. If a dequeue transaction fails because the server process dies (including ALTER SYSTEM KILL SESSION) or SHUTDOWN ABORT on the instance, then RETRY_COUNT is not incremented. Note that max_retries is supported for all single consumer queues and 8.1-compatible or higher multiconsumer queues but not for 8.0-compatible multiconsumer queues.
retry_delay	Delay time, in seconds, before this message is scheduled for processing again after an application rollback. The default is 0, which means the message can be retried as soon as possible. This parameter has no effect if max_retries is set to 0. Note that retry_delay is supported for single consumer queues and 8.1-compatible or higher multiconsumer queues but not for 8.0-compatible multiconsumer queues.
retention_time	Number of seconds for which a message is retained in the queue table after being dequeued from the queue. INFINITE means the message is retained forever. NUMBER is the number of seconds for which to retain the messages. The default is 0, no retention.

Table 17–20 (Cont.) CREATE_QUEUE Procedure Parameters

Parameter	Description
dependency_tracking	Reserved for future use. FALSE is the default. TRUE is not permitted in this release.
comment	User-specified description of the queue. This user comment is added to the queue catalog.
auto_commit	TRUE causes the current transaction, if any, to commit before the CREATE_QUEUE operation is carried out. The CREATE_QUEUE operation becomes persistent when the call returns. This is the default. FALSE means the operation is part of the current transaction and becomes persistent only when the caller enters a commit. Caution: This parameter has been deprecated.

Usage Notes

All queue names must be unique within a schema. After a queue is created with CREATE_QUEUE, it can be enabled by calling START_QUEUE. By default, the queue is created with both enqueue and dequeue disabled.

CREATE_QUEUE_TABLE Procedure

This procedure creates a queue table for messages of a predefined type.

Syntax

```
DBMS_AQADM.CREATE_QUEUE_TABLE (
   queue_table          IN      VARCHAR2,
   queue_payload_type   IN      VARCHAR2,
   [storage_clause      IN      VARCHAR2        DEFAULT NULL,]
   sort_list            IN      VARCHAR2        DEFAULT NULL,
   multiple_consumers   IN      BOOLEAN         DEFAULT FALSE,
   message_grouping     IN      BINARY_INTEGER  DEFAULT NONE,
   comment              IN      VARCHAR2        DEFAULT NULL,
   auto_commit          IN      BOOLEAN         DEFAULT TRUE,
   primary_instance     IN      BINARY_INTEGER  DEFAULT 0,
   secondary_instance   IN      BINARY_INTEGER  DEFAULT 0,
   compatible           IN      VARCHAR2        DEFAULT NULL,
   secure               IN      BOOLEAN         DEFAULT FALSE);
```

Parameters

Table 17–21 CREATE_QUEUE_TABLE Procedure Parameters

Parameter	Description
queue_table	Name of a queue table to be created
queue_payload_type	Type of the user data stored. See Type Name on page 16-4 for valid values for this parameter.
storage_clause	Storage parameter. The storage parameter is included in the CREATE TABLE statement when the queue table is created. The storage_clause argument can take any text that can be used in a standard CREATE TABLE storage_clause argument.The storage parameter can be made up of any combinations of the following parameters: PCTFREE, PCTUSED, INITRANS, MAXTRANS, TABLEPSACE, LOB, and a table storage clause. If a tablespace is not specified here, then the queue table and all its related objects are created in the default user tablespace. If a tablespace is specified here, then the queue table and all its related objects are created in the tablespace specified in the storage clause. See *Oracle Database SQL Reference* for the usage of these parameters.

Table 17–21 (Cont.) CREATE_QUEUE_TABLE Procedure Parameters

Parameter	Description
sort_list	The columns to be used as the sort key in ascending order. This parameter has the following format: '*sort_column_1*,*sort_column_2*' The allowed column names are priority and enq_time. If both columns are specified, then *sort_column_1* defines the most significant order. After a queue table is created with a specific ordering mechanism, all queues in the queue table inherit the same defaults. The order of a queue table cannot be altered after the queue table has been created. If no sort list is specified, then all the queues in this queue table are sorted by the enqueue time in ascending order. This order is equivalent to FIFO order. Even with the default ordering defined, a dequeuer is allowed to choose a message to dequeue by specifying its msgid or correlation. msgid, correlation, and sequence_deviation take precedence over the default dequeueing order, if they are specified.
multiple_consumers	FALSE means queues created in the table can only have one consumer for each message. This is the default. TRUE means queues created in the table can have multiple consumers for each message.
message_grouping	Message grouping behavior for queues created in the table. NONE means each message is treated individually. TRANSACTIONAL means messages enqueued as part of one transaction are considered part of the same group and can be dequeued as a group of related messages.
comment	User-specified description of the queue table. This user comment is added to the queue catalog.
auto_commit	TRUE causes the current transaction, if any, to commit before the CREATE_QUEUE_TABLE operation is carried out. The CREATE_QUEUE_TABLE operation becomes persistent when the call returns. This is the default. FALSE means the operation is part of the current transaction and becomes persistent only when the caller enters a commit. Note: This parameter has been deprecated.
primary_instance	The primary owner of the queue table. Queue monitor scheduling and propagation for the queues in the queue table are done in this instance. The default value for primary instance is 0, which means queue monitor scheduling and propagation will be done in any available instance.
secondary_instance	The queue table fails over to the secondary instance if the primary instance is not available. The default value is 0, which means that the queue table will fail over to any available instance.
compatible	The lowest database version with which the queue is compatible. Currently the possible values are either 8.0, 8.1, or 10.0. If the database is in 10.1-compatible mode, the default value is 10.0. If the database is in 8.1-compatible or 9.2-compatible mode, the default value is 8.1. If the database is in 8.0 compatible mode, the default value is 8.0.

Table 17–21 (Cont.) CREATE_QUEUE_TABLE Procedure Parameters

Parameter	Description
secure	This parameter must be set to TRUE if you want to use the queue table for secure queues. Secure queues are queues for which AQ agents must be associated explicitly with one or more database users who can perform queue operations, such as enqueue and dequeue. The owner of a secure queue can perform all queue operations on the queue, but other users cannot perform queue operations on a secure queue, unless they are configured as secure queue users.

Usage Notes

The sort keys for dequeue ordering, if any, must be defined at table creation time. The following objects are created at this time:

- aq$_*queue_table_name*_e, a default exception queue associated with the queue table
- aq$*queue_table_name*, a read-only view, which is used by Oracle Streams AQ applications for querying queue data
- aq$_*queue_table_name*_t, an index (or an index organized table (IOT) in the case of multiple consumer queues) for the queue monitor operations
- aq$_*queue_table_name*_i, an index (or an index organized table in the case of multiple consumer queues) for dequeue operations

For 8.1-compatible or higher queue tables, the following index-organized tables are created:

- aq$_*queue_table_name*_s, a table for storing information about the subscribers
- aq$_*queue_table_name*_r, a table for storing information about rules on subscriptions

aq$_*queue_table_name*_h, an index-organized table for storing the dequeue history data

CLOB, BLOB, and BFILE are valid attributes for Oracle Streams AQ object type payloads. However, only CLOB and BLOB can be propagated using Oracle Streams AQ propagation in Oracle8*i* release 8.1.5 or later. See the *Oracle Streams Advanced Queuing User's Guide and Reference* for more information.

The default value of the compatible parameter depends on the database compatibility mode in the init.ora. If the database is in 10.1-compatible mode, the default value is 10.0. If the database is in 8.1-compatible or 9.2-compatible mode, the default value is 8.1. If the database is in 8.0 compatible mode, the default value is 8.0

You can specify and modify the primary_instance and secondary_instance only in 8.1-compatible or higher mode. You cannot specify a secondary instance unless there is a primary instance.

DEL_ALIAS_FROM_LDAP Procedure

This procedure drops an alias for a queue, agent, or JMS ConnectionFactory in LDAP.

Syntax

```
DBMS_AQ.DEL_ALIAS_FROM_LDAP(
   alias IN VARCHAR2);
```

Parameters

Table 17–22 DEL_ALIAS_FROM_LDAP Procedure Parameters

Parameter	Description
alias	The alias to be removed.

DISABLE_DB_ACCESS Procedure

This procedure revokes the privileges of a specific database user from an Oracle Streams AQ Internet agent.

Syntax

```
DBMS_AQADM.DISABLE_DB_ACCESS (
  agent_name                IN VARCHAR2,
  db_username               IN VARCHAR2)
```

Parameters

Table 17–23 DISABLE_DB_ACCESS Procedure Parameters

Parameter	Description
agent_name	Specifies the username of the Oracle Streams AQ Internet agent.
db_username	Specifies the database user whose privileges are to be revoked from the Oracle Streams AQ Internet agent.

Usage Notes

The Oracle Streams AQ Internet agent should have been previously granted those privileges using the ENABLE_DB_ACCESS Procedure.

DISABLE_PROPAGATION_SCHEDULE Procedure

This procedure disables a propagation schedule.

Syntax

```
DBMS_AQADM.DISABLE_PROPAGATION_SCHEDULE (
   queue_name            IN   VARCHAR2,
   destination           IN   VARCHAR2 DEFAULT NULL,
   destination_queue  IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 17–24 DISABLE_PROPAGATION_SCHEDULE Procedure Parameters

Parameter	Description
queue_name	Name of the source queue whose messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the user.
destination	Destination database link. Messages in the source queue for recipients at this destination are propagated. If it is NULL, then the destination is the local database and messages are propagated to other queues in the local database. The length of this field is currently limited to 128 bytes, and if the name is not fully qualified, then the default domain name is used.
destination_queue	Name of the target queue to which messages are to be propagated in the form of a dblink

DROP_AQ_AGENT Procedure

This procedure drops an agent that was previously registered for Oracle Streams AQ Internet access.

Syntax

```
DBMS_AQADM.DROP_AQ_AGENT (
  agent_name                IN VARCHAR2)
```

Parameters

Table 17–25 DROP_AQ_AGENT Procedure Parameters

Parameter	Description
agent_name	Specifies the username of the Oracle Streams AQ Internet agent

DROP_QUEUE Procedure

This procedure drops an existing queue.

Syntax

```
DBMS_AQADM.DROP_QUEUE (
   queue_name        IN    VARCHAR2,
   auto_commit       IN    BOOLEAN DEFAULT TRUE);
```

Parameters

Table 17–26 DROP_QUEUE Procedure Parameters

Parameter	Description
queue_name	Name of the queue that is to be dropped.
auto_commit	TRUE causes the current transaction, if any, to commit before the DROP_QUEUE operation is carried out. The DROP_QUEUE operation becomes persistent when the call returns. This is the default. FALSE means the operation is part of the current transaction and becomes persistent only when the caller enters a commit. Caution: This parameter has been deprecated.

Usage Notes

DROP_QUEUE is not allowed unless STOP_QUEUE has been called to disable the queue for both enqueuing and dequeuing. All the queue data is deleted as part of the drop operation.

DROP_QUEUE_TABLE Procedure

This procedure drops an existing queue table.

Syntax

```
DBMS_AQADM.DROP_QUEUE_TABLE (
   queue_table       IN    VARCHAR2,
   force             IN    BOOLEAN DEFAULT FALSE,
   auto_commit       IN    BOOLEAN DEFAULT TRUE);
```

Parameters

Table 17–27 DROP_QUEUE_TABLE Procedure Parameters

Parameter	Description
queue_table	Name of a queue table to be dropped.
force	FALSE means the operation does not succeed if there are any queues in the table. This is the default. TRUE means all queues in the table are stopped and dropped automatically.
auto_commit	TRUE causes the current transaction, if any, to commit before the DROP_QUEUE_TABLE operation is carried out. The DROP_QUEUE_TABLE operation becomes persistent when the call returns. This is the default. FALSE means the operation is part of the current transaction and becomes persistent only when the caller enters a commit. Caution: This parameter has been deprecated.

Usage Notes

All the queues in a queue table must be stopped and dropped before the queue table can be dropped. You must do this explicitly unless the `force` option is used, in which case this is done automatically.

ENABLE_DB_ACCESS Procedure

This procedure grants an Oracle Streams AQ Internet agent the privileges of a specific database user.

Syntax

```
DBMS_AQADM.ENABLE_DB_ACCESS (
  agent_name                IN VARCHAR2,
  db_username               IN VARCHAR2)
```

Parameters

Table 17–28 ENABLE_DB_ACCESS Procedure Parameters

Parameter	Description
agent_name	Specifies the username of the Oracle Streams AQ Internet agent.
db_username	Specified the database user whose privileges are to be granted to the Oracle Streams AQ Internet agent.

Usage Notes

The Oracle Streams AQ Internet agent should have been previously created using the CREATE_AQ_AGENT Procedure.

For secure queues, the sender and receiver agent of the message must be mapped to the database user performing the enqueue or dequeue operation.

> **See Also:** *Oracle Streams Concepts and Administration* for information about secure queues

The SYS.AQ$INTERNET_USERS view has a list of all Oracle Streams AQ Internet agents and the names of the database users whose privileges are granted to them.

ENABLE_JMS_TYPES Procedure

Enqueue of JMS types and XML types does not work with Oracle Streams `Sys.Anydata` queues unless you call this procedure after `DBMS_STREAMS_ADM.SET_UP_QUEUE`. Enabling an Oracle Streams queue for these types may affect import/export of the queue table.

Syntax

```
DBMS_AQADM.ENABLE_JMS_TYPES (
   queue_table   IN   VARCHAR2);
```

Parameters

Table 17–29 ENABLE_JMS_TYPES Procedure Parameters

Parameter	Description
`queue_table`	Specifies name of the queue table to be enabled for JMS and XML types.

ENABLE_PROPAGATION_SCHEDULE Procedure

This procedure enables a previously disabled propagation schedule.

Syntax

```
DBMS_AQADM.ENABLE_PROPAGATION_SCHEDULE (
   queue_name          IN   VARCHAR2,
   destination         IN   VARCHAR2 DEFAULT NULL,
   destination_queue   IN   VARCHAR2  DEFAULT NULL);
```

Parameters

Table 17–30 ENABLE_PROPAGATION_SCHEDULE Procedure Parameters

Parameter	Description
`queue_name`	Name of the source queue whose messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the user.
`destination`	Destination database link. Messages in the source queue for recipients at this destination are propagated. If it is `NULL`, then the destination is the local database and messages are propagated to other queues in the local database. The length of this field is currently limited to 128 bytes, and if the name is not fully qualified, then the default domain name is used.
`destination_queue`	Name of the target queue to which messages are to be propagated in the form of a `dblink`

GET_WATERMARK Procedure

This procedure retrieves the value of watermark set by `SET_WATERMARK`.

Syntax

```
DBMS_AQADM.GET_WATERMARK (
   wmvalue     OUT      NUMBER);
```

Parameters

Table 17–31 GET_WATERMARK Procedure Parameter

Parameter	Description
`wmvalue`	Watermark value in megabytes.

GRANT_QUEUE_PRIVILEGE Procedure

This procedure grants privileges on a queue to users and roles. The privileges are ENQUEUE or DEQUEUE. Initially, only the queue table owner can use this procedure to grant privileges on the queues.

Syntax

```
DBMS_AQADM.GRANT_QUEUE_PRIVILEGE (
   privilege        IN    VARCHAR2,
   queue_name       IN    VARCHAR2,
   grantee          IN    VARCHAR2,
   grant_option     IN    BOOLEAN := FALSE);
```

Parameters

Table 17–32 GRANT_QUEUE_PRIVILEGE Procedure Parameters

Parameter	Description
privilege	The Oracle Streams AQ queue privilege to grant. The options are ENQUEUE, DEQUEUE, and ALL. ALL means both ENQUEUE and DEQUEUE.
queue_name	Name of the queue.
grantee	Grantee(s). The grantee(s) can be a user, a role, or the PUBLIC role.
grant_option	Specifies if the access privilege is granted with the GRANT option or not. If the privilege is granted with the GRANT option, then the grantee is allowed to use this procedure to grant the access privilege to other users or roles, regardless of the ownership of the queue table. The default is FALSE.

GRANT_SYSTEM_PRIVILEGE Procedure

This procedure grants Oracle Streams AQ system privileges to users and roles. The privileges are ENQUEUE_ANY, DEQUEUE_ANY, and MANAGE_ANY. Initially, only SYS and SYSTEM can use this procedure successfully.

Syntax

```
DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE (
   privilege         IN    VARCHAR2,
   grantee           IN    VARCHAR2,
   admin_option      IN    BOOLEAN := FALSE);
```

Parameters

Table 17–33 GRANT_SYSTEM_PRIVILEGE Procedure Parameters

Parameter	Description
privilege	The Oracle Streams AQ system privilege to grant. The options are ENQUEUE_ANY, DEQUEUE_ANY, and MANAGE_ANY. ENQUEUE_ANY means users granted this privilege are allowed to enqueue messages to any queues in the database. DEQUEUE_ANY means users granted this privilege are allowed to dequeue messages from any queues in the database. MANAGE_ANY means users granted this privilege are allowed to run DBMS_AQADM calls on any schemas in the database.
grantee	Grantee(s). The grantee(s) can be a user, a role, or the PUBLIC role.
admin_option	Specifies if the system privilege is granted with the ADMIN option or not. If the privilege is granted with the ADMIN option, then the grantee is allowed to use this procedure to grant the system privilege to other users or roles. The default is FALSE.

MIGRATE_QUEUE_TABLE Procedure

This procedure upgrades an 8.0-compatible queue table to an 8.1-compatible or higher queue table, or downgrades an 8.1-compatible or higher queue table to an 8.0-compatible queue table.

Syntax

```
DBMS_AQADM.MIGRATE_QUEUE_TABLE (
   queue_table   IN   VARCHAR2,
   compatible    IN   VARCHAR2);
```

Parameters

Table 17–34 MIGRATE_QUEUE_TABLE Procedure Parameters

Parameter	Description
`queue_table`	Specifies name of the queue table to be migrated.
`compatible`	Set this to `8.1` to upgrade an 8.0-compatible queue table, or set this to `8.0` to downgrade an 8.1-compatible queue table.

PURGE_QUEUE_TABLE Procedure

This procedure purges messages from queue tables. You can perform various purge operations on both single-consumer and multiconsumer queue tables for persistent and buffered messages.

Syntax

```
DBMS_AQADM.PURGE_QUEUE_TABLE(
   queue_table        IN   VARCHAR2,
   purge_condition    IN   VARCHAR2,
   purge_options      IN   aq$_purge_options_t);
```

where type aq$_purge_options_t is described in Chapter 184, "Oracle Streams AQ TYPEs".

Parameters

Table 17–35 PURGE_QUEUE_TABLE Procedure Parameters

Parameter	Description
queue_table	Specifies the name of the queue table to be purged.
purge_condition	Specifies the purge condition to use when purging the queue table. The purge condition must be in the format of a SQL WHERE clause, and it is case-sensitive. The condition is based on the columns of aq$queue_table_name view. When specifying the purge_condition, qualify the column names in aq$queue_table_name view with qtview. To purge all queues in a queue table, set purge_condition to either NULL (a bare null word, no quotes) or '' (two single quotes).
purge_options	Type aq$_purge_options_t contains a block parameter and a delivery_mode parameter. ■ If block is TRUE, then an exclusive lock on all the queues in the queue table is held while purging the queue table. This will cause concurrent enqueuers and dequeuers to block while the queue table is purged. The purge call always succeeds if block is TRUE. The default for block is FALSE. This will not block enqueuers and dequeuers, but it can cause the purge to fail with an error during high concurrency times. ■ delivery_mode is used to specify whether DBMS_AQADM.PERSISTENT, DBMS_AQADM.BUFFERED or DBMS_AQADM.PERSISTENT_OR_BUFFERED types of messages are to be purged. You cannot implement arbitrary purge conditions if buffered messages have to be purged.

Usage Notes

- You an purge selected messages from the queue table by specifying a purge_condition. Table 17–35 describes these parameters. Messages can be enqueued to and dequeued from the queue table while the queue table is being purged.

- A trace file is generated in the `udump` destination when you run this procedure. It details what the procedure is doing.
- This procedure commits batches of messages in autonomous transactions. Several such autonomous transactions may get executed as a part of one `purge_queue_table` call depending on the number of messages in the queue table.

QUEUE_SUBSCRIBERS Function

This function returns the subscribers to an 8.0-compatible multiconsumer queue in the PL/SQL index by table collection type `DBMS_AQADM.AQ$_subscriber_list_t`. Each element of the collection is of type `sys.aq$_agent`. This functionality is provided for 8.1-compatible queues by the `AQ$`*`queue_table_name`*`_S` view.

Syntax

```
DBMS_AQADM.QUEUE_SUBSCRIBERS (
   queue_name         IN         VARCHAR2);
RETURN aq$_subscriber_list_t IS
```

Parameters

Table 17–36 QUEUE_SUBSCRIBERS Function Parameters

Parameter	Description
`queue_name`	Specifies the queue whose subscribers are to be printed.

REMOVE_SUBSCRIBER Procedure

This procedure removes a default subscriber from a queue. This operation takes effect immediately, and the containing transaction is committed. All references to the subscriber in existing messages are removed as part of the operation.

Syntax

```
DBMS_AQADM.REMOVE_SUBSCRIBER (
   queue_name          IN          VARCHAR2,
   subscriber          IN          sys.aq$_agent);
```

Parameters

Table 17–37 REMOVE_SUBSCRIBER Procedure Parameters

Parameter	Description
queue_name	Name of the queue.
subscriber	Agent who is being removed. See AQ$_AGENT Type on page 184-3.

REVOKE_QUEUE_PRIVILEGE Procedure

This procedure revokes privileges on a queue from users and roles. The privileges are ENQUEUE or DEQUEUE.

Syntax

```
DBMS_AQADM.REVOKE_QUEUE_PRIVILEGE (
   privilege         IN      VARCHAR2,
   queue_name        IN      VARCHAR2,
   grantee           IN      VARCHAR2);
```

Parameters

Table 17–38 REVOKE_QUEUE_PRIVILEGE Procedure Parameters

Parameter	Description
privilege	The Oracle Streams AQ queue privilege to revoke. The options are ENQUEUE, DEQUEUE, and ALL. ALL means both ENQUEUE and DEQUEUE.
queue_name	Name of the queue.
grantee	Grantee(s). The grantee(s) can be a user, a role, or the PUBLIC role. If the privilege has been propagated by the grantee through the GRANT option, then the propagated privilege is also revoked.

Usage Notes

To revoke a privilege, the revoker must be the original grantor of the privilege. The privileges propagated through the GRANT option are revoked if the grantor's privileges are revoked.

REVOKE_SYSTEM_PRIVILEGE Procedure

This procedure revokes Oracle Streams AQ system privileges from users and roles. The privileges are ENQUEUE_ANY, DEQUEUE_ANY and MANAGE_ANY. The ADMIN option for a system privilege cannot be selectively revoked.

Syntax

```
DBMS_AQADM.REVOKE_SYSTEM_PRIVILEGE (
   privilege         IN   VARCHAR2,
   grantee           IN   VARCHAR2);
```

Parameters

Table 17–39 REVOKE_SYSTEM_PRIVILEGE Procedure Parameters

Parameter	Description
privilege	The Oracle Streams AQ system privilege to revoke. The options are ENQUEUE_ANY, DEQUEUE_ANY, and MANAGE_ANY. The ADMIN option for a system privilege cannot be selectively revoked.
grantee	Grantee(s). The grantee(s) can be a user, a role, or the PUBLIC role.

SCHEDULE_PROPAGATION Procedure

This procedure schedules propagation of messages from a queue to a destination identified by a specific database link.

Syntax

```
DBMS_AQADM.SCHEDULE_PROPAGATION (
   queue_name          IN    VARCHAR2,
   destination         IN    VARCHAR2 DEFAULT NULL,
   start_time          IN    DATE     DEFAULT SYSDATE,
   duration            IN    NUMBER   DEFAULT NULL,
   next_time           IN    VARCHAR2 DEFAULT NULL,
   latency             IN    NUMBER   DEFAULT 60,
   destination_queue   IN    VARCHAR2 DEFAULT NULL);
```

Parameters

Table 17–40 SCHEDULE_PROPAGATION Procedure Parameters

Parameter	Description
queue_name	Name of the source queue whose messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the administrative user.
destination	Destination database link. Messages in the source queue for recipients at this destination are propagated. If it is NULL, then the destination is the local database and messages are propagated to other queues in the local database. The length of this field is currently limited to 128 bytes, and if the name is not fully qualified, then the default domain name is used.
start_time	Initial start time for the propagation window for messages from the source queue to the destination.
duration	Duration of the propagation window in seconds. A NULL value means the propagation window is forever or until the propagation is unscheduled.
next_time	Date function to compute the start of the next propagation window from the end of the current window. If this value is NULL, then propagation is stopped at the end of the current window. For example, to start the window at the same time every day, next_time should be specified as SYSDATE + 1 - duration/86400.
latency	Maximum wait, in seconds, in the propagation window for a message to be propagated after it is enqueued. For example, if the latency is 60 seconds and there are no messages to be propagated during the propagation window, then messages from that queue for the destination are not propagated for at least 60 more seconds. It is at least 60 seconds before the queue is checked again for messages to be propagated for the specified destination. If the latency is 600, then the queue is not checked for 10 minutes, and if the latency is 0, then a job queue process will be waiting for messages to be enqueued for the destination. As soon as a message is enqueued, it is propagated.
destination_queue	Name of the target queue to which messages are to be propagated in the form of a dblink

Usage Notes

Messages may also be propagated to other queues in the same database by specifying a NULL destination. If a message has multiple recipients at the same destination in

either the same or different queues, the message is propagated to all of them at the same time.

SET_WATERMARK Procedure

This procedure is used for Oracle Streams AQ notification to specify and limit memory use.

Syntax

```
DBMS_AQADM.SET_WATERMARK (
   wmvalue     IN      NUMBER);
```

Parameters

Table 17–41 SET_WATERMARK Procedure Parameter

Parameter	Description
wmvalue	Watermark value in megabytes.

START_QUEUE Procedure

This procedure enables the specified queue for enqueuing or dequeuing.

Syntax

```
DBMS_AQADM.START_QUEUE (
   queue_name       IN     VARCHAR2,
   enqueue          IN     BOOLEAN DEFAULT TRUE,
   dequeue          IN     BOOLEAN DEFAULT TRUE);
```

Parameters

Table 17–42 START_QUEUE Procedure Parameters

Parameter	Description
queue_name	Name of the queue to be enabled
enqueue	Specifies whether ENQUEUE should be enabled on this queue. TRUE means enable ENQUEUE. This is the default. FALSE means do not alter the current setting.
dequeue	Specifies whether DEQUEUE should be enabled on this queue. TRUE means enable DEQUEUE. This is the default. FALSE means do not alter the current setting.

Usage Notes

After creating a queue, the administrator must use START_QUEUE to enable the queue. The default is to enable it for both ENQUEUE and DEQUEUE. Only dequeue operations are allowed on an exception queue. This operation takes effect when the call completes and does not have any transactional characteristics.

STOP_QUEUE Procedure

This procedure disables enqueuing or dequeuing on the specified queue.

Syntax

```
DBMS_AQADM.STOP_QUEUE (
   queue_name      IN   VARCHAR2,
   enqueue         IN   BOOLEAN DEFAULT TRUE,
   dequeue         IN   BOOLEAN DEFAULT TRUE,
   wait            IN   BOOLEAN DEFAULT TRUE);
```

Parameters

Table 17–43 STOP_QUEUE Procedure Parameters

Parameter	Description
queue_name	Name of the queue to be disabled
enqueue	Specifies whether ENQUEUE should be disabled on this queue. TRUE means disable ENQUEUE. This is the default. FALSE means do not alter the current setting.
dequeue	Specifies whether DEQUEUE should be disabled on this queue. TRUE means disable DEQUEUE. This is the default. FALSE means do not alter the current setting.
wait	Specifies whether to wait for the completion of outstanding transactions. TRUE means wait if there are any outstanding transactions. In this state no new transactions are allowed to enqueue to or dequeue from this queue. FALSE means return immediately either with a success or an error.

Usage Notes

By default, this call disables both ENQUEUE and DEQUEUE. A queue cannot be stopped if there are outstanding transactions against the queue. This operation takes effect when the call completes and does not have any transactional characteristics.

UNSCHEDULE_PROPAGATION Procedure

This procedure unschedules previously scheduled propagation of messages from a queue to a destination identified by a specific database link.

Syntax

```
DBMS_AQADM.UNSCHEDULE_PROPAGATION (
   queue_name         IN  VARCHAR2,
   destination        IN  VARCHAR2 DEFAULT NULL
   destination_queue  IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 17–44 UNSCHEDULE_PROPAGATION Procedure Parameters

Parameter	Description
queue_name	Name of the source queue whose messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the administrative user.
destination	Destination database link. Messages in the source queue for recipients at this destination are propagated. If it is NULL, then the destination is the local database and messages are propagated to other queues in the local database. The length of this field is currently limited to 128 bytes, and if the name is not fully qualified, then the default domain name is used.
destination_queue	Name of the target queue to which messages are to be propagated in the form of a dblink

VERIFY_QUEUE_TYPES Procedure

This procedure verifies that the source and destination queues have identical types. The result of the verification is stored in the table `sys.aq$_message_types`, overwriting all previous output of this command.

Syntax

```
DBMS_AQADM.VERIFY_QUEUE_TYPES (
   src_queue_name    IN    VARCHAR2,
   dest_queue_name   IN    VARCHAR2,
   destination       IN    VARCHAR2 DEFAULT NULL,
   rc                OUT   BINARY_INTEGER);
```

Parameters

Table 17–45 VERIFY_QUEUE_TYPES Procedure Parameters

Parameter	Description
`src_queue_name`	Name of the source queue whose messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the user.
`dest_queue_name`	Name of the destination queue where messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the user.
`destination`	Destination database link. Messages in the source queue for recipients at this destination are propagated. If it is `NULL`, then the destination is the local database and messages are propagated to other queues in the local database. The length of this field is currently limited to 128 bytes, and if the name is not fully qualified, then the default domain name is used.
`rc`	Return code for the result of the procedure. If there is no error, and if the source and destination queue types match, then the result is 1. If they do not match, then the result is 0. If an Oracle error is encountered, then it is returned in `rc`.

18

DBMS_AQELM

The DBMS_AQELM package provides subprograms to manage the configuration of Oracle Streams Advanced Queuing (AQ) asynchronous notification by e-mail and HTTP.

> **See Also:** *Oracle Streams Advanced Queuing User's Guide and Reference* for detailed information about DBMS_AQELM

This chapter contains the following topic:

Summary of DBMS_AQELM Subprograms

Table 18–1 DBMS_ALERT Package Subprograms

Subprogram	Description
SET_MAILHOST Procedure on page 18-3	Sets the host name for the SMTP server that the database will uses send out e-mail notifications
SET_MAILPORT Procedure on page 18-4	Sets the port number for the SMTP server
SET_SENDFROM Procedure on page 18-5	Sets the sent-from e-mail address

SET_MAILHOST Procedure

This procedure sets the host name for the SMTP server. The database uses this SMTP server host name to send out e-mail notifications.

Syntax

```
DBMS_AQELM.SET_MAILHOST (
   mailhost  IN  VARCHAR2);
```

Parameters

Table 18–2 SET_MAILHOST Procedure Parameters

Parameter	Description
mailhost	SMTP server host name.

Usage Notes

As part of the configuration for e-mail notifications, a user with AQ_ADMINISTRATOR_ROLE or with EXECUTE permissions on the DBMS_AQELM package needs to set the host name before registering for e-mail notifications.

SET_MAILPORT Procedure

This procedure sets the port number for the SMTP server.

Syntax

```
DBMS_AQELM.SET_MAILPORT (
   mailport  IN NUMBER);
```

Parameters

Table 18–3 SET_MAILPORT Procedure Parameters

Parameter	Description
mailport	SMTP server port number.

Usage Notes

As part of the configuration for e-mail notifications, a user with AQ_ADMINISTRATOR_ROLE or with EXECUTE permissions on DBMS_AQELM package needs to set the port number before registering for e-mail notifications. The database uses this SMTP server port number to send out e-mail notifications. If not set, the SMTP mailport defaults to 25

SET_SENDFROM Procedure

This procedure sets the sent-from e-mail address. This e-mail address is used in the sent-from field in all the e-mail notifications sent out by the database to the registered e-mail addresses.

Syntax

```
DBMS_AQELM.SET_SENDFROM (
   sendfrom  IN  VARCHAR2);
```

Parameters

Table 18–4 SET_SENDFROM Procedure Parameters

Parameter	Description
sendfrom	The sent-from e-mail address.

Usage Notes

As part of the configuration for e-mail notifications, a user with AQ_ADMINISTRATOR_ROLE or with EXECUTE permissions on the DBMS_AQELM package should set the sent-from address before registering for e-mail notifications

19

DBMS_AQIN

The DBMS_AQIN package plays a part in providing secure access to the Oracle JMS interfaces.

> **See Also:** *Oracle Streams Advanced Queuing User's Guide and Reference* for detailed information about DBMS_AQIN

This chapter contains the following topic:

- Using DBMS_AQIN
  - Over view

Using DBMS_AQIN

This section contains topics which relate to using the `DBMS_AQIN` package.

Overview

While you should not call any subprograms in the DBMS_AQIN package directly, you must have the EXECUTE privilege on the DBMS_AQIN and DBMS_AQJMS packages to use the Oracle JMS interfaces. Use the following syntax to accomplish this with regard to the DBMS_AQIN package:

```
GRANT EXECUTE ON DBMS_AQIN to user;
```

Note that you can also acquire these rights through the AQ_USER_ROLE or the AQ_ADMINSTRATOR_ROLE.

20

DBMS_CAPTURE_ADM

The DBMS_CAPTURE_ADM package, one of a set of Streams packages, provides subprograms for starting, stopping, and configuring a capture process. The source of the captured changes is the redo logs, and the repository for the captured changes is a queue.

> **See Also:** *Oracle Streams Concepts and Administration* and *Oracle Streams Replication Administrator's Guide* for more information about this package and capture processes

This chapter contains the following topic:

Summary of DBMS_CAPTURE_ADM Subprograms

Table 20–1 DBMS_CAPTURE_ADM Package Subprograms

Subprogram	Description
ABORT_GLOBAL_INSTANTIATION Procedure on page 20-3	Reverses the effects of running the PREPARE_GLOBAL_INSTANTIATION, PREPARE_SCHEMA_INSTANTIATION, and PREPARE_TABLE_INSTANTIATION procedures
ABORT_SCHEMA_INSTANTIATION Procedure on page 20-4	Reverses the effects of running the PREPARE_SCHEMA_INSTANTIATION and PREPARE_TABLE_INSTANTIATION procedures
ABORT_TABLE_INSTANTIATION Procedure on page 20-5	Reverses the effects of running the PREPARE_TABLE_INSTANTIATION procedure
ALTER_CAPTURE Procedure on page 20-6	Alters a capture process
BUILD Procedure on page 20-11	Extracts the data dictionary of the current database to the redo logs and automatically specifies database supplemental logging for all primary key and unique key columns
CREATE_CAPTURE Procedure on page 20-12	Creates a capture process
DROP_CAPTURE Procedure on page 20-17	Drops a capture process
INCLUDE_EXTRA_ATTRIBUTE Procedure on page 20-19	Includes or excludes an extra attribute in logical change records (LCRs) captured by the specified capture process
PREPARE_GLOBAL_INSTANTIATION Procedure on page 20-21	Performs the synchronization necessary for instantiating all the tables in the database at another database and can enable supplemental logging for key columns or all columns in these tables
PREPARE_SCHEMA_INSTANTIATION Procedure on page 20-22	Performs the synchronization necessary for instantiating all tables in the schema at another database and can enable supplemental logging for key columns or all columns in these tables
PREPARE_TABLE_INSTANTIATION Procedure on page 20-23	Performs the synchronization necessary for instantiating the table at another database and can enable supplemental logging for key columns or all columns in the table
SET_PARAMETER Procedure on page 20-24	Sets a capture process parameter to the specified value
START_CAPTURE Procedure on page 20-26	Starts the capture process, which mines redo logs and enqueues the mined redo information into the associated queue
STOP_CAPTURE Procedure on page 20-27	Stops the capture process from mining redo logs

Note: All subprograms commit unless specified otherwise.

ABORT_GLOBAL_INSTANTIATION Procedure

This procedure reverses the effects of running the PREPARE_GLOBAL_INSTANTIATION, PREPARE_SCHEMA_INSTANTIATION, and PREPARE_TABLE_INSTANTIATION procedures.

Specifically, this procedure performs the following actions:

- Removes data dictionary information related to the database, schema, and table instantiations
- Removes any supplemental logging enabled by the PREPARE_GLOBAL_INSTANTIATION, PREPARE_SCHEMA_INSTANTIATION, and PREPARE_TABLE_INSTANTIATION procedures

Syntax

```
DBMS_CAPTURE_ADM.ABORT_GLOBAL_INSTANTIATION();
```

ABORT_SCHEMA_INSTANTIATION Procedure

This procedure reverses the effects of running the PREPARE_SCHEMA_INSTANTIATION procedure. It also reverses the effects of running the PREPARE_TABLE_INSTANTIATION procedure on tables in the specified schema.

Specifically, this procedure performs the following actions:

- Removes data dictionary information related to schema instantiations and table instantiations of tables in the schema
- Removes any supplemental logging enabled by the PREPARE_SCHEMA_INSTANTIATION procedure
- Removes any supplemental logging enabled by the PREPARE_TABLE_INSTANTIATION procedure for tables in the specified schema

Syntax

```
DBMS_CAPTURE_ADM.ABORT_SCHEMA_INSTANTIATION(
   schema_name  IN  VARCHAR2);
```

Parameter

Table 20–2 ABORT_SCHEMA_INSTANTIATION Procedure Parameter

Parameter	Description
schema_name	The name of the schema for which to abort the effects of preparing instantiation

ABORT_TABLE_INSTANTIATION Procedure

This procedure reverses the effects of running the PREPARE_TABLE_INSTANTIATION procedure.

Specifically, this procedure performs the following actions:

- Removes data dictionary information related to the table instantiation
- Removes any supplemental logging enabled by the PREPARE_TABLE_INSTANTIATION procedure

Syntax

```
DBMS_CAPTURE_ADM.ABORT_TABLE_INSTANTIATION(
   table_name  IN  VARCHAR2);
```

Parameter

Table 20–3 ABORT_TABLE_INSTANTIATION Procedure Parameter

Parameter	Description
table_name	The name of the table for which to abort the effects of preparing instantiation, specified as [*schema_name*.]*object_name*. For example, hr.employees. If the schema is not specified, then the current user is the default.

ALTER_CAPTURE Procedure

This procedure alters a capture process.

> **See Also:** *Oracle Streams Concepts and Administration* for more information about altering a capture process

Syntax

```
DBMS_CAPTURE_ADM.ALTER_CAPTURE(
   capture_name              IN  VARCHAR2,
   rule_set_name             IN  VARCHAR2  DEFAULT NULL,
   remove_rule_set           IN  BOOLEAN   DEFAULT FALSE,
   start_scn                 IN  NUMBER    DEFAULT NULL,
   use_database_link         IN  BOOLEAN   DEFAULT NULL,
   first_scn                 IN  NUMBER    DEFAULT NULL,
   negative_rule_set_name    IN  VARCHAR2  DEFAULT NULL,
   remove_negative_rule_set  IN  BOOLEAN   DEFAULT FALSE,
   capture_user              IN  VARCHAR2  DEFAULT NULL,
   checkpoint_retention_time IN  NUMBER    DEFAULT NULL);
```

Parameters

Table 20–4 ALTER_CAPTURE Procedure Parameters

Parameter	Description
`capture_name`	The name of the capture process being altered. You must specify an existing capture process name. Do not specify an owner.
`rule_set_name`	The name of the positive rule set for the capture process. The positive rule set contains the rules that instruct the capture process to capture changes. If you want to use a positive rule set for the capture process, then you must specify an existing rule set in the form `[`*`schema_name`*`.]`*`rule_set_name`*. For example, to specify a positive rule set in the `HR` schema named `job_capture_rules`, enter `hr.job_capture_rules`. If the schema is not specified, then the current user is the default. An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the `DBMS_STREAMS_ADM` package or the `DBMS_RULE_ADM` package. If you specify `NULL` and the `remove_rule_set` parameter is set to `FALSE`, then the procedure retains any existing positive rule set. If you specify `NULL` and the `remove_rule_set` parameter is set to `TRUE`, then the procedure removes any existing positive rule set. See Also**:** *Oracle Streams Concepts and Administration* for more information about the changes that can be captured by a capture process

Table 20–4 (Cont.) ALTER_CAPTURE Procedure Parameters

Parameter	Description
remove_rule_set	If TRUE, then the procedure removes the positive rule set for the specified capture process. If you remove a positive rule set for a capture process, and the capture process does not have a negative rule set, then the capture process captures all supported changes to all objects in the database, excluding database objects in the SYS and SYSTEM schemas. If you remove a positive rule set for a capture process, and the capture process has a negative rule set, then the capture process captures all supported changes that are not discarded by the negative rule set. If FALSE, then the procedure retains the positive rule set for the specified capture process. If the rule_set_name parameter is non-NULL, then this parameter should be set to FALSE.
start_scn	A valid SCN for the database from which the capture process should start capturing changes. The SCN value specified must be greater than or equal to the first SCN for the capture process. An error is returned if an invalid SCN is specified.
use_database_link	If TRUE, then the capture process at a downstream database uses a database link to the source database for administrative purposes relating to the capture process. If you want a capture process that is not using a database link currently to begin using a database link, then specify TRUE. In this case, a database link with the same name as the global name of the source database must exist at the downstream database. If FALSE, then either the capture process is running on the source database, or the capture process at a downstream database does not use a database link to the source database. If you want a capture process that is using a database link currently to stop using a database link, then specify FALSE. In this case, you must prepare source database objects for instantiation manually when you add or change capture process rules that pertain to these objects. If NULL, then the current value of this parameter for the capture process is not changed.

Table 20–4 (Cont.) ALTER_CAPTURE Procedure Parameters

Parameter	Description
`first_scn`	Specifies the lowest SCN in the redo log from which a capture process can capture changes. If you specify a new first SCN for the capture process, then the specified first SCN must meet the following requirements: ■ It must be greater than the current first SCN for the capture process. ■ It must be less than or equal to the current applied SCN for the capture process. However, this requirement does not apply if the current applied SCN for the capture process is zero. ■ It must be less than or equal to the required checkpoint SCN for the capture process. An error is returned if the specified SCN does not meet the first three requirements. See "Usage Notes" on page 20-10 for information about determining an SCN value that meets all of these conditions. When the first SCN is modified, the capture process purges information from its LogMiner data dictionary that is required to restart it at an earlier SCN. Also, if the specified first SCN is higher than the current start SCN for the capture process, then the start SCN is set automatically to the new value of the first SCN.
`negative_rule_set_name`	The name of the negative rule set for the capture process. The negative rule set contains the rules that instruct the capture process to discard changes. If you want to use a negative rule set for the capture process, then you must specify an existing rule set in the form `[`*`schema_name`*`.]`*`rule_set_name`*. For example, to specify a negative rule set in the `hr` schema named `neg_capture_rules`, enter `hr.neg_capture_rules`. If the schema is not specified, then the current user is the default. An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the `DBMS_STREAMS_ADM` package or the `DBMS_RULE_ADM` package. If you specify `NULL` and the `remove_negative_rule_set` parameter is set to `FALSE`, then the procedure retains any existing negative rule set. If you specify `NULL` and the `remove_negative_rule_set` parameter is set to `TRUE`, then the procedure removes any existing negative rule set. If you specify both a positive and a negative rule set for a capture process, then the negative rule set is always evaluated first.

Table 20–4 (Cont.) ALTER_CAPTURE Procedure Parameters

Parameter	Description
remove_negative_rule_set	If TRUE, then the procedure removes the negative rule set for the specified capture process. If you remove a negative rule set for a capture process, and the capture process does not have a positive rule set, then the capture process captures all supported changes to all objects in the database, excluding database objects in the SYS and SYSTEM schemas. If you remove a negative rule set for a capture process, and a positive rule set exists for the capture process, then the capture process captures all changes that are not discarded by the positive rule set. If FALSE, then the procedure retains the negative rule set for the specified capture process. If the negative_rule_set_name parameter is non-NULL, then this parameter should be set to FALSE.
capture_user	The user in whose security domain a capture process captures changes that satisfy its rule sets and runs custom rule-based transformations configured for capture process rules. If NULL, then the capture user is not changed. To change the capture user, the user who invokes the ALTER_CAPTURE procedure must be granted DBA role. Only the SYS user can set the capture_user to SYS. If you change the capture user, then this procedure grants the new capture user enqueue privilege on the queue used by the capture process and configures the user as a secure queue user of the queue. In addition, make sure the capture user has the following privileges: ■ EXECUTE privilege on the rule sets used by the capture process ■ EXECUTE privilege on all rule-based transformation functions used in the rule set ■ EXECUTE privilege on all packages, including Oracle-supplied packages, that are invoked in rule-based transformations run by the capture process These privileges must be granted directly to the capture user. They cannot be granted through roles. By default, this parameter is set to the user who created the capture process by running either the CREATE_CAPTURE procedure in this package or one of the following procedures in the DBMS_STREAMS_ADM package with the streams_type parameter set to capture: ADD_GLOBAL_RULES ADD_SCHEMA_RULES ADD_TABLE_RULES ADD_SUBSET_RULES **Note:** If the specified user is dropped using DROP USER... CASCADE, then the capture_user setting for the capture process is set to NULL automatically. You must specify a capture user before the capture process can run.

Table 20–4 (Cont.) ALTER_CAPTURE Procedure Parameters

Parameter	Description
checkpoint_retention_time	Either the number of days that a capture process should retain checkpoints before purging them automatically, or DBMS_CAPTURE_ADM.INFINITE if checkpoints should not be purged automatically. If NULL, then the checkpoint retention time is not changed. If a number is specified, then a capture process purges a checkpoint the specified number of days after the checkpoint was taken. Partial days can be specified using decimal values. For example, .25 specifies 6 hours. When a checkpoint is purged, LogMiner data dictionary information for the archived redo log file that corresponds to the checkpoint is purged, and the first_scn of the capture process is reset to the SCN value corresponding to the first change in the next archived redo log file. **See Also:** *Oracle Streams Concepts and Administration* for more information about checkpoint retention time

Usage Notes

If you want to alter the first SCN for a capture process, then the value specified must meet the conditions in the description for the first_scn parameter. The following query determines the current first SCN, applied SCN, and required checkpoint SCN for each capture process in a database:

```
SELECT CAPTURE_NAME, FIRST_SCN, APPLIED_SCN, REQUIRED_CHECKPOINT_SCN
   FROM DBA_CAPTURE;
```

Also, a capture process is stopped and restarted automatically when you change the value of one or more of the following ALTER_CAPTURE procedure parameters:

- start_scn
- capture_user

BUILD Procedure

This procedure extracts the data dictionary of the current database to the redo log and automatically specifies database supplemental logging by running the following SQL statement:

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;
```

This procedure is overloaded. One version of this procedure contains the `OUT` parameter `first_scn`, and the other does not.

Syntax

```
DBMS_CAPTURE_ADM.BUILD(
   first_scn  OUT  NUMBER);

DBMS_CAPTURE_ADM.BUILD();
```

Parameters

Table 20–5 BUILD Procedure Parameter

Parameter	Description
`first_scn`	Contains the lowest SCN value corresponding to the data dictionary extracted to the redo log that can be specified as a first SCN for a capture process

Usage Notes

You can run this procedure multiple times at a source database.

If you plan to capture changes originating at a source database with a capture process, then this procedure must be executed at the source database at least once. When the capture process is started, either at a local source database or at a downstream database, the capture process uses the extracted information in the redo log to create a LogMiner data dictionary.

After executing this procedure, you can query the `FIRST_CHANGE#` column of the `V$ARCHIVED_LOG` dynamic performance view where the `DICTIONARY_BEGIN` column is `YES` to determine the lowest SCN value for the database that can be specified as a first SCN for a capture process. The first SCN for a capture process is the lowest SCN in the redo log from which the capture process can capture changes.You can specify the first SCN for a capture process when you run the `CREATE_CAPTURE` or `ALTER_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package.

CREATE_CAPTURE Procedure

This procedure creates a capture process.

See Also:

- *Oracle Streams Concepts and Administration* for more information about creating a capture process
- Chapter 92, "DBMS_RULE_ADM" for more information about rules and rule sets

Syntax

```
DBMS_CAPTURE_ADM.CREATE_CAPTURE(
   queue_name                 IN  VARCHAR2,
   capture_name               IN  VARCHAR2,
   rule_set_name              IN  VARCHAR2  DEFAULT NULL,
   start_scn                  IN  NUMBER    DEFAULT NULL,
   source_database            IN  VARCHAR2  DEFAULT NULL,
   use_database_link          IN  BOOLEAN   DEFAULT FALSE,
   first_scn                  IN  NUMBER    DEFAULT NULL,
   logfile_assignment         IN  VARCHAR2  DEFAULT 'implicit',
   negative_rule_set_name     IN  VARCHAR2  DEFAULT NULL,
   capture_user               IN  VARCHAR2  DEFAULT NULL,
   checkpoint_retention_time  IN  NUMBER    DEFAULT 60);
```

Parameters

Table 20–6 CREATE_CAPTURE Procedure Parameters

Parameter	Description
queue_name	The name of the queue into which the capture process enqueues changes. You must specify an existing queue in the form [*schema_name*.]*queue_name*. For example, to specify a queue in the hr schema named streams_queue, enter hr.streams_queue. If the schema is not specified, then the current user is the default. **Note:** The queue_name setting cannot be altered after the capture process is created.
capture_name	The name of the capture process being created. A NULL specification is not allowed. Do not specify an owner. **Note:** The capture_name setting cannot be altered after the capture process is created.

Table 20–6 (Cont.) CREATE_CAPTURE Procedure Parameters

Parameter	Description
rule_set_name	The name of the positive rule set for the capture process. The positive rule set contains the rules that instruct the capture process to capture changes. If you want to use a positive rule set for the capture process, then you must specify an existing rule set in the form [*schema_name*.]*rule_set_name*. For example, to specify a positive rule set in the hr schema named job_capture_rules, enter hr.job_capture_rules. If the schema is not specified, then the current user is the default. An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the DBMS_STREAMS_ADM package or the DBMS_RULE_ADM package. If you specify NULL, and no negative rule set is specified, then the capture process captures all supported changes to all objects in the database, excluding database objects in the SYS and SYSTEM schemas. If you specify NULL, and a negative rule set exists for the capture process, then the capture process captures all changes that are not discarded by the negative rule set. **See Also:** *Oracle Streams Concepts and Administration* for more information about the changes that can be captured by a capture process
start_scn	A valid SCN for the database from which the capture process should start capturing changes. If the specified value is lower than the current SCN of the source database, then either the first_scn should be specified, or the SCN value specified for start_scn must be greater than or equal to the first SCN of an existing capture process which has taken at least one checkpoint. If start_scn is NULL and no value is specified for the first_scn parameter, then the current SCN of the database is used as start SCN. If start_scn is NULL and a non-NULL value is specified for the first_scn parameter, then the first_scn value is used. If a value is specified for both start_scn and first_scn, then the start_scn value must be greater than or equal to the first_scn value. An error is returned if an invalid SCN is specified.
source_database	The global name of the source database. The source database is where the changes to be captured originated. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically. If NULL, or if the specified name is the same as the global name of the current database, then local capture is assumed and only the default values for use_database_link and first_scn can be specified.

Table 20–6 (Cont.) CREATE_CAPTURE Procedure Parameters

Parameter	Description
use_database_link	If TRUE, then the capture process at a downstream database uses a database link to the source database for administrative purposes relating to the capture process. The capture process uses the database link to prepare database objects for instantiation at the source database and run the DBMS_CAPTURE_ADM.BUILD procedure at the source database, if necessary. If FALSE, then either the capture process is running on the source database, or the capture process at a downstream database does not use a database link to the source database. In this case, you must perform the following administrative tasks manually: ■ Run the DBMS_CAPTURE_ADM.BUILD procedure at the source database to extract the data dictionary at the source database to the redo log when a capture process is created. ■ Obtain the first SCN for the downstream capture process if the first SCN is not specified during capture process creation. The first SCN is needed to create and maintain a capture process. ■ Prepare source database objects for instantiation.
first_scn	Specifies the lowest SCN in the redo log from which a capture process can capture changes. A non-NULL value for this parameter is valid only if the DBMS_CAPTURE_ADM.BUILD procedure has been run at least once at the source database. You can query the FIRST_CHANGE# column of the V$ARCHIVED_LOG dynamic performance view where the DICTIONARY_BEGIN column is YES to determine whether the DBMS_CAPTURE_ADM.BUILD procedure has been run on a source database. Any of the values returned by such a query can be used as a first_scn value if the redo log containing that SCN value is still available.
logfile_assignment	If implicit, the default, then the capture process at a downstream database scans all redo log files added by redo transport services or manually from the source database to the downstream database. If explicit, then a redo log file is scanned by a capture process at a downstream database only if the capture process name is specified in the FOR *logminer_session_name* clause when the redo log file is added manually to the downstream database. If explicit, then redo transport services cannot be used to add redo log files to the capture process being created. If you specify explicit for this parameter for a local capture process, then the local capture process cannot use the online redo log to find changes. In this case, the capture process must use the archived redo log. See Also: "Usage Notes" on page 20-16 for information about adding redo log files manually

Table 20–6 (Cont.) CREATE_CAPTURE Procedure Parameters

Parameter	Description
negative_rule_set_name	The name of the negative rule set for the capture process. The negative rule set contains the rules that instruct the capture process to discard changes. If you want to use a negative rule set for the capture process, then you must specify an existing rule set in the form [*schema_name*.]*rule_set_name*. For example, to specify a negative rule set in the hr schema named neg_capture_rules, enter hr.neg_capture_rules. If the schema is not specified, then the current user is the default. If you specify NULL, and no positive rule set is specified, then the capture process captures all supported changes to all objects in the database, excluding database objects in the SYS and SYSTEM schemas. If you specify NULL, and a positive rule set exists for the capture process, then the capture process captures all changes that are not discarded by the positive rule set. An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the DBMS_STREAMS_ADM package or the DBMS_RULE_ADM package. If you specify both a positive and a negative rule set for a capture process, then the negative rule set is always evaluated first.
capture_user	The user in whose security domain a capture process captures changes that satisfy its rule sets and runs custom rule-based transformations configured for capture process rules. If NULL, then the user who runs the CREATE_CAPTURE procedure is used. Only a user who is granted DBA role can set a capture user. Only the SYS user can set the capture_user to SYS. **Note:** If the specified user is dropped using DROP USER... CASCADE, then the capture_user setting for the capture process is set to NULL automatically. You must specify a capture user before the capture process can run.
checkpoint_retention_time	Either the number of days that a capture process should retain checkpoints before purging them automatically, or DBMS_CAPTURE_ADM.INFINITE if checkpoints should not be purged automatically. If a number is specified, then a capture process purges a checkpoint the specified number of days after the checkpoint was taken. Partial days can be specified using decimal values. For example, .25 specifies 6 hours. When a checkpoint is purged, LogMiner data dictionary information for the archived redo log file that corresponds to the checkpoint is purged, and the first_scn of the capture process is reset to the SCN value corresponding to the first change in the next archived redo log file. **See Also:** *Oracle Streams Concepts and Administration* for more information about checkpoint retention time

Usage Notes

The user who invokes this procedure must be granted DBA role.

The capture_user parameter specifies the user who captures changes that satisfy the capture process rule sets. This user must have the necessary privileges to capture changes. This procedure grants the capture user enqueue privilege on the queue used by the capture process and configures the user as a secure queue user of the queue.

In addition, make sure the capture user has the following privileges:

- EXECUTE privilege on the rule sets used by the capture process
- EXECUTE privilege on all rule-based transformation functions used in the rule set
- EXECUTE privilege on all packages, including Oracle-supplied packages, that are invoked in rule-based transformations run by the capture process

These privileges must be granted directly to the capture user. They cannot be granted through roles.

Note:

- A capture user does not require privileges on a database object to capture changes to the database object. The capture process can pass these changes to a rule-based transformation function. Therefore, make sure you consider security implications when you configure a capture process.
- Creation of the first capture process in a database might take some time because the data dictionary is duplicated during this creation.

If you specify explicit for the logfile_assignment parameter, then you add a redo log file manually to a downstream database using the following statement:

```
ALTER DATABASE REGISTER LOGICAL LOGFILE
   file_name FOR capture_process;
```

Here, *file_name* is the name of the redo log file being added and *capture_process* is the name of the capture process that will use the redo log file at the downstream database. The *capture_process* is equivalent to the *logminer_session_name* and must be specified. The redo log file must be present at the site running the downstream database. You must transfer this file manually to the site running the downstream database using the DBMS_FILE_TRANSFER package, FTP, or some other transfer method.

See Also: *Oracle Database SQL Reference* for more information about the ALTER DATABASE statement and *Oracle Data Guard Concepts and Administration* for more information registering redo log files

DROP_CAPTURE Procedure

This procedure drops a capture process.

Note: A capture process must be stopped before it can be dropped.

See Also: "STOP_CAPTURE Procedure" on page 20-27

Syntax

```
DBMS_CAPTURE_ADM.DROP_CAPTURE(
   capture_name           IN  VARCHAR2,
   drop_unused_rule_sets  IN  BOOLEAN  DEFAULT FALSE);
```

Parameters

Table 20–7 DROP_CAPTURE Procedure Parameters

Parameter	Description
capture_name	The name of the capture process being dropped. Specify an existing capture process name. Do not specify an owner.
drop_unused_rule_sets	If TRUE, then the procedure drops any rule sets, positive and negative, used by the specified capture process if these rule sets are not used by any other Streams client. Streams clients include capture processes, propagations, apply processes, and messaging clients. If this procedure drops a rule set, then this procedure also drops any rules in the rule set that are not in another rule set. If FALSE, then the procedure does not drop the rule sets used by the specified capture process, and the rule sets retain their rules.

Usage Notes

When you use this procedure to drop a capture process, rules-related information for the capture process created by the DBMS_STREAMS_ADM package is removed from the data dictionary views for Streams rules. Information about such a rule is removed even if the rule is not in either rule set for the capture process.

The following are the data dictionary views for Streams rules:

- ALL_STREAMS_GLOBAL_RULES
- DBA_STREAMS_GLOBAL_RULES
- ALL_STREAMS_MESSAGE_RULES
- DBA_STREAMS_MESSAGE_RULES
- ALL_STREAMS_SCHEMA_RULES
- DBA_STREAMS_SCHEMA_RULES
- ALL_STREAMS_TABLE_RULES
- DBA_STREAMS_TABLE_RULES
- ALL_STREAMS_RULES

- DBA_STREAMS_RULES

See Also: *Oracle Streams Concepts and Administration* for more information about Streams data dictionary views

INCLUDE_EXTRA_ATTRIBUTE Procedure

This procedure includes or excludes an extra attribute in logical change records (LCRs) captured by the specified capture process.

Syntax

```
DBMS_CAPTURE_ADM.INCLUDE_EXTRA_ATTRIBUTE(
   capture_name    IN  VARCHAR2,
   attribute_name  IN  VARCHAR2,
   include         IN  BOOLEAN   DEFAULT TRUE);
```

Parameters

Table 20–8 INCLUDE_EXTRA_ATTRIBUTE Procedure Parameters

Parameter	Description
capture_name	The name of the capture process. Specify an existing capture process name. Do not specify an owner.
attribute_name	The name of the attribute to be included in or excluded from LCRs captured by the capture process. The following names are valid settings: ■ row_id The rowid of the row changed in a row LCR. This attribute is not included in DDL LCRs, or in row LCRs for index-organized tables. The type is VARCHAR2. ■ serial# The serial number of the session that performed the change captured in the LCR. The type is NUMBER. ■ session# The identifier of the session that performed the change captured in the LCR. The type is NUMBER. ■ thread# The thread number of the instance in which the change captured in the LCR was performed. Typically, the thread number is relevant only in a Real Application Clusters environment. The type is NUMBER. ■ tx_name The name of the transaction that includes the LCR. The type is VARCHAR2. ■ username The name of the user who performed the change captured in the LCR. The type is VARCHAR2.
include	If TRUE, then the specified attribute is included in LCRs captured by the capture process If FALSE, then the specified attribute is excluded from LCRs captured by the capture process

Usage Notes

The redo log contains information about each change made to a database, and some of this information is not captured by a capture process unless you use this procedure to instruct a capture process to capture it. This procedure enables you to specify extra

information in the redo log that a capture process should capture. If you want to exclude an extra attribute that is being captured by a capture process, then specify the attribute and specify `FALSE` for the `include` parameter.

PREPARE_GLOBAL_INSTANTIATION Procedure

This procedure performs the synchronization necessary for instantiating all the tables in the database at another database and can enable supplemental logging for key columns or all columns in these tables.

This procedure records the lowest SCN of each object in the database for instantiation. SCNs subsequent to the lowest SCN for an object can be used for instantiating the object. Running this procedure prepares all current and future objects in the database for instantiation.

> **See Also:** *Oracle Streams Replication Administrator's Guide* for more information about instantiation and supplemental logging

Syntax

```
DBMS_CAPTURE_ADM.PREPARE_GLOBAL_INSTANTIATION
   supplemental_logging  IN  VARCHAR2  DEFAULT 'keys');
```

Parameter

Table 20–9 PREPARE_GLOBAL_INSTANTIATION Procedure Parameter

Parameter	Description
`supplemental_logging`	Either `none`, `keys`, or `all`. If `none` is specified, then this procedure does not enable supplemental logging for any columns in the tables in the database. This procedure does not remove existing supplemental logging specifications for these tables. If `keys` is specified, then this procedure enables supplemental logging for primary key, unique key, bitmap index, and foreign key columns in the tables in the database and for any table added to the database in the future. Primary key columns are logged unconditionally. Unique key, bitmap index, and foreign key columns are logged conditionally. Specifying `keys` does not enable supplemental logging of bitmap join index columns. If `all` is specified, then this procedure enables supplemental logging for all columns in the tables in the database and for any table added to the database in the future. The columns are logged unconditionally. Supplemental logging is not enabled for columns of the following types: LOB, `LONG`, `LONG RAW`, and user-defined type.

Usage Notes

Run this procedure at the source database.

If you use a capture process to capture all of the changes to a database, then use this procedure to prepare the tables in the database for instantiation after the capture process has been configured.

PREPARE_SCHEMA_INSTANTIATION Procedure

This procedure performs the synchronization necessary for instantiating all tables in the schema at another database and can enable supplemental logging for key columns or all columns in these tables.

This procedure records the lowest SCN of each object in the schema for instantiation. SCNs subsequent to the lowest SCN for an object can be used for instantiating the object. Running this procedure prepares all current and future objects in the schema for instantiation.

> **See Also:** *Oracle Streams Replication Administrator's Guide* for more information about instantiation and supplemental logging

Syntax

```
DBMS_CAPTURE_ADM.PREPARE_SCHEMA_INSTANTIATION(
   schema_name           IN  VARCHAR2,
   supplemental_logging  IN  VARCHAR2  DEFAULT 'keys');
```

Parameters

Table 20–10 PREPARE_SCHEMA_INSTANTIATION Procedure Parameters

Parameter	Description
`schema_name`	The name of the schema. For example, `hr`.
`supplemental_logging`	Either `none`, `keys`, or `all`.
	If `none` is specified, then this procedure does not enable supplemental logging for any columns in the tables in the schema. This procedure does not remove existing supplemental logging specifications for these tables.
	If `keys` is specified, then this procedure enables supplemental logging for primary key, unique key, bitmap index, and foreign key columns in the tables in the schema and for any table added to this schema in the future. Primary key columns are logged unconditionally. Unique key, bitmap index, and foreign key columns are logged conditionally. Specifying `keys` does not enable supplemental logging of bitmap join index columns.
	If `all` is specified, then this procedure enables supplemental logging for all columns in the tables in the schema and for any table added to this schema in the future. The columns are logged unconditionally. Supplemental logging is not enabled for columns of the following types: LOB, `LONG`, `LONG RAW`, and user-defined type.

Usage Notes

Run this procedure at the source database. If you use a capture process to capture all of the changes to a schema, then use this procedure to prepare the tables in the schema for instantiation after the capture process has been configured.

PREPARE_TABLE_INSTANTIATION Procedure

This procedure performs the synchronization necessary for instantiating the table at another database and can enable supplemental logging for key columns or all columns in the table.

This procedure records the lowest SCN of the table for instantiation. SCNs subsequent to the lowest SCN for an object can be used for instantiating the object.

> **See Also:** *Oracle Streams Replication Administrator's Guide* for more information about instantiation and supplemental logging

Syntax

```
DBMS_CAPTURE_ADM.PREPARE_TABLE_INSTANTIATION(
   table_name            IN  VARCHAR2,
   supplemental_logging  IN  VARCHAR2  DEFAULT 'keys');
```

Parameters

Table 20–11 PREPARE_TABLE_INSTANTIATION Procedure Parameters

Parameter	Description
`table_name`	The name of the table specified as `[`*`schema_name`*`.]`*`object_name`*. For example, `hr.employees`. If the schema is not specified, then the current user is the default.
`supplemental_logging`	Either `none`, `keys`, or `all`.
	If `none` is specified, then this procedure does not enable supplemental logging for any columns in the table. This procedure does not remove existing supplemental logging specifications for the table.
	If `keys` is specified, then this procedure enables supplemental logging for primary key, unique key, bitmap index, and foreign key columns in the table. The procedure places the key columns for the table in three separate log groups: the primary key columns in an unconditional log group, the unique key columns and bitmap index columns in a conditional log group, and the foreign key columns in a conditional log group. Specifying `keys` does not enable supplemental logging of bitmap join index columns.
	If `all` is specified, then this procedure enables supplemental logging for all columns in the table. The procedure places all of the columns for the table in an unconditional log group. Supplemental logging is not enabled for columns of the following types: LOB, `LONG`, `LONG RAW`, and user-defined type.

Usage Notes

Run this procedure at the source database. If you use a capture process to capture all of the changes to a table, then use this procedure to prepare the table for instantiation after the capture process has been configured.

SET_PARAMETER Procedure

This procedure sets a capture process parameter to the specified value.

Syntax

```
DBMS_CAPTURE_ADM.SET_PARAMETER(
   capture_name  IN  VARCHAR2,
   parameter     IN  VARCHAR2,
   value         IN  VARCHAR2);
```

Parameters

Table 20–12 SET_PARAMETER Procedure Parameters

Parameter	Description
capture_name	The name of the capture process. Do not specify an owner.
parameter	The name of the parameter you are setting. See "Capture Process Parameters" on page 20-24 for a list of these parameters.
value	The value to which the parameter is set

Usage Notes

When you alter a parameter value, a short amount of time might pass before the new value for the parameter takes effect.

Capture Process Parameters

The following table lists the parameters for the capture process.

Table 20–13 Capture Process Parameters

Parameter Name	Possible Values	Default	Description
disable_on_limit	Y or N	N	If Y, then the capture process is disabled because it reached a value specified by the time_limit parameter or message_limit parameter. If N, then the capture process is restarted immediately after stopping because it reached a limit.
downstream_real_time_mine	Y or N	Y for local capture processes N for downstream capture processes	If Y, then the capture process is a real-time downstream capture process. After setting this parameter to y, switch the redo log file at the source database using the SQL statement ALTER SYSTEM ARCHIVE LOG CURRENT to begin real-time downstream capture. If this parameter is set to Y, then redo data from the source database must be sent to the standby redo log at the downstream database. See *Oracle Streams Concepts and Administration* for information about creating a real-time downstream capture process. If N, then the capture process is an archived-log downstream capture process. This parameter is ignored for local capture processes.
maximum_scn	A valid SCN or infinite	infinite	The capture process is disabled before capturing a change record with an SCN greater than or equal to the value specified. If infinite, then the capture process runs regardless of the SCN value.

Table 20–13 (Cont.) Capture Process Parameters

Parameter Name	Possible Values	Default	Description
message_limit	A positive integer or infinite	infinite	The capture process stops after capturing the specified number of messages. If infinite, then the capture process continues to run regardless of the number of messages captured.
parallelism	A positive integer	1	The number of parallel execution servers that can concurrently mine the redo log Setting the parallelism parameter to a number higher than the number of available parallel execution servers might disable the capture process. Make sure the PROCESSES and PARALLEL_MAX_SERVERS initialization parameters are set appropriately when you set the parallelism capture process parameter. **Note: When you change the value of this parameter, the capture process is stopped and restarted automatically.**
startup_seconds	0, a positive integer, or infinite	0	The maximum number of seconds to wait for another instantiation of the same capture process to finish. If the other instantiation of the same capture process does not finish within this time, then the capture process does not start. This parameter is useful only if you are starting the capture process manually. If infinite, then the capture process does not start until another instantiation of the same capture process finishes.
time_limit	A positive integer or infinite	infinite	The capture process stops as soon as possible after the specified number of seconds since it started. If infinite, then the capture process continues to run until it is stopped explicitly.
trace_level	0 or a positive integer	0	Set this parameter only under the guidance of Oracle Support Services.
write_alert_log	Y or N	Y	If Y, then the capture process writes a message to the alert log on exit. If N, then the capture process does not write a message to the alert log on exit. The message specifies the reason the capture process stopped.

Note:

- For all parameters that are interpreted as positive integers, the maximum possible value is 4,294,967,295. Where applicable, specify infinite for larger values.
- For parameters that require an SCN setting, any valid SCN value can be specified.

START_CAPTURE Procedure

This procedure starts the capture process, which mines redo logs and enqueues the mined redo information into the associated queue.

The start status is persistently recorded. Hence, if the status is ENABLED, then the capture process is started upon database instance startup.

The capture process is a background Oracle process and is prefixed by c.

The enqueue and dequeue state of DBMS_AQADM.START_QUEUE and DBMS_AQADM.STOP_QUEUE have no effect on the start status of a capture process.

See Also: Chapter 106, "DBMS_STREAMS_ADM"

Syntax

```
DBMS_CAPTURE_ADM.START_CAPTURE(
   capture_name  IN  VARCHAR2);
```

Parameters

Table 20–14 START_CAPTURE Procedure Parameter

Parameter	Description
capture_name	The name of the capture process. Do not specify an owner. The capture process uses LogMiner to capture changes in the redo information. A NULL setting is not allowed.

Usage Notes

The capture process status is persistently recorded. Hence, if the status is ENABLED, then the capture process is started upon database instance startup. A capture process (c*nnn*) is an Oracle background process.

STOP_CAPTURE Procedure

This procedure stops the capture process from mining redo logs.

The stop status is persistently recorded. Hence, if the status is DISABLED, then the capture process is not started upon database instance startup.

The enqueue and dequeue state of DBMS_AQADM.START_QUEUE and DBMS_AQADM.STOP_QUEUE have no effect on the stop status of a capture process.

Syntax

```
DBMS_CAPTURE_ADM.STOP_CAPTURE(
   capture_name  IN  VARCHAR2,
   force         IN  BOOLEAN  DEFAULT FALSE);
```

Parameters

Table 20–15 STOP_CAPTURE Procedure Parameters

Parameter	Description
capture_name	The name of the capture process. A NULL setting is not allowed. Do not specify an owner.
force	This parameter is reserved for future use. Currently, valid BOOLEAN settings are ignored.

Usage Notes

The capture process status is persistently recorded. Hence, if the status is DISABLED or ABORTED, then the capture process is not started upon database instance startup. A capture process (*cnnn*) is an Oracle background process.

21

DBMS_CDC_PUBLISH

The DBMS_CDC_PUBLISH package, one of a set of Change Data Capture packages, is used by a publisher to set up an Oracle Change Data Capture system to capture and publish change data from one or more Oracle relational source tables.

Change Data Capture captures and publishes only committed data. Oracle Change Data Capture identifies new data that has been added to, updated in, or removed from relational tables, and publishes the change data in a form that is usable by subscribers.

Typically, a Change Data Capture system has one **publisher** who captures and publishes changes for any number of Oracle relational source tables. The publisher then provides subscribers (applications or individuals) with access to the published data. Subscribers access the published data using the DBMS_CDC_SUBSCRIBE package.

Note: In previous releases, this package was named DBMS_LOGMNR_CDC_PUBLISH. Beginning with Oracle Database 10*g*, the LOGMNR string has been removed from the name, resulting in the name DBMS_CDC_PUBLISH. Although both variants of the name are still supported, the variant with the LOGMNR string has been deprecated and may not be supported in a future release.

See Also: *Oracle Database Data Warehousing Guide* for information regarding Oracle Change Data Capture

This chapter contains the following topics:

Using DBMS_CDC_PUBLISH

This section contains the following topics, which relate to using the `DBMS_CDC_PUBLISH` package:

Overview

Through the DBMS_CDC_PUBLISH package, the publisher creates and maintains change sources, change sets, and change tables, and eventually drops them when they are no longer useful.

The publisher, typically a database administrator, is concerned primarily with the source of the data and with creating the schema objects that describe the structure of the capture system: change sources, change sets, and change tables.

Most Change Data Capture systems have one publisher and many subscribers. The publisher accomplishes the following main objectives:

1. Determines which source table changes need to be published.
2. Decides whether to capture changes asynchronously or synchronously.
3. Uses the subprograms in the DBMS_CDC_PUBLISH package to capture change data from the source tables and make it available by creating and administering the change source, change set, and change table objects.
4. Allows controlled access to subscribers by using the SQL GRANT and REVOKE statements to grant and revoke the SELECT privilege on change tables for users and roles. (This is necessary to allow the subscribers to subscribe to the change data using the DBMS_CDC_SUBSCRIBE package.)

See Also: Chapter 22, "DBMS_CDC_SUBSCRIBE" for information on the package used to subscribe to published change data

Deprecated Subprograms

> **Note:** Oracle recommends that you do not use deprecated procedures in new applications. Support for deprecated features is for backward compatibility only.

The following subprograms are deprecated with Oracle Database 10*g*:

- `DBMS_CDC_PUBLISH.DROP_SUBSCRIPTION` with a subscription handle

  When dropping a subscription, the publisher should now specify the name of the subscription to be dropped, not the subscription handle.

- `DBMS_CDC_PUBLISH.DROP_SUBSCRIBER_VIEW`

  Dropping a subscriber view is now performed automatically by Change Data Capture.

Security Model

You must have the EXECUTE_CATALOG_ROLE role to use the DBMS_CDC_PUBLISH package. Additional privileges and roles are required depending on the publishing mode and whether the publisher is on the source or staging database. See the section on Granting Privileges and Roles to the Publisher in *Oracle Database Data Warehousing Guide* for details.

Views

The DBMS_CDC_PUBLISH package uses the views listed in the section on Getting Information About the Change Data Capture Environment in *Oracle Database Data Warehousing Guide*.

Summary of DBMS_CDC_PUBLISH Subprograms

Table 21–1 describes the subprograms in the DBMS_CDC_PUBLISH supplied package and the mode or modes with which each can be used. A value of All in the Mode column indicates that the subprogram can be used with synchronous and all modes of asynchronous Change Data Capture, a value of Asynchronous in the Mode column indicates that the subprogram can be used with all modes of asynchronous Change Data Capture (HotLog, Distributed HotLog, and AutoLog).

Table 21–1 DBMS_CDC_PUBLISH Package Subprograms

Subprogram	Mode	Description
ALTER_AUTOLOG_CHANGE_SOURCE Procedure on page 21-8	Asynchronous AutoLog	Changes one or more properties of an existing AutoLog change source
ALTER_CHANGE_SET Procedure on page 21-10	All	Changes one or more of the properties of an existing change set
ALTER_CHANGE_TABLE Procedure on page 21-13	All	Adds or drops columns for an existing change table, or changes the properties of an existing change table
ALTER_HOTLOG_CHANGE_SOURCE Procedure on page 21-15	Asynchronous Distributed HotLog	Changes one or more properties of an existing Distributed HotLog change source
CREATE_AUTOLOG_CHANGE_SOURCE Procedure on page 21-17	Asynchronous AutoLog	Creates an AutoLog change source
CREATE_CHANGE_SET Procedure on page 21-19	All	Creates a change set
CREATE_CHANGE_TABLE Procedure on page 21-22	All	Creates a change table in a specified schema
CREATE_HOTLOG_CHANGE_SOURCE Procedure on page 21-26	Asynchronous Distributed HotLog	Creates a Distributed HotLog change source
DROP_CHANGE_SET Procedure on page 21-28	All	Drops an existing change set
DROP_CHANGE_SOURCE Procedure on page 21-29	Asynchronous Autolog and Asynchronous Distributed Hotlog	Drops an existing AutoLog or Distributed HotLog change source
DROP_CHANGE_TABLE Procedure on page 21-30	All	Drops an existing change table
DROP_SUBSCRIPTION Procedure on page 21-31	All	Allows a publisher to drop a subscription that was created by a subscriber
PURGE Procedure on page 21-32	All	Removes unneeded rows from all change tables in the staging database
PURGE_CHANGE_SET Procedure on page 21-33	All	Removes unneeded rows from all change tables in a specified change set
PURGE_CHANGE_TABLE Procedure on page 21-34	All	Removes unneeded rows from a specified change table

ALTER_AUTOLOG_CHANGE_SOURCE Procedure

This procedure changes the properties of an existing AutoLog change source.

Syntax

```
DBMS_CDC_PUBLISH.ALTER_AUTOLOG_CHANGE_SOURCE(
     change_source_name  IN VARCHAR2,
     description         IN VARCHAR2 DEFAULT NULL,
     remove_description  IN CHAR DEFAULT 'N',
     first_scn           IN NUMBER DEFAULT NULL);
```

Parameters

Table 21–2 ALTER_AUTOLOG_CHANGE_SOURCE Procedure Parameters

Parameter	Description
`change_source_name`	Name of an existing AutoLog change source. Change source names follow Oracle schema object naming rules.
`description`	New description of the change source. The description must be specified using 255 or fewer characters.
`remove_description`	A value of `'Y'` or `'N'`. If the value is `'Y'`, then the current description is changed to `NULL`. If the value is `'N'`, then the current description is unchanged. Do not specify the description parameter with this parameter.
`first_scn`	New first SCN.

Exceptions

Table 21–3 ALTER_AUTOLOG_CHANGE_SOURCE Procedure Exceptions

Exception	Description
`ORA-31401`	Specified change source is not an existing change source
`ORA-31452`	Invalid value for parameter, expecting: `Y` or `N`
`ORA-31455`	Nothing to `ALTER`
`ORA-31497`	Invalid value specified for `first_scn`
`ORA-31498`	The `description` and `remove_description` parameters cannot both be specified
`ORA-31499`	Null value specified for required parameter
`ORA-31501`	Specified change source is not an AutoLog change source
`ORA-31504`	Cannot alter or drop predefined change source
`ORA-31507`	Specified parameter value longer than maximum length

Usage Notes

- Properties supplied to this procedure with a `NULL` value are unchanged.
- This procedure can be used to change more than one property at a time.
- This procedure can be used in making SCN adjustments after determining which redo logs are no longer needed for an asynchronous AutoLog change set.

See Also: The section on asynchronous Change Data Capture and redo log files in *Oracle Database Data Warehousing Guide* for information on how the publisher can use the `ALTER_AUTOLOG_CHANGE_SOURCE` procedure in making SCN adjustments after determining which redo logs are no longer needed for an asynchronous AutoLog change set.

ALTER_CHANGE_SET Procedure

This procedure changes the properties of an existing change set that was created with the CREATE_CHANGE_SET procedure.

Syntax

```
DBMS_CDC_PUBLISH.ALTER_CHANGE_SET(
     change_set_name         IN VARCHAR2,
     description             IN VARCHAR2 DEFAULT NULL,
     remove_description      IN CHAR DEFAULT 'N',
     enable_capture          IN CHAR DEFAULT NULL,
     recover_after_error     IN CHAR DEFAULT NULL,
     remove_ddl              IN CHAR DEFAULT NULL,
     stop_on_ddl             IN CHAR DEFAULT NULL);
```

Parameters

Table 21–4 ALTER_CHANGE_SET Procedure Parameters

Parameter	Description
change_set_name	Name of an existing change set. Change set names follow the Oracle schema object naming rules.
description	New description of the change set. Specify using 255 or fewer characters.
remove_ description	A value of 'Y' or 'N'. If the value is 'Y', then the current description is changed to NULL. If the value is 'N', then the current description is unchanged. Do not specify the description parameter with this parameter.
enable_capture	A value of 'Y' or 'N'. If the value is 'Y', then change data capture is enabled for this change set. If the value is 'N', then change data capture is disabled for this change set. Synchronous change sets are created with change data capture enabled and cannot be disabled. Asynchronous change sets are created with change data capture disabled.
recover_after_ error	A value of 'Y' or 'N'. If the value is 'Y', then Change Data Capture will attempt to recover from earlier capture errors. If the value is 'N', then Change Data Capture will not attempt to recover from earlier capture errors.
remove_ddl	A value of 'Y' or 'N'. If the value is 'Y' and the value of the recover_after_error parameter is 'Y', then any DDL records that may have caused capture errors will be filtered out during recovery. If the value is 'N', then DDL records that may have caused capture errors will not be filtered out during recovery. This parameter has meaning only when the recover_after_error parameter is specified with a value of 'Y'.

Table 21–4 (Cont.) ALTER_CHANGE_SET Procedure Parameters

Parameter	Description
stop_on_ddl	A value of 'Y' or 'N'.
	If the value is 'Y', then Change Data Capture stops when a DDL event is detected.
	If the value is 'N', then Change Data Capture continues when a DDL event is detected.
	See the Usage Notes for additional information about this parameter.

Exceptions

Table 21–5 ALTER_CHANGE_SET Procedure Exceptions

Exception	Description
ORA-31410	Specified change set is not an existing change set
ORA-31452	Invalid value for parameter, expecting: Y or N
ORA-31455	Invalid lock handle while acquiring lock
ORA-31468	Cannot process DDL change record
ORA-31469	Cannot enable Change Data Capture for change set
ORA-31485	Invalid database link
ORA-31498	The description and remove_description parameters cannot both be specified
ORA-31499	Null value specified for required parameter
ORA-31505	Cannot alter or drop predefined change set
ORA-31507	Specified parameter value longer than maximum length
ORA-31508	Invalid parameter value for synchronous change set
ORA-31514	Change set disabled due to capture error

Usage Notes

- The publisher can use this procedure for asynchronous and synchronous Change Data Capture. However, the predefined synchronous change set, SYNC_SET, cannot be altered, and the following parameters cannot be altered for publisher-defined synchronous change sets: enable_capture, recover_after_error, remove_ddl, and stop_on_ddl.
- Properties supplied to this procedure with a NULL value are unchanged.
- This procedure can alter more than one parameter at a time.
- Enabling or disabling an asynchronous HotLog or AutoLog change set starts or stops the Oracle Streams capture process and apply process underlying the change set. Enabling or disabling an asynchronous Distributed HotLog change set starts or stops the Oracle Streams apply process underlying the change set.
- The effect of the stop_on_ddl parameter is as follows:
  - When the stop_on_ddl parameter is set to 'Y', asynchronous Change Data Capture stops if DDL is encountered during change data capture. Some DDL statements can adversely affect capture, such as a statement that drops a source table column that is being captured. The publisher has an opportunity

to analyze and adjust to DDL changes that may adversely affect change tables while capture is stopped, thereby preventing possible errors during capture.

Because these statements do not affect the column data itself, Change Data Capture does not stop capturing change data when the `stop_on_ddl` parameter is set to `'Y'` and any of the following statements is encountered:

* `ANALYZE TABLE`
* `LOCK TABLE`
* `GRANT` privileges to access a table
* `REVOKE` privileges to access a table
* `COMMENT` on a table
* `COMMENT` on a column

These statements can be issued on the source database without concern for their impact on Change Data Capture processing.

– When the `stop_on_ddl` parameter is set to `'N'`, Change Data Capture does not stop if DDL is encountered during change data capture. If a change set does not stop on DDL, but a DDL change occurs that affects change tables, that change can result in a capture error. There are also system conditions that can cause capture errors, such as being out of disk space.

See Also: *Oracle Database Data Warehousing Guide* for information on the effects of, and how to recover from, a capture error

Whenever a DDL statement causes processing to stop, a message is written to the alert log indicating for which change set processing has been stopped and the DDL statement that caused it to be stopped. Similarly, whenever DDL statements are ignored by Change Data Capture and processing continues, a message is written to the alert log indicating which DDL statement was ignored.

- The publisher can attempt to recover an asynchronous change set after a capture error by specifying `'Y'` for the `recover_after_error` parameter. Capture errors can occur when any of the following is true:
  – The `stop_on_ddl` parameter is set to `'Y'` and there is a DDL record in the change data. In this case, to recover from the error, the publisher must also specify `'Y'` for the `remove_ddl` parameter.
  – The `stop_on_ddl` parameter is set to `'N'` and there is a DDL record that affects capture. For example, if the publisher drops and re-creates a change table, it causes an error the next time that Change Data Capture attempts to add change data to the named change table.
  – A miscellaneous error occurs, such as running out of disk space, or a redo log file error (such as ORA-01688: unable to extend table *string.string* partition *string* by *string* in tablespace *string*).

  See Also: *Oracle Database Data Warehousing Guide* for more information on how to recover from a capture error.

ALTER_CHANGE_TABLE Procedure

This procedure adds columns to, or drops columns from, or changes the properties of, a change table that was created with the CREATE_CHANGE_TABLE procedure.

Syntax

```
DBMS_CDC_PUBLISH.ALTER_CHANGE_TABLE(
     owner                  IN VARCHAR2,
     change_table_name      IN VARCHAR2,
     operation              IN VARCHAR2,
     column_list            IN VARCHAR2,
     rs_id                  IN CHAR,
     row_id                 IN CHAR,
     user_id                IN CHAR,
     timestamp              IN CHAR,
     object_id              IN CHAR,
     source_colmap          IN CHAR,
     target_colmap          IN CHAR);
```

Parameters

Table 21–6 ALTER_CHANGE_TABLE Procedure Parameters

Parameter	Description
owner	The schema that owns the change table.
change_table_name	The change table that is being altered. Change table names follow the Oracle schema object naming rules.
operation	Either the value ADD or DROP to indicate whether to add or drop the user columns specified with the column_list parameter and any control columns specified by other parameters.
column_list	User column names and datatypes for each column of the source table that should be added to, or dropped from, the change table. The list is comma-delimited.
rs_id row_id user_id timestamp object_id source_colmap target_colmap	Each listed parameter specifies a particular control column, as follows: ■ The rs_id parameter specifies the RSID$ control column. ■ The row_id parameter specifies the ROW_ID$ control column. ■ The user_id parameter specifies the USERNAME$ control column. ■ The timestamp parameter specifies the TIMESTAMP$ control column. ■ The object_id parameter specifies the SYS_NC_OID$ control column. ■ The source_colmap parameter specifies the SOURCE_COLMAP$ control column. ■ The target_colmap parameter specifies the TARGET_COLMAP$ control column. Each parameter must have a value of either 'Y' or 'N', where: ■ 'Y': Adds the specified control column to, or drops it from the change table, as indicated by the operation parameter. ■ 'N': Neither adds the specified control column, nor drops it from the change table.

See Also: *Oracle Database Data Warehousing Guide* for a complete description of control columns.

Exceptions

Table 21–7 ALTER_CHANGE_TABLE Procedure Exceptions

Exception	Description
ORA-31403	Specified change table already contains the specified column
ORA-31409	One or more values for input parameters are incorrect
ORA-31415	Specified change set does not exist
ORA-31416	Invalid SOURCE_COLMAP value
ORA-31417	Column list contains control column *control-column-name*
ORA-31421	Change table does not exist
ORA-31422	Specified owner schema does not exist
ORA-31423	Specified change table does not contain the specified column
ORA-31454	Invalid value specified for operation parameter, expecting ADD or DROP
ORA-31455	Nothing to alter
ORA-31456	Error executing a procedure in the DBMS_CDC_UTILITY package
ORA-31459	System triggers for DBMS_CDC_PUBLISH package are not installed
ORA-31471	Invalid OBJECT_ID value

Usage Notes

- The publisher can use this procedure for asynchronous and synchronous Change Data Capture.
- The publisher cannot add and drop user columns in the same call to the ALTER_CHANGE_TABLE procedure; these schema changes require separate calls.
- The publisher must not specify the name of the control columns in the column_list parameter.
- When altering an asynchronous change table, the publisher must accept the default value or specify 'N' for the source_colmap and object_id parameters. In addition, for the asynchronous Distributed HotLog mode, the publisher also must accept the default value or specify 'N' for the row_id and username parameters when the change source is 9.2 or 10.1.

  See Also: *Oracle Database Data Warehousing Guide* for information about the impact on subscriptions when a publisher adds a column to a change table.

ALTER_HOTLOG_CHANGE_SOURCE Procedure

This procedure changes the properties of an existing Distributed HotLog change source.

Syntax

```
DBMS_CDC_PUBLISH.ALTER_HOTLOG_CHANGE_SOURCE(
     change_source_name  IN VARCHAR2,
     description         IN VARCHAR2 DEFAULT NULL,
     remove_description  IN CHAR DEFAULT 'N',
     enable_source       IN CHAR DEFAULT NULL);
```

Parameters

Table 21–8 ALTER_HOTLOG_CHANGE_SOURCE Procedure Parameters

Parameter	Description
change_source_name	Name of an existing Distributed HotLog change source. Change source names follow Oracle schema object naming rules.
description	New description of the change source. The description must be specified using 255 or fewer characters.
remove_description	A value of 'Y' or 'N'. If the value is 'Y', then the current description is changed to NULL. If the value is 'N', then the current description is unchanged. Do not specify the description parameter with this parameter.
enable_source	A value of 'Y' or 'N'. If the value is 'Y', then the change source is enabled. If the value is 'N', then the change source is disabled.

Exceptions

Table 21–9 ALTER_HOTLOG_CHANGE_SOURCE Procedure Exceptions

Exception	Description
ORA-31401	Change source is not an existing change source
ORA-31455	Nothing to ALTER
ORA-31480	Staging database and source database cannot be the same
ORA-31481	Change source is not a HotLog change source
ORA-31482	Invalid option for non-distributed HotLog change source
ORA-31484	Source database must be at least 9.2.0.6 or greater
ORA-31485	Invalid database link
ORA-31498	The description and remove_description parameters cannot both be specified
ORA-31499	Null value specified for required parameter
ORA-31504	Cannot alter or drop predefined change source
ORA-31507	Parameter value longer than maximum length
ORA-31532	Cannot enable change source

Table 21–9 (Cont.) ALTER_HOTLOG_CHANGE_SOURCE Procedure Exceptions

Exception	Description
ORA-31534	Change Data Capture publisher is missing DBA role

Usage Notes

- Properties supplied to this procedure with a NULL value are unchanged.
- This procedure can be used to change more than one property at a time.
- Enabling or disabling a Distributed HotLog change source starts or stops the Oracle Streams capture process that underlies the change source.
- This procedure cannot be used to alter the change source for the asynchronous HotLog mode of Change Database Capture. The change source for the asynchronous HotLog mode is the predefined change source, HOTLOG_SOURCE, which cannot be altered.

CREATE_AUTOLOG_CHANGE_SOURCE Procedure

This procedure creates an AutoLog change source. An AutoLog change source is based on of a set of redo log files automatically copied by redo transport services to the system on which the staging database resides.

Syntax

```
DBMS_CDC_PUBLISH.CREATE_AUTOLOG_CHANGE_SOURCE(
     change_source_name  IN VARCHAR2,
     description         IN VARCHAR2 DEFAULT NULL,
     source_database     IN VARCHAR2,
     first_scn           IN NUMBER,
     online_log          IN CHAR DEFAULT 'N');
```

Parameters

Table 21–10 CREATE_AUTOLOG_CHANGE_SOURCE Procedure Parameters

Parameter	Description
change_source_name	Name of the change source. Change source names follow the Oracle schema object naming rules.
description	Description of the change source. Specify using 255 or fewer characters.
source_database	Global name of the change source's source database instance.
first_scn	The SCN of the start of a LogMiner dictionary that is in the change source's archived redo log files.
online_log	A value of 'Y' or 'N' If the value is 'Y', then the change source uses the AutoLog online option to hot-mine the source database online redo log to gather change data. There can only be one change source with online_log='Y' on a given staging database. If the value is 'N', then the change source uses the AutoLog archive option to get change data from archived redo log files. There can be one or more change sources with online_log='N' on a given staging database.

Exceptions

Table 21–11 CREATE_AUTOLOG_CHANGE_SOURCE Procedure Exceptions

Exception	Description
ORA-31436	Duplicate change source specified
ORA-31497	Invalid value specified for first_scn
ORA-31499	Null value specified for required parameter
ORA-31507	Specified parameter value is longer than the maximum length
ORA-31508	Invalid parameter value for synchronous change set
ORA-31535	Cannot support change source in this configuration

Usage Notes

- The publisher can use this procedure for asynchronous Change Data Capture only.

- The publisher must take care when specifying a value for the `source_database` parameter. Change Data Capture does not validate this value when creating the change source. The publisher can query the `GLOBAL_NAME` column in the `GLOBAL_NAME` view at the source database for the `source_database` parameter value.
- The publisher must configure redo transport services to automatically copy the log files to the system on which the staging database resides.

  See Also: The section on performing asynchronous AutoLog publishing in *Oracle Database Data Warehousing Guide* for information on configuring redo transport services to automatically copy the log files to the system on which the staging database resides.

- An AutoLog change source must begin with an archived redo log file that contains a LogMiner dictionary. The `CREATE_AUTOLOG_CHANGE_SOURCE` `first_scn` parameter indicates the SCN for this dictionary extraction and is the point at which the change source can begin capturing changes. The publisher can determine the value for the `first_scn` parameter using either of the following methods:
  - Direct `DBMS_CAPTURE_ADM.BUILD` to return the value when the dictionary is built:

    ```
    SET SERVEROUTPUT ON
    VARIABLE FSCN NUMBER;
    BEGIN
     :FSCN := 0;
     DBMS_CAPTURE_ADM.BUILD(:FSCN);
     DBMS_OUTPUT.PUT_LINE('The first_scn value is ' || :FSCN);
    END;
    /
    The first_scn value is 207722
    ```

  - Make the following query on the source database. If this query returns multiple distinct values for `first_change#`, then the data dictionary has been extracted more than once and the publisher should choose the `first_change#` value that is the most appropriate to the change source.

    ```
    SELECT DISTINCT FIRST_CHANGE#, NAME
       FROM V$ARCHIVED_LOG
       WHERE DICTIONARY_BEGIN = 'YES';
    ```

    See Also: The section on performing asynchronous AutoLog publishing in *Oracle Database Data Warehousing Guide* for information on archived redo log files and the LogMiner dictionary.

- For the asynchronous mode of Change Data Capture, the amount of change data captured is dependent on the level of supplemental logging enabled at the source database.

  See Also: *Oracle Database Data Warehousing Guide* for information about supplemental logging.

CREATE_CHANGE_SET Procedure

This procedure allows the publisher to create a change set. For asynchronous HotLog and AutoLog Change Data Capture, the publisher can optionally provide beginning and ending date values at which to begin and end change data capture.

Syntax

```
DBMS_CDC_PUBLISH.CREATE_CHANGE_SET(
     change_set_name        IN VARCHAR2,
     description            IN VARCHAR2 DEFAULT NULL,
     change_source_name     IN VARCHAR2,
     stop_on_ddl            IN CHAR DEFAULT 'N',
     begin_date             IN DATE DEFAULT NULL,
     end_date               IN DATE DEFAULT NULL);
```

Parameters

Table 21–12 CREATE_CHANGE_SET Procedure Parameters

Parameter	Description
change_set_name	Name of the change set. Change set names follow the Oracle schema object naming rules.
description	Description of the change set. Specify using 255 or fewer characters.
change_source_name	Name of the existing change source to contain this change set.
stop_on_ddl	A value of 'Y' or 'N'. If the value is 'Y', then Change Data Capture stops when a DDL event is detected. If the value is 'N', then Change Data Capture continues when a DDL event is detected. See the Usage Notes for additional information about this parameter.
begin_date	Date on which the publisher wants the change set to begin capturing changes. A value for this parameter is valid for the asynchronous HotLog and AutoLog modes of Change Data Capture only.
end_date	Date on which the publisher wants the change set to stop capturing changes. A value for this parameter is valid for the asynchronous HotLog and AutoLog modes of Change Data Capture only.

Exceptions

Table 21–13 CREATE_CHANGE_SET Procedure Exceptions

Exception	Description
ORA-31401	Specified change source is not an existing change source
ORA-31407	The end_date must be greater than the begin_date
ORA-31408	Invalid value specified for begin_scn or end_scn
ORA-31437	Duplicate change set specified
ORA-31452	Invalid value for parameter, expecting: Y or N
ORA-31483	Cannot have spaces in the parameter
ORA-31485	Invalid database link

Table 21–13 (Cont.) CREATE_CHANGE_SET Procedure Exceptions

Exception	Description
ORA-31487	Cannot support begin dates or end dates in this configuration
ORA-31488	Cannot support change set in this configuration
ORA-31499	Null value specified for required parameter
ORA-31503	Invalid date supplied for begin_date or end_date
ORA-31507	Specified parameter value longer than maximum length
ORA-31508	Invalid parameter value for synchronous change set

Usage Notes

- The publisher can use this procedure for asynchronous and synchronous Change Data Capture. However, the default values for the following parameters are the only supported values for synchronous change sets: begin_date, end_date, and stop_on_ddl. The default values for the following parameters are the only supported values for asynchronous Distributed HotLog change sets: begin_date and end_date.
- When the change source is Distributed HotLog on a release of Oracle Database earlier than 10.2, Change Data Capture inserts rows into the CHANGE_PROPAGATION and CHANGE_PROPAGATION_SETS views on the staging database.
- An AutoLog online change source (created with online_log='Y') can only contain one change set.
- The begin_date and end_date parameters are optional. The publisher can specify neither of them, one of them, or both. The effect of these parameters is as follows:
  - When a begin_date is specified, changes from transactions that begin on or after that date are captured.
  - When a begin_date is not specified, capture starts with the earliest available change data.
  - When an end_date is specified, changes from transactions that are committed on or before that date are captured.
  - When an end_date is not specified, Change Data Capture continues indefinitely.
- The effect of the stop_on_ddl parameter is as follows:
  - When the stop_on_ddl parameter is set to 'Y', asynchronous Change Data Capture stops if DDL is encountered during change data capture. Some DDL statements can adversely affect capture, such as a statement that drops a source table column that is being captured. The publisher has an opportunity to analyze and adjust to DDL changes that may adversely affect change tables while capture is stopped, thereby preventing possible errors during capture.

    Because these statements do not affect the column data itself, Change Data Capture does not stop capturing change data when the stop_on_ddl parameter is set to 'Y' and any of the following statements is encountered:

    * ANALYZE TABLE
    * LOCK TABLE
    * GRANT privileges to access a table

  * `REVOKE` privileges to access a table
  * `COMMENT` on a table
  * `COMMENT` on a column

  These statements can be issued on the source database without concern for their impact on Change Data Capture processing.

– When the `stop_on_ddl` parameter is set to `'N'`, Change Data Capture does not stop if DDL is encountered during change data capture. If a change set does not stop on DDL, but a DDL change occurs that affects capture, that change can result in a capture error.

  See Also: *Oracle Database Data Warehousing Guide* for information on the effects of, and how to recover from, a capture error.

Whenever a DDL statement causes processing to stop, a message is written to the alert log indicating for which change set processing has been terminated and the DDL statement that caused it to be terminated. Similarly, whenever DDL statements are ignored by Change Data Capture and processing continues, a message is written to the alert log indicating which DDL statement was ignored.

CREATE_CHANGE_TABLE Procedure

This procedure creates a change table in a specified schema.

> **Note:** Oracle recommends that the publisher be certain that the source table that will be referenced in a `CREATE_CHANGE_TABLE` procedure has been created prior to calling this procedure, particularly if the change set that will be specified in the procedure has the `stop_on_ddl` parameter set to `'Y'`.

Syntax

```
DBMS_CDC_PUBLISH.CREATE_CHANGE_TABLE(
     owner                  IN VARCHAR2,
     change_table_name      IN VARCHAR2,
     change_set_name        IN VARCHAR2,
     source_schema          IN VARCHAR2,
     source_table           IN VARCHAR2,
     column_type_list       IN VARCHAR2,
     capture_values         IN VARCHAR2,
     rs_id                  IN CHAR,
     row_id                 IN CHAR,
     user_id                IN CHAR,
     timestamp              IN CHAR,
     object_id              IN CHAR,
     source_colmap          IN CHAR,
     target_colmap          IN CHAR,
     options_string         IN VARCHAR2);
```

Parameters

Table 21–14 CREATE_CHANGE_TABLE Procedure Parameters

Parameter	Description
`owner`	Name of the schema that owns the change table.
`change_table_name`	Name of the change table that is being created. Change table names follow the Oracle schema object naming rules.
`change_set_name`	Name of the change set in which this change table resides.
`source_schema`	The schema where the source table is located.
`source_table`	The source table from which the change records are captured.
`column_type_list`	The user columns and datatypes that are being tracked. Specify using a comma-delimited list.
`capture_values`	One of the following capture values for update operations: ■ `OLD`: Captures the original values from the source table. ■ `NEW`: Captures the changed values from the source table. ■ `BOTH`: Captures the original and changed values from the source table.

Table 21–14 (Cont.) CREATE_CHANGE_TABLE Procedure Parameters

Parameter	Description
rs_id row_id user_id timestamp object_id source_colmap target_colmap	Each listed parameter specifies a particular control column as follows: ■ The rs_id parameter specifies the RSID$ control column. ■ The row_id parameter specifies the ROW_ID$ control column. ■ The user_id parameter specifies the USERNAME$ control column. ■ The timestamp parameter specifies the TIMESTAMP$ control column. ■ The object_id parameter specifies the SYS_NC_OID$ control column. ■ The source_colmap parameter specifies the SOURCE_COLMAP$ control column. ■ The target_colmap parameter specifies the TARGET_COLMAP$ control column. Each parameter can have a value of 'Y' or 'N', where: ■ 'Y': Adds the specified control column to the change table. ■ 'N': Does not add the specified control column to the change table.
options_string	The syntactically correct options to be passed to a CREATE TABLE DDL statement. The options string is appended to the generated CREATE TABLE DDL statement after the closing parenthesis that defines the columns of the table. See the Usage Notes for more information.

See Also: *Oracle Database Data Warehousing Guide* for a complete description of control columns

Exceptions

Table 21–15 CREATE_CHANGE_TABLE Procedure Exceptions

Exception	Description
ORA-31402	Unrecognized parameter specified
ORA-31409	One or more values for input parameters are incorrect
ORA-31415	Specified change set does not exist
ORA-31416	Invalid SOURCE_COLMAP value
ORA-31417	Column list contains control column *control-column-name*
ORA-31418	Specified source schema does not exist
ORA-31419	Specified source table does not exist
ORA-31420	Unable to submit the purge job
ORA-31421	Change table does not exist
ORA-31422	Owner schema does not exist
ORA-31438	Duplicate change table
ORA-31447	Cannot create change tables in the SYS schema
ORA-31450	Invalid value for change_table_name

Table 21–15 (Cont.) CREATE_CHANGE_TABLE Procedure Exceptions

Exception	Description
ORA-31451	Invalid value for capture_values, expecting: OLD, NEW, or BOTH
ORA-31452	Invalid value for parameter, expecting: Y or N
ORA-31459	System triggers for DBMS_CDC_PUBLISH package are not installed
ORA-31467	No column found in the source table
ORA-31471	Invalid OBJECT_ID value

Usage Notes

- The publisher can use this procedure for asynchronous and synchronous Change Data Capture.
- A change table is a database table that contains the change data resulting from DML statements (INSERT, UPDATE, and DELETE) made to a source table. A given change table can capture changes from only one source table.
- A change table is a database table that contains two types of columns:
  - User columns, which are copies of actual columns of source tables that reside in the change table.
  - Control columns, which maintain special metadata for each change row in the change table. Information such as the DML operation performed, the capture time (time stamp), and changed column vectors are examples of control columns. The publisher must not specify the name of the control columns in the user column list.
- If there are multiple publishers on the staging database for the Distributed HotLog mode of Change Data capture, and one publisher defines a change table in another publisher's Distributed HotLog change set, then Change Data Capture uses the database link established by the publisher who created the change set to access the source database. Therefore, the database link to the source database established by the publisher who created the change set must be intact for the change table to be successfully created. If the change set publisher's database link is not present when creating a change table, an error is returned indicating that the connection description for the remote database was not found.
- The publisher must not attempt to control a change table's partitioning properties. Change Data Capture automatically manages the change table partitioning as part of its change table management.
- When creating a change table for any mode of asynchronous Change Data Capture, the publisher must accept the default value or specify 'N' for the source_colmap and object_id parameters. In addition, for the asynchronous Distributed HotLog mode of Change Data Capture, the publisher also must accept the default value or specify 'N' for the row_id and username parameters when the change source is 9.2 or 10.1.
- When the publisher specifies the rs_id parameter, the RSID$ column is added to the change table. The RSID$ column value reflects an operation's capture order within a transaction, but not across transactions. The publisher cannot use the RSID$ column value by itself to order committed operations across transactions; it must be used in conjunction with the CSCN$ column value.
- The publisher can control a change table's physical properties, tablespace properties, and so on, by specifying the options_string parameter. With the

options_string parameter, the publisher can set any option that is valid for the CREATE TABLE DDL statement (except for partitioning properties).

Note: How the publisher defines the options_string parameter can have an effect on the performance and operations in a Change Data Capture system. For example, if the publisher places several constraints in the options column, it can have a noticeable effect on performance. Also, if the publisher uses NOT NULL constraints and a particular column is not changed in an incoming change row, then the constraint can cause the INSERT operation to fail and the transaction that contains the INSERT operation to be rolled back.

- Oracle recommends that change tables not be created in system tablespaces. This can be accomplished if the publisher's default tablespace is not the system tablespace or if the publisher specifies a tablespace in the options_string parameter. If a tablespace is not specified by the publisher, and the publisher's default table space is the system tablespace, then Change Data Capture creates change tables in the system tablespace.

  See Also: *Oracle Database Data Warehousing Guide* for more information on, and examples of, creating change tables in tablespaces managed by the publisher.

CREATE_HOTLOG_CHANGE_SOURCE Procedure

This procedure creates a Distributed HotLog change source on the source database when the publisher runs this procedure from the staging database. A Distributed HotLog change source is based on data in the online redo log files that is automatically transferred to the staging database by Oracle Streams propagation.

Syntax

```
DBMS_CDC_PUBLISH.CREATE_HOTLOG_CHANGE_SOURCE(
     change_source_name     IN VARCHAR2,
     description            IN VARCHAR2 DEFAULT NULL,
     source_database        IN VARCHAR2);
```

Parameters

Table 21–16 CREATE_HOTLOG_CHANGE_SOURCE Procedure Parameters

Parameters	Description
change_source_name	Name of the Distributed HotLog change source to be created. Each change source name must be unique and must follow the Oracle schema object naming rules.
description	Description of the change source. Specify using 255 or fewer characters.
source_database	The name of the database link defined from the staging database to the source database, where the source database is Oracle9*i* Database, Database 10*g* Release 1, or Oracle Database 10*g*Release 2. See *Oracle Database Data Warehousing Guide* for information on creating database links for the Distributed HotLog mode of Change Data Capture.

Exceptions

Table 21–17 CREATE_HOTLOG_CHANGE_SOURCE Procedure Exceptions

Exception	Description
ORA-31436	Duplicate change source
ORA-31480	Staging database and source database cannot be the same
ORA-31483	Cannot have spaces in the parameter
ORA-31484	Source database must be at least 9.2.0.6 or greater
ORA-31485	Invalid database link
ORA-31499	Null value specified for required parameter
ORA-31507	Parameter value longer than the maximum length
ORA-31534	Change Data Capture publisher is missing DBA role

Usage Notes

- The publisher can use this procedure for the asynchronous Distributed HotLog mode of Change Data Capture only.

  This procedure cannot be used to create a change source for the asynchronous HotLog mode of Change Database Capture. The publisher must use the predefined change source, HOTLOG_SOURCE, for the asynchronous HotLog mode of Change Data Capture.

- A Distributed HotLog change source can contain one or more change sets, but they must all be on the same staging database.
- A staging database publisher cannot create multiple Distributed HotLog change sources with the same name, even when those change sources are on different source databases.
- When the publisher creates a change source on a release of Oracle Database earlier than 10.2, Change Data Capture:
  - Generates names for the Streams capture process, capture queue, and propagation based on the change source name. If a generated name is already in use, an error indicating that the capture process, queue, or propagation cannot be created is returned.
  - Inserts a row into the `CHANGE_SOURCES` view on the staging database where the `SOURCE_TYPE` column of the inserted row indicates that the source Oracle Database release is earlier than 10.2.
- Note that the database link indicated by the `source_database` parameter must exist when creating, altering, or dropping a Distributed HotLog change source and the change sets and change tables it contains. However, this database link is not required for change capture to occur. Once the required Distributed HotLog change sources, change sets and change tables are in place and enabled, this database link can be dropped without interrupting change capture. This database link would need to be recreated to create, alter, or drop Distributed HotLog change sources, change sets and change tables.

DROP_CHANGE_SET Procedure

This procedure drops an existing change set that was created with the CREATE_CHANGE_SET procedure.

Syntax

```
DBMS_CDC_PUBLISH.DROP_CHANGE_SET(
     change_set_name     IN VARCHAR2);
```

Parameters

Table 21–18 DROP_CHANGE_SET Procedure Parameters

Parameter	Description
change_set_name	Name of the change set to be dropped. Change set names follow the Oracle schema object naming rules.

Exceptions

Table 21–19 DROP_CHANGE_SET Procedure Exceptions

Exception	Description
ORA-31410	Specified change set is not an existing change set
ORA-31411	Specified change set is referenced by a change table
ORA-31485	Invalid database link
ORA-31499	Null value specified for required parameter
ORA-31505	Cannot alter or drop predefined change set
ORA-31507	Specified parameter value is longer than maximum length

Usage Notes

- The publisher can use this procedure for asynchronous and synchronous Change Data Capture.
- The change set to be dropped cannot contain any change tables.
- The predefined synchronous change set, SYNC_SET, cannot be dropped.

DROP_CHANGE_SOURCE Procedure

This procedure drops an existing AutoLog change source that was created with the CREATE_AUTOLOG_CHANGE_SOURCE procedure or an existing Distributed HotLog change source that was created with the CREATE_HOTLOG_CHANGE_SOURCE procedure.

Syntax

```
DBMS_CDC_PUBLISH.DROP_CHANGE_SOURCE(
     change_source_name    IN VARCHAR2);
```

Parameters

Table 21–20 DROP_CHANGE_SOURCE Procedure Parameters

Parameter	Description
change_source_name	Name of the change source to be dropped. Change source names follow the Oracle schema object naming rules.

Exceptions

Table 21–21 DROP_CHANGE_SOURCE Procedure Exceptions

Exception	Description
ORA-31401	Specified change source is not an existing change source
ORA-31406	Specified change source is referenced by a change set
ORA-31499	Null value specified for required parameter
ORA-31504	Cannot alter or drop predefined change source
ORA-31507	Specified parameter value longer than maximum length

Usage Notes

- The change source to be dropped cannot contain any change sets.
- The predefined change sources, HOTLOG_SOURCE and SYNC_SOURCE, cannot be dropped.

DROP_CHANGE_TABLE Procedure

This procedure drops an existing change table that was created with the CREATE_CHANGE_TABLE procedure.

Syntax

```
DBMS_CDC_PUBLISH.DROP_CHANGE_TABLE(
     owner              IN VARCHAR2,
     change_table_name  IN VARCHAR2,
     force_flag         IN CHAR);
```

Parameters

Table 21–22 DROP_CHANGE_TABLE Procedure Parameters

Parameter	Description
owner	Name of the schema that owns the change table.
change_table_name	Name of the change table to be dropped. Change table names follow the Oracle schema object naming rules.
force_flag	Drops the change table, depending on whether or not there are subscriptions to it, as follows: ■ 'Y': Drops the change table even if there are subscriptions to it. ■ 'N': Drops the change table only if there are no subscriptions to it.

Exceptions

Table 21–23 DROP_CHANGE_TABLE Procedure Exceptions

Exception	Description
ORA-31421	Change table does not exist
ORA-31422	Specified owner schema does not exist
ORA-31424	Change table has active subscriptions
ORA-31441	Table is not a change table

Usage Notes

- The publisher can use this procedure for asynchronous and synchronous Change Data Capture.
- If the publisher wants to drop a change table while there are active subscriptions to that table, he or she must call the DROP_CHANGE_TABLE procedure using the force_flag => 'Y' parameter. This tells Change Data Capture to override its normal safeguards and allow the change table to be dropped despite active subscriptions. The subscriptions that include the dropped table will no longer be valid, and subscribers will lose access to the change data.

DROP_SUBSCRIPTION Procedure

This procedure allows a publisher to drop a subscription that was created by a subscriber with a prior call to the DBMS_CDC_SUBSCRIBE.CREATE_SUBSCRIPTION procedure.

Syntax

```
DBMS_CDC_PUBLISH.DROP_SUBSCRIPTION(
     subscription_name  IN VARCHAR2);
```

Parameters

Table 21–24 DROP_SUBSCRIPTION Procedure Parameters

Parameter	Description
subscription_name	Name of the subscription that was specified by a previous call to the DBMS_CDC_SUBSCRIBE.CREATE_SUBSCRIPTION procedure. Subscription names follow the Oracle schema object naming rules.

Exceptions

Table 21–25 DROP_SUBSCRIPTION Procedure Exceptions

Exception	Description
ORA-31409	One or more values for input parameters are incorrect
ORA-31425	Subscription does not exist
ORA-31432	Invalid source table

Usage Notes

- The publisher can use this procedure for asynchronous and synchronous Change Data Capture.
- This procedure works the same way as the DBMS_CDC_SUBSCRIBE.DROP_SUBSCRIPTION procedure.
- This procedure provides the publisher with a way to drop subscriptions that have not been dropped by the subscriber. It is possible that a subscription that is no longer needed still exists and is holding change data in a change table indefinitely. The publisher can use this procedure to remove such a subscription so that a purge operation can clean up its change data. Oracle recommends that the publisher attempt to verify that the subscription is not needed prior to dropping it. If that is not possible, the publisher should inform the subscription owner that the subscription has been dropped. Ideally, subscribers drop subscriptions that are no longer needed using the DBMS_CDC_SUBSCRIBE.DROP_SUBSCRIPTION procedure and the publisher need not use the DBMS_CDC_SUBSCRIBE.DROP_SUBSCRIPTION procedure.

PURGE Procedure

This procedure monitors change table usage by all subscriptions, determines which rows are no longer needed by any subscriptions, and removes the unneeded rows to prevent change tables from growing indefinitely. When called, this procedure purges all change tables on the staging database.

Syntax

```
DBMS_CDC_PUBLISH.PURGE;
```

Exceptions

Only standard Oracle exceptions (for example, a privilege violation) are returned during a purge operation.

Usage Notes

- The publisher can use this procedure for asynchronous and synchronous Change Data Capture.
- The publisher can run this procedure manually or automatically:
  - The publisher can run this procedure manually from the command line to purge data from change tables.
  - The publisher can run this procedure in a script to routinely perform a purge operation and control the growth of change tables.
- Note that the `DBMS_CDC_PUBLISH.PURGE` procedure (used by the publisher and the Change Data Capture default purge job) is distinct from the `DBMS_CDC_SUBSCRIBE.PURGE_WINDOW` procedure (used by subscribers). A call to the `DBMS_CDC_PUBLISH.PURGE` procedure physically removes unneeded rows from change tables. A call to the `DBMS_CDC_SUBSCRIBE.PURGE_WINDOW` procedure, logically removes change rows from a subscription window, but does not physically remove rows from the underlying change tables.

PURGE_CHANGE_SET Procedure

This procedure removes unneeded rows from all change tables in the named change set. This procedure allows a finer granularity purge operation than the basic PURGE procedure.

Syntax

```
DBMS_CDC_PUBLISH.PURGE_CHANGE_SET(
 change_set_name IN VARCHAR2);
```

Parameters

Table 21–26 PURGE_CHANGE_SET Procedure Parameters

Parameter	Description
change_set_name	Name of an existing change set. Change set names follow the Oracle schema object naming rules.

Exceptions

Table 21–27 PURGE_CHANGE_SET Procedure Exceptions

Exception	Description
ORA-31410	Change set is not an existing change set

Usage Notes

- The publisher can use this procedure for asynchronous and synchronous Change Data Capture.
- The publisher can run this procedure manually from the command line or in a script to purge unneeded rows from change tables in a specific change set.
- Note that the DBMS_CDC_PUBLISH.PURGE_CHANGE_SET procedure (used by the publisher) is distinct from the DBMS_CDC_SUBSCRIBE.PURGE_WINDOW procedure (used by subscribers). A call to the DBMS_CDC_PUBLISH.PURGE_CHANGE_SET procedure physically removes unneeded rows from change tables in the specified change set. A call to the DBMS_CDC_SUBSCRIBE.PURGE_WINDOW procedure, logically removes change rows from a subscription window, but does not physically remove rows from the underlying change tables.

PURGE_CHANGE_TABLE Procedure

This procedure removes unneeded rows from the named change table. This procedure allows a finer granularity purge operation than the basic PURGE procedure or the PURGE_CHANGE_SET procedure.

Syntax

```
DBMS_CDC_PUBLISH.PURGE_CHANGE_TABLE(
     owner              IN VARCHAR2,
     change_table_name  IN VARCHAR2);
```

Parameters

Table 21–28 PURGE_CHANGE_TABLE Procedure Parameters

Parameter	Description
owner	Owner of the named change table.
change_table_name	Name of an existing change table. Change table names follow the Oracle schema object naming rules.

Exceptions

Table 21–29 PURGE_CHANGE_TABLE Procedure Exceptions

Exception	Description
ORA-31421	Change table does not exist

Usage Notes

- The publisher can use this procedure for asynchronous and synchronous Change Data Capture.
- The publisher can run this procedure manually from the command line or in a script to purge unneeded rows from a specified change table.
- Note that the DBMS_CDC_PUBLISH.PURGE_CHANGE_TABLE procedure (used by the publisher) is distinct from the DBMS_CDC_SUBSCRIBE.PURGE_WINDOW procedure (used by subscribers). A call to the DBMS_CDC_PUBLISH.PURGE_CHANGE_TABLE procedure physically removes unneeded rows from the specified change table. A call to the DBMS_CDC_SUBSCRIBE.PURGE_WINDOW procedure, logically removes change rows from a subscription window, but does not physically remove rows from the underlying change tables.

22

DBMS_CDC_SUBSCRIBE

The DBMS_CDC_SUBSCRIBE package, one of a set of Change Data Capture packages, lets subscribers view and query change data that was captured and published with the DBMS_CDC_PUBLISH package.

A Change Data Capture system usually has one publisher and many subscribers. The **subscribers** (applications or individuals), use the Oracle supplied package, DBMS_CDC_SUBSCRIBE, to access published data.

Note: In previous releases, this package was named DBMS_LOGMNR_CDC_SUBSCRIBE. Beginning with Oracle Database 10*g*, the LOGMNR string has been removed from the name, resulting in the name DBMS_CDC_SUBSCRIBE. Although both variants of the name are still supported, the variant with the LOGMNR string has been deprecated and may not be supported in a future release.

See Also: *Oracle Database Data Warehousing Guide* for information regarding Oracle Change Data Capture.

This chapter contains the following topics:

Using DBMS_CDC_SUBSCRIBE

This section contains the following topics, which relate to using the `DBMS_CDC_SUBSCRIBE` package:

Overview

The primary role of the subscriber is to use the change data. Through the DBMS_CDC_SUBSCRIBE package, each subscriber registers interest in source tables by subscribing to them.

Once the publisher sets up the system to capture data into change tables (which are viewed as publications by subscribers) and grants subscribers access to the change tables, subscribers can access and query the published change data for any of the source tables of interest. Using the subprograms in the DBMS_CDC_SUBSCRIBE package, the subscriber accomplishes the following main objectives:

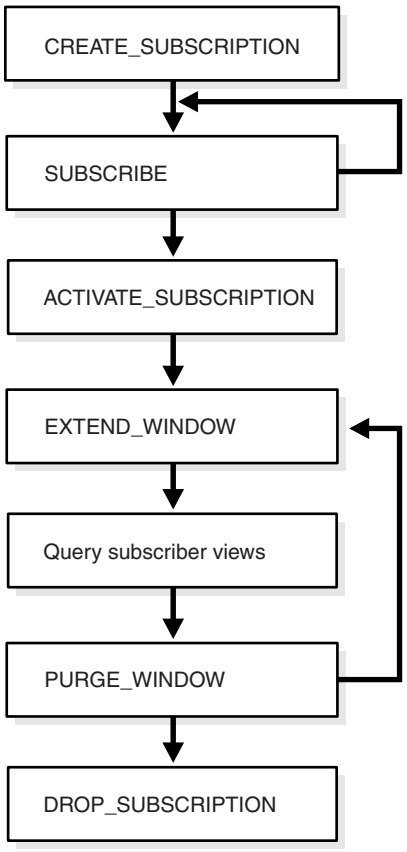
1. Indicates the change data of interest by creating a subscription and associated subscriber views on published source tables and source columns
2. Activates the subscription to indicate that the subscriber is ready to receive change data
3. Extends the subscription window to receive a new set of change data
4. Uses SQL SELECT statements to retrieve change data from the subscriber views
5. Purges the subscription window when finished processing a block of changes
6. Drops the subscription when finished with the subscription

Figure 22–1 provides a graphical flowchart of the order in which subscribers most typically use the subprograms in the DBMS_CDC_SUBSCRIBE package (which are listed in Table 22–1). A subscriber would typically create a subscription, subscribe to one or more source tables and columns, activate the subscription, extend the subscription window, query the subscriber views, purge the subscription window, and then either extend the subscription window again or drop the subscription.

Note: If a subscriber uses the PURGE_WINDOW procedure immediately after using an EXTEND_WINDOW procedure, then change data may be lost without ever being processed.

See Also: Chapter 21, "DBMS_CDC_PUBLISH" for information on the package for publishing change data.

Figure 22–1 Subscription Flow



Deprecated Subprograms

> **Note:** Oracle Corporation recommends that you do not use deprecated procedures in new applications. Support for deprecated features is for backward compatibility only.

The following subprograms are deprecated with Oracle Database 10*g*:

- `DROP_SUBSCRIBER_VIEW`

  Subscribers no longer need to drop subscriber views. This work is now done automatically by Change Data Capture.

- `GET_SUBSCRIPTION_HANDLE`

  Subscribers no longer explicitly specify subscription handles. Subscribers should use the `CREATE_SUBSCRIPTION` procedure instead to specify a subscription name.

- `PREPARE_SUBSCRIBER_VIEW`

  Subscribers no longer need to prepare subscriber views. This work is now done automatically by Change Data Capture.

Changes in Behavior

If an existing application uses these deprecated `DBMS_CDC_SUBSCRIBE` subprograms with Oracle Database 10*g*, note the following changes in behavior:

- Subscriber views are persistent for the life of the subscription.
- Some error conditions, particularly with regard to subscriber view management, no longer occur.
- If a publisher alters a publication such that it contains different control columns, the subscriber must call `DBMS_CDC_SUBSCRIBE.EXTEND_WINDOW` to see the new column structure.

Deprecated Parameters

The use of the `subscription_handle` parameter with the following `DBMS_CDC_SUBSCRIBE` procedures has been deprecated beginning with Oracle Database 10*g*:

- `SUBSCRIBE`
- `ACTIVATE_SUBSCRIPTION`
- `EXTEND_WINDOW`
- `PURGE_WINDOW`
- `DROP_SUBSCRIPTION`

Security Model

Change Data Capture grants `EXECUTE` privileges to `PUBLIC` on the `DBMS_CDC_SUBSCRIBE` package.

Views

The DBMS_CDC_SUBSCRIBE package uses the views listed in the section on Getting Information About the Change Data Capture Environment in *Oracle Database Data Warehousing Guide*.

Summary of DBMS_CDC_SUBSCRIBE Subprograms

Table 22–1 DBMS_CDC_SUBSCRIBE Package Subprograms

Subprogram	Description
ACTIVATE_SUBSCRIPTION Procedure on page 22-9	Indicates that a subscription is ready to start accessing change data
CREATE_SUBSCRIPTION Procedure on page 22-10	Creates a subscription and associates it with one change set
DROP_SUBSCRIPTION Procedure on page 22-12	Drops a subscription that was created with a prior call to the `CREATE_SUBSCRIPTION` procedure
EXTEND_WINDOW Procedure on page 22-13	Sets a subscription window high boundary so that new change data can be seen
PURGE_WINDOW Procedure on page 22-14	Sets the low boundary for a subscription window to notify Change Data Capture that the subscriber is finished processing a set of change data
SUBSCRIBE Procedure on page 22-15	Specifies a source table and the source columns for which the subscriber wants to access change data and specifies the subscriber view through which the subscriber sees change data for the source table

ACTIVATE_SUBSCRIPTION Procedure

This procedure indicates that a subscription is ready to start accessing change data.

Syntax

```
DBMS_CDC_SUBSCRIBE.ACTIVATE_SUBSCRIPTION (
     subscription_name  IN VARCHAR2);
```

Parameters

Table 22–2 ACTIVATE_SUBSCRIPTION Procedure Parameters

Parameter	Description
subscription_name	The name of the subscription that was specified for a previous call to the CREATE_SUBSCRIPTION procedure. Subscription names follow the Oracle schema object naming rules.

Exceptions

Table 22–3 ACTIVATE_SUBSCRIPTION Procedure Exceptions

Exception	Description
ORA-31409	One or more values for input parameters are incorrect
ORA-31425	Subscription does not exist
ORA-31426	Cannot modify active subscriptions
ORA-31469	Cannot enable Change Data Capture for change set
ORA-31514	Change set disabled due to capture error

Usage Notes

- The ACTIVATE_SUBSCRIPTION procedure indicates that the subscriber is finished subscribing to tables, and the subscription is ready to start accessing change data.
- Once the subscriber activates the subscription:
  - No additional source tables can be added to the subscription.
  - Change Data Capture holds the available data for the source tables and sets the subscription window to empty.
  - The subscriber must use the EXTEND_WINDOW procedure to see the initial set of change data.
  - The subscription cannot be activated again.
- A subscription cannot be activated if the underlying change set has reached its end_date parameter value.

CREATE_SUBSCRIPTION Procedure

This procedure creates a subscription that is associated with one change set. This procedure replaces the deprecated GET_SUBSCRIPTION_HANDLE procedure.

Syntax

```
DBMS_CDC_SUBSCRIBE.CREATE_SUBSCRIPTION (
     change_set_name      IN  VARCHAR2,
     description          IN  VARCHAR2,
     subscription_name    IN  VARCHAR2);
```

Parameters

Table 22–4 CREATE_SUBSCRIPTION Procedure Parameters

Parameter	Description
change_set_name	The name of an existing change set to which the subscriber subscribes
description	A description of the subscription (which might include, for example, the purpose for which it is used). The description must be specified using 255 or fewer characters.
subscription_name	A unique name for a subscription that must consist of 30 characters or fewer and cannot have a prefix of CDC$. Subscription names follow the Oracle schema object naming rules.

Exceptions

Table 22–5 CREATE_SUBSCRIPTION Procedure Exceptions

Exception	Description
ORA-31409	One or more values for input parameters are incorrect
ORA-31415	Specified change set does not exist
ORA-31449	Invalid value for change_set_name
ORA-31457	Maximum length of description field exceeded
ORA-31469	Cannot enable Change Data Capture for change set
ORA-31506	Duplicate subscription name specified
ORA-31510	Name uses reserved prefix CDC$
ORA-31511	Name exceeds maximum length of 30 characters

Usage Notes

- The CREATE_SUBSCRIPTION procedure allows a subscriber to register interest in a change set associated with source tables of interest.
- A subscriber can query the ALL_PUBLISHED_COLUMNS view to see all the published source tables for which the subscriber has privileges and the change sets in which the source table columns are published.
- Subscriptions are not shared among subscribers; rather, each subscription name is validated against a given subscriber's login ID.

- Subscriptions cannot be created if the underlying change set has reached its `end_date` parameter value.

DROP_SUBSCRIPTION Procedure

This procedure drops a subscription.

Syntax

```
DBMS_CDC_SUBSCRIBE.DROP_SUBSCRIPTION (
     subscription_name  IN VARCHAR2);
```

Parameters

Table 22–6 DROP_SUBSCRIPTION Procedure Parameters

Parameter	Description
subscription_name	The name of the subscription that was specified for a previous call to the CREATE_SUBSCRIPTION procedure. Subscription names follow the Oracle schema object naming rules.

Exceptions

Table 22–7 DROP_SUBSCRIPTION Procedure Exceptions

Exception	Description
ORA-31409	One or more values for input parameters are incorrect
ORA-31425	Subscription does not exist

Usage Notes

Subscribers should be diligent about dropping subscriptions that are no longer needed so that change data will not be held in the change tables unnecessarily.

EXTEND_WINDOW Procedure

This procedure sets the subscription window high boundary so that new change data can be seen.

Syntax

```
DBMS_CDC_SUBSCRIBE.EXTEND_WINDOW (
     subscription_name  IN VARCHAR2);
```

Parameters

Table 22–8 EXTEND_WINDOW Procedure Parameters

Parameter	Description
subscription_name	The unique name of the subscription that was specified by a previous call to the CREATE_SUBSCRIPTION procedure. Subscription names follow the Oracle schema object naming rules.

Exceptions

Table 22–9 EXTEND_WINDOW Procedure Exceptions

Exception	Description
ORA-31409	One or more values for input parameters are incorrect
ORA-31425	Subscription does not exist
ORA-31429	Subscription has not been activated
ORA-31432	Invalid source table
ORA-31469	Cannot enable Change Data Capture for change set
ORA-31509	Publication does not exist
ORA-31514	Change set disabled due to capture error

Usage Notes

- Until the subscriber calls the EXTEND_WINDOW procedure to begin receiving change data, the subscription window remains empty.
  - The first time that the subscriber calls the EXTEND_WINDOW procedure, it establishes the initial boundaries for the subscription window.
  - Subsequent calls to the EXTEND_WINDOW procedure extend the high boundary of the subscription window so that new change data can be seen.
- Oracle recommends that subscribers not view change tables directly. Instead, subscribers should use the DBMS_CDC_SUBSCRIBE package and access data through subscriber views only. Control column values are guaranteed to be consistent only when viewed through subscriber views that have been updated with a call to the EXTEND_WINDOW procedure.
- When the underlying change set for a subscription has reached its end_date parameter value, subsequent calls to the EXTEND_WINDOW procedure will not raise the high boundary.

PURGE_WINDOW Procedure

This procedure sets the low boundary of the subscription window so that the subscription no longer sees any change data, effectively making the subscription window empty. The subscriber calls this procedure to notify Change Data Capture that the subscriber is finished processing a block of change data.

Syntax

```
DBMS_CDC_SUBSCRIBE.PURGE_WINDOW (
     subscription_name   IN VARCHAR2);
```

Parameters

Table 22–10 PURGE_WINDOW Procedure Parameters

Parameter	Description
subscription_name	The name of the subscription that was specified for a previous call to the CREATE_SUBSCRIPTION procedure. Subscription names follow the Oracle schema object naming rules.

Exceptions

Table 22–11 PURGE_WINDOW Procedure Exceptions

Exception	Description
ORA-31409	One or more values for input parameters are incorrect
ORA-31425	Subscription does not exist
ORA-31429	Subscription has not been activated
ORA-31432	Invalid source table
ORA-31469	Cannot enable Change Data Capture for change set
ORA-31514	Change set disabled due to capture error

Usage Notes

- When finished with a set of changes, the subscriber purges the subscription window with the PURGE_WINDOW procedure. By this action, the subscriber performs the following functions:
  - Informs Change Data Capture that the subscriber is finished with the current set of change data.
  - Enables Change Data Capture to remove change data that is no longer needed by any subscribers.

  Change Data Capture manages the change data to ensure that it is available as long as there are subscribers who need it.
- When the underlying change set for a subscription has reached its end_date parameter value, subsequent calls to the PURGE_WINDOW procedure will not move the low boundary.

SUBSCRIBE Procedure

This procedure specifies a source table and the source columns for which the subscriber wants to access change data. In addition, it specifies the subscriber view through which the subscriber sees change data for the source table.

Syntax

There are two versions of syntax for the SUBSCRIBE procedure, as follow:

- Using source schema and source table

  When this syntax is used, Change Data Capture will attempt to find a single publication ID that contains the specified source_table and column_list. If such a publication cannot be found, then Change Data Capture returns an error.

```
DBMS_CDC_SUBSCRIBE.SUBSCRIBE (
     subscription_name    IN VARCHAR2,
     source_schema        IN VARCHAR2,
     source_table         IN VARCHAR2,
     column_list          IN VARCHAR2,
     subscriber_view      IN VARCHAR2);
```

- Using publication IDs

  When this syntax is used, Change Data Capture will use the publication ID to identify the change table. If the columns specified in the column_list parameter are not in the identified change table, then Change Data Capture returns an error.

```
DBMS_CDC_SUBSCRIBE.SUBSCRIBE (
   subscription_name    IN VARCHAR2,
   publication_id       IN NUMBER,
   column_list          IN VARCHAR2,
   subscriber_view      IN VARCHAR2);
```

Parameters

Table 22–12 SUBSCRIBE Procedure Parameters

Parameter	Description
subscription_name	The name of a subscription that was specified for, or returned by, a previous call to the CREATE_SUBSCRIPTION procedure. Subscription names follow the Oracle schema object naming rules.
source_schema	The name of the schema where the source table resides
source_table	The name of a published source table
column_list	A comma-delimited list of columns from the published source table or publication
subscriber_view	Unique name for the subscriber view for this source table or publication that must consist of 30 or fewer characters and must not have a prefix of CDC$. Subscriber view names follow the Oracle schema object naming rules.
publication_id	A valid publication_id, which the subscriber can obtain from the ALL_PUBLISHED_COLUMNS view.

Exceptions

Table 22–13 SUBSCRIBE Procedure Exceptions

Exception	Description
ORA-31409	One or more values for input parameters are incorrect
ORA-31425	Subscription does not exist
ORA-31426	Cannot modify active subscriptions
ORA-31427	Specified source table already subscribed
ORA-31428	No publication contains all the specified columns
ORA-31432	Invalid source table
ORA-31466	No publications found
ORA-31469	Cannot enable Change Data Capture for change set
ORA-31510	Name uses reserved prefix CDC$
ORA-31511	Name exceeds maximum length of 30 characters

Usage Notes

- The SUBSCRIBE procedure allows a subscriber to subscribe to one or more published source tables and to specific columns in each source table. Each call to the SUBSCRIBE procedure can specify only a single source table or publication ID. The subscriber can make multiple calls to the SUBSCRIBE procedure to include multiple source tables or publications IDs in a subscription.
- If the columns of interest are all in a single publication, the subscriber can call the SUBSCRIBE procedure using the source_schema and source_table parameters or using the publication_id parameter. However, if there are multiple publications on a single source table and these publications share some columns, and if any of the shared columns will be used by a single subscription, then the subscriber should call the SUBSCRIBE procedure using the publication_id parameter.
- The subscriber can subscribe to any valid publication ID on which the subscriber has privileges to access. The subscriber can find valid publication IDs on which the subscriber has access by querying the ALL_PUBLISHED_COLUMNS view.
- A subscriber can query the ALL_PUBLISHED_COLUMNS view to see all the published source table columns accessible to the subscriber.
- Subscriptions must be created before a subscriber calls the SUBSCRIBE procedure. Change Data Capture does not guarantee that there will be any change data available at the moment the subscription is created.
- Subscribers can subscribe only to published columns from the source table. All of the columns specified in a single call to the SUBSCRIBE procedure must come from the same publication. Any control columns associated with the underlying change table are added to the subscription automatically.
- All specified source tables or publications must be in the change set that is associated with the named subscription.
- A single source table can have more than one publication defined on it. A subscriber can subscribe to one or more of these publications. However a subscriber can subscribe to a particular publication only once.

- Each publication in a subscription has its own subscriber view. Subscriber views are used to query the change data encompassed by the subscription's current window. Subscriber views are created in the schema of the subscriber.
- A subscriber cannot subscribe to a publication within a change set that has reached its `end_date` parameter value.

23

DBMS_CHANGE_NOTIFICATION

The DBMS_CHANGE_NOTIFICATION package is part of the database change notification feature that provides the functionality to create registration on queries designated by a client application and so to receive notifications in response to DML or DDL changes on the objects associated with the queries. The notifications are published by the database when the DML or DDL transaction commits.

> **See Also:** *Oracle Database Application Developer's Guide - Fundamentals* regarding implementing database change notification.

This chapter contains the following topics:

Using DBMS_CHANGE_NOTIFICATION

Overview

The DBMS_CHANGE_NOTIFICATION package provides PL/SQL based registration interfaces. A client can use this interface to create registrations on queries based on objects of interest and specify a PL/SQL callback handler to receive notifications. When a transaction changes any of the objects associated with the registered queries and COMMITs, this invokes the PL/SQL callback specified during registration. The application can define client-specific processing inside the implementation of its PL/SQL callback handler.

The interface lets you define a registration block (using a mechanism similar to a BEGIN-END block). The recipient of notifications namely the name of the PL/SQL callback handler and a few other registration properties like time-outs can be specified during the BEGIN phase. Any queries executed subsequently (inside the registration block) are considered "interesting queries" and objects referenced by those queries during query execution are registered. The registration is completed by ENDing the registration block.

The registration block lets you create new registrations or add objects to existing registrations.

When a registration is created through the PL/SQL interface, a unique registration ID is assigned to the registration by the RDBMS. The client application can use the registration ID to keep track of registrations created by it. When a notification is published by the RDBMS, the registration ID will be part of the notification.

Typical Applications

This functionality is useful for example to applications that cache query result sets on mostly read-only objects in the mid-tier to avoid network round trips to the database. Such an application can create a registration on the queries it is interested in caching. On changes to objects referenced inside those queries, the database publishes a notification when the underlying transaction commits. In response to the notification, the mid-tier application can refresh its cache by re-executing the query/queries.

Security Model

The DBMS_CHANGE_NOTIFICATION package requires that the user have the CHANGE NOTIFICATION system privilege in order to receive notifications, and be granted EXECUTE privilege on the DBMS_CHANGE_NOTIFICATION package.

In addition the user is required to have SELECT privileges on all objects to be registered. Note that if the SELECT privilege on an object was granted at the time of registration creation but lost subsequently (due to a revoke), then the registration will be purged and a notification to that effect will be published.

Constants

The DBMS_CHANGE_NOTIFICATION package uses the constants shown in Table 23–1. The constants are used as flag parameters either during registration or when received during the notification.

The DBMS_CHANGE_NOTIFCATION package has sets of constants:

- EVENT_STARTUP, EVENT_SHUTDOWN, EVENT_SHUTDOWN_ANY, EVENT_DEREG describe the type of the notification published by the database.
- INSERTOP, DELETEOP, UPDATEOP, ALTEROP, DROPOP and UNKNOWNOP describe the type of operation on a table (during a notification published by the database).
- QOS_RELIABLE, QOS_DEREG_NFY, QOS_ROWIDs describe registration Quality of Service properties that the client requires. These are specified during registration.

Table 23–1 DBMS_CHANGE_NOTIFICATION Constants

Name	Type	Value	Description
ALL_OPERATIONS	BINARY_INTEGER	0	Interested in being notified on all operations, specified as a parameter during registration
ALL_ROWS BINARY_ INTEGER	BINARY_INTEGER	1	All rows within the table may have been potentially modified
EVENT_STARTUP	BINARY_INTEGER	1	Instance startup notification
EVENT_SHUTDOWN	BINARY_INTEGER	2	Instance shutdown notification
EVENT_SHUTDOWN_ANY	BINARY_INTEGER	3	Any instance shutdown when running RAC
EVENT_DEREG	BINARY_INTEGER	5	Registration has been removed
EVENT_OBJCHANGE	BINARY_INTEGER	6	Notification for object change
INSERTOP	BINARY_INTEGER	2	Insert operation
UPDATEOP	BINARY_INTEGER	4	Update operation
DELETEOP	BINARY_INTEGER	8	Delete operation
ALTEROP	BINARY_INTEGER	16	Table altered
DROPOP	BINARY_INTEGER	32	Table dropped
UNKNOWNOP	BINARY_INTEGER	64	Unknown operation
QOS_RELIABLE	BINARY_INTEGER	1	Reliable or persistent notification. Also implies that the notifications will be inserted into the persistent storage atomically with the committing transaction that results in an object change.
QOS_DEREG_NFY	BINARY_INTEGER	2	Purge registration on first notification
QOS_ROWIDS	BINARY_INTEGER	4	Require rowids of modified rows

Operational Notes

- The notifications are published by the database when a transaction changes the registered objects and COMMITs.
- All objects referenced in the queries executed inside the registration block starting from the previous NEW_REG_START or ENABLE_REG to REG_END are considered interesting objects and added to the registration.

Troubleshooting

If you have created a registration and seem to not receive notifications when the underlying tables are changed, please check the following.

- Is the job_queue_processes parameter set to a non-zero value? This parameter needs to be configured to a non-zero value in order to receive PL/SQL notifications via the handler.
- Are the registrations being created as a non-SYS user?
- If you are attempting DML changes on the registered object, are you COMMITing the transaction? Please note that the notifications are transactional and will be generated when the transaction COMMITs.
- It maybe possible that there are run-time errors during the execution of the PL/SQL callback due to implementation errors. If so, they would be logged to the trace file of the JOBQ process that attempts to execute the procedure. The trace file would be usually named <ORACLE_SID>_j*_<PID>.trc. '

  For example, if the ORACLE_SID is 'dbs1' and the process is 12483, the trace file might be named 'dbs1_j000_12483.trc.

  Suppose a registration is created with 'chnf_callback as the notification handler and with registration_id 100. Let us suppose the user forgets to define the chnf_callback procedure. Then the JOBQ trace file might contain a message of the following form.

```
Runtime error during execution of PL/SQL cbk chnf_callback for reg CHNF100
   Error in PLSQL notification of msgid:
   Queue :
   Consumer Name :
   PLSQL function :chnf_callback
   Exception Occured, Error msg:
   ORA-00604: error occurred at recursive SQL level 2
   ORA-06550: line 1, column 7:
   PLS-00201: identifier 'CHNF_CALLBACK' must be declared
   ORA-06550: line 1, column 7:
   PL/SQL: Statement ignored
```

  See Also: For more information about troubleshooting Database Change Notification, see *Oracle Database Application Developer's Guide - Fundamentals*.

Examples

Suppose that a mid-tier application has a lot of queries on the HR.EMPLOYEES table. If the EMPLOYEES table is infrequently updated, it can obtain better performance by caching rows from the table because that would avoid a round-trip to the backend database server and server side execution latency. Let us assume that the application has implemented a mid-tier HTTP listener that listens for notifications and updates the mid-tier cache in response to a notification.

The DBMS_CHANGE_NOTIFICATION package can be utilized in this scenario to send notifications about changes to the table by means of the following steps:

1. Implement a mid-tier listener component of the cache management system (for example, using HTTP) that listens to notification messages sent from the database and refreshes the mid-tier cache in response to the notification.
2. Create a server side stored procedure to process notifications

```
CONNECT / AS SYSDBA;
GRANT CHANGE NOTIFICATION TO hr;
GRANT EXECUTE ON DBMS_CHANGE_NOTIFICATION TO hr;

Rem Enable job queue processes to receive notifications.
ALTER SYSTEM SET "job_queue_processes"=2;

CONNECT hr/hr;
Rem Create a table to record notification events
CREATE TABLE nfevents(regid number, event_type number);

Rem create a table to record changes to registered tables
CREATE TABLE nftablechanges(regid number, table_name varchar2(100),
                            table_operation number);

Rem create a table to record rowids of changed rows.
CREATE TABLE nfrowchanges(regid number, table_name varchar2(100),
                          row_id varchar2(30));

Rem Create a PL/SQL callback handler to process notifications.
CREATE OR REPLACE PROCEDURE chnf_callback(ntfnds IN SYS.CHNF$_DESC) IS
   regid           NUMBER;
   tbname          VARCHAR2(60);
   event_type      NUMBER;
   numtables       NUMBER;
   operation_type  NUMBER;
   numrows         NUMBER;
   row_id          VARCHAR2(20);
 BEGIN
     regid      := ntfnds.registration_id;
     numtables  := ntfnds.numtables;
     event_type := ntfnds.event_type;

  INSERT INTO nfevents VALUES(regid, event_type);
  IF (event_type = DBMS_CHANGE_NOTIFICATION.EVENT_OBJCHANGE) THEN
    FOR i IN 1..numtables LOOP
      tbname          := ntfnds.table_desc_array(i).table_name;
      operation_type  := ntfnds.table_desc_array(I). Opflags;
      INSERT INTO nftablechanges VALUES(regid, tbname, operation_type);
      /* Send the table name and operation_type to client side listener using UTL_
HTTP */
      /* If interested in the rowids, obtain them as follows */
```

```
      IF (bitand(operation_type, DBMS_CHANGE_NOTIFICATION.ALL_ROWS) = 0) THEN
        numrows := ntfnds.table_desc_array(i).numrows;
      ELSE
        numrows :=0;   /* ROWID INFO NOT AVAILABLE */
      END IF;

      /* The body of the loop is not executed when numrows is ZERO */
      FOR j IN 1..numrows LOOP
          Row_id := ntfnds.table_desc_array(i).row_desc_array(j).row_id;
          INSERT INTO nfrowchanges VALUES(regid, tbname, Row_id);
         /* optionally Send out row_ids to client side listener using UTL_HTTP; */
      END LOOP;

    END LOOP;
  END IF;
  COMMIT;
END;
/
```

In Step 2 we can send as much information about the invalidation as the mid-tier application needs based on the information obtained from the notification descriptor.

3. Create a registrations on the tables that we wish to be notified about. We pass in the previously defined procedure name (`chnf_callback`) as the name of the server side PL/SQL procedure to be executed when a notification is generated.

```
Rem Create a REGISTRATION on the EMPLOYEES TABLE
DECLARE
  REGDS      SYS.CHNF$_REG_INFO;
  regid      NUMBER;
  mgr_id     NUMBER;
  dept_id    NUMBER;
  qosflags   NUMBER;
BEGIN
 qosflags := DBMS_CHANGE_NOTIFICATION.QOS_RELIABLE +
         DBMS_CHANGE_NOTIFICATION.QOS_ROWIDS;
REGDS := SYS.CHNF$_REG_INFO ('chnf_callback', qosflags, 0,0,0);
regid := DBMS_CHANGE_NOTIFICATION.NEW_REG_START (REGDS);
SELECT manager_id INTO mgr_id FROM EMPLOYEES WHERE employee_id = 200;
DBMS_CHANGE_NOTIFICATION.REG_END;
END;
/
```

Once the registration is created in Step 3 above, the server side PL/SQL procedure defined in Step 2 is executed in response to any `COMMIT`ted changes to the `HR.EMPLOYEES` table. As an example, let us assume that the following update is performed on the employees table.

```
UPDATE employees SET salary=salary*1.05 WHERE employee_id=203;
COMMIT;
```

Once the notification is processed, you will find rows which might look like the following in the `nfevents`, `nftablechanges` and `nfrowchanges` tables.

```
SQL> SELECT * FROM nfevents;

     REGID     EVENT_TYPE
---------------------------
     20045          6


SQL> SELECT * FROM nftablechanges;

  REGID      TABLE_NAME         TABLE_OPERATION
```

```
-----------------------------------------
  20045      HR.EMPLOYEES            4


SQL> select * from nfrowchanges;

    REGID      TABLE_NAME          ROW_ID
-----------------------------------------------------
    20045     HR.EMPLOYEES    AAAKB/AABAAAJ8zAAF
```

Notes

1. In the above example, a registration was created on the EMPLOYEES table with 'chnf_callback' as the PL/SQL handler for notifications. During registration, the client specified reliable notifications (QOS_RELIABLE) and rowid notifications (QOS_ROWIDS)
2. The handler accesses the table descriptor array from the notification descriptor only if the notification type is of EVENT_OBJCHANGE. In all other cases (e.g EVENT_DEREG, EVENT_SHUTDOWN), the table descriptor array should not be accessed.
3. The handler accesses the row descriptor array from the table notification descriptor only if the ALL_ROWS bit is not set in the table operation flag. If the ALL_ROWS bit is set in the table operation flag, then it means that all rows within the table may have been potentially modified. In addition to operations like TRUNCATE that affect all rows in the tables, this bit may also be set if individual rowids have been rolled up into a FULL table invalidation.

   This can occur if too many rows were modified on a given table in a single transaction (more than 80) or the total shared memory consumption due to rowids on the RDBMS is determined too large (exceeds 1 % of the dynamic shared pool size). In this case, the recipient must conservatively assume that the entire table has been invalidated and the callback/application must be able to handle this condition.

   Also note that the implementation of the user defined callback is up to the developer. In the above example, the callback was used to record event details into database tables. The application can additionally send the notification details to a mid-tier HTTP listener of its cache management system (as in the example) using UTL_HTTP. The listener could then refresh its cache by querying from the back-end database.

Data Structures

The DBMS_CHANGE_NOTIFICATION package uses the following OBJECT types.

OBJECT Types

- SYS.CHNF$_DESC Object Type
- SYS.CHNF$_TDESC Object Type
- SYS.CHNF$_TDESC_ARRAY Object (Array) Type
- SYS.CHNF$_RDESC Object Type
- SYS.CHNF$_RDESC_ARRAY Object (Array) Type
- SYS.CHNF$_REG_INFO Object Type

SYS.CHNF$_DESC Object Type

This is the top level change notification descriptor type.

Syntax

```
TYPE SYS.CHNF$_DESC IS OBJECT(
   registration_id    NUMBER,
   transaction_id     RAW(8),
   dbname             VARCHAR2(30),
   event_type         NUMBER,
   numtables          NUMBER,
   table_desc_array   CHNF$_TDESC_ARRAY)
```

Attributes

Table 23–2 SYS.CHNF$_DESC Object Type

Attribute	Description
registration_id	Registration ID returned during registration
transaction_id	Transaction ID. transaction_id of the transaction that made the change. Will be NULL unless the event_type is EVENT_OBJCHANGE
dbname	Name of database
event_type	Database event associated with the notification. Can be one of EVENT_OBJCHANGE (change to a registered object), EVENT_STARTUP, EVENT_SHUTDOWN or EVENT_DEREG (registration has been removed due to a timeout or other reason)
numtables	Number of modified tables. Will be NULL unless the event_type is EVENT_OBJCHANGE.
table_desc_array	Array of table descriptors. Will be NULL unless the event_type is EVENT_OBJCHANGE.

SYS.CHNF$_TDESC Object Type

This object type is the table descriptor that contains an array of row descriptors.

Syntax

```
TYPE SYS.CHNF$_TDESC IS OBJECT OF (
   opflags         NUMBER,
   table_name      VARCHAR2(64),
   numrows         NUMBER,
   row_desc_array  CHNF$_RDESC_ARRAY);
```

Attributes

Table 23–3 TYPE SYS.CHNF$_TDESC Object Type

Attribute	Description
opflags	Table level operation flags. This is a flag field (bit-vector) which describes the operations that occurred on the table. It can be an OR of the following bit fields - INSERTOP, UPDATEOP, DELETEOP, DROPOP, ALTEROP, ALL_ROWS. If the ALL_ROWS (0x1) bit is set it means that either the entire table is modified (for example, DELETE * FROM t) or row level granularity of information is not requested or not available in the notification and the receiver has to conservatively assume that the entire table has been invalidated.
table_name	Name of modified table
numrows	Number of modified rows within the table. numrows will be NULL and hence should not be accessed if the ALL_ROWS bit is set in the table change descriptor.
row_desc_array	Array of row descriptors. This field will be NULL if the ALL_ROWS bit is set in opflags.

SYS.CHNF$_TDESC_ARRAY Object (Array) Type

This object is the table descriptor type that describes the operations that occurred on a registered table.

Syntax

```
TYPE SYS.CHNF$_TDESC_ARRAY IS VARRAY (1024) OF CHNF$_TDESC;
```

SYS.CHNF$_RDESC Object Type

This object type is the notification descriptor received as an argument of the server side PL/SQL procedure which contains the details of the invalidation. This object type describes modifications to an individual row within a changed table.

An array of CHNF$_RDESC is embedded inside a CHNF$_TDESC (table change descriptor) if the QOS_ROWIDS option was chosen at the time of registration and the ALL_ROWS bit is not set in the opflags field of the table change descriptor.

Syntax

```
TYPE SYS.CHNF$_RDESC IS OBJECT OF (
   opflags     NUMBER,
   row_id      VARCHAR2(2000));
```

Attributes

Table 23–4 TYPE SYS.CHNF$_RDESC Object Type

Attribute	Description
opflags	Row level operation flags. The flag field (bit vector) describes the operations in the row (could be INSERTOP, UPDATEOP or DELETEOP).
row_id	The rowid of the modified row

SYS.CHNF$_RDESC_ARRAY Object (Array) Type

This object type corresponds to an array of row change notification descriptors and is embedded inside the table change descriptor (CHNF$_TDESC) if QOS_ROWIDS was specified during registration and the ALL_ROWS bit is not set in the opflags field of the table change descriptor.

Syntax

```
TYPE SYS.CHNF$_RDESC_ARRAY IS VARRAY (1024) OF CHNF$_RDESC;
```

SYS.CHNF$_REG_INFO Object Type

The object type describes the attributes associated with creating a new registration.

Syntax

```
TYPE SYS.CHNF$_REG_INFO IS OBJECT (
  callback                       VARCHAR2(20),
  quosflags                      NUMBER,
  timeout                        NUMBER,
  operations_filter              NUMBER,
  transaction_lag                NUMBER);
```

Attributes

Table 23–5 TYPE SYS.CHNF$_REG_INFO Object Type

Attribute	Description
`callback`	Name of the server side PL/SQL procedure to be executed on a notification. Prototype is `<call_backname>(ntfnds IN SYS.chnf$_desc)`
`qosflags`	Quality of service flags. Can be set to an `OR` of `QOS_RELIABLE` / `QOS_DEREG_NFY` / `QOS_ROWIDS`: ▪ `0x1` (`QOS_RELIABLE`): Notifications are reliable (persistent) and survive instance death. This means that on an instance death in a RAC cluster, surviving instances will be able to deliver any queued invalidations. Similarly, pending invalidations can be delivered on instance restart, in a single instance configuration. The disadvantage is that there is a CPU cost/ latency involved in inserting the invalidation message to a persistent store. If this parameter is false, then server side CPU and latency are minimized, because invalidations are buffered into an in memory queue but the client could lose invalidation messages on an instance shutdown. ▪ `0x2` (`QOS_DEREG_NFY`): The registration will be expunged on the first notification ▪ `0x4` (`QOS_ROWIDS`): The notification needs to include information about the rowids that were modified
`timeout`	If set to a non-zero value, specifies the time in seconds after which the registration is automatically expunged by the database. If zero / `NULL`, the registration lives until explicitly deregistered. Note that the `timeout` option can be combined with the purge on notification (QOS_DEREG_NFY) option as well.

Table 23–5 (Cont.) TYPE SYS.CHNF$_REG_INFO Object Type

Attribute	Description
operations_filter	if non-zero, specifies a filter to be selectively notified on certain operations. These flags can be used to filter based on specific operation types: ■ 0: Notify on all operations (DBMS_CHANGE_NOTIFICATION.ALL_OPERATIONS) ■ 0x2: Notify on every INSERT (DBMS_CHANGE_NOTIFICATION.INSERTOP) ■ 0x4: Notify on every UPDATE (DBMS_CHANGE_NOTIFICATION.UPDATEOP) ■ 0x8: Notify on every DELETE (DBMS_CHANGE_NOTIFICATION.DELETEOP) A combination of operations can be specified by using a bitwise OR.
transaction_lag	Lag between consecutive notifications in units of transactions. Can be used to specify the number of transactions/database changes, by which the client is willing to lag behind the database. If 0, it means that the client needs to receive an invalidation message as soon as it is generated

Usage Notes

- In response to a database change, the server side PL/SQL procedure specified by "callback" is executed. The PL/SQL procedure name has to be specified in the format schema_name.procedure_name. The procedure must have the following signature:

  ```
  PROCEDURE <procedure_name>(ntfnds IN SYS.chnf$_desc)
  ```

  CHNF$_DESC describes the change notification descriptor.

- The init.ora parameter job_queue_processes must be set to a non-zero value to receive PL/SQL notifications, because the specified procedure is executed inside a job queue process when a notification is generated.

Summary of DBMS_CHANGE_NOTIFICATION Subprograms

Table 23–6 DBMS_CHANGE_NOTIFICATION Package Subprograms

Subprogram	Description
DEREGISTER Procedure on page 23-19	De-subscribes the client with the supplied registration identifier (ID)
ENABLE_REG Procedure on page 23-20	Begins a registration block using an existing registration identifier (ID)
NEW_REG_START Function on page 23-21	Begins a new registration block
REG_END Procedure on page 23-22	Ends the registration boundary

DEREGISTER Procedure

This procedure desubscribes the client with the specified registration identifier (ID).

Syntax

```
DBMS_CHANGE_NOTIFICATION.DEREGISTER (
  regid IN NUMBER);
```

Parameters

Table 23–7 DEREGISTER Procedure Parameters

Parameter	Description
regid	Client registration ID

Usage Notes

Only the user that created the registration (or the SYS user) will be able to desubscribe the registration.

ENABLE_REG Procedure

This procedure adds objects to an existing registration identifier (ID). This subprogram is similar to the interface for creating a new registration, except that it takes an existing `regid` to which to add objects.

Subsequent execution of queries causes the objects referenced in the queries to be added to the specified `regid`, and the registration is completed on invoking the REG_END Procedure.

Syntax

```
DBMS_CHANGE_NOTIFICATION.ENABLE_REG (
  regid IN NUMBER);
```

Parameters

Table 23–8 ENABLE_REG Procedure Parameters

Parameter	Description
`regid`	Client registration ID

Usage Notes

Only the user that created the registration will be able to add further objects to the registration.

NEW_REG_START Function

This procedure begins a new registration block. Any objects referenced by queries executed within the registration block are considered interesting objects and added to the registration. The registration block ends upon calling the REG_END procedure.

Syntax

```
DBMS_CHANGE_NOTIFICATION.NEW_REG_START (
  regds IN sys.chnf$_reg_info)
 RETURN NUMBER;
```

Parameters

Table 23–9 NEW_REG_START Function Parameters

Parameter	Description
sys.chnf$_reg_info	Registration descriptor describing the notification handler and other properties of the registration

Return Values

The procedure returns a registration-id which is a unique integer assigned by the database to this registration. The registration-id will be echoed back in every notification received for this registration.

Usage Notes

- The only operations permitted inside a registration block are queries (the ones the user wishes to register). DML and DDL operations are not permitted.
- The registration block is a session property and implicitly terminates upon exiting the session. While the registration block is a session property, the registration itself is a persistent database entity. Once created, the registration survives until explicitly deregistered by the client application or timed-out or removed by the database for some other reason (such as loss of privileges).
- The user must have the CHANGE NOTIFICATION system privilege and SELECT privileges on any objects to be registered.
- The SYS user will not be permitted to create new registrations.
- Nesting of registration block is not permitted.

REG_END Procedure

This procedure marks the end of the registration block. No newly executed queries are tracked.

Syntax

```
DBMS_CHANGE_NOTIFICATION.REG_END;
```

24

DBMS_CRYPTO

DBMS_CRYPTO provides an interface to encrypt and decrypt stored data, and can be used in conjunction with PL/SQL programs running network communications. It provides support for several industry-standard encryption and hashing algorithms, including the Advanced Encryption Standard (AES) encryption algorithm. AES has been approved by the National Institute of Standards and Technology (NIST) to replace the Data Encryption Standard (DES).

> **See Also:** *Oracle Database Security Guide* for further information about using this package and about encrypting data in general.

This chapter contains the following topics:

Using the DBMS_CRYPTO Subprograms

Overview

DBMS_CRYPTO contains basic cryptographic functions and procedures. To use this package correctly and securely, a general level of security expertise is assumed.

The DBMS_CRYPTO package enables encryption and decryption for common Oracle datatypes, including RAW and large objects (LOBs), such as images and sound. Specifically, it supports BLOBs and CLOBs. In addition, it provides Globalization Support for encrypting data across different database character sets.

The following cryptographic algorithms are supported:

- Data Encryption Standard (DES), Triple DES (3DES, 2-key and 3-key)
- Advanced Encryption Standard (AES)
- MD5, MD4, and SHA-1 cryptographic hashes
- MD5 and SHA-1 Message Authentication Code (MAC)

Block cipher modifiers are also provided with DBMS_CRYPTO. You can choose from several padding options, including PKCS (Public Key Cryptographic Standard) #5, and from four block cipher chaining modes, including Cipher Block Chaining (CBC).

Table 24–1 lists the DBMS_CRYPTO package features in comparison to the other PL/SQL encryption package, the DBMS_OBFUSCATION_TOOLKIT.

Table 24–1 DBMS_CRYPTO and DBMS_OBFUSCATION_TOOLKIT Feature Comparison

Package Feature	DBMS_CRYPTO	DBMS_OBFUSCATION_TOOLKIT
Cryptographic algorithms	DES, 3DES, AES, RC4, 3DES_2KEY	DES, 3DES
Padding forms	PKCS5, zeroes	none supported
Block cipher chaining modes	CBC, CFB, ECB, OFB	CBC
Cryptographic hash algorithms	MD5, SHA-1, MD4	MD5
Keyed hash (MAC) algorithms	HMAC_MD5, HMAC_SH1	none supported
Cryptographic pseudo-random number generator	RAW, NUMBER, BINARY_INTEGER	RAW, VARCHAR2
Database types	RAW, CLOB, BLOB	RAW, VARCHAR2

DBMS_CRYPTO is intended to replace the DBMS_OBFUSCATION_TOOLKIT, providing greater ease of use and support for a range of algorithms to accommodate new and existing systems. Specifically, 3DES_2KEY and MD4 are provided for backward compatibility. It is not recommended that you use these algorithms because they do not provide the same level of security as provided by 3DES, AES, MD5, or SHA-1.

Security Model

Oracle Database installs this package in the `SYS` schema. You can then grant package access to existing users and roles as needed.

Types

Parameters for the DBMS_CRYPTO subprograms use these datatypes:

Table 24–2 DBMS_CRYPTO Datatypes

Type	Description
BLOB	A source or destination binary LOB
CLOB	A source or destination character LOB (excluding NCLOB)
PLS_INTEGER	Specifies a cryptographic algorithm type (used with BLOB, CLOB, and RAW datatypes)
RAW	A source or destination RAW buffer

Algorithms

The following cryptographic algorithms, modifiers, and cipher suites are predefined in this package.

Table 24–3 DBMS_CRYPTO Cryptographic Hash Functions

Name	Description
HASH_MD4	Produces a 128-bit hash, or message digest of the input message
HASH_MD5	Also produces a 128-bit hash, but is more complex than MD4
HASH_SH1	Secure Hash Algorithm (SHA). Produces a 160-bit hash.

Table 24–4 DBMS_CRYPTO MAC (Message Authentication Code) Functions

Name	Description
HMAC_MD5[1]	Same as MD5 hash function, except it requires a secret key to verify the hash value.
HMAC_SH1[1]	Same as SHA hash function, except it requires a secret key to verify the hash value.

[1] Complies with IETF RFC 2104 standard

Table 24–5 DBMS_CRYPTO Encryption Algorithms

Name	Description
ENCRYPT_DES	Data Encryption Standard. Block cipher. Uses key length of 56 bits.
ENCRYPT_3DES_2KEY	Data Encryption Standard. Block cipher. Operates on a block 3 times with 2 keys. Effective key length of 112 bits.
ENCRYPT_3DES	Data Encryption Standard. Block cipher. Operates on a block 3 times.
ENCRYPT_AES128	Advanced Encryption Standard. Block cipher. Uses 128-bit key size.
ENCRYPT_AES192	Advanced Encryption Standard. Block cipher. Uses 192-bit key size.
ENCRYPT_AES256	Advanced Encryption Standard. Block cipher. Uses 256-bit key size.
ENCRYPT_RC4	Stream cipher. Uses a secret, randomly generated key unique to each session.

Table 24–6 DBMS_CRYPTO Block Cipher Suites

Name	Description
DES_CBC_PKCS5	ENCRYPT_DES[1] + CHAIN_CBC[2]+ PAD_PKCS5[3]
DES3_CBC_PKCS5	ENCRYPT_3DES[1] + CHAIN_CBC[2] + PAD_PKCS5[3]

[1] See Table 24–5, " DBMS_CRYPTO Encryption Algorithms"
[2] See Table 24–7, " DBMS_CRYPTO Block Cipher Chaining Modifiers"
[3] See Table 24–8, " DBMS_CRYPTO Block Cipher Padding Modifiers"

Table 24–7 DBMS_CRYPTO Block Cipher Chaining Modifiers

Name	Description
CHAIN_ECB	Electronic Codebook. Encrypts each plaintext block independently.
CHAIN_CBC	Cipher Block Chaining. Plaintext is XORed with the previous ciphertext block before it is encrypted.
CHAIN_CFB	Cipher-Feedback. Enables encrypting units of data smaller than the block size.
CHAIN_OFB	Output-Feedback. Enables running a block cipher as a synchronous stream cipher. Similar to CFB, except that *n* bits of the previous output block are moved into the right-most positions of the data queue waiting to be encrypted.

Table 24–8 DBMS_CRYPTO Block Cipher Padding Modifiers

Name	Description
PAD_PKCS5	Provides padding which complies with the PKCS #5: Password-Based Cryptography Standard
PAD_NONE	Provides option to specify no padding. Caller must ensure that blocksize is correct, else the package returns an error.
PAD_ZERO	Provides padding consisting of zeroes.

Restrictions

The VARCHAR2 datatype is not directly supported by DBMS_CRYPTO. Before you can perform cryptographic operations on data of the type VARCHAR2, you must convert it to the uniform database character set AL32UTF8, and then convert it to the RAW datatype. After performing these conversions, you can then encrypt it with the DBMS_CRYPTO package.

> **See Also:** "Conversion Rules" on page 24-11 for information about converting datatypes.

Exceptions

Table 24–9 lists exceptions that have been defined for DBMS_CRYPTO.

Table 24–9 DBMS_CRYPTO Exceptions

Exception	Code	Description
CipherSuiteInvalid	28827	The specified cipher suite is not defined.
CipherSuiteNull	28829	No value has been specified for the cipher suite to be used.
KeyNull	28239	The encryption key has not been specified or contains a NULL value.
KeyBadSize	28234	DES keys: Specified key size is too short. DES keys must be at least 8 bytes (64 bits). AES keys: Specified key size is not supported. AES keys must be 128, 192, or 256 bits in length.
DoubleEncryption	28233	Source data was previously encrypted.

Operational Notes

When to Use Encrypt and Decrypt Procedures or Functions

This package includes both `ENCRYPT` and `DECRYPT` procedures and functions. The procedures are used to encrypt or decrypt `LOB` datatypes (overloaded for `CLOB` and `BLOB` datatypes). In contrast, the `ENCRYPT` and `DECRYPT` functions are used to encrypt and decrypt `RAW` datatypes. Data of type `VARCHAR2` must be converted to `RAW` before you can use `DBMS_CRYPTO` functions to encrypt it.

When to Use Hash or Message Authentication Code (MAC) Functions

This package includes two different types of one-way hash functions: the `HASH` function and the `MAC` function. Hash functions operate on an arbitrary-length input message, and return a fixed-length hash value. One-way hash functions work in one direction only. It is easy to compute a hash value from an input message, but it is extremely difficult to generate an input message that hashes to a particular value. Note that hash values should be at least 128 bits in length to be considered secure.

You can use hash values to verify whether data has been altered. For example, before storing data, Laurel runs `DBMS_CRYPTO.HASH` against the stored data to create a hash value. When she retrieves the stored data at a later date, she can again run the hash function against it, using the same algorithm. If the second hash value is identical to the first one, then the data has not been altered. Hash values are similar to "file fingerprints" and are used to ensure data integrity.

The `HASH` function included with `DBMS_CRYPTO`, is a one-way hash function that you can use to generate a hash value from either `RAW` or `LOB` data. The `MAC` function is also a one-way hash function, but with the addition of a secret key. It works the same way as the `DBMS_CRYPTO.HASH` function, except only someone with the key can verify the hash value.

MACs can be used to authenticate files between users. They can also be used by a single user to determine if her files have been altered, perhaps by a virus. A user could compute the MAC of his files and store that value in a table. If the user did not use a MAC function, then the virus could compute the new hash value after infection and replace the table entry. A virus cannot do that with a MAC because the virus does not know the key.

About Generating and Storing Encryption Keys

The `DBMS_CRYPTO` package can generate random material for encryption keys, but it does not provide a mechanism for maintaining them. Application developers must take care to ensure that the encryption keys used with this package are securely generated and stored. Also note that the encryption and decryption operations performed by `DBMS_CRYPTO` occur on the server, not on the client. Consequently, if the key is sent over the connection between the client and the server, the connection must be protected by using network encryption. Otherwise, the key is vulnerable to capture over the wire.

Although DBMS_CRYPTO cannot generate keys on its own, it does provide tools you can use to aid in key generation. For example, you can use the RANDOMBYTES function to generate random material for keys. (Calls to the RANDOMBYTES function behave like calls to the DESGETKEY and DES3GETKEY functions of the DBMS_OBFUSCATION_TOOLKIT package.)

When generating encryption keys for DES, it is important to remember that some numbers are considered weak and semiweak keys. Keys are considered weak or semiweak when the pattern of the algorithm combines with the pattern of the initial key value to produce ciphertext that is more susceptible to cryptanalysis. To avoid this, filter out the known weak DES keys. Lists of the known weak and semiweak DES keys are available on several public Internet sites.

See Also:

- *Oracle Database Advanced Security Administrator's Guide* for information about configuring network encryption and SSL.
- "Key Management" on page 62-5 for a full discussion about securely storing encryption keys
- "RANDOMBYTES Function" on page 24-20

Conversion Rules

- To convert VARCHAR2 to RAW, use the UTL_I18N.STRING_TO_RAW function to perform the following steps:

  1. Convert VARCHAR2 in the current database character set to VARCHAR2 in the AL32UTF8 database character.
  2. Convert VARCHAR2 in the AL32UTF8 database character set to RAW.

  Syntax example:

  ```
  UTL_I18N.STRING_TO_RAW (string, 'AL32UTF8');
  ```

- To convert RAW to VARCHAR2, use the UTL_I18N.RAW_TO_CHAR function to perform the following steps:

  1. Convert RAW to VARCHAR2 in the AL32UTF8 database character set.
  2. Convert VARCHAR2 in the AL32UTF8 database character set to VARCHAR2 in the database character set you wish to use.

  Syntax example:

  ```
  UTL_I18N.RAW_TO_CHAR (data, 'AL32UTF8');
  ```

  See Also: Chapter 169, "UTL_I18N" for information about using the UTL_I18N PL/SQL package.

- If you want to store encrypted data of the RAW datatype in a VARCHAR2 database column, then use RAWTOHEX or UTL_ENCODE.BASE64_ENCODE to make it suitable for VARCHAR2 storage. These functions expand data size by 2 and 4/3, respectively.

Examples

The following listing shows PL/SQL block encrypting and decrypting pre-defined '`input_string`' using 256-bit AES algorithm with Cipher Block Chaining and PKCS#5 compliant padding.

```
DECLARE
   input_string       VARCHAR2 (200) :=  'Secret Message';
   output_string      VARCHAR2 (200);
   encrypted_raw      RAW (2000);             -- stores encrypted binary text
   decrypted_raw      RAW (2000);             -- stores decrypted binary text
   num_key_bytes      NUMBER := 256/8;        -- key length 256 bits (32 bytes)
   key_bytes_raw      RAW (32);               -- stores 256-bit encryption key
   encryption_type    PLS_INTEGER :=          -- total encryption type
                         DBMS_CRYPTO.ENCRYPT_AES256
                       + DBMS_CRYPTO.CHAIN_CBC
                       + DBMS_CRYPTO.PAD_PKCS5;
BEGIN
   DBMS_OUTPUT.PUT_LINE ( 'Original string: ' || input_string);
   key_bytes_raw := DBMS_CRYPTO.RANDOMBYTES (num_key_bytes);
   encrypted_raw := DBMS_CRYPTO.ENCRYPT
      (
         src => UTL_I18N.STRING_TO_RAW (input_string,  'AL32UTF8'),
         typ => encryption_type,
         key => key_bytes_raw
      );
    -- The encrypted value "encrypted_raw" can be used here

   decrypted_raw := DBMS_CRYPTO.DECRYPT
      (
         src => encrypted_raw,
         typ => encryption_type,
         key => key_bytes_raw
      );
   output_string := UTL_I18N.RAW_TO_CHAR (decrypted_raw, 'AL32UTF8');

   DBMS_OUTPUT.PUT_LINE ('Decrypted string: ' || output_string);
END;
```

Summary of DBMS_CRYPTO Subprograms

Table 24–10 DBMS_CRYPTO Package Subprograms

Subprogram	Description
DECRYPT Function on page 24-14	Decrypts `RAW` data using a stream or block cipher with a user supplied key and optional IV (initialization vector)
DECRYPT Procedures on page 24-15	Decrypts `LOB` data using a stream or block cipher with a user supplied key and optional IV
ENCRYPT Function on page 24-16	Encrypts `RAW` data using a stream or block cipher with a user supplied key and optional IV
ENCRYPT Procedures on page 24-17	Encrypts `LOB` data using a stream or block cipher with a user supplied key and optional IV
HASH Function on page 24-18	Applies one of the supported cryptographic hash algorithms (MD4, MD5, or SHA-1) to data
MAC Function on page 24-19	Applies Message Authentication Code algorithms (MD5 or SHA-1) to data to provide keyed message protection
RANDOMBYTES Function on page 24-20	Returns a `RAW` value containing a cryptographically secure pseudo-random sequence of bytes, and can be used to generate random material for encryption keys
RANDOMINTEGER Function on page 24-21	Returns a random `BINARY_INTEGER`
RANDOMNUMBER Function on page 24-22	Returns a random 128-bit integer of the `NUMBER` datatype

DECRYPT Function

This function decrypts RAW data using a stream or block cipher with a user supplied key and optional IV (initialization vector).

Syntax

```
DBMS_CRYPTO.DECRYPT(
   src IN RAW,
   typ IN PLS_INTEGER,
   key IN RAW,
   iv  IN RAW          DEFAULT NULL)
 RETURN RAW;
```

Pragmas

```
pragma restrict_references(decrypt,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 24–11 DECRYPT Function Parameters

Parameter Name	Description
src	RAW data to be decrypted.
typ	Stream or block cipher type and modifiers to be used.
key	Key to be used for decryption.
iv	Optional initialization vector for block ciphers. Default is NULL.

Usage Notes

- To retrieve original plaintext data, DECRYPT must be called with the same cipher, modifiers, key, and IV that was used to encrypt the data originally.

  See Also: "Usage Notes" for the ENCRYPT function on page 24-16 for additional information about the ciphers and modifiers available with this package.

- If VARCHAR2 data is converted to RAW before encryption, then it must be converted back to the appropriate database character set by using the UTL_I18N package.

  See Also: "Conversion Rules" on page 24-11 for a discussion of the VARCHAR2 to RAW conversion process.

DECRYPT Procedures

These procedures decrypt LOB data using a stream or block cipher with a user supplied key and optional IV (initialization vector).

Syntax

```
DBMS_CRYPTO.DECRYPT(
   dst IN OUT NOCOPY BLOB,
   src IN            BLOB,
   typ IN            PLS_INTEGER,
   key IN            RAW,
   iv  IN            RAW          DEFAULT NULL);

DBMS_CRYPT.DECRYPT(
   dst IN OUT NOCOPY CLOB         CHARACTER SET ANY_CS,
   src IN            BLOB,
   typ IN            PLS_INTEGER,
   key IN            RAW,
   iv  IN            RAW          DEFAULT NULL);
```

Pragmas

```
pragma restrict_references(decrypt,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 24–12 DECRYPT Procedure Parameters

Parameter Name	Description
dst	LOB locator of output data. The value in the output LOB <dst> will be overwritten.
src	LOB locator of input data.
typ	Stream or block cipher type and modifiers to be used.
key	Key to be used for decryption.
iv	Optional initialization vector for block ciphers. Default is all zeroes.

ENCRYPT Function

This function encrypts RAW data using a stream or block cipher with a user supplied key and optional IV (initialization vector).

Syntax

```
DBMS_CRYPTO.ENCRYPT(
   src IN RAW,
   typ IN PLS_INTEGER,
   key IN RAW,
   iv  IN RAW          DEFAULT NULL)
 RETURN RAW;
```

Pragmas

```
pragma restrict_references(encrypt,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 24–13 ENCRYPT Function Parameters

Parameter Name	Description
src	RAW data to be encrypted.
typ	Stream or block cipher type and modifiers to be used.
key	Encryption key to be used for encrypting data.
iv	Optional initialization vector for block ciphers. Default is NULL.

Usage Notes

- Block ciphers may be modified with chaining and padding type modifiers. The chaining and padding type modifiers are added to the block cipher to produce a cipher suite. Cipher Block Chaining (CBC) is the most commonly used chaining type, and PKCS #5 is the recommended padding type. See Table 24–7 and Table 24–8 on page 24-7 for block cipher chaining and padding modifier constants that have been defined for this package.
- To improve readability, you can define your own package-level constants to represent the cipher suites you use for encryption and decryption. For example, the following example defines a cipher suite that uses DES, cipher block chaining mode, and no padding:

  ```
  DES_CBC_NONE CONSTANT PLS_INTEGER := DBMS_CRYPTO.ENCRYPT_DES
                                     + DBMS_CRYPTO.CHAIN_CBC
                                     + DBMS_CRYPTO.PAD_NONE;
  ```

  See Table 24–6 on page 24-6 for the block cipher suites already defined as constants for this package.
- To encrypt VARCHAR2 data, it should first be converted to the AL32UTF8 character set.

  See Also: "Conversion Rules" on page 24-11 for a discussion of the conversion process.

- Stream ciphers, such as RC4, are not recommended for stored data encryption.

ENCRYPT Procedures

These procedures encrypt LOB data using a stream or block cipher with a user supplied key and optional IV (initialization vector).

Syntax

```
DBMS_CRYPTO.ENCRYPT(
   dst IN OUT NOCOPY BLOB,
   src IN            BLOB,
   typ IN            PLS_INTEGER,
   key IN            RAW,
   iv  IN            RAW          DEFAULT NULL);

DBMS_CRYPTO.ENCRYPT(
   dst IN OUT NOCOPY BLOB,
   src IN            CLOB         CHARACTER SET ANY_CS,
   typ IN            PLS_INTEGER,
   key IN            RAW,
   iv  IN            RAW          DEFAULT NULL);
```

Pragmas

```
pragma restrict_references(encrypt,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 24–14 ENCRYPT Procedure Parameters

Parameter Name	Description
dst	LOB locator of output data. The value in the output LOB <dst> will be overwritten.
src	LOB locator of input data.
typ	Stream or block cipher type and modifiers to be used.
key	Encryption key to be used for encrypting data.
iv	Optional initialization vector for block ciphers. Default is NULL.

Usage Notes

See "Conversion Rules" on page 24-11 for usage notes about using the ENCRYPT procedure.

HASH Function

A one-way hash function takes a variable-length input string, the data, and converts it to a fixed-length (generally smaller) output string called a *hash value*. The hash value serves as a unique identifier (like a fingerprint) of the input data. You can use the hash value to verify whether data has been changed or not.

Note that a one-way hash function is a hash function that works in one direction. It is easy to compute a hash value from the input data, but it is hard to generate data that hashes to a particular value. Consequently, one-way hash functions work well to ensure data integrity. Refer to "When to Use Hash or Message Authentication Code (MAC) Functions" on page 24-10 for more information about using one-way hash functions.

This function applies to data one of the supported cryptographic hash algorithms listed in Table 24–3 on page 24-6.

Syntax

```
DBMS_CRYPTO.Hash (
   src IN RAW,
   typ IN PLS_INTEGER)
 RETURN RAW;

DBMS_CRYPTO.Hash (
   src IN BLOB,
   typ IN PLS_INTEGER)
 RETURN RAW;

DBMS_CRYPTO.Hash (
   src IN CLOB CHARACTER SET ANY_CS,
   typ IN PLS_INTEGER)
 RETURN RAW;
```

Pragmas

```
pragma restrict_references(hash,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 24–15 HASH Function Parameters

Parameter Name	Description
`src`	The source data to be hashed.
`typ`	The hash algorithm to be used.

Usage Note

Oracle recommends that you use the SHA-1 (Secure Hash Algorithm), specified with the constant, `HASH_SH1`, because it is more resistant to brute-force attacks than MD4 or MD5. If you must use a Message Digest algorithm, then MD5 provides greater security than MD4.

MAC Function

A Message Authentication Code, or MAC, is a key-dependent one-way hash function. MACs have the same properties as the one-way hash function described in "HASH Function" on page 24-18, but they also include a key. Only someone with the identical key can verify the hash. Also refer to "When to Use Hash or Message Authentication Code (MAC) Functions" on page 24-10 for more information about using MACs.

This function applies MAC algorithms to data to provide keyed message protection. See Table 24–4 on page 24-6 for a list of MAC algorithms that have been defined for this package.

Syntax

```
DBMS_CRYPTO.MAC (
   src IN RAW,
   typ IN PLS_INTEGER,
   key IN RAW)
 RETURN RAW;

DBMS_CRYPTO.MAC (
   src IN BLOB,
   typ IN PLS_INTEGER
   key IN RAW)
 RETURN RAW;

DBMS_CRYPTO.MAC (
   src IN CLOB CHARACTER SET ANY_CS,
   typ IN PLS_INTEGER
   key IN RAW)
 RETURN RAW;
```

Pragmas

```
pragma restrict_references(mac,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 24–16 MAC Function Parameters

Parameter Name	Description
src	Source data to which MAC algorithms are to be applied.
typ	MAC algorithm to be used.
key	Key to be used for MAC algorithm.

RANDOMBYTES Function

This function returns a RAW value containing a cryptographically secure pseudo-random sequence of bytes, which can be used to generate random material for encryption keys. The RANDOMBYTES function is based on the RSA X9.31 PRNG (Pseudo-Random Number Generator), and it draws its entropy (seed) from the sqlnet.ora file parameter SQLNET.CRYPTO_SEED.

Syntax

```
DBMS_CRYPTO.RANDOMBYTES (
   number_bytes IN POSITIVE)
 RETURN RAW;
```

Pragmas

```
pragma restrict_references(randombytes,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 24–17 RANDOMBYTES Function Parameter

Parameter Name	Description
number_bytes	The number of pseudo-random bytes to be generated.

Usage Note

- The number_bytes value should not exceed the maximum length of a RAW variable.
- The SQLNET.CRYPTO_SEED parameter can be set by entering 10 to 70 random characters with the following syntax in the sqlnet.ora file:

  ```
  SQLNET.CRYPTO_SEED = <10 to 70 random characters>
  ```

  See Also: *Oracle Database Advanced Security Administrator's Guide* for more information about the SQLNET.CRYPTO_SEED parameter and its use.

RANDOMINTEGER Function

This function returns an integer in the complete range available for the Oracle `BINARY_INTEGER` datatype.

Syntax

```
DBMS_CRYPTO.RANDOMINTEGER
 RETURN BINARY_INTEGER;
```

Pragmas

```
pragma restrict_references(randominteger,WNDS,RNDS,WNPS,RNPS);
```

RANDOMNUMBER Function

This function returns an integer in the Oracle `NUMBER` datatype in the range of [0..2**128-1].

Syntax

```
DBMS_CRYPTO.RANDOMNUMBER
 RETURN NUMBER;
```

Pragmas

```
pragma restrict_references(randomnumber,WNDS,RNDS,WNPS,RNPS);
```

25

DBMS_DATA_MINING

Oracle Data Mining (ODM) is designed for programmers, systems analysts, project managers, and others who develop data mining applications. Data mining discovers hidden patterns within the data and uses that knowledge to make predictions and summaries.

The `DBMS_DATA_MINING` package is an interface to ODM. With `DBMS_DATA_MINING`, you can build a mining model, test the model, and apply the model to your data.

See Also:

- Chapter 26, "DBMS_DATA_MINING_TRANSFORM". This package supports data pre-processing for data mining.
- Chapter 72, "DBMS_PREDICTIVE_ANALYTICS". This package automates the entire process of predictive data mining, from data preprocessing through model building to scoring new data.
- *Oracle Database SQL Reference* for information about the SQL scoring functions for data mining.
- *Oracle Data Mining Administrator's Guide* for information about sample data mining programs.
- *Oracle Data Mining Application Developer's Guide* for information about developing data mining applications in SQL and Java.

This chapter contains the following topics:

Using DBMS_DATA_MINING

This section contains topics which relate to using the `DBMS_DATA_MINING` package.

Overview

Oracle Data Mining (ODM) embeds data mining functionality in the Oracle Database. The data never leaves the database — the data, its preparation, model building, and model scoring (applying) all remain in the database. This enables Oracle to provide an infrastructure for data analysts and application developers to integrate data mining seamlessly with database applications.

Oracle Data Mining Concepts

ODM supports both predictive and descriptive data mining. Predictive data mining uses an historical model to predict a target value. Descriptive data mining identifies natural groupings within a given data set.

Predictive data mining functions include:

- Classification
- Regression
- Attribute Importance

Descriptive data mining functions include:

- Clustering
- Association
- Feature Extraction

The steps you use to build and score a model depend on the data mining function and the algorithm being used. To learn more about ODM functions and algorithms, refer to *Oracle Data Mining Concepts*.

Oracle Data Mining APIs

ODM provides a graphical user interface (Oracle Data Miner), as well as application programming interfaces for SQL and Java. The SQL interface consists of PL/SQL packages and SQL functions. The Java interface is an Oracle implementation of the JDM 1.0 standard for data mining. The SQL and Java APIs are fully interoperable.

The SQL functions for data mining, new in 10*g* Release 2 (10.2), return the results of model scoring. These functions apply pre-existing models within the context of a SQL statement. The ODM scoring functions include: `CLUSTER_ID`, `CLUSTER_PROBABILITY`, `CLUSTER_SET`, `FEATURE_ID`, `FEATURE_SET`, `FEATURE_VALUE`, `PREDICTION`, `PREDICTION_COST`, `PREDICTION_DETAILS`, `PREDICTION_PROBABILITY`, `PREDICTION_SET`. The ODM scoring functions are documented in *Oracle Database SQL Reference*.

The `DBMS_DATA_MINING` package supports the process of building, testing, and scoring models for all ODM mining functions. Most mining data requires preprocessing before mining activities can begin. For this, you can use the `DBMS_DATA_MINING_TRANSFORM` package or third-party utilities. To automate the entire process of predictive data mining, use the `DBMS_PREDICTIVE_ANALYTICS` package.

> **See Also:** Sample data mining programs are available with Oracle Data Mining. Instructions for using the sample programs are provided in the *Oracle Data Mining Administrator's Guide*. Additional information about the Oracle Data Mining interfaces is available in the *Oracle Data Mining Application Developer's Guide*.

New Functionality

In Oracle Database 10*g* Release 2, the `DBMS_DATA_MINING` package includes the following new functionality:

- Decision Tree algorithm for classification
- Orthogonal Partitioning Clustering (O-Cluster) algorithm for clustering
- One-Class Support Vector Machine, which supports Anomaly Detection
- Active learning for Support Vector Machine

For detailed information about new features in Oracle Data Mining, see *Oracle Data Mining Concepts* and *Oracle Database New Features*

Model Names

The names of ODM models must be valid schema object names. However, the naming rules for models are more restrictive than the naming rules for schema objects. A model name must satisfy the following additional requirements:

- It must be 25 or fewer characters long.
- It must be a nonquoted identifier. Oracle requires that nonquoted identifiers contain only alphanumeric characters, the underscore (_), dollar sign ($), and pound sign (#); the initial character must be alphabetic. Oracle strongly discourages the use of the dollar sign and pound sign in nonquoted literals.

Naming requirements for schema objects are fully documented in *Oracle Database SQL Reference*.

Constants

Oracle Data Mining uses constants to specify the mining function, its algorithm, and other details about a model. The function, or type, of a model is specified when the model is created. Non-default characteristics of the model are specified in a settings table associated with the model.

The settings table is a user-created table with the following columns:

```
(setting_name        VARCHAR2(30),
 setting_value       VARCHAR2(128))
```

Each setting in the `setting_name` column is a constant, and many of the values that can be specified in the `setting_value` column are also constants. Numeric values are implicitly converted to `VARCHAR2`. To explicitly convert them, use the `TOCHAR` function.

> **See Also:** *Oracle Data Mining Application Developer's Guide* for information about creating a settings table, and for default setting values and ranges.

Constants that Specify the Mining Function

Oracle Data Mining supports a number of predictive and descriptive mining functions. The mining function is specified as a parameter to the `CREATE_MODEL` procedure. See "CREATE_MODEL Procedure" on page 25-30 for more information.

The `mining_function` parameter has a `VARCHAR2(30)` data type; it can have the values listed in Table 25–1.

Table 25–1 Mining Functions

Constant	Description
`association`	Association is a descriptive mining function. An association model identifies relationships and the probability of their occurrence within a data set.
`attribute_importance`	Attribute Importance is a predictive mining function. An attribute importance model identifies the relative importance of an attribute in predicting a given outcome.
`classification`	Classification is a predictive mining function. A classification model uses historical data to predict new discrete or categorical data. The `classification` function can also be used for anomaly detection. In this case, the SVM algorithm with a null target is used (One-Class SVM).
`clustering`	Clustering is a descriptive mining function. A clustering model identifies natural groupings within a data set.
`feature_extraction`	Feature Extraction is a descriptive mining function. A feature extraction model creates an optimized data set on which to base a model.
`regression`	Regression is a predictive mining function. A regression model uses historical data to predict new continuous, numerical data.

Constants that Specify Information About Mining Functions

Every model is based on one of the mining functions described in Table 25–1. You can configure the mining function with the settings described in Table 25–2.

Table 25–2 Mining Function Settings

Constant	Description
`algo_name`	Setting that specifies the algorithm used by the model. The following constants can be values for this setting:
`algo_adaptive_bayes_network`	Adaptive Bayes Network (ABN), the default algorithm for classification.
`algo_ai_mdl`	Minimum Description Length (MDL) algorithm for attribute importance.
`algo_apriori_association_rules`	Apriori algorithm for association.
`algo_decision_tree`	Decision Tree (DT) algorithm for classification.
`algo_kmeans`	*k*-Means (KM), the default algorithm for clustering.
`algo_naive_bayes`	Naive Bayes (NB) algorithm for classification.
`algo_nonnegative_matrix_factor`	Non-Negative Matrix Factorization (NMF) algorithm for feature selection.
`algo_o_cluster`	O-Cluster (OC) algorithm for clustering.
`algo_support_vector_machines`	Support Vector Machine (SVM) algorithm for classification or regression.
	One-Class SVM, used for anomaly detection, is SVM classification with a null target.
`asso_max_rule_length`	Setting that specifies the maximum length of a rule used by an association model.
`asso_min_confidence`	Setting that specifies the minimum confidence for an association model.
`asso_min_support`	Setting that specifies the minimum support for an association model.
`clas_cost_table_name`	Setting that specifies the name of the cost matrix table for a classification model.
`clas_priors_table_name`	Setting that specifies the name of the prior probability table for NB and ABN models. Decision Tree is the only classification algorithm that does not use priors.
	For SVM classification models, this setting specifies the name of a table of weights.
`clus_num_clusters`	Setting that specifies the number of clusters for a clustering model.
`feat_num_features`	Setting that specifies the number of features for a feature selection model.

Constants that Specify Information About Algorithms

The algorithm used by a model is specified by the `algo_name` setting (described in Table 25–2). You can configure the algorithm with the settings described in Table 25–3.

Table 25–3 Algorithm Settings

Algorithm	Constant	Description
ABN	abns_max_build_minutes	Setting that specifies the maximum time threshold to complete an ABN model build.
	abns_max_nb_predictors	Setting that specifies the maximum number of predictors, measured by their MDL ranking, to be considered for building an ABN model of type abns_naive_bayes.
	abns_max_predictors	Setting that specifies the maximum number of predictors, measured by their MDL ranking, to be considered for building an ABN model of type abns_single_feature or abns_multi_feature.
	abns_model_type	Setting that specifies the ABN model type. The following constants can be values for this setting:
	abns_multi_feature	Multifeature ABN model.
	abns_naive_bayes	Naive Bayes ABN model.
	abns_single_feature	Single feature ABN model.
DT	tree_impurity_metric	Setting that specifies the tree impurity metric for Decision Tree. The following constants can be values for this setting:
	tree_impurity_entropy	Entropy.
	tree_impurity_gini	Gini.
	tree_term_max_depth	Setting that specifies the maximum tree depth termination criteria for Decision Tree.
	tree_term_minpct_node	Setting name that specifies the minimum percent of records in a parent node termination criteria for Decision Tree.
	tree_term_minpct_split	Setting name that specifies the minimum percent of records in a parent node splitting criteria for Decision Tree.
	tree_term_minrec_node	Setting name that specifies the minimum number of records in a parent node termination criteria for Decision Tree.
	tree_term_minrec_split	Setting that specifies the minimum number of records in a parent node splitting criteria for Decision Tree.
KM	kmns_block_growth	Setting that specifies the growth factor for memory allocated to hold cluster data for *k*-Means.
	kmns_conv_tolerance	Setting that specifies the convergence tolerance for *k*-Means.
	kmns_distance	Setting that specifies the distance function for *k*-Means. The following constants can be values for this setting:
	kmns_cosine	Cosine distance function.
	kmns_euclidean	Euclidean distance function.
	kmns_fast_cosine	Fast cosine distance function.
	kmns_iterations	Setting that specifies the number of iterations for *k*-Means.
	kmns_min_pct_attr_support	Setting that specifies the minimum percentage support required for attributes in rules for *k*-Means clusters.
	kmns_num_bins	Setting that specifies the number of histogram bins *k*-Means.

Table 25–3 (Cont.) Algorithm Settings

Algorithm	Constant	Description
	kmns_split_criterion	Setting that specifies the split criterion for *k*-Means. The following constants can be values for this setting:
	kmns_size	Size as the split criterion.
	kmns_variance	Variance as the split criterion.
NB	nabs_pairwise_threshold	Setting that specifies pair-wise threshold for Naive Bayes.
	nabs_singleton_threshold	Setting that specifies singleton threshold for Naive Bayes.
NMF	nmfs_conv_tolerance	Setting that specifies convergence tolerance for NMF.
	nmfs_num_iterations	Setting that specifies the number of iterations for NMF.
	nmfs_random_seed	Setting that specifies the random seed for NMF.
OC	oclt_max_buffer	Setting that specifies buffer size for O-Cluster.
	oclt_sensitivity	Setting that specifies sensitivity for O-Cluster.
SVM	svms_active_learning	Setting that specifies whether active learning is enabled or disabled. The following constants can be values for this setting:
	svms_al_disable	Active learning is disabled.
	svms_al_enable	Active learning is enabled.
	svms_complexity_factor	Setting that specifies the complexity factor for SVM.
	svms_conv_tolerance	Setting that specifies tolerance for SVM.
	svms_epsilon	Setting that specifies epsilon for SVM Regression.
	svms_kernel_cache_size	Setting that specifies the Gaussian kernel cache size for SVM.
	svms_kernel_function	Setting that specifies the kernel function for SVM. The following constants can be values for this setting:
	svms_gaussian	Gaussian kernel.
	svms_linear	Linear kernel.
	svms_outlier_rate	Setting that specifies the desired rate of outliers in the training data. Valid for One-Class SVM models only.
	svms_std_dev	Setting that specifies standard deviation for SVM Gaussian kernel.

Data Types

The DBMS_DATA_MINING package includes a number of table functions that return algorithm-specific information about models. These functions take a model name as input and return the requested information as a collection of rows. The table functions are named GET_*n*, where *n* identifies the type of information to return. For a list of the ODM GET functions, see "Summary of DBMS_DATA_MINING Subprograms" on page 25-15.

All the GET functions use **pipelining**, which causes each row of output to be materialized as it is read from model storage, without waiting for the generation of the complete table object. For more information on pipelined, parallel table functions, consult the *Oracle Database PL/SQL User's Guide and Reference.*

The virtual table returned by the GET functions is an object data type. Another object data type defines the rows. Some of the columns have object data types that define nested tables.

ODM also uses object data types for handling wide data. These types, DM_NESTED_NUMERICALS and DM_NESTED_CATEGORICALS define nested tables that can be used for storing a set of mining attributes in a single column. For more information on wide data, see the *Oracle Data Mining Application Developer's Guide*.

The ODM object data types are described in Table 25–4.

Table 25–4 DBMS_DATA_MINING Summary of Data Types

Data Type	Purpose
DM_ABN_DETAIL	Represents information about an ABN model.
DM_ABN_DETAILS	Represents a collection of DM_ABN_DETAIL. It is returned by GET_MODEL_DETAILS_ABN.
DM_CENTROID	Represents the centroid of a cluster.
DM_CENTROIDS	Represents a collection of DM_CENTROID. It is returned by GET_MODEL_DETAILS_KM and GET_MODEL_DETAILS_OC.
DM_CHILD	Represents a child node of a cluster.
DM_CHILDREN	Represents a collection of DM_CHILD.
DM_CLUSTER	Represents a cluster.
DM_CLUSTERS	Represents a collection of DM_CLUSTER. It is returned by GET_MODEL_DETAILS_KM and GET_MODEL_DETAILS_OC.
DM_CONDITIONAL	Represents a conditional probability associated with a mining attribute used in an NB or ABN model.
DM_CONDITIONALS	Represents a collection of DM_CONDITIONAL. It is returned by GET_MODEL_DETAILS_NB and GET_MODEL_DETAILS_ABN.
DM_HISTOGRAM_BIN	Represents a histogram associated with a cluster identifier.
DM_HISTOGRAMS	Represents a collection of DM_HISTOGRAM_BIN. It is returned by GET_MODEL_DETAILS_KM.
DM_ITEMS	Represents items.
DM_ITEMSET	Represents a collection of DM_ITEMS.
DM_ITEMSETS	Represents a collection of DM_ITEMSET. These are frequent sets of items in Association models.

Table 25–4 (Cont.) DBMS_DATA_MINING Summary of Data Types

Data Type	Purpose
DM_MODEL_SETTING	Represents a setting/value combination from the settings table for the model.
DM_MODEL_SETTINGS	Represents a collection of DM_MODEL_SETTING. It is returned by GET_MODEL_SETTINGS.
DM_MODEL_SIGNATURE_ ATTRIBUTE	Represents an attribute of the model signature.
DM_MODEL_SIGNATURE	Represents a collection of DM_MODEL_SIGNATURE. It is returned by GET_MODEL_SIGNATURE.
DM_NB_DETAIL	Represents information about an NB model.
DM_NB_DETAILS	Represents a collection of DM_DB_DETAIL. It is returned by GET_MODEL_DETAILS_NB.
DM_NESTED_CATEGORICAL	Represents a nested table of categorical attributes.
DM_NESTED_CATEGORICALS	Represents a collection of DM_NESTED_CATEGORICAL. It is used for representing wide data.
DM_NESTED_NUMERICAL	Represents a nested table of numerical attributes.
DM_NESTED_NUMERICALS	Represents a collection of DM_NESTED_NUMERICAL. It is used for representing wide data.
DM_NMF_ATTRIBUTE	Represents a mining attribute for an NMF model.
DM_NMF_ATTRIBUTE_SET	Represents a collection of DM_NMF_ATTRIBUTE. It is returned by GET_MODEL_DETAILS_NMF.
DM_NMF_FEATURE	Represents a feature in an NMF model.
DM_NMF_FEATURE_SET	Represents a collection of DM_NMF_FEATURE. It is returned by GET_MODEL_DETAILS_NMF.
DM_PREDICATE	Represents either the antecedent or the consequent of a rule.
DM_PREDICATES	Represents a collection of DM_PREDICATE.
DM_RANKED_ATTRIBUTE	Represents an entry in the set of attributes ranked by the attribute's importance.
DM_RANKED_ATTRIBUTES	Represents a collection of DM_RANKED_ATTRIBUTE. It is returned by GET_MODEL_DETAILS_AI.
DM_RULE	Represents a model rule.
DM_RULES	Represents a collection of DM_RULE. It is returned by GET_ ASSOCIATION_RULES for *k*-means models, by GET_ MODEL_DETAILS_KM, and by GET_MODEL_DETAILS_OC.
DM_SVM_ATTRIBUTE	Represents an attribute for an SVM model.
DM_SVM_ATTRIBUTE_SET	Represents a collection of DM_SVM_ATTRIBUTE. It is returned by GET_MODEL_DETAILS_SVM for a linear model.
DM_SVM_LINEAR_COEFF	Represents an SVM linear coefficient.
DM_SVM_LINEAR_COEFF_SET	Represents a collection of DM_SVM_LINEAR_COEFF. It is returned by GET_MODEL_DETAILS_SVM for an SVM model built using the linear kernel.

Exceptions

The following table lists the exceptions raised by DBMS_DATA_MINING.

Table 25–5 Exceptions raised by DBMS_DATA_MINING

Oracle Error	Description
ORA-40201	Invalid input parameter %s
ORA-40202	Column %s does not exist in the input table %s
ORA-40203	Model %s does not exist
ORA-40204	Model %s already exists
ORA-40205	Invalid setting name %s
ORA-40206	Invalid setting value for setting name %s
ORA-40207	Duplicate or multiple function settings
ORA-40208	Duplicate or multiple algorithm settings for function %s
ORA-40209	Setting % is invalid for % function
ORA-40211	Algorithm name %s is invalid
ORA-40212	Invalid target data type in input data for %s function
ORA-40213	Contradictory values for settings: %s, %s
ORA-40214	Duplicate setting: %s
ORA-40215	Model %s is incompatible with current operation
ORA-40216	Feature not supported
ORA-40217	Priors table mismatched with training data
ORA-40219	Apply result table %s is incompatible with current operation
ORA-40220	Maximum number of attributes exceeded
ORA-40221	Maximum target cardinality exceeded
ORA-40222	Data mining model export failed, job name=%s, error=%s
ORA-40223	Data mining model import failed, job name=%s, error=%s
ORA-40225	Model is currently in use by another process
ORA-40226	Model upgrade/downgrade must be performed by SYS
ORA-40251	No support vectors were found
ORA-40252	No target values were found
ORA-40253	No target counter examples were found
ORA-40254	Priors cannot be specified for one-class models
ORA-40261	Input data for model build contains negative values
ORA-40262	NMF: number of features not between [1, %s]
ORA-40271	No statistically significant features were found
ORA-40272	Apply rules prohibited for this model mode
ORA-40273	Invalid model type %s for Adaptive Bayes network algorithm
ORA-40281	Invalid model name
ORA-40282	Invalid cost matrix

Table 25–5 (Cont.) Exceptions raised by DBMS_DATA_MINING

Oracle Error	Description
ORA-40283	Missing cost matrix
ORA-40284	Model does not exist
ORA-40285	Label not in the model
ORA-40286	Remote operations not permitted on mining models
ORA-40287	Invalid data for model -- cosine distance out of bounds
ORA-40289	Duplicate attributes provided for data mining function
ORA-40290	Model incompatible with data mining function
ORA-40291	Model cost not available
ORA-40301	Invalid cost matrix specification
ORA-40302	Invalid classname %s in cost matrix specification
ORA-40303	Invalid prior probability specification
ORA-40304	Invalid classname %s in prior probability specification
ORA-40305	Invalid impurity metric specified
ORA-40306	Wide data not supported for decision tree model create
ORA-40321	Invalid bin number, is zero or negative value
ORA-40322	Bin number too large

User Views

Table 25–6 describes the DM_USER_MODELS view, which provides information about the models in the user's schema.

Table 25–6 DM_USER_MODELS

Column	Data Type	NULL	Description
name	VARCHAR2(25)	NOT NULL	Name of the model
function_name	VARCHAR2(30)		The model function. See Table 25–1.
algorithm_name	VARCHAR2(30)		The algorithm used by the model. See Table 25–2.
creation_date	DATE		The date on which the model was created
build_duration	NUMBER		The duration of the model build process
target_attribute	VARCHAR2(30)		The attribute designated as the target of a classification model
model_size	NUMBER		The size of the model in megabytes

Summary of DBMS_DATA_MINING Subprograms

Table 25–7 summarizes the subprograms included in the DBMS_DATA_MINING package.

Table 25–7 DBMS_DATA_MINING Package Subprograms

Data Type	Purpose
APPLY Procedure on page 25-17	Applies a model to a data set (scores the data)
COMPUTE_CONFUSION_MATRIX Procedure on page 25-20	Computes the confusion matrix from the APPLY results on test data for a classification model; also provides the accuracy of the model
COMPUTE_LIFT Procedure on page 25-23	Computes lift for a given positive target value from the APPLY results on test data for a classification model
COMPUTE_ROC Procedure on page 25-26	Computes Receiver Operating Characteristic (ROC) for a classification model
CREATE_MODEL Procedure on page 25-30	Creates (builds) a mining model
DROP_MODEL Procedure on page 25-33	Drops a model
EXPORT_MODEL Procedure on page 25-34	Exports a model into a dump file
GET_ASSOCIATION_RULES Function on page 25-37	Returns the rules from an Association model
GET_DEFAULT_SETTINGS Function on page 25-40	Returns all the default settings for all mining functions and algorithms
GET_FREQUENT_ITEMSETS Function on page 25-41	Returns the frequent itemsets from an Association model
GET_MODEL_DETAILS_ABN Function on page 25-43	Returns the details of an Adaptive Bayes Network model
GET_MODEL_DETAILS_AI Function on page 25-45	Returns the details of an Attribute Importance model
GET_MODEL_DETAILS_KM Function on page 25-46	Returns the details of a *k*-Means model
GET_MODEL_DETAILS_NB Function on page 25-49	Returns the details of a Naive Bayes model
GET_MODEL_DETAILS_NMF Function on page 25-51	Returns the details of an NMF model
GET_MODEL_DETAILS_OC Function on page 25-52	Returns the details of an O-Cluster model
GET_MODEL_DETAILS_SVM Function on page 25-55	Returns the details of a SVM model with a linear kernel
GET_MODEL_DETAILS_XML Function on page 25-57	Returns the details of a decision tree model
GET_MODEL_SETTINGS Function on page 25-58	Returns the settings used to build a model
GET_MODEL_SIGNATURE Function on page 25-59	Returns the signature of a model

Table 25–7 (Cont.) DBMS_DATA_MINING Package Subprograms

Data Type	Purpose
IMPORT_MODEL Procedure on page 25-60	Imports a specified model into a user schema
RANK_APPLY Procedure on page 25-63	Ranks the predictions from the `APPLY` results for a classification model
RENAME_MODEL Procedure on page 25-66	Renames a model

APPLY Procedure

This procedure applies a mining model to the data of interest, and generates the APPLY results in a table. The APPLY operation is also referred to as **scoring**.

For predictive mining functions, the APPLY operation generates predictions in a target column. For descriptive mining functions such as clustering, the APPLY operation assigns each case to a cluster with a probability.

The APPLY operation is not applicable to association models and attribute importance models.

> **Note:** You can use the ODM scoring functions as an alternative to the DBMS_DATA_MINING.APPLY procedure. These SQL functions are documented in the *Oracle Database SQL Reference*. Additional information and code samples are provided in the *Oracle Data Mining Application Developer's Guide*.

Syntax

```
DBMS_DATA_MINING.APPLY (
      model_name           IN VARCHAR2,
      data_table_name      IN VARCHAR2,
      case_id_column_name  IN VARCHAR2,
      result_table_name    IN VARCHAR2,
      data_schema_name     IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 25–8 APPLY Procedure Parameters

Parameter	Description
model_name	Name of the model
data_table_name	Name of table or view representing data to be scored
case_id_column_name	Name of the case identifier column
result_table_name	Name of the table to store apply results
data_schema_name	Name of the schema containing the data to be scored

Usage Notes

The data provided for APPLY should match the data provided to CREATE_MODEL in terms of the schema definition and relevant content. The GET_MODEL_SIGNATURE function provides this information. If the data provided as input to CREATE_MODEL has been pre-processed, then the data input to APPLY must be pre-processed in the same way. The case identifier is not considered to be a mining attribute during APPLY.

You must provide the name of the table in which the results of the apply operation are to be stored. APPLY creates a table with an algorithm-specific fixed schema in the user schema that owns the model.

The behavior of an APPLY operation is analogous to a SQL query operation, even though it is packaged as a procedure. It does not update the model contents and does not have any contention with CREATE_MODEL, DROP_MODEL, or RENAME_MODEL operations. The corollary is that if you potentially drop or rename a model while a

model is being applied to scoring data, the APPLY operation may discontinue with partial or unpredictable results.

The schema for the apply results from each of the supported algorithms is listed in subsequent sections. The case_id column will match the case identifier column name provided by you. The type of incoming case_id column is preserved in APPLY output.

Classification Algorithms

The table containing the APPLY results for all classification models has the same definition. For numerical targets, the results table will have the following columns.

```
case_id      VARCHAR2/NUMBER
prediction   NUMBER
probability  NUMBER
```

For categorical targets, the results table will have the following columns.

```
case_id      VARCHAR2/NUMBER
prediction   VARCHAR2
probability  NUMBER
```

One-Class SVM (Anomaly Detection)

The results table will have the following columns.

```
case_id       VARCHAR2/NUMBER
prediction    NUMBER
probability   NUMBER
```

Values in the prediction column can be either 0 or 1. When the prediction is 1, the case is a typical example. When the prediction is 0, the case is an outlier.

Regression using SVM

The results table will have the following columns.

```
case_id     VARCHAR2/NUMBER
prediction  NUMBER
```

Clustering using *k*-Means and O-Cluster

Clustering is an unsupervised mining function, and hence there are no targets. The results of an APPLY operation will contain simply the cluster identifier corresponding to a case, and the associated probability. The results table will have the following columns.

```
case_id      VARCHAR2/NUMBER
cluster_id   NUMBER
probability  NUMBER
```

Feature Extraction using NMF

Feature extraction is also an unsupervised mining function, and hence there are no targets. The results of an APPLY operation will contain simply the feature identifier corresponding to a case, and the associated match quality. The results table will have the following columns

```
case_id        VARCHAR2/NUMBER
feature_id     NUMBER
match_quality  NUMBER
```

Examples

```
BEGIN
/* build a model with name census_model.
 * (See example under CREATE_MODEL)
 */

/* if build data was pre-processed in any manner,
 * perform the same pre-processing steps on the
 * scoring data also.
 * (See examples in the section on DBMS_DATA_MINING_TRANSFORM)
 */

/* apply the model to data to be scored */
DBMS_DATA_MINING.APPLY(
  model_name          => 'census_model',
  data_table_name     => 'census_2d_apply',
  case_id_column_name => 'person_id',
  result_table_name   => 'census_apply_result');
END;
/

-- View Apply Results
SELECT case_id, prediction, probability
  FROM census_apply_result;
```

COMPUTE_CONFUSION_MATRIX Procedure

This procedure computes the confusion matrix for a classification model and also provides the accuracy of the model. See *Oracle Data Mining Concepts* for a description of confusion matrix.

Before executing a COMPUTE_CONFUSION_MATRIX procedure:

- Apply the model on the test data
- Create a target table or view containing only the case identifier and target columns from the test data

You will specify this table or view and the apply results table as input to the procedure.

Syntax

```
DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX (
      accuracy                    OUT NUMBER,
      apply_result_table_name     IN  VARCHAR2,
      target_table_name           IN  VARCHAR2,
      case_id_column_name         IN  VARCHAR2,
      target_column_name          IN  VARCHAR2,
      confusion_matrix_table_name IN  VARCHAR2,
      score_column_name           IN  VARCHAR2 DEFAULT 'PREDICTION',
      score_criterion_column_name IN  VARCHAR2 DEFAULT 'PROBABILITY',
      cost_matrix_table_name      IN  VARCHAR2 DEFAULT NULL,
      apply_result_schema_name    IN  VARCHAR2 DEFAULT NULL,
      target_schema_name          IN  VARCHAR2 DEFAULT NULL,
      cost_matrix_schema_name     IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 25–9 COMPUTE_CONFUSION_MATRIX Procedure Parameters

Parameter	Description
accuracy	Accuracy of the model
apply_result_table_name	Name of the table containing the results of an APPLY operation on the test dataset (see Usage Notes)
target_table_name	Name of the table or view containing only the case identifier column and target column values (see Usage Notes)
case_id_column_name	Name of the case identifier column in the test data set. This must be common across the target table and the apply results table.
target_column_name	Name of the target column in the target table
confusion_matrix_table_name	Name of the table into which the confusion matrix is to be generated
score_column_name	Name of the column representing the score from the apply results table. In the fixed schema table generated by APPLY, this column has the name PREDICTION, which is the default.
score_criterion_column_name	Name of the column representing the ranking factor for the score from the apply results table. In the fixed schema table generated by APPLY for classification models, this column has the name PROBABILITY, which is the default. Values in this column must be represented numerically.

Table 25–9 (Cont.) COMPUTE_CONFUSION_MATRIX Procedure Parameters

Parameter	Description
cost_matrix_table_name	Name of the fixed-schema cost matrix table
apply_result_schema_name	Name of the schema hosting the APPLY results table
target_schema_name	Name of the schema hosting the targets table
cost_matrix_schema_name	Name of the schema hosting the cost matrix table

Usage Notes

You can also provide a cost matrix as an optional input in order to have the cost of predictions reflected in the results.

It is important to note that the inputs to COMPUTE_CONFUSION_MATRIX do not always have to be generated using APPLY. As long as the definition of the two input tables matches the ones discussed in this section, with appropriate content, the procedure can produce the confusion matrix and accuracy. The quality of the results depends on the quality of the data.

The data provided for testing your classification model must match the data provided to CREATE_MODEL in schema and relevant content. If the data provided as input to CREATE_MODEL has been pre-processed, then the data input to APPLY must also be pre-processed using the statistics from the CREATE_MODEL data pre-processing.

Before you use the COMPUTE_CONFUSION_MATRIX procedure, you must prepare two data input streams from your test data.

First, you must APPLY the model on your test data. Use the result table name from APPLY as apply_result_table_name in the COMPUTE_CONFUSION_MATRIX procedure.

Next, you must create a table or view containing only the case identifier column and the target column in its schema. Use the name of this second table as target_table_name.

The definition for the second view or table name for a numerical target attribute is:

```
(case_identifier_column_name  VARCHAR2/NUMBER,
target_column_name            NUMBER)
```

The definition for the second view or table name for a categorical target attribute is:

```
(case_identifier_column_name  VARCHAR2/NUMBER,
target_column_name            NUMBER)
```

You must provide the name of the table in which the confusion matrix is to be generated. The resulting fixed schema table will always be created in the schema owning the model.

For numerical target attributes, the confusion matrix table will have the definition:

```
(actual_target_value    NUMBER,
predicted_target_value  NUMBER,
value                   NUMBER)
```

For categorical target attributes, the confusion matrix table will have the definition:

```
(actual_target_value     VARCHAR2,
```

```
predicted_target_value   VARCHAR2,
value                    NUMBER)
```

Examples

Assume that you have built a classification model `census_model` using the Naive Bayes algorithm, and you have been provided the test data in a table called `census_2d_test`, with case identifier column name `person_id`, and the target column name `class`.

```
DECLARE
  v_sql_stmt VARCHAR2(4000);
  v_accuracy NUMBER;
BEGIN

/* apply the model census_model on test data */
DBMS_DATA_MINING.APPLY(
  model_name           => 'census_model',
  data_table_name      => 'census_2d_test',
  case_id_column_name  => 'person_id',
  result_table_name    => 'census_test_result');
CREATE VIEW census_2d_test_view as select person_id, class from census_2d_test;

/* now compute the confusion matrix from the two
 * data streams, also providing a cost matrix as input.
 */
DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX (
  accuracy                     => v_accuracy,
  apply_result_table_name      => 'census_test_result',
  target_table_name            => 'census_2d_test_view',
  case_id_column_name          => 'person_id',
  target_column_name           => 'class',
  confusion_matrix_table_name  => 'census_confusion_matrix',
  cost_matrix_table_name       => 'census_cost_matrix');
DBMS_OUTPUT.PUT_LINE('Accuracy of the model: ' || v_accuracy);
END;
/

-- View the confusion matrix using Oracle SQL
SELECT actual_target_value, predicted_target_value, value
  FROM census_confusion_matrix;
```

COMPUTE_LIFT Procedure

This procedure computes a lift table for a given positive target for a classification model. See *Oracle Data Mining Concepts* for a description of lift.

Before executing a COMPUTE_LIFT procedure:

- Apply the model on the test data
- Create a target table or view containing only the case identifier and target columns from the test data

You will specify this table or view and the apply results table as input to the procedure.

Syntax

```
DBMS_DATA_MINING.COMPUTE_LIFT (
      apply_result_table_name      IN VARCHAR2,
      target_table_name            IN VARCHAR2,
      case_id_column_name          IN VARCHAR2,
      target_column_name           IN VARCHAR2,
      lift_table_name              IN VARCHAR2,
      positive_target_value        IN VARCHAR2,
      score_column_name            IN VARCHAR2 DEFAULT 'PREDICTION',
      score_criterion_column_name  IN VARCHAR2 DEFAULT 'PROBABILITY',
      num_quantiles                IN NUMBER DEFAULT 10,
      cost_matrix_table_name       IN VARCHAR2 DEFAULT NULL,
      apply_result_schema_name     IN VARCHAR2 DEFAULT NULL,
      target_schema_name           IN VARCHAR2 DEFAULT NULL,
      cost_matrix_schema_name      IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 25–10 COMPUTE_LIFT Procedure Parameters

Parameter	Description
apply_result_table_name	Name of the table containing the results of an APPLY operation on the test dataset (see Usage Notes)
target_table_name	Name of the table or view containing only the case identifier column and target column values (see Usage Notes)
case_id_column_name	Name of the case identifier column in the test data set. This must be common across the targets table and the apply results table.
target_column_name	Name of the target column
lift_table_name	Name of the table into which the lift table is to be generated
positive_target_value	Value of the positive target. If the target column is of NUMBER type, use TO_CHAR() operator to provide the value as a string.
score_column_name	Name of the column representing the score in the apply results table. In the fixed schema table generated by APPLY, this column has the name PREDICTION, which is the default.

Table 25–10 (Cont.) COMPUTE_LIFT Procedure Parameters

Parameter	Description
score_criterion_column_name	Name of the column representing the ranking factor for the score in the apply results table. In the fixed schema table generated by APPLY for classification models, this column has the name PROBABILITY, which is the default. This column must be a numerical type.
num_quantiles	Number of quantiles required in the lift table
cost_matrix_table_name	Name of the cost matrix table
apply_result_schema_name	Name of the schema hosting the APPLY results table
target_schema_name	Name of the schema hosting the targets table
cost_matrix_schema_name	Name of the schema hosting the cost matrix table

Usage Notes

You can also provide a cost matrix as an optional input to have the cost of predictions reflected in the results.

It is important to note that the data inputs to COMPUTE_LIFT do not always have to be generated using APPLY. As long as the schema of the two input tables matches the ones discussed in this section, with appropriate content, the procedure can provide the lift table as output. The quality of the results depends on the quality of the data.

The data provided for testing your classification model must match the data provided to CREATE_MODEL in schema and relevant content. If the data provided as input to CREATE_MODEL has been pre-processed, then the data input to APPLY must also be pre-processed using the same binning table used in build pre-processing.

Before you use the COMPUTE_LIFT procedure, you must prepare two data input streams from your test data.

First, you must APPLY the model on your test data. The parameter apply_result_table_name in the COMPUTE_LIFT procedure represents the table that will be generated in your schema as a result of the APPLY operation.

Next, you must create a table or view containing only the case identifier column and the target column in its schema. The parameter target_table_name reflects this input. The definition for this view or table name for a numerical target attribute is:

```
(case_identifier_column_name  VARCHAR2/NUMBER,
target_column_name             NUMBER)
```

The definition for this view or table name for a categorical target attribute is:

```
(case_identifier_column_name  VARCHAR2/NUMBER,
target_column_name            NUMBER)
```

You must provide the name of the table in which the lift table is to be generated. The resulting fixed schema table is always created in the schema that owns the model.

The resulting lift table will have the following definition:

```
(quantile_number               NUMBER,
 probability_threshold         NUMBER,
 gain_cumulative               NUMBER,
 quantile_total_count          NUMBER,
 quantile_target_count         NUMBER,
 percent_records_cumulative    NUMBER,
```

```
 lift_cumulative              NUMBER,
 target_density_cumulative    NUMBER,
 targets_cumulative           NUMBER,
 non_targets_cumulative       NUMBER,
 lift_quantile                NUMBER,
 target_density               NUMBER)
```

When a cost matrix is passed to the `COMPUTE_LIFT` procedure, the cost threshold is returned in the `probability_threshold` column.

The output columns are explained in *Oracle Data Mining Concepts*.

Examples

Assume that you have built a classification model `census_model` using the Naive Bayes algorithm, and you have been provided the test data in a table called `census_2d_test`, with case identifier column name `person_id`, and the target column name `class`.

```
DECLARE
  v_sql_stmt VARCHAR2(4000);
BEGIN

/* apply the model census_model on test data */
DBMS_DATA_MINING.APPLY(
  model_name           => 'census_model',
  data_table_name      => 'census_2d_test,
  case_id_column_name  => 'person_id',
  result_table_name    => 'census_test_result');

/* next create a view from test data that projects
 * only the case identifier and target column
 */

/* now compute lift with the default 10 quantiles
 * from the two data streams
 */
DBMS_DATA_MINING.COMPUTE_LIFT (
  apply_result_table_name   => 'census_test_result',
  target_table_name         => 'census_2d_test_view',
  case_id_column_name       => 'person_id',
  target_column_name        => 'class',
  lift_table_name           => 'census_lift',
  positive_target_value     => '1',
  cost_matrix_table_name    => 'census_cost_matrix');
END;
/

-- View the lift table contents using SQL
SELECT *
  FROM census_lift;
```

COMPUTE_ROC Procedure

This procedure computes the receiver operating characteristic (ROC) for a binary classification model. See *Oracle Data Mining Concepts* for a description of receiver operating characteristic.

Before executing a `COMPUTE_ROC` procedure:

- Apply the model on the test data
- Create a target table or view containing only the case identifier and target columns from the test data

You will specify this table or view and the apply results table as input to the procedure.

Syntax

```
DBMS_DATA_MINING.COMPUTE_ROC (
      roc_area_under_curve         OUT NUMBER,
      apply_result_table_name      IN  VARCHAR2,
      target_table_name            IN  VARCHAR2,
      case_id_column_name          IN  VARCHAR2,
      target_column_name           IN  VARCHAR2,
      roc_table_name               IN  VARCHAR2,
      positive_target_value        IN  VARCHAR2,
      score_column_name            IN  VARCHAR2 DEFAULT 'PREDICTION',
      score_criterion_column_name  IN  VARCHAR2 DEFAULT 'PROBABILITY',
      apply_result_schema_name     IN  VARCHAR2 DEFAULT NULL,
      target_schema_name           IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 25–11 COMPUTE_ROC Procedure Parameters

Parameter	Description
`roc_area_under_the_curve`	A measure of model accuracy, specifically, the probability that the model will correctly rank a randomly chosen pair of rows of opposite classes.
`apply_result_table_name`	Name of the table containing the results of an `APPLY` operation on the test dataset (see Usage Notes)
`target_table_name`	Name of the table or view containing the case identifiers and target values from the test data. (See the Usage Notes.)
`case_id_column_name`	Name of the case identifier column in the test data set. This must be common across the targets table and the apply results table.
`target_column_name`	Name of the target column
`roc_table_name`	Name of the table into which ROC results are to be generated. See Table 25–12, " COMPUTE_ROC Output".
`positive_target_value`	Value of the positive target. If the target column is of `NUMBER` type, use `TO_CHAR()` operator to provide the value as a string.

Table 25–11 (Cont.) COMPUTE_ROC Procedure Parameters

Parameter	Description
score_column_name	Name of the column representing the score in the apply results table. In the fixed schema table generated by APPLY, this column has the name PREDICTION, which is the default.
score_criterion_column_name	Name of the column representing the ranking factor for the score in the apply results table. In the fixed schema table generated by APPLY for classification models, this column has the name PROBABILITY, which is the default. Values in this column must be represented numerically.
apply_result_schema_name	Name of the schema hosting the APPLY results table
target_schema_name	Name of the schema hosting the targets table

Usage Notes

It is important to note that the data inputs to COMPUTE_ROC do not always have to be generated using APPLY. As long as the schema of the two input tables matches the ones discussed in this section, with appropriate content, the procedure can provide the ROC table as output. The quality of the results depends on the quality of the data.

The data provided for testing your classification model must match the data provided to CREATE_MODEL in schema and relevant content. If the data provided as input to CREATE_MODEL has been pre-processed, then the data input to APPLY must also be pre-processed using the statistics from the CREATE_MODEL data pre-processing.

Before you use the COMPUTE_ROC procedure, you must prepare two data input streams from your test data.

First, you must APPLY the model on your test data. The parameter apply_result_table_name in the COMPUTE_ROC procedure identifies the table that will be generated in your schema as a result of the APPLY operation.

Next, you must create a table or view containing only the case identifiers and target values from the test data. The parameter target_table_name identifies this table. For a numerical target attribute, the columns of this table are:

```
case_identifier_column_name  VARCHAR2/NUMBER,
target_column_name           NUMBER
```

For a categorical target attribute, the columns of this table are:

```
case_identifier_column_name  VARCHAR2/NUMBER,
target_column_name           VARCHAR2
```

You must provide the name of the table in which the ROC table is to be generated. The resulting table will always be created in the schema that owns the model, and it will always have the following columns.

```
(probability              NUMBER,
 true_positives           NUMBER,
 false_negatives          NUMBER,
 false_positives          NUMBER,
 true_negatives           NUMBER,
 true_positive_fraction   NUMBER,
 false_positive_fraction  NUMBER)
```

The output columns are explained in Table 25–12.

Table 25–12 COMPUTE_ROC Output

Output Column	Description
probability	Minimum predicted positive class probability resulting in a positive class prediction. Thus, different threshold values result in different hit rates and false_alarm_rates.
true_negatives	Negative cases in the test data with predicted probabilities below the probability_threshold (correctly predicted)
true_positives	Positive cases in the test data with predicted probabilities above the probability_threshold (correctly predicted)
false_negatives	Positive cases in the test data with predicted probabilities below the probability_threshold (incorrectly predicted)
false_positives	Negative cases in the test data with predicted probabilities above the probability_threshold (incorrectly predicted)
true_positive_ fraction	true_positives/(true_positives + false_ negatives)
false_positive_ fraction	false_positives/(false_positives + true_ negatives)

The typical use scenario is to examine the true_positive_fraction and false_ positive_fraction to determine the most desirable probability_threshold. This threshold is then used to predict class values in subsequent apply operations. For example, to identify positively predicted cases in probability rank order from an apply result table, given a probability_threshold:

```
select case_id_column_name from apply_result_table_name where probability >
probability_threshold order by probability DESC;
```

There are two procedures one might use to identify the most desirable probability_ threshold. One procedure applies when the relative cost of positive class versus negative class prediction errors are known to the user. The other applies when such costs are not well known to the user. In the first instance, one can apply the relative costs to the ROC table to compute the minimum cost probability_threshold. Suppose the relative cost ratio, Positive Class Error Cost / Negative Class Error Cost = 20. Then execute a query like:

```
WITH cost AS (
  SELECT probability_threshold, 20 * false_negatives + false positives cost
    FROM ROC_table
  GROUP BY probability_threshold),
    minCost AS (
      SELECT min(cost) minCost
        FROM cost)
      SELECT max(probability_threshold)probability_threshold
        FROM cost, minCost
    WHERE cost = minCost;
```

If relative costs are not well known, the user simply scans the values in the table (in sorted order) and makes a determination about which of the displayed trade-offs (misclassified positives versus misclassified negatives) is most desirable:

```
select * from ROC_table order by probability_threshold
```

Examples

Assume that you have built a classification model census_model using the SVM algorithm, and you have been provided the test data in a table called census_2d_

test, with case identifier column name person_id, and the target column name class.

```
DECLARE
  v_sql_stmt VARCHAR2(4000);
  v_accuracy NUMBER;
BEGIN

/* apply the model census_model on test data */
DBMS_DATA_MINING.APPLY(
  model_name           => 'census_model',
  data_table_name      => 'census_2d_test',
  case_id_column_name  => 'person_id',
  result_table_name    => 'census_test_result');

/* next create a view from test data that projects
 * only the case identifier and target column
 */
v_sql_stmt :=
'CREATE VIEW census_2d_test_view AS ' ||
'SELECT person_id, class FROM census_2d_test';
EXECUTE IMMEDIATE v_sql_stmt;

/* now compute the receiver operating characterestics from
 * the two data streams, also providing a cost matrix
 * as input.
 */
DBMS_DATA_MINING.COMPUTE_ROC (
  accuracy                => v_accuracy,
  apply_result_table_name  => 'census_test_result',
  target_table_name        => 'census_2d_test_view',
  case_id_column_name      => 'person_id',
  target_column_name       => 'class',
  roc_table_name           => 'census_roc',
  cost_matrix_table_name   => 'census_cost_matrix');
END;
/

-- View the ROC results using Oracle SQL
SELECT *
  FROM census_roc;
```

CREATE_MODEL Procedure

This procedure creates a mining model for a given mining function

Syntax

```
DBMS_DATA_MINING.CREATE_MODEL (
      model_name            IN VARCHAR2,
      mining_function       IN VARCHAR2,
      data_table_name       IN VARCHAR2,
      case_id_column_name   IN VARCHAR2,
      target_column_name    IN VARCHAR2 DEFAULT NULL,
      settings_table_name   IN VARCHAR2 DEFAULT NULL,
      data_schema_name      IN VARCHAR2 DEFAULT NULL,
      settings_schema_name  IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 25–13 CREATE_MODEL Procedure Parameters

Parameter	Description
model_name	Name of the model. (See "Model Names" on page 25-5)
mining_function	Constant representing the mining function. See "Constants that Specify the Mining Function" on page 25-6
data_table_name	Name of the table or view containing the training data
case_id_column_name	Name of the case identifier column
target_column_name	Name of the target column — NULL for descriptive models and for One-Class SVM models
settings_table_name	Name of the table or view containing mining function settings and algorithm settings
data_schema_name	Name of the schema hosting the training data
settings_schema_name	Name of the schema hosting the settings table/view

Usage Notes

The data provided to all subsequent operations such as APPLY must match the data provided to CREATE_MODEL in schema and relevant content. If the data provided as input to CREATE_MODEL has been pre-processed, then the data input to subsequent operations such as APPLY must also be pre-processed using the statistics from the CREATE_MODEL data pre-processing. The case identifier column is not considered to be a mining attribute during CREATE_MODEL.

You can view the default settings for each algorithm through GET_DEFAULT_SETTINGS. You can override the defaults by providing a settings table specifying your choice of mining algorithm and relevant overriding algorithm settings.

Once a model has been built, information about the attributes used for model build can be obtained from GET_MODEL_SIGNATURE. To inspect or review model contents, you can use any of the algorithm-specific GET_MODEL_DETAILS functions.

The behavior of the CREATE_MODEL is analogous to a SQL DDL CREATE operation. It contends with RENAME_MODEL and DROP_MODEL operations.

Note: The CREATE_MODEL operation creates a set of tables in the owner's schema to store the patterns and information that constitute a mining model for a particular algorithm.The names of these tables have the prefix DM$. The number, schema, and content of these tables is Oracle proprietary and may change from release to release. You must not direct any queries or updates against these system tables.

Examples

The first example builds a classification model using the Support Vector Machine algorithm.

```
/* prepare a settings table to override default
 * settings (Naive Bayes is the default classifier)
 */
CREATE TABLE census_settings (
  setting_name  VARCHAR2(30),
  setting_value VARCHAR2(128));

BEGIN
/* indicate that SVM is the chosen classifier */
INSERT INTO census_settings VALUES (
DBMS_DATA_MINING.ALGO_NAME, DBMS_DATA_MINING.ALGO_SUPPORT_VECTOR_MACHINES);

/* override the default value for complexity factor */
INSERT INTO census_settings (setting_name, setting_value)
VALUES (dbms_data_mining.svms_complexity_factor, TO_CHAR(0.081));
COMMIT;

/* build a model with name census_model */
DBMS_DATA_MINING.CREATE_MODEL(
  model_name           => 'census_model',
  mining_function      => DBMS_DATA_MINING.CLASSIFICATION,
  data_table_name      => 'census_2d_build',
  case_id_column_name  => 'person_id',
  target_column_name   => 'class',
  settings_table_name  => 'census_settings');
END;
/
```

You use similar code to build a One-Class SVM model. The main difference is that the target column is empty.

```
/* prepare a settings table to override default
 * settings (Naive Bayes is the default classifier)
 */
CREATE TABLE census_settings (
  setting_name  VARCHAR2(30),
  setting_value VARCHAR2(128));

BEGIN
/* indicate that SVM is the chosen classifier */
INSERT INTO census_settings VALUES (
DBMS_DATA_MINING.ALGO_NAME, DBMS_DATA_MINING.ALGO_SUPPORT_VECTOR_MACHINES);

/* override the default value for outlier rate */
INSERT INTO census_settings (setting_name, setting_value)
VALUES (dbms_data_mining.svms_outlier_rate, TO_CHAR(0.05));
COMMIT;
```

```
/* build a model with name census_model */
DBMS_DATA_MINING.CREATE_MODEL(
  model_name           => 'census_model',
  mining_function      => DBMS_DATA_MINING.CLASSIFICATION,
  data_table_name      => 'census_2d_build',
  case_id_column_name  => 'person_id',
  target_column_name   => NULL,
  settings_table_name  => 'census_settings');
END;
/
```

DROP_MODEL Procedure

This procedure drops an existing mining model from the user's schema.

Syntax

```
DBMS_DATA_MINING.DROP_MODEL (model_name IN VARCHAR2);
```

Parameters

Table 25–14 DROP_MODEL Procedure Parameters

Parameter	Description
model_name	Name of the model

Usage Notes

You can use DROP_MODEL to drop an existing mining model.

The behavior of the DROP_MODEL is similar to a SQL DDL DROP operation. It blocks RENAME_MODEL and CREATE_MODEL operations. It does not block or block on APPLY, which is a SQL query-like operation that does not update any model data.

If an APPLY operation is using a model, and you attempt to drop the model during that time, the DROP will succeed and APPLY will return indeterminate results. This is in line with the conventional behavior in the RDBMS, where DDL operations do not block on query operations.

Examples

Assume the existence of a model census_model. The following example shows how to drop this model.

```
BEGIN
  DBMS_DATA_MINING.DROP_MODEL(model_name => 'census_model');
END;
/
```

EXPORT_MODEL Procedure

This procedure exports the specified data mining models to a dump file set. You can import from the dump file set using the IMPORT_MODEL procedure. Both EXPORT_MODEL and IMPORT_MODEL use Oracle Data Pump technology.

> **See Also:** *Oracle Data Mining Administrator's Guide* for more information on model export and import.

Syntax

```
DBMS_DATA_MINING.EXPORT_MODEL (
      filename          IN VARCHAR2,
      directory         IN VARCHAR2,
      model_filter      IN VARCHAR2 DEFAULT NULL,
      filesize          IN VARCHAR2 DEFAULT NULL,
      operation         IN VARCHAR2 DEFAULT NULL,
      remote_link       IN VARCHAR2 DEFAULT NULL,
      jobname           IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 25–15 EXPORT_MODEL Procedure Parameters

Parameter	Description
filename	Name of the dump file set to which the models should be exported. The name must be unique within the schema. The dump file set can contain one or more files. The number of files in a dump file set is determined by the size of the models being exported (both metadata and data) and a specified or estimated maximum file size. You can specify the file size in the filesize parameter, or you can use the operation parameter to cause Oracle Data Pump to estimate the file size. If the size of the models to export is greater than the maximum file size, one or more additional files are created. When the export operation completes successfully, the name of the dump file set is automatically expanded to *filename01.dmp*, even if there is only one file in the dump set. If there are additional files, they are named sequentially as *filename02.dmp*, *filename03.dmp*, and so forth.
directory	Name of a pre-defined directory object that specifies where the dump file set should be created. You must have read/write privileges on the directory object and on the file system directory that it identifies.
model_filter	Optional parameter that specifies which model or models to export. If you do not specify a value for model_filter, all models in the schema are exported. You can also specify NULL (the default) or 'ALL' to export all models. You can export individual models by name and groups of models that share a given characteristic. For instance, you could export all Naive Bayes models or all models that use the same target attribute. See the Usage Notes for more information. Examples are provided in Table 25–16.
filesize	Optional parameter that specifies the maximum size of a file in the dump file set. The size may be specified in bytes, kilobytes (K), megabytes (M), or gigabytes (G). The default size is 50 MB. If the size of the models to export is larger than filesize, one or more additional files are created within the dump set. See the description of the filename parameter for more information.

Table 25–15 (Cont.) EXPORT_MODEL Procedure Parameters

Parameter	Description
operation	Optional parameter that specifies whether or not to estimate the size of the files in the dump set. By default the size is not estimated and the value of the filesize parameter determines the size of the files. You can specify either of the following values for operation: ▪ 'EXPORT' — Export all or the specified models. (Default) ▪ 'ESTIMATE' — Estimate the size of the exporting models.
remote_link	Optional parameter not used in this release. Set to NULL.
jobname	Optional parameter that specifies the name of the export job. By default, the name has the form *username*_exp_*nnnn*, where *nnnn* is a number. For example, a job name in the SCOTT schema might be SCOTT_exp_134. If you specify a job name, it must be unique within the schema. The maximum length of the job name is 30 characters. A log file for the export job, named *jobname.log*, is created in the same directory as the dump file set.

Usage Notes

The model_filter parameter specifies which models to export. You can list the models by name, or you can identify a group of models that share a given characteristic. To specify models by name, provide a single model name or a comma-delimited list of model names. To specify a group of models that share a characteristic, use a conditional expression that completes the WHERE clause of a query against the DM_USER_MODELS view. DM_USER_MODELS lists the models in the current schema. It has the following columns.

```
Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 NAME                                      NOT NULL VARCHAR2(25)
 FUNCTION_NAME                                      VARCHAR2(30)
 ALGORITHM_NAME                                     VARCHAR2(30)
 CREATION_DATE                                      DATE
 BUILD_DURATION                                     NUMBER
 TARGET_ATTRIBUTE                                   VARCHAR2(30)
 MODEL_SIZE                                         NUMBER
```

For descriptions of the columns in DM_USER_MODELS, see "User Views" on page 25-14.

To construct a conditional expression for model_filter, specify a column name, a supported conditional operator, and a value. The supported conditional operators are: <, <=, =, =>, >, LIKE, IN. For information on conditional operators and WHERE clauses, see *Oracle Database SQL Reference*.

Examples of model filters are provided in Table 25–16.

Table 25–16 Sample Values for the Model Filter Parameter

Sample Value	Meaning
'mymodel'	Export the model named mymodel
'mymodel2, mymodel3'	Export the models named mymodel2 and mymodel3
'name= ''mymodel'''	Export the model named mymodel
'name IN (''mymodel2'',''mymodel3'')'	Export the models named mymodel2 and mymodel3

Table 25–16 (Cont.) Sample Values for the Model Filter Parameter

Sample Value	Meaning
`'name LIKE ''AI%'''`	Export all models that have names starting with `AI`
`'ALGORITHM_NAME = ''NAIVE_BAYES'''`	Export all Naive Bayes models. See Table 25–2 for a list of algorithm names.
`'FUNCTION_NAME =''CLASSIFICATION'''`	Export all classification models. See Table 25–1 for a list of mining functions.

Examples

The following statement exports all the models in the `DMUSER3` schema to a dump file set called `models_out` in the directory `$ORACLE_HOME/rdbms/log`. This directory is mapped to a directory object called `DATA_PUMP_DIR`. The `DMUSER3` user has read/write access to the directory and to the directory object.

```
SQL>execute dbms_data_mining.export_model ('models_out', 'DATA_PUMP_DIR');
```

You can exit SQL*Plus and list the resulting dump file and log file.

```
SQL>exit
>cd $ORACLE_HOME/rdbms/log
>ls
>DMUSER3_exp_1027.log  models_out01.dmp
```

The following example uses the same directory object and is executed by the same user. It exports the models called `NMF_SH_SAMPLE` and `SVMR_SH_REGR_SAMPLE` to a different dump file set in the same directory.

```
SQL>execute dbms_data_mining.export_model ( 'models2_out', 'DATA_PUMP_DIR',
            'name in (''NMF_SH_SAMPLE'', ''SVMR_SH_REGR_SAMPLE'')');
SQL>exit
>cd $ORACLE_HOME/rdbms/log
>ls
>DMUSER3_exp_1027.log  models_out01.dmp
 DMUSER3_exp_924.log  models2_out01.dmp
```

Using the same directory object and schema, this example exports all models whose target is `AFFINITY_CARD`.

```
SQL>execute dbms_data_mining.export_model ('models050402_out',
                'DATA_PUMP_DIR', 'target_attribute = ''AFFINITY_CARD''',
                '1M', 'EXPORT', NULL, 'models050402_job');
SQL>exit
>cd $ORACLE_HOME/rdbms/log
>ls
>DMUSER3_exp_1027.log  models_out01.dmp
 DMUSER3_exp_924.log   models2_out01.dmp
 models050402_job.log  models050402_out01.dmp   models050402_out02.dmp
```

GET_ASSOCIATION_RULES Function

This table function returns the rules from an Association model.

You can specify filtering criteria to cause GET_ASSOCIATION_RULES to return a subset of the rules. Filtering criteria can improve the performance of the table function. If the number of rules is large, the greatest performance improvement will result from specifying the topn parameter.

Syntax

```
DBMS_DATA_MINING.GET_ASSOCIATION_RULES (
   model_name            IN VARCHAR2,
   topn                  IN NUMBER DEFAULT NULL,
   rule_id               IN INTEGER DEFAULT NULL,
   min_confidence        IN NUMBER DEFAULT NULL,
   min_support           IN NUMBER DEFAULT NULL,
   max_rule_length       IN INTEGER DEFAULT NULL,
   min_rule_length       IN INTEGER DEFAULT NULL,
   sort_order            IN DMSYS.ORA_MINING_VARCHAR2_NT DEFAULT NULL,
   antecedent_items      IN DYSYS.ORA_MINING_VARCHAR2_NT DEFAULT NULL,
   consequent_items      IN DYSYS.ORA_MINING_VARCHAR2_NT DEFAULT NULL)
 RETURN DM_RULES PIPELINED;
```

Parameters

Table 25–17 GET_ASSOCIATION_RULES Function Parameters

Parameter	Description
model_name	Name of the model. This is the only required parameter of GET_ASSOCIATION_RULES. All other parameters specify optional filters on the rules to return.
topn	Return the *n* top rules ordered by confidence and then support, both descending. If you specify a sort order, the top *n* rules are derived after the sort is performed. If topn is specified and no maximum or minimum rule length is specified, then the only columns allowed in the sort order are RULE_CONFIDENCE and RULE_SUPPORT. If *topn* is specified and a maximum or minimum rule length *is* specified, then RULE_CONFIDENCE, RULE_SUPPORT, and NUMBER_OF_ITEMS are allowed in the sort order.
rule_id	Identifier of the rule to return. If you specify a value for rule_id, do not specify values for the other filtering parameters.
min_confidence	Return the rules with confidence greater than or equal to this number
min_support	Return the rules with support greater than or equal to this number
max_rule_length	Return the rules with a length less than or equal to this number. Rule length refers to the number of items in the rule (See NUMBER_OF_ITEMS in Table 25–18). For example, in the rule A=>B (if A, then B), the number of items is 2. If max_rule_length is specified, then the NUMBER_OF_ITEMS column is permitted in the sort order.

Table 25–17 (Cont.) GET_ASSOCIATION_RULES Function Parameters

Parameter	Description
min_rule_length	Return the rules with a length greater than or equal to this number. See max_rule_length for a description of rule length. If min_rule_length is specified, then the NUMBER_OF_ITEMS column is permitted in the sort order.
sort_order	Sort the rules by the values in one or more of the returned columns. Specify one or more column names, each followed by ASC for ascending order or DESC for descending order. For example, to sort the result set in descending order first by the NUMBER_OF_ITEMS column, then by the RULE_CONFIDENCE column, you would specify: DMSYS.ORA_MINING_VARCHAR2_NT('NUMBER_OF_ITEMS DESC', 'RULE_CONFIDENCE DESC') If you specify topn, the results will vary depending on the sort order. By default, the results are sorted by confidence in descending order, then by support in descending order. See the examples.
antecedent_items	Return the rules with these items in the antecedent. See the examples.
consequent_items	Return the rules with this item in the consequent. See the examples.

Return Values

Table 25–18 GET_ASSOCIATION RULES Function Return Values

Return Value	Description
DM_RULES	Represents a set of rows of type DM_RULE. The rows have the following columns: (rule_id INTEGER, antecedent DM_PREDICATES, consequent DM_PREDICATES, rule_support NUMBER, rule_confidence NUMBER, antecedent_support NUMBER, consequent_support NUMBER, number_of_items INTEGER) The antecedent and consequent columns each return nested tables of type DM_PREDICATES.The rows, of type DM_PREDICATE, have the following columns: (attribute_name VARCHAR2(30), conditional_operator CHAR(2)/*=,<>,<,>,<=,>=*/, attribute_num_value NUMBER, attribute_str_value VARCHAR2(4000), attribute_support NUMBER, attribute_confidence NUMBER)

Usage Notes

This table function pipes out rows of type DM_RULES. For information on ODM data types and piped output from table functions, see "Data Types" on page 25-10.

The DMSYS.ORA_MINING_VARCHAR2_NT type is defined as a table of VARCHAR2(4000).

Examples

The following example demonstrates an Association model build followed by several invocations of the GET_ASSOCIATION_RULES table function.

```
-- prepare a settings table to override default settings
CREATE TABLE market_settings AS
SELECT *
  FROM TABLE(DBMS_DATA_MINING.GET_DEFAULT_SETTINGS)
 WHERE setting_name LIKE 'ASSO_%';
BEGIN
-- update the value of the minimum confidence
UPDATE census_settings
   SET setting_value = TO_CHAR(0.081)
 WHERE setting_name = DBMS_DATA_MINING.asso_min_confidence;

-- build an AR model
DBMS_DATA_MINING.CREATE_MODEL(
  model_name => 'market_model',
  function => DBMS_DATA_MINING.ASSOCIATION,
  data_table_name => 'market_build',
  case_id_column_name => 'item_id',
  target_column_name => NULL,
  settings_table_name => 'census_settings');
END;
/
-- View the (unformatted) rules
SELECT rule_id, antecedent, consequent, rule_support,
       rule_confidence
  FROM TABLE(DBMS_DATA_MINING.GET_ASSOCIATION_RULES('market_model'));
```

In the previous example, you view all rules. To view just the top 20 rules, use the following statement.

```
-- View the top 20 (unformatted) rules
SELECT rule_id, antecedent, consequent, rule_support,
       rule_confidence
  FROM TABLE(DBMS_DATA_MINING.GET_ASSOCIATION_RULES('market_model', 20));
```

The following example returns all the rules which have 'AQUATIC' or 'EGGS' in the antecedent, and has 'VENOMOUS' as the consequent. The rules are sorted first by NUMBER_OF_ITEMS in descending order, then by RULE_CONFIDENCE in descending order, and finally by RULE_SUPPORT in descending order.

```
SELECT * FROM TABLE
(    DBMS_DATA_MINING.GET_ASSOCIATION_RULES
       ('AR_Model_31', 120, NULL, 1, .51, 7,
         DMSYS.ORA_MINING_VARCHAR2_NT
          ('NUMBER_OF_ITEMS DESC', 'RULE_CONFIDENCE DESC', 'RULE_SUPPORT DESC'),
         DMSYS.ORA_MINING_VARCHAR2_NT('AQUATIC', 'EGGS'),
         DMSYS.ORA_MINING_VARCHAR2_NT('VENOMOUS')));
```

GET_DEFAULT_SETTINGS Function

This table function returns the default settings for all mining functions and algorithms supported in the DBMS_DATA_MINING package.

Syntax

```
DBMS_DATA_MINING.GET_DEFAULT_SETTINGS
  RETURN DM_MODEL_SETTINGS PIPELINED;
```

Return Values

Table 25–19 GET_DEFAULT_SETTINGS Function Return Values

Return Value	Description
DM_MODEL_SETTINGS	Represents a set of rows of type DM_MODEL_SETTING. The rows have the following columns: (setting_name VARCHAR2(30), setting_value VARCHAR2(128))

Usage Notes

This table function pipes out rows of type DM_MODEL_SETTING. For information on ODM data types and ODM data types and piped output from table functions, see "Data Types" on page 25-10.

This function is particularly useful if you do not know what settings are associated with a particular function or algorithm, and you want to override some or all of them.

Examples

For example, if you want to override some or all of *k*-Means clustering settings, you can create a settings table as shown, and update individual settings as required.

```
BEGIN
  CREATE TABLE mysettings AS
  SELECT *
  FROM TABLE(DBMS_DATA_MINING.GET_DEFAULT_SETTINGS)
   WHERE setting_name LIKE 'KMNS%';
  -- now update individual settings as required
  UPDATE mysettings
     SET setting_value = 0.02
   WHERE setting_name = DBMS_DATA_MINING.KMNS_MIN_PCT_ATTR_SUPPORT;
END;
/
```

GET_FREQUENT_ITEMSETS Function

This table function returns a set of rows that represent the frequent itemsets from an Association model. For a detailed description of frequent itemsets, consult *Oracle Data Mining Concepts*.

Syntax

```
DBMS_DATA_MINING.GET_FREQUENT_ITEMSETS (
    model_name        IN VARCHAR2,
    topn              IN NUMBER DEFAULT NULL)
  RETURN DM_ITEMSETS PIPELINED;
```

Parameters

Table 25–20 GET_FREQUENT_ITEMSETS Function Parameters

Parameter	Description
model_name	Name of the model
topn	When not NULL, return the top *n* rows ordered by support in descending order

Return Values

Table 25–21 GET_FREQUENT_ITEMSETS Function Return Values

Return Value	Description
DM_ITEMSETS	Represents a set of rows of type DM_ITEMSET. The rows have the following columns: (itemsets_id NUMBER, items DM_ITEMS, support NUMBER, number_of_items NUMBER) The items column returns a nested table of type DM_ITEMS. The table has one column of type VARCHAR2(4000), which contains individual item names.

Usage Notes

This table function pipes out rows of type DM_ITEMSETS. For information on ODM data types and piped output from table functions, see "Data Types" on page 25-10.

Examples

The following example demonstrates an Association model build followed by an invocation of GET_FREQUENT_ITEMSETS table function from Oracle SQL.

```
-- prepare a settings table to override default settings
CREATE TABLE market_settings AS
    SELECT *
  FROM TABLE(DBMS_DATA_MINING.GET_DEFAULT_SETTINGS)
 WHERE setting_name LIKE 'ASSO_%';
BEGIN
-- update the value of the minimum confidence
UPDATE market_settings
   SET setting_value = TO_CHAR(0.081)
 WHERE setting_name = DBMS_DATA_MINING.asso_min_confidence;
```

```
/* build a AR model */
DBMS_DATA_MINING.CREATE_MODEL(
  model_name           => 'market_model',
  function             => DBMS_DATA_MINING.ASSOCIATION,
  data_table_name      => 'market_build',
  case_id_column_name  => 'item_id',
  target_column_name   => NULL,
  settings_table_name  => 'census_settings');
END;
/

-- View the (unformatted) Itemsets from SQL*Plus
SELECT itemset_id, items, support, number_of_items
  FROM TABLE(DBMS_DATA_MINING.GET_FREQUENT_ITEMSETS('market_model'));
```

In the example above, you view all itemsets. To view just the top 20 itemsets, use the following statement:

```
-- View the top 20 (unformatted) Itemsets from SQL*Plus
SELECT itemset_id, items, support, number_of_items
  FROM TABLE(DBMS_DATA_MINING.GET_FREQUENT_ITEMSETS('market_model', 20));
```

GET_MODEL_DETAILS_ABN Function

This table function returns a set of rows that provide the details of an Adaptive Bayes Network model.

Syntax

```
DBMS_DATA_MINING.GET_MODEL_DETAILS_ABN (
    model_name          IN VARCHAR2)
  RETURN DM_ABN_DETAILS PIPELINED;
```

Parameters

Table 25–22 GET_MODEL_DETAILS_ABN Function Parameters

Parameter	Description
model_name	Name of the model

Return Values

Table 25–23 GET_MODEL_DETAILS_ABN Function Return Values

Return Value	Description
DM_ABN_DETAILS	Represents a set of rows of type DM_ABN_DETAIL. The rows have the following columns: (rule_id INTEGER, antecedent DM_PREDICATES, consequent DM_PREDICATES, rule_support NUMBER) The antecedent and consequent columns of DM_ABN_DETAIL each return nested tables of type DM_PREDICATES. The rows, of type DM_PREDICATE, have the following columns: (attribute_name VARCHAR2(30), conditional_operator CHAR(2), /*=,<>,<,>,<=,>=*/ attribute_num_value NUMBER, attribute_str_value VARCHAR2(4000), attribute_support NUMBER, attribute_confidence NUMBER)

Usage Notes

This table function pipes out rows of type DM_ABN_DETAIL. For information on ODM data types and piped output from table functions, see "Data Types" on page 25-10.

This function returns details only for a single feature ABN model.

Examples

The following example demonstrates an ABN model build followed by an invocation of GET_MODEL_DETAILS_ABN table function from Oracle SQL.

```
BEGIN
  -- prepare a settings table to override default algorithm and model type
  CREATE TABLE abn_settings (setting_name VARCHAR2(30),
  setting_value
VARCHAR2(128));
  INSERT INTO abn_settings VALUES (DBMS_DATA_MINING.ALGO_NAME,
```

```
    DBMS_DATA_MINING.ALGO_ADAPTIVE_BAYES_NETWORK);
  INSERT INTO abn_settings VALUES
    (DBMS_DATA_MINING.ABNS_MODEL_TYPE,
     DBMS_DATA_MINING.ABNS_SINGLE_FEATURE);
   COMMIT;
  -- create a model
  DBMS_DATA_MINING.CREATE_MODEL (
    model_name           => 'abn_model',
    function             => DBMS_DATA_MINING.CLASSIFICATION,
    data_table_name      => 'abn_build',
    case_id_column_name  => 'id',
    target_column_name   => NULL,
    settings_table_name  => 'abn_settings');
END;
/
-- View the (unformatted) results from SQL*Plus
SELECT *
    FROM TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_ABN('abn_model'));
```

GET_MODEL_DETAILS_AI Function

This table function returns a set of rows that provide the details of an Attribute Importance model.

Syntax

```
DBMS_DATA_MINING.GET_MODEL_DETAILS_AI (
  model_name          IN VARCHAR2)
 RETURN DM_RANKED_ATTRIBUTES PIPELINED;
```

Parameters

Table 25–24 GET_MODEL_DETAILS_AI Function Parameters

Parameter	Description
model_name	Name of the model

Return Values

Table 25–25 GET_MODEL_DETAILS_AI Function Return Values

Return Value	Description
DM_RANKED_ATTRIBUTES	Represents a set of rows of type DM_RANKED_ATTRIBUTE. The rows have the following columns: (attribute_name VARCHAR2(30), importance_value NUMBER, rank NUMBER(38))

GET_MODEL_DETAILS_KM Function

This table function returns a set of rows that provide the details of a *k*-Means clustering model.

You can provide input to `GET_MODEL_DETAILS_KM` to request specific information about the model, thus improving the performance of the query. If you do not specify filtering parameters, `GET_MODEL_DETAILS_KM` returns all the information about the model.

Syntax

```
DBMS_DATA_MINING.GET_MODEL_DETAILS_KM (
          model_name         VARCHAR2,
          cluster_id         NUMBER    DEFAULT NULL,
          attribute          VARCHAR2  DEFAULT NULL,
          centroid           NUMBER    DEFAULT 1,
          histogram          NUMBER    DEFAULT 1,
          rules              NUMBER    DEFAULT 2)
RETURN DM_CLUSTERS PIPELINED;
```

Parameters

Table 25–26 GET_MODEL_DETAILS_KM Function Parameters

Parameter	Description
`model_name`	Name of the model
`cluster_id`	The ID of a cluster in the model. When a valid cluster ID is specified, only the details of this cluster are returned. Otherwise the details for all clusters are returned.
`attribute`	The name of an attribute. When a valid attribute name is specified, only the details of this attribute are returned. Otherwise the details for all attributes are returned
`centroid`	This parameter accepts the following values: ■ 1 — Details about centroids are returned (default) ■ 0 — Details about centroids are not returned
`histogram`	This parameter accepts the following values: ■ 1 — Details about histograms are returned (default) ■ 0 — Details about histograms are not returned
`rules`	This parameter accepts the following values: ■ 2 — Details about rules are returned (default) ■ 1 — Rule summaries are returned ■ 0 — No information about rules is returned

Return Values

Table 25–27 GET_MODEL_DETAILS_KM Function Return Values

Return Value	Description
DM_CLUSTERS	Represents a set of rows of type DM_CLUSTER. The rows have the following columns: (id INTEGER, record_count NUMBER, parent NUMBER, tree_level NUMBER, dispersion NUMBER, split_predicate DM_PREDICATES, child DM_CHILDREN, centroid DM_CENTROIDS, histogram DM_HISTOGRAMS, rule DM_RULE) The split_predicate column of DM_CLUSTER returns a nested table of type DM_PREDICATES. Each row, of type DM_PREDICATE, has the following columns: (attribute_name VARCHAR2(30), conditional_operator CHAR(2) /*=,<>,<,>,<=,>=*/, attribute_num_value NUMBER, attribute_str_value VARCHAR2(4000), attribute_support NUMBER, attribute_confidence NUMBER) The child column of DM_CLUSTER returns a nested table of type DM_CHILDREN. The rows, of type DM_CHILD, have a single column of type NUMBER, which contains the identifiers of each child. The centroid column of DM_CLUSTER returns a nested table of type DM_CENTROIDS. The rows, of type DM_CENTROID, have the following columns: (attribute_name VARCHAR2(30), mean NUMBER, mode_value VARCHAR2(4000), variance NUMBER) The histogram column of DM_CLUSTER returns a nested table of type DM_HISTOGRAMS. The rows, of type DM_HISTOGRAM_BIN, have the following columns: (attribute_name VARCHAR2(30), bin_id NUMBER, lower_bound NUMBER, upper_bound NUMBER, label VARCHAR2(4000), count NUMBER) The rule column of DM_CLUSTER returns a single row of type DM_RULE. The columns are: (rule_id INTEGER, antecedent DM_PREDICATES, consequent DM_PREDICATES, rule_support NUMBER, rule_confidence NUMBER)

Table 25–27 (Cont.) GET_MODEL_DETAILS_KM Function Return Values

Return Value	Description

	The `antecedent` and `consequent` columns of `DM_RULE` each return nested tables of type `DM_PREDICATES`. The rows, of type `DM_PREDICATE`, have the following columns:
	`(attribute_name VARCHAR2(30),`
	`conditional_operator CHAR(2)/*=,<>,<,>,<=,>=*/,`
	`attribute_num_value NUMBER,`
	`attribute_str_value VARCHAR2(4000),`
	`attribute_support NUMBER,`
	`attribute_confidence NUMBER)`

Usage Notes

The table function pipes out rows of type `DM_CLUSTERS`. For information on ODM data types and piped output from table functions, see "Data Types" on page 25-10.

Examples

The following example demonstrates a *k*-Means clustering model build followed by an invocation of `GET_MODEL_DETAILS_KM` table function from Oracle SQL.

```
BEGIN
-- create a settings table
UPDATE cluster_settings
   SET setting_value = 3
 WHERE setting_name = DBMS_DATA_MINING.KMEANS_BLOCK_GROWTH;

/* build a k-Means clustering model */
DBMS_DATA_MINING.CREATE_MODEL(
  model_name           => 'eight_clouds',
  function             => DBMS_DATA_MINING.CLUSTERING,
  data_table_name      => 'eight_clouds_build',
  case_id_column_name  => 'id',
  target_column_name   => NULL,
  settings_table_name  => 'cluster_settings');
END;
/

-- View the (unformatted) rules from SQL*Plus
SELECT id, record_count, parent, tree_level, dispersion,
       child, centroid, histogram, rule
  FROM TABLE(DBMS_DATA_MINING_GET_MODEL_DETAILS_KM('eight_clouds'));
```

GET_MODEL_DETAILS_NB Function

This table function returns a set of rows that provide the details of a Naive Bayes model.

Syntax

```
DBMS_DATA_MINING.GET_MODEL_DETAILS_NB (
   model_name      IN       VARCHAR2)
 RETURN DM_NB_DETAILS PIPELINED;
```

Parameters

Table 25–28 GET_MODEL_DETAILS_NB Function Parameters

Parameter	Description
model_name	Name of the model

Return Values

Table 25–29 GET_MODEL_DETAILS_NB Function Return Values

Return Value	Description
DM_NB_DETAILS	Represents a set of rows of type DM_NB_DETAIL. The rows have the following columns: (target_attribute_name VARCHAR2(30), target_attribute_str_value VARCHAR2(4000), target_attribute_num_value NUMBER, prior_probability NUMBER, conditionals DM_CONDITIONALS) The conditionals column of DM_NB_DETAIL returns a nested table of type DM_CONDITIONALS. The rows, of type DM_CONDITIONAL, have the following columns: (attribute_name VARCHAR2(30), attribute_str_value VARCHAR2(4000), attribute_num_value NUMBER, conditional_probability NUMBER)

Usage Notes

The table function pipes out rows of type DM_NB_DETAILS. For information on ODM data types and piped output from table functions, see "Data Types" on page 25-10.

Examples

Assume that you have built a classification model census_model using the Naive Bayes algorithm. You can retrieve the model details as shown in this example.

```
-- You can view the Naive Bayes model details in many ways
-- Consult the Oracle Application Developer's Guide -
-- Object-Relational Features for different ways of
-- accessing Oracle Objects.

-- View the (unformatted) details from SQL*Plus
SELECT attribute_name, attribute_num_value, attribute_str_value,
       prior_probability, conditionals,
```

```
FROM TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_NB('census_model');
```

See `nbdemo.sql` for generation of formatted rules.

GET_MODEL_DETAILS_NMF Function

This table function returns a set of rows that provide the details of a Non-Negative Matrix Factorization model.

Syntax

```
DBMS_DATA_MINING.GET_MODEL_DETAILS_NMF (
   model_name        IN        VARCHAR2)
 RETURN DM_NMF_FEATURE_SET PIPELINED;
```

Parameters

Table 25–30 GET_MODEL_DETAILS_NMF Function Parameters

Parameter	Description
model_name	Name of the model

Return Values

Table 25–31 GET_MODEL_DETAILS_NMF Function Return Values

Return Value	Description
DM_NMF_FEATURE_SET	Represents a set of rows of DM_NMF_FEATURE. The rows have the following columns: (feature_id NUMBER, attribute_set DM_NMF_ATTRIBUTE_SET) The attribute_set column of DM_NMF_FEATURE returns a nested table of type DM_NMF_ATTRIBUTE_SET. The rows, of type DM_NMF_ATTRIBUTE, have the following columns: (attribute_name VARCHAR2(30), attribute_value VARCHAR2(4000), coefficient NUMBER)

Usage Notes

The table function pipes out rows of type DM_NMF_FEATURE_SET. For information on ODM data types and piped output from table functions, see "Data Types" on page 25-10.

Examples

Assume you have built an NMF model called my_nmf_model. You can retrieve model details as shown:

```
--View (unformatted) details from SQL*Plus
SELECT feature_id, attribute_set
FROM TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_NMF(
        'my_nmf_model'));
```

GET_MODEL_DETAILS_OC Function

This table function returns a set of rows that provide the details of an O-Cluster clustering model. The rows are an enumeration of the clustering patterns generated during the creation of the model.

You can provide input to GET_MODEL_DETAILS_OC to request specific information about the model, thus improving the performance of the query. If you do not specify filtering parameters, GET_MODEL_DETAILS_OC returns all the information about the model.

Syntax

```
DBMS_DATA_MINING.GET_MODEL_DETAILS_OC (
         model_name         VARCHAR2,
         cluster_id         NUMBER    DEFAULT NULL,
         attribute          VARCHAR2  DEFAULT NULL,
         centroid           NUMBER    DEFAULT 1,
         histogram          NUMBER    DEFAULT 1,
         rules              NUMBER    DEFAULT 2)
RETURN DM_CLUSTERS PIPELINED;
```

Parameters

Table 25–32 GET_MODEL_DETAILS_OC Function Parameters

Parameter	Description
model_name	Name of the model
cluster_id	The ID of a cluster in the model. When a valid cluster ID is specified, only the details of this cluster are returned. Otherwise the details for all clusters are returned.
attribute	The name of an attribute. When a valid attribute name is specified, only the details of this attribute are returned. Otherwise the details for all attributes are returned
centroid	This parameter accepts the following values: ■ 1 — Details about centroids are returned (default) ■ 0 — Details about centroids are not returned
histogram	This parameter accepts the following values: ■ 1 — Details about histograms are returned (default) ■ 0 — Details about histograms are not returned
rules	This parameter accepts the following values: ■ 2 — Details about rules are returned (default) ■ 1 — Rule summaries are returned ■ 0 — No information about rules is returned

Return Values

Table 25–33 GET_MODEL_DETAILS_OC Function Return Values

Return Value	Description
DM_CLUSTERS	Represents a set of rows of type DM_CLUSTER. The rows have the following columns: (id INTEGER, record_count NUMBER, parent NUMBER, tree_level NUMBER, dispersion NUMBER, split_predicate DM_PREDICATES, child DM_CHILDREN, centroid DM_CENTROIDS, histogram DM_HISTOGRAMS, rule DM_RULE) The split_predicate column of DM_CLUSTER returns a nested table of type DM_PREDICATES. Each row, of type DM_PREDICATE, has the following columns: (attribute_name VARCHAR2(30), conditional_operator CHAR(2) /*=,<>,<,>,<=,>=*/, attribute_num_value NUMBER, attribute_str_value VARCHAR2(4000), attribute_support NUMBER, attribute_confidence NUMBER) The child column of DM_CLUSTER returns a nested table of type DM_CHILDREN. The rows, of type DM_CHILD, have a single column of type NUMBER, which contains the identifiers of each child. The centroid column of DM_CLUSTER returns a nested table of type DM_CENTROIDS. The rows, of type DM_CENTROID, have the following columns: (attribute_name VARCHAR2(30) mean NUMBER, mode_value VARCHAR2(4000), variance NUMBER) The histogram column of DM_CLUSTER returns a nested table of type DM_HISTOGRAMS. The rows, of type DM_HISTOGRAM_BIN, have the following columns: (attribute_name VARCHAR2(30), bin_id NUMBER, lower_bound NUMBER, upper_bound NUMBER, label VARCHAR2(4000), count NUMBER) The rule column of DM_CLUSTER returns a single row of type DM_RULE. The columns are: (rule_id INTEGER, antecedent DM_PREDICATES, consequent DM_PREDICATES, rule_support NUMBER, rule_confidence NUMBER)

Table 25–33 (Cont.) GET_MODEL_DETAILS_OC Function Return Values

Return Value	Description

	The `antecedent` and `consequent` columns each return nested tables of type `DM_PREDICATES`.The rows, of type `DM_PREDICATE`, have the following columns:
	`(attribute_name VARCHAR2(30),`
	`conditional_operator CHAR(2)/*=,<>,<,>,<=,>=*/,`
	`attribute_num_value NUMBER,`
	`attribute_str_value VARCHAR2(4000),`
	`attribute_support NUMBER,`
	`attribute_confidence NUMBER)`

Usage Notes

The table function pipes out rows of type `DM_CLUSTER`. For information about ODM data types and piped output from table functions, see "Data Types" on page 25-10.

Examples

Assume you have built an OC model called `my_oc_model`. You can retrieve information from the model details as shown:

```
--View (unformatted) details from SQL*Plus
SELECT T.id           clu_id,
       T.record_count rec_cnt,
       T.parent       parent,
       T.tree_level   tree_level
  FROM (SELECT *
          FROM TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_OC(
                 'my_oc_model'))
        ORDER BY id) T
 WHERE ROWNUM < 11;
```

GET_MODEL_DETAILS_SVM Function

This table function returns a set of rows that provide the details of a Support Vector Machine model. This is applicable only for classification or regression models built using a linear kernel. For any other kernel, the table function returns ORA-40215.

Syntax

```
DBMS_DATA_MINING.GET_MODEL_DETAILS_SVM (
  model_name      IN       VARCHAR2)
 RETURN DM_SVM_LINEAR_COEFF_SET PIPELINED;
```

Parameters

Table 25–34 GET_MODEL_DETAILS_SVM Function Parameters

Parameter	Description
model_name	Name of the model

Return Values

Table 25–35 GET_MODEL_DETAILS_SVM Function Return Values

Return Value	Description
DM_SVM_LINEAR_COEFF_SET	Represents a set of rows of type DM_SVM_LINEAR_COEFF. The rows have the following columns: (class VARCHAR2(4000), attribute_set DM_SVM_ATTRIBUTE_SET) The attribute_set column returns a nested table of type DM_SVM_ATTRIBUTE_SET. The rows, of type DM_SVM_ATTRIBUTE, have the following columns: (attribute_name VARCHAR2(30), attribute_value VARCHAR2(4000), coefficient NUMBER) See Usage Notes.

Usage Notes

The table function pipes out rows of type DM_SVM_LINEAR_COEFF. For information on ODM data types and piped output from table functions, see "Data Types" on page 25-10.

The class column of DM_SVM_LINEAR_COEFF represents classification target values. For regression targets, class is NULL. For each classification target value for classification models, a set of coefficients is returned. For binary classification, one-class classifier, and regression models, only a single set of coefficients is returned.

The attribute_value column in the nested table DM_SVM_ATTRIBUTE_SET is used for categorical attributes. The coefficient column is the linear coefficient value.

Examples

The following example demonstrates an SVM model build followed by an invocation of GET_MODEL_DETAILS_SVM table function from Oracle SQL:

```
-- Create SVM model
```

```
BEGIN
  dbms_data_mining.create_model(
    model_name           => 'SVM_Clas_sample',
    mining_function      => dbms_data_mining.classification,
    data_table_name      => 'svmc_sample_build_prepared',
    case_id_column_name  => 'id',
    target_column_name   => 'affinity_card',
    settings_table_name  => 'svmc_sample_settings');
END;
/
-- Display model details
SELECT *
  FROM TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_SVM('SVM_Clas_sample'))
ORDER BY class;
```

GET_MODEL_DETAILS_XML Function

This table function returns an XML object that provides the details of a Decision Tree model.

Syntax

```
DBMS_DATA_MINING.GET_MODEL_DETAILS_XML (
  model_name      IN       VARCHAR2)
 RETURN XMLTYPE;
```

Parameters

Table 25–36 GET_MODEL_DETAILS_XML Function Parameters

Parameter	Description
model_name	Name of the model

Return Values

Table 25–37 GET_MODEL_DETAILS_XML Function Return Value

Return Value	Description
XMLTYPE	The PMML 2.1 XML definition for the decision tree model.

Usage Notes

The function returns the XML representing the decision tree; the definition is the one specified in the Data Mining Group Predictive Model Markup Language (PMML) version 2.1 specification. The specification is available at http://www.dmg.org.

GET_MODEL_SETTINGS Function

This table function returns the list of settings that were used to build the model.

Syntax

```
DBMS_DATA_MINING.GET_MODEL_SETTINGS(
   model_name           IN VARCHAR2)
 RETURN DM_MODEL_SETTINGS PIPELINED;
```

Parameters

Table 25–38 GET_MODEL_SETTINGS Function Parameters

Parameter	Description
model_name	Name of the model

Return Values

Table 25–39 GET_MODEL_SETTINGS Function Return Values

Return Value	Description
DM_MODEL_SETTINGS	Represents a set of rows of type DM_MODEL_SETTING. The rows have the following columns: (setting_name VARCHAR2(30), setting_value VARCHAR2(128))

Usage Notes

The table function pipes out rows of type DM_MODEL_SETTING. For information about ODM data types and piped output from table functions, see "Data Types" on page 25-10.

You can use this table function to determine the settings that were used to build the model. This is purely for informational purposes only — you cannot alter the model to adopt new settings.

Examples

Assume that you have built a classification model census_model using the Naive Bayes algorithm. You can retrieve the model settings using Oracle SQL as follows:

```
SELECT setting_name, setting_value
  FROM TABLE(DBMS_DATA_MINING.GET_MODEL_SETTINGS('census_model'));
```

GET_MODEL_SIGNATURE Function

This table function returns the model signature, which is a set of rows that provide the name and type of each attribute required as input to the APPLY operation.

The case identifier is not considered a mining attribute. For classification and regression models, the target attribute is also not considered part of the model signature.

Syntax

```
DBMS_DATA_MINING.GET_MODEL_SIGNATURE(
  model_name           IN VARCHAR2)
RETURN DM_MODEL_SIGNATURE PIPELINED;
```

Parameters

Table 25–40 GET_MODEL_SIGNATURE Function Parameters

Parameter	Description
model_name	Name of the model

Return Values

Table 25–41 GET_MODEL_SIGNATURE Function Return Values

Return Value	Description
DM_MODEL_SIGNATURE	Represents a set of rows of type DM_MODEL_SIGNATURE_ATTRIBUTE. The rows have the following columns: (attribute_name VARCHAR2(30), attribute_type VARCHAR2(106))

Usage Notes

This table function pipes out rows of type DM_MODEL_SIGNATURE. For information on ODM data types and piped output from table functions, see "Data Types" on page 25-10.

You can use this table function to get the list of attributes used for building the model. This is particularly helpful to describe a model when an APPLY operation on test or scoring data is done a significant time period after the model is built, or after it is imported into another definition.

Examples

Assume that you have built a classification model census_model using the Naive Bayes algorithm. You can retrieve the model details using Oracle SQL as follows:

```
SELECT attribute_name, attribute_type
  FROM TABLE(DBMS_DATA_MINING.GET_MODEL_SIGNATURE('census_model');
```

IMPORT_MODEL Procedure

This procedure imports the specified data mining models from a dump file set that was created with EXPORT_MODEL or with the expdp export utility. Both IMPORT_MODEL and EXPORT_MODEL use Oracle Data Pump technology.

> **See Also:** *Oracle Data Mining Administrator's Guide* for more information on model export and import.

Syntax

```
DBMS_DATA_MINING.IMPORT_MODEL (
    filename             IN  VARCHAR2,
    directory            IN  VARCHAR2,
    model_filter         IN  VARCHAR2 DEFAULT NULL,
    operation            IN  VARCHAR2 DEFAULT NULL,
    remote_link          IN  VARCHAR2 DEFAULT NULL,
    jobname              IN  VARCHAR2 DEFAULT NULL,
    schema_remap         IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 25–42 IMPORT_MODEL Procedure Parameters

Parameter	Description
filename	Name of the dump file set from which the models should be imported. The dump file set must have been created by the EXPORT_MODEL procedure or the expdp export utility of Oracle Data Pump. The dump file set can contain one or more files. (Refer to "EXPORT_MODEL Procedure" on page 25-34 for details.) If the dump file set contains multiple files, you can specify '*filename*%U' instead of listing them. For example, if your dump file set contains 3 files, archive01.dmp, archive02.dmp, and archive03.dmp, you can import them by specifying 'archive%U'.
directory	Name of a pre-defined directory object that specifies where the dump file set is located. You must have read/write privileges on the directory object and on the file system directory that it identifies.
model_filter	Optional parameter that specifies which model or models to import. If you do not specify a value for model_filter, all models in the dump file set are imported. You can also specify NULL (the default) or 'ALL' to import all models. You can import individual models by name and groups of models that share a given characteristic. For instance, you could import all Naive Bayes models or all models that use the same target attribute. See the Usage Notes for more information. Examples are provided in Table 25–43.
operation	Optional parameter that specifies whether to import the models or the SQL statements that create the models. By default, the models are imported. You can specify either of the following values for operation: ■ 'IMPORT' — Import the models (Default) ■ 'SQL_FILE'— Write the SQL DDL for creating the models to a text file. The text file is named *job_name*.sql and is located in the dump set directory.
remote_link	Optional parameter not used in this release. Set to NULL

Table 25–42 (Cont.) IMPORT_MODEL Procedure Parameters

Parameter	Description
jobname	Optional parameter that specifies the name of the import job. By default, the name has the form *username*_imp_*nnnn*, where *nnnn* is a number. For example, a job name in the SCOTT schema might be SCOTT_imp_134. If you specify a job name, it must be unique within the schema. The maximum length of the job name is 30 characters. A log file for the import job, named *jobname*.log, is created in the same directory as the dump file set.
schema_remap	Optional parameter for importing into a different schema. By default, models are exported and imported within the same schema. If the dump file set belongs to a different schema, you must specify a schema mapping in the form *export_user*:*import_user*. For example, you would specify 'SCOTT:MARY' to import a model exported by SCOTT into the MARY schema. NOTE: In some cases, you may need to have the IMPORT_FULL_DATABASE privilege or the SYS role to import a model from a different schema.

Usage Notes

The model_filter parameter specifies which models to import. You can list the models by name, or you can identify a group of models that share a given characteristic. To specify models by name, provide a single model name or a comma-delimited list of model names. To specify a group of models that share a characteristic, use a conditional expression that completes the WHERE clause of a query against the DM_USER_MODELS view. DM_USER_MODELS lists the models in the current schema. It has the following columns.

```
Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 NAME                                      NOT NULL VARCHAR2(25)
 FUNCTION_NAME                                      VARCHAR2(30)
 ALGORITHM_NAME                                     VARCHAR2(30)
 CREATION_DATE                                      DATE
 BUILD_DURATION                                     NUMBER
 TARGET_ATTRIBUTE                                   VARCHAR2(30)
 MODEL_SIZE                                         NUMBER
```

For descriptions of the columns in DM_USER_MODELS, see "User Views" on page 25-14.

To construct a conditional expression for model_filter, specify a column name, a supported conditional operator, and a value. The supported conditional operators are: <, <=, =, =>, >, LIKE, IN. For information on conditional operators and WHERE clauses, see *Oracle Database SQL Reference*

Examples of model filters are provided in Table 25–43.

Table 25–43 Sample Values for the Model Filter Parameter

Sample Value	Meaning
'mymodel'	Import the model named mymodel.
'mymodel2, mymodel3'	Import the models named mymodel2 and mymodel3.
'name= ''mymodel'''	Import the model named mymodel.
'name IN (''mymodel2'',''mymodel3'')'	Import the models named mymodel2 and mymodel3.

Table 25–43 (Cont.) Sample Values for the Model Filter Parameter

Sample Value	Meaning
`'name LIKE ''AI%'''`	Import all models that have names starting with `AI`.
`'ALGORITHM_NAME = ''NAIVE_BAYES'''`	Import all Naive Bayes models. See Table 25–2 for a list of algorithm names.
`'FUNCTION_NAME =''CLASSIFICATION'''`	Import all classification models. See Table 25–1 for a list of mining functions.

Examples

This example shows a model being exported and imported within the schema `dmuser2`. Then the same model is imported into the `dmuser3` schema. The `dmuser3` user has the `IMPORT_FULL_DATABASE` privilege.

```
SQL>CONNECT dmuser2/dmuser2_psw
SQL>SELECT name FROM dm_user_models;
    NAME
    ------------------------
    NMF_SH_SAMPLE
    SVMO_SH_CLAS_SAMPLE
    SVMR_SH_REGR_SAMPLE

-- export the model called NMF_SH_SAMPLE to a dump file in same schema
SQL>EXECUTE DBMS_DATA_MINING.EXPORT_MODEL ('NMF_SH_SAMPLE_out', 'DATA_PUMP_DIR',
                            'name = ''NMF_SH_SAMPLE''');
-- import the model back into the same schema
SQL>EXECUTE DBMS_DATA_MINING.IMPORT_MODEL ('NMF_SH_SAMPLE_out01.dmp',
                            'DATA_PUMP_DIR', 'name = ''NMF_SH_SAMPLE''');

-- connect as different user
-- import same model into that schema
SQL>CONNECT dmuser3/dmuser3_psw
SQL>EXECUTE DBMS_DATA_MINING.IMPORT_MODEL ('NMF_SH_SAMPLE_out01.dmp',
                            'DATA_PUMP_DIR', 'name = ''NMF_SH_SAMPLE''',
                            'IMPORT', NULL, 'nmf_imp_job', 'dmuser2:dmuser3');
```

The following example shows user `MARY` importing all models from a dump file, `model_exp_001.dmp`, which was created by user `SCOTT`. The dump file is located in the file system directory mapped to a directory object called `DM_DUMP`. If user `MARY` does not have `IMPORT_FULL_DATABASE` privileges, `IMPORT_MODEL` will raise an error.

```
-- import all models
DECLARE
  file_name       VARCHAR2(40);
BEGIN
  file_name := 'model_exp_001.dmp';
  DBMS_DATA_MINING.IMPORT_MODEL(
                filename=>file_name,
               directory=>'DM_DUMP',
                schema_remap=>'SCOTT:MARY');
  DBMS_OUTPUT.PUT_LINE(
'DBMS_DATA_MINING.IMPORT_MODEL of all models from SCOTT done!');
END;
/
```

RANK_APPLY Procedure

This procedure ranks the results of an APPLY operation based on a top-N specification for predictive and descriptive model results. For classification models, you can provide a cost matrix as input, and obtain the ranked results with costs applied to the predictions.

Syntax

```
DBMS_DATA_MINING.RANK_APPLY (
      apply_result_table_name        IN VARCHAR2,
      case_id_column_name            IN VARCHAR2,
      score_column_name              IN VARCHAR2,
      score_criterion_column_name    IN VARCHAR2,
      ranked_apply_result_tab_name   IN VARCHAR2,
      top_N                          IN INTEGER DEFAULT 1,
      cost_matrix_table_name         IN VARCHAR2 DEFAULT NULL,
      apply_result_schema_name       IN VARCHAR2 DEFAULT NULL,
      cost_matrix_schema_name        IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 25–44 RANK_APPLY Procedure Parameters

Parameter	Description
apply_result_table_name	Name of the table or view containing the results of an APPLY operation on the test dataset (see Usage Notes)
case_id_column_name	Name of the case identifier column. This must be the same as the one used for generating APPLY results.
score_column_name	Name of the prediction column in the apply results table
score_criterion_column_name	Name of the probability column in the apply results table
ranked_apply_result_tab_name	Name of the table containing the ranked apply results
top_N	Top N predictions to be considered from the APPLY results for precision recall computation
cost_matrix_table_name	Name of the cost matrix table
apply_result_schema_name	Name of the schema hosting the APPLY results table
cost_matrix_schema_name	Name of the schema hosting the cost matrix table

Usage Notes

You can use RANK_APPLY to generate ranked apply results, based on a top-N filter and also with application of cost for predictions, if the model was built with costs.

The behavior of RANK_APPLY is similar to that of APPLY with respect to other DDL-like operations such as CREATE_MODEL, DROP_MODEL, and RENAME_MODEL. The procedure does not depend on the model; the only input of relevance is the apply results generated in a fixed schema table from APPLY.

The main intended use of RANK_APPLY is for the generation of the final APPLY results against the scoring data in a production setting. You can apply the model against test

data using APPLY, compute various test metrics against various cost matrix tables, and use the candidate cost matrix for RANK_APPLY.

The schema for the apply results from each of the supported algorithms is listed in subsequent sections. The case_id column will be the same case identifier column as that of the apply results.

Classification Models — NB, ABN, SVM

For numerical targets, the ranked results table will have the definition as shown:

```
(case_id       VARCHAR2/NUMBER,
prediction     NUMBER,
probability    NUMBER,
cost           NUMBER,
rank           INTEGER)
```

For categorical targets, the ranked results table will have the following definition:

```
(case_id       VARCHAR2/NUMBER,
prediction     VARCHAR2,
probability    NUMBER,
cost           NUMBER,
rank           INTEGER)
```

Clustering using *k*-Means or O-Cluster

Clustering is an unsupervised mining function, and hence there are no targets. The results of an APPLY operation contains simply the cluster identifier corresponding to a case, and the associated probability. Cost matrix is not considered here. The ranked results table will have the definition as shown, and contains the cluster ids ranked by top-N.

```
(case_id       VARCHAR2/NUMBER,
cluster_id     NUMBER,
probability    NUMBER,
rank           INTEGER)
```

Feature Extraction using NMF

Feature extraction is also an unsupervised mining function, and hence there are no targets. The results of an APPLY operation contains simply the feature identifier corresponding to a case, and the associated match quality. Cost matrix is not considered here. The ranked results table will have the definition as shown, and contains the feature ids ranked by top-N.

```
(case_id        VARCHAR2/NUMBER,
feature_id      NUMBER,
match_quality   NUMBER,
rank            INTEGER)
```

Examples

```
BEGIN
/* build a model with name census_model.
 * (See example under CREATE_MODEL)
 */

/* if build data was pre-processed in any manner,
 * perform the same pre-processing steps on apply
 * data also.
 * (See examples in the section on DBMS_DATA_MINING_TRANSFORM)
```

```
 */

/* apply the model to data to be scored */
DBMS_DATA_MINING.RANK_APPLY(
  apply_result_table_name       => 'census_apply_result',
  case_id_column_name           => 'person_id',
  score_column_name             => 'prediction',
  score_criterion_column_name   => 'probability
  ranked_apply_result_tab_name  => 'census_ranked_apply_result',
  top_N                         => 3,
  cost_matrix_table_name        => 'census_cost_matrix');
END;
/

-- View Ranked Apply Results
SELECT *
  FROM census_ranked_apply_result;
```

RENAME_MODEL Procedure

This procedure renames a mining model to a specified new name.

Syntax

```
DBMS_DATA_MINING.RENAME_MODEL (
     model_name            IN VARCHAR2,
     new_model_name        IN VARCHAR2);
```

Parameters

Table 25–45 RENAME_MODEL Procedure Parameters

Parameter	Description
model_name	Old name of the model
new_model_name	New name of the model (See "Model Names" on page 25-5)

Usage Notes

You can use RENAME_MODEL to rename an existing mining model.

The behavior of the RENAME_MODEL is similar to a SQL DDL RENAME operation. It blocks DROP_MODEL and CREATE_MODEL operations. It does not block APPLY, which is a SQL query-like operation that does not update any model data.

If an APPLY operation is using a model, and you attempt to rename the model during that time, the RENAME will succeed and APPLY will return indeterminate results. This is in line with the conventional behavior in the RDBMS, where DDL operations do not block on query operations.

Examples

Assume the existence of a model census_model. The following example shows how to rename this model.

```
BEGIN
  DBMS_DATA_MINING.RENAME_MODEL(
    model_name      => 'census_model',
    new_model_name  => 'census_new_model');
END;
/
```

26

DBMS_DATA_MINING_TRANSFORM

The DBMS_DATA_MINING_TRANSFORM package contains a set of data transformation utilities that prepare data for data mining. Once you have prepared the data, you can use it to build and score models using the DBMS_DATA_MINING package or the Oracle Data Mining (ODM) Java API. You can also score models using the SQL scoring functions for data mining.

DBMS_DATA_MINING_TRANSFORM is an open-source PL/SQL package. You can use the routines in this package to prepare your data for data mining, or you can develop your own routines based on the public source code.

The source code, interface definitions, and inline documentation are available in $ORACLE_HOME/rdbms/admin/dbmsdmxf.sql.

See Also:

- Chapter 25, "DBMS_DATA_MINING". This package provides routines for building, scoring, exporting, and importing models. It also provides functions that return information about models.
- Chapter 72, "DBMS_PREDICTIVE_ANALYTICS". This package automates the entire process of predictive data mining, from data preprocessing through model building to scoring new data.
- *Oracle Data Mining Administrator's Guide* for information about sample data mining programs.
- *Oracle Data Mining Concepts* for Data Mining concepts.
- *Oracle Data Mining Application Developer's Guide* for information about developing data mining applications in SQL and Java.

This chapter contains the following topics:

Using DBMS_DATA_MINING_TRANSFORM

This section contains topics which relate to using the `DBMS_DATA_MINING_TRANSFORM` package.

Overview

The DBMS_DATA_MINING_TRANSFORM package serves two purposes:

- It is a basic utility package for preprocessing data for data mining.
- It is an open-source learning tool that shows how to use SQL to perform common data transformations for data mining. The inputs and outputs for routines in this package are simple views and tables that are not Oracle proprietary. You can study the source code to help you create data transforms that are specific to your own application data. The source code for this package is in dbmsdmxf.sql in the rdbms/admin directory under $ORACLE_HOME.

Note: Use of the DBMS_DATA_MINING_TRANSFORM package is not required by Oracle Data Mining. You can develop your own preprocessing utilities or use third-party tools customized for your application.

The main principle behind the design of DBMS_DATA_MINING_TRANSFORM is the fact that SQL has enough power to perform most of the common mining transforms efficiently. For example, binning can be done using CASE expressions or DECODE functions, and linear normalization is a simple algebraic expression of the form (*x* - shift)/scale where *x* is the data value that is being transformed.

However, the queries that perform the transforms can be rather lengthy. So it is desirable to have some convenience routines that will help in generating queries. Thus, the goal of this package is to provide query generation services for the most common mining transforms, as well as to provide a framework that can be easily extended for implementing other transforms.

Note on Notation: This chapter uses standard interval notation for number sets:

- **[a,b]** is the set of all real numbers greater than or equal to **a** and less than or equal to **b**
- **[a,b)** is the set of all real numbers greater than or equal to **a** and less than **b**.

(b is in the set [a,b]; b is not in the set [a,b).)

Subscripts do not conform to standard notation; instead "X_N" is used for " X_N."

See Also: Sample data mining programs are available with Oracle Data Mining. These programs include sample data transformations using DBMS_DATA_MINING_TRANSFORM. Instructions for using the sample programs are provided in the *Oracle Data Mining Administrator's Guide*.

Types

Table 26–1 Summary of Data Types

Data Type	Purpose
`Column_List`	List of column names representing mining attributes, defined to be `VARRAY(1000)` of `VARCHAR2(32)`.

Transformation Methods

The DBMS_DATA_MINING_TRANSFORM package supports binning, normalization, winsorizing and clipping, and missing value transformations.

Binning

Binning involves mapping both continuous and discrete values to discrete values of reduced cardinality. For example, the age of persons can be binned into discrete numeric bins: 1-20 to 1, 21-40 to 2, and so on. Popular car manufacturers such as Ford, Chrysler, BMW, Volkswagen can be binned into discrete categorical bins: {Ford, Chrysler} to US_Car_Makers, and {BMW, Volkswagen} to European_Car_Makers.

DBMS_DATA_MINING_TRANSFORM supports binning for both categorical and numerical attributes. Categorical attributes have VARCHAR2/CHAR data types; numerical attributes have NUMBER data types.

Top-N Frequency Categorical Binning

The bin definition for each attribute is computed based on the occurrence frequency of values that are computed from the data. The user specifies a particular number of bins, say N. Each of the bins bin_1,..., bin_N corresponds to the values with top frequencies. The bin bin_N+1 corresponds to all remaining values.

Equi-Width Numerical Binning

The bin definition for each attribute is computed based on the minimum and maximum values that are computed from the data. The user specifies a particular number of bins, say N. Each of the bins bin_1,..., bin_N span ranges of equal width of size inc = (max – min)/N, bin_0 spans (–inf, min) and bin_(N+1) spans (max, + inf). When N is not specified, it can be estimated from the data.

Quantile Numerical Binning

The definition for each relevant column is computed based on the minimum values for each quantile, where quantiles are computed from the data using NTILE function. Bins bin_1,..., bin_N span the following ranges: bin_1 spans [min_1,min_2]; bin_2,..., bin_i,..., bin_N-1 span (min_i, min_(i+1)] and bin_N spans (min_N, max_N]. Bins with equal left and right boundaries are collapsed.

Normalization

Normalization involves scaling continuous values down to a specific range, such as [–1.0,1.0] or [0.0,1.0] such that `x_new = (x_old-shift)/scale`. Normalization applies only to numerical attributes.

Min-Max Normalization

The normalization definition for each attribute is computed based on the minimum and maximum values of the data. The values for `shift` and `scale` are `shift = min`, and `scale = (max - min)` respectively.

Scale Normalization

The normalization definition for each attribute is computed based on the minimum and maximum values of the data. The values for `shift` and `scale` are `shift = 0` and `scale = max{abs(max), abs(min)}`.

Z-Score Normalization

The normalization definition for each attribute is computed based on the values for mean and standard deviation that are computed from the data. The values for `shift`

and scale are computed to be shift = mean, and scale = standard deviation respectively.

Winsorizing and Trimming (Clipping)

Some computations on attribute values can be significantly affected by extreme values. One approach to achieving a more robust computation is to either Winsorize or trim the data as a preprocessing step.

Winsorizing involves setting the tail values of a particular attribute to some specified value. For example, for a 90% Winsorization, the bottom 5% are set equal to the minimum value in the 6th percentile, while the upper 5% are set equal to the value corresponding to the maximum value in the 95th percentile.

Trimming "removes" the tails in the sense that trimmed values are ignored in further values. This is achieved by setting the tails to NULL.

Missing Value Treatment

Missing Value treatment involves replacing NULL values in the data. Missing Value treatment is recommended when the fraction of missing values is high compared to the overall attribute value set. If the data contains relatively few missing values, you might choose to simply delete those records for the purpose of data mining.

If you want to replace missing values and you know or suspect what the values should be, you can use that knowledge to replace the NULLs. If you suspect that the NULLs may be random omissions, you can determine a meaningful value for them.

DBMS_DATA_MINING_TRANSFORM INSERT routines handle missing values by replacing NULLs in numerical attributes with the mean attribute value, and by replacing NULLs in categorical attributes with the mode.

Steps in Defining a Transformation

DBMS_DATA_MINING_TRANSFORM provides routines that define CREATE, INSERT, and XFORM operations. To define a data transformation, perform the following steps:

1. Use a CREATE routine to create a transformation definition table with a pre-defined set of columns.
2. Use an INSERT routine to populate the table with transformation definitions for selected attributes.
3. Use an XFORM routine to create a view of the transformation definition table.

Creating a Transformation Definition Table

Use the following procedures to create transformation definition tables:

- CREATE_BIN_NUM and CREATE_BIN_CAT to create bin definition tables.
- CREATE_NORM_LIN to create a normalization definition table.
- CREATE_CLIP to create a clipping definition table.
- CREATE_MISS_NUM and CREATE_MISS_CAT to create missing value treatment definition tables.

Usually, the consistency and integrity of transform definition tables is guaranteed by the creation process. Alternatively, it can be achieved by leveraging an integrity constraints mechanism. This can be done either by altering the tables created with CREATE routines, or by creating the tables manually with the necessary integrity constraints.

Defining a Transformation

The most common way of defining a transformation (populating the transformation definition tables) for each attribute is based on data inspection using some predefined methods (also known as automatic transform definition). Some of the most popular methods have been captured by the INSERT routines in DBMS_DATA_MINING_TRANSFORM. For example, the z-score normalization method estimates mean and standard deviation from the data to be used as a shift and scale parameters of the linear normalization transform.

Use the following procedures to populate the transformation definition tables:

- INSERT_BIN_NUM_EQWIDTH, INSERT_BIN_NUM_QTILE, and INSERT_AUTOBIN_NUM_EQWIDTH for binning numerical attributes, and INSERT_BIN_CAT_FREQ for binning categorical attributes.
- INSERT_NORM_LIN_MINMAX, INSERT_NORM_LIN_SCALE, or INSERT_NORM_LIN_ZSCORE for normalizing numerical attributes.
- INSERT_CLIP_WINZOR_TAIL for winsorizing numerical attributes or INSERT_CLIP_TRIM_TAIL for clipping numerical attributes.
- INSERT_MISS_NUM_MEAN for missing value treatment of numerical attributes, and INSERT_MISS_CAT_MODE for missing value treatment of categorical attributes.

You can invoke these routines several times to transform all relevant attributes from various data sources until the definition table fully represents all mining attributes for a given problem.

After performing automatic transform definitions, some or all of the definitions can be adjusted by issuing SQL DML statements against the transform definition tables, thus providing virtually infinite flexibility in defining custom transforms.

The `INSERT` routines enable flexible transformation definitions in several ways:

- The data provided to the `INSERT` routines does not necessarily have to be the data used for a particular model creation. It can be any data that contains adequate representation of the mining attributes.
- The `INSERT` routines can be called any number of times against the same or different dataset until all the attributes have their transformations defined. You can selectively exclude one or more attributes for a particular iteration of the `INSERT`. In the most extreme case, each individual attribute can potentially have a unique transformation definition.
- You do not have to separately feed in numerical and categorical attributes, since categorical binning automatically skips over `NUMBER` columns in your table, and numerical binning, normalization, and clipping routines skip over `VARCHAR2/CHAR` columns in your input data.

Generating the Query for a Transform

Query generation is driven by the simple transform-specific definition tables with predefined columns. Query generation routines should be viewed as macros, and transformation definition tables as parameters used in macro expansions. Similar to using #define macros in the C language, the invoker is responsible for ensuring the correctness of the expanded macro, that is, that the result is a valid SQL query.

You can generate the views representing the transformation queries with the following procedures:

- `XFORM_BIN_NUM`, and `XFORM_BIN_CAT` for binning
- `XFORM_NORM_LIN` for normalization
- `XFORM_CLIP` for clipping
- `XFORM_MISS_NUM` and `XFORM_MISS_CAT` for missing value treatment

If your data contains a combination of numerical and categorical attributes, you must essentially feed the results of one transformation step to the next step. For example, the results of `XFORM_BIN_CAT` can be fed to `XFORM_BIN_NUM` or vice versa. The order is irrelevant since numerical and categorical transforms work on disjoint sets of attributes.

Sample Transformation

Given a dataset for a particular mining problem, any preprocessing and transformations on the mining data must be uniform across all mining operations. In other words, if the build data is preprocessed according to a particular transformation definition, then it follows that the test data and the scoring data must be preprocessed using the same definition.

The general usage of routines in this package can be explained using this example. Assume that your input table contains both numerical and categorical data that requires binning. A possible sequence of operations will be:

1. Invoke `CREATE_BIN_NUM` to generate an empty numerical bin definition table.
2. Invoke `INSERT_BIN_NUM_EQWIDTH` to define the transformations for all numerical attributes in the build data input. (For the sake of simplicity, let us assume that all numerical values are to be binned into 10 bins.) If you are binning for an O-Cluster model, use `INSERT_AUTOBIN_NUM_EQWIDTH`.
3. Next invoke `XFORM_BIN_NUM` with the numerical bin table and the build data table as inputs. The resulting object is a view that represents a SQL query against the build data table that performs numerical binning. Assume that you have named this result object `build_bin_num_view`.
4. Since you still have the categorical attributes to be binned, invoke `CREATE_BIN_CAT` to create a categorical bin definition table.
5. Next, invoke `INSERT_BIN_CAT_FREQ` to define the transforms for all categorical attributes. (For the sake of simplicity, let us assume that all categorical attributes are to be binned into 10 bins.)
6. As the final step, invoke `XFORM_BIN_CAT` with the categorical bin table and the view name provided by `XFORM_BIN_NUM`, namely `build_bin_num_view`, as the inputs. This essentially amounts to combining the transformations from both stages.
7. The object resulting from this operation is a view that represents a SQL query against your build data table, influenced by the contents of the bin definition tables. Provide this view name as the data input to the `CREATE_MODEL` procedure in the `DBMS_DATA_MINING` package.

If this happens to be a classification model, and you want to `APPLY` this model to scoring data, you must prepare the scoring data similar to the build data. You can achieve this in two simple steps:

1. First, call `XFORM_BIN_NUM` with the scoring data table and the numerical bin boundary table as inputs. The resulting object is a view that represents an SQL query against your scoring data table, influenced by the contents of the numerical bin boundary table. Assume that you have named this result object `apply_bin_num_view`.
2. As the next and final step, invoke `XFORM_BIN_CAT` with the categorical bin table and the view name provided by `XFORM_BIN_NUM`, namely `apply_bin_num_view`, as the inputs.
3. The object resulting from this operation is now a view that represents a SQL query against your scoring data table, influenced by the contents of the bin definition tables. Provide this view name as the data input to the `APPLY` procedure in the `DBMS_DATA_MINING` package.

Summary of DBMS_DATA_MINING_TRANSFORM Subprograms

Table 26–2 DBMS_DATA_MINING_TRANSFORM Package Subprograms

Subprogram	Purpose
CREATE_BIN_CAT Procedure on page 26-12	Creates a categorical bin definition table
CREATE_BIN_NUM Procedure on page 26-13	Creates a numerical bin definition table
CREATE_CLIP Procedure on page 26-14	Creates a clipping definition table
CREATE_MISS_CAT Procedure on page 26-15	Creates a categorical missing value treatment definition table
CREATE_MISS_NUM Procedure on page 26-16	Creates a numerical missing value treatment definition table
CREATE_NORM_LIN Procedure on page 26-17	Creates a normalization definition table
INSERT_AUTOBIN_NUM_ EQWIDTH Procedure on page 26-18	Populates the numerical bin definition table, using the number of bins estimated from the data
INSERT_BIN_CAT_FREQ Procedure on page 26-20	Populates the categorical bin definition table, applying frequency-based binning to the categorical input data
INSERT_BIN_NUM_EQWIDTH Procedure on page 26-22	Populates the numerical bin definition table, applying equi-width binning to the numerical input data
INSERT_BIN_NUM_QTILE Procedure on page 26-24	Populates the numerical bin definition table, applying quantile binning to the numerical input data
INSERT_CLIP_TRIM_TAIL Procedure on page 26-26	Populates the clipping definition table, applying trimming based on tail fraction to the numerical input data
INSERT_CLIP_WINSOR_TAIL Procedure on page 26-28	Populates the clipping definition table, applying Winsorizing based on tail fraction to the numerical input data
INSERT_MISS_CAT_MODE Procedure on page 26-30	Populates the categorical missing value treatment definition table, applying the mode to each missing value
INSERT_MISS_NUM_MEAN Procedure on page 26-31	Populates the numerical missing value treatment definition table, applying the mean to each missing value
INSERT_NORM_LIN_MINMAX Procedure on page 26-32	Populates the normalization definition table, applying min-max normalization to the numerical input data
INSERT_NORM_LIN_SCALE Procedure on page 26-33	Populates the normalization definition table, applying scale normalization to the numerical input data
INSERT_NORM_LIN_ZSCORE Procedure on page 26-34	Populates the normalization definition table applying z-score normalization to the numerical input data
XFORM_BIN_CAT Procedure on page 26-35	Creates the view representing the transformed output with binned categorical data
XFORM_BIN_NUM Procedure on page 26-37	Creates the view representing the transformed output with binned numerical data
XFORM_CLIP Procedure on page 26-40	Creates the view representing the transformed output with clipped numerical data

Table 26–2 (Cont.) DBMS_DATA_MINING_TRANSFORM Package Subprograms

Subprogram	Purpose
XFORM_MISS_CAT Procedure on page 26-42	Creates the view representing the transformed output with categorical missing value treatment
XFORM_MISS_NUM Procedure on page 26-43	Creates the view representing the transformed output with numerical missing value treatment
XFORM_NORM_LIN Procedure on page 26-44	Creates the view representing the transformed output with normalized numerical data

CREATE_BIN_CAT Procedure

This procedure creates a categorical binning definition table. This table is used as input to the INSERT_BIN_CAT_FREQ and XFORM_BIN_CAT procedures.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.CREATE_BIN_CAT (
     bin_table_name     IN VARCHAR2,
     bin_schema_name    IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 26–3 CREATE_BIN_CAT Procedure Parameters

Parameter	Description
bin_table_name	Name of the bin definition table
bin_schema_name	Name of the schema hosting the bin definition table

Usage Notes

The generated bin definition table will have the following columns.

Column Name	Data Type
col	VARCHAR2(30)
val	VARCHAR2(4000)
bin	VARCHAR2(4000)

Examples

```
BEGIN
   DBMS_DATA_MINING_TRANSFORM.CREATE_BIN_CAT('build_bin_cat_table');
END;
```

CREATE_BIN_NUM Procedure

This procedure creates a numerical binning definition table. This table is used as input to the INSERT_BIN_NUM_EQWIDTH, INSERT_BIN_NUM_QTILE, INSERT_AUTOBIN_NUM_EQWIDTH, and XFORM_BIN_NUM procedures.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.CREATE_BIN_NUM (
     bin_table_name    IN VARCHAR2,
     bin_schema_name   IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 26–4 CREATE_BIN_NUM Procedure Parameters

Parameter	Description
bin_table_name	Name of the bin definition table
bin_schema_name	Name of the schema hosting the bin definition table

Usage Notes

The generated bin definition table will have the following columns.

Column Name	Data Type
col	VARCHAR2(30)
val	NUMBER
bin	VARCHAR2(4000)

Examples

```
BEGIN
   DBMS_DATA_MINING_TRANSFORM.CREATE_BIN_NUM('build_bin_num_table');
END;
```

CREATE_CLIP Procedure

This procedure creates a clipping definition table. This table is used as input to the INSERT_CLIP_WINSOR_TAIL, INSERT_CLIP_TRIM_TAIL, and XFORM_CLIP procedures.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.CREATE_CLIP (
     clip_table_name    IN VARCHAR2,
     clip_schema_name   IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 26–5 CREATE_CLIP Procedure Parameters

Parameter	Description
clip_table_name	Name of the clipping definition table
clip_schema_name	Name of the schema hosting the clipping definition table

Usage Notes

The generated clipping definition table will have the following columns.

Column Name	Data Type
col	VARCHAR2(30)
lcut	NUMBER
lval	NUMBER
rcut	NUMBER
rval	NUMBER

Examples

```
BEGIN
  DBMS_DATA_MINING_TRANSFORM.CREATE_CLIP('build_clip_table');
END;
```

CREATE_MISS_CAT Procedure

This procedure creates a categorical missing value treatment definition table. This table is used as input to the INSERT_MISS_CAT_MODE procedure.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.CREATE_MISS_CAT (
     miss_table_name       IN VARCHAR2,
     miss_schema_name      IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 26–6 CREATE_MISS_CAT Procedure Parameters

Parameter	Description
miss_table_name	Name of the categorical missing value treatment definition table
miss_schema_name	Name of the schema hosting the categorical missing value treatment definition table

Usage Notes

The generated categorical missing value treatment definition table will have the following columns.

Column Name	Data Type
col	VARCHAR2(30)
val	VARCHAR2(4000)

Examples

```
BEGIN
   DBMS_DATA_MINING_TRANSFORM.CREATE_MISS_CAT('build_miss_cat_table');
END;
```

CREATE_MISS_NUM Procedure

This procedure creates a numerical missing value treatment definition table. This table is used as input to the INSERT_MISS_NUM_MEAN procedure.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.CREATE_MISS_NUM (
     miss_table_name       IN VARCHAR2,
     miss_schema_name      IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 26–7 CREATE_MISS_NUM Procedure Parameters

Parameter	Description
miss_table_name	Name of the numeric missing value treatment definition table
miss_schema_name	Name of the schema hosting the numeric missing value treatment definition table

Usage Notes

The generated numeric missing value definition table will have the following columns.

Column Name	Data Type
col	VARCHAR2(30)
val	NUMBER

Example

```
BEGIN
  DBMS_DATA_MINING_TRANSFORM.CREATE_MISS_NUM('build_miss_num_table');
END;
```

CREATE_NORM_LIN Procedure

This procedure creates a linear normalization definition table. This table is used as input to the INSERT_NORM_LIN_MINMAX, INSERT_NORM_LIN_SCALE, INSERT_NORM_LIN_ZSCORE, and XFORM_NORM_LIN procedures.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.CREATE_NORM_LIN (
     norm_table_name       IN VARCHAR2,
     norm_schema_name      IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 26–8 CREATE_NORMALIZE_LIN Procedure Parameters

Parameter	Description
norm_table_name	Name of the normalization definition table
norm_schema_name	Name of the schema hosting the normalization definition table

Usage Notes

The generated linear normalization definition table will have the following columns.

Column Name	Data Type
col	VARCHAR2(30)
shift	NUMBER
scale	NUMBER

Examples

```
BEGIN
   DBMS_DATA_MINING_TRANSFORM.CREATE_NORM_LIN('build_norm_table');
END;
```

INSERT_AUTOBIN_NUM_EQWIDTH Procedure

This procedure finds the numerical binning definition for every numerical column in the data table that is not specified in the exclusion list and inserts the definition into the numerical binning definition table that was created using CREATE_BIN_NUM. Based on the statistical information it collects on the input data, this procedure calculates the number of bins.

Definition for each relevant column is computed based on the minimum and maximum values that are computed from the data table.

N, the number of bins, is computed for each column separately and is based on the number of non-NULL values (cnt), the maximum (max), the minimum (min), the standard deviation (dev) and the constant C=3.49/0.9 as follows:

```
N=floor(power(cnt,1/3)*(max-min)/(c*dev))
```

Each of the bin_num (= N) bins bin_1,..., bin_N span ranges of equal width inc = (max – min) / N where bin_I = I when N > 0 or bin_I = N+1–I when N < 0, and bin_0 = bin_(N+1) = NULL. The values of the val column are rounded to round_num significant digits prior to scoring them in the definition table.

The parameter bin_num is used to adjust N to be at least bin_num. No adjustment is done when bin_num is NULL or 0. The parameter max_bin_num is used to adjust N to be at most max_bin_num. No adjustment is done when bin_num is NULL or 0. For columns with all integer values (discrete columns), N is adjusted to be at most the maximum number of distinct values in the observed range max-min+1.

The parameter sample_size is used to adjust cnt to be at most sample_size. No adjustment is done when sample_size is NULL or 0.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_AUTOBIN_NUM_EQWIDTH (
     bin_table_name        IN VARCHAR2,
     data_table_name       IN VARCHAR2,
     bin_num               IN PLS_INTEGER DEFAULT 3,
     max_bin_num           IN PLS_INTEGER DEFAULT 100,
     exclude_list          IN Column_List DEFAULT NULL,
     round_num             IN PLS_INTEGER DEFAULT 6,
     sample_size           IN PLS_INTEGER DEFAULT 50000,
     bin_schema_name       IN VARCHAR2 DEFAULT NULL,
     data_schema_name      IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 26–9 INSERT_AUTOBIN_EQWIDTH Procedure Parameters

Parameter	Description
bin_table_name	Name of the categorical bin table generated using CREATE_BIN_NUM procedure
data_table_name	Name of the table containing the data
bin_num	Minimum number of bins; default number is 3
max_bin_num	Maximum number of bins that sets the upper limit for estimates of bin numbers; default is 100

Table 26–9 (Cont.) INSERT_AUTOBIN_EQWIDTH Procedure Parameters

Parameter	Description
exclude_list	List of columns (attributes) to be excluded from this iteration of the binning process; categorical attributes are automatically excluded
round_num	Number of significant digits; default is 6
sample_size	Size of the data sample; default is 50,000
bin_schema_name	Name of the schema hosting the bin definition table; default is user schema
data_schema_name	Name of the schema hosting the table with data; default is user schema

Usage Notes

For a given input table, you can call this routine several times with different specifications for number of bins for a given input table. For each call, you can selectively exclude attributes (that is, column names) using the exclude_list parameter for a particular binning specification.

Columns with all NULL values or only one unique value are ignored. The sign of bin_num, max_bin_num, and sample_size have no effect on the result; absolute values are used. The value adjustment of N is done in the following order: First bin_num, next, max_bin_num, and, finally, discrete column adjustment.

Examples

The simplest invocation of this routine populates bin definitions in the num_bin_table for all the numerical attributes found in build_table.

```
BEGIN
   DBMS_DATA_MINING_TRANSFORM.INSERT_AUTOBIN_NUM_EQUIWIDTH(
      'num_bin_table', 'build_data_table');
END;
/
```

INSERT_BIN_CAT_FREQ Procedure

This procedure finds the categorical binning definition for every VARCHAR2 and CHAR column in the data table that is not specified in the exclusion list and inserts the definition into the categorical binning definition table created using CREATE_BIN_CAT.

Definition for each relevant column is computed based on the occurrence frequency of column values that are computed from the data table. Each of the bin_num(N) bins bin_1, ..., bin_N corresponds to the values with top frequencies when N > 0 or bottom frequencies when N < 0, and bin_(N+1) to all remaining values, where bin_I = I. Ordering ties among identical frequencies are broken by ordering on column values (ASC for N > 0 or DESC for N < 0). When the number of distinct values C < N only C+1 bins will be created.

The parameter default_num (D) is used for pruning based on the number of values that fall into the default bin. When D > 0 only columns that have at least D defaults are kept while others are ignored. When D < 0 only columns that have at most D values are kept. No pruning is done when D is NULL or D = 0. Parameter bin_support (SUP) is used for restricting bins to frequent (SUP > 0) values frq >= SUP*tot, or infrequent (SUP < 0) ones frq <= –SUP*tot, where frq is a given value count and tot is a sum of all counts as computed from the data. No support filtering is done when SUP is NULL or when SUP = 0.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_BIN_CAT_FREQ (
     bin_table_name       IN VARCHAR2,
     data_table_name      IN VARCHAR2,
     bin_num              IN PLS_INTEGER DEFAULT 9,
     exclude_list         IN Column_List DEFAULT NULL,
     default_num          IN PLS_INTEGER DEFAULT 2,
     bin_support          NUMBER DEFAULT NULL,
     bin_schema_name      IN VARCHAR2 DEFAULT NULL,
     data_schema_name     IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 26–10 INSERT_BIN_CAT_FREQ Procedure Parameters

Parameter	Description
bin_table_name	Name of the categorical bin table generated using CREATE_BIN_CAT procedure
data_table_name	Name of the table containing the data
bin_num	Number of bins
exclude_list	List of columns (attributes) to be excluded from this iteration of the binning process
default_num	Number of default values
bin_support	Bin support as a fraction
bin_schema_name	Name of the schema hosting the bin definition table
data_schema_name	Name of the schema hosting the table with data

Usage Notes

For a given input table, you can iteratively call this routine several times with different specifications for number of bins for a given input table. For each iteration, you can selectively exclude attributes (that is, column names) using the `exclude_list` parameter for a particular binning specification.

Columns with all `NULL`s are ignored. No bin definitions are populated when bin_num = 0, or bin_num, is `NULL`.

Examples

The simplest invocation of this routine populates bin definitions in the `cat_bin_table` for all the categorical attributes found in `build_table`.

```
BEGIN
   DBMS_DATA_MINING_TRANSFORM.INSERT_BIN_NUM(
      'cat_bin_table', 'build_table');
END;
/
```

INSERT_BIN_NUM_EQWIDTH Procedure

This procedure finds the numerical binning definition for every NUMBER column in the data table that is not specified in the exclusion list and inserts the definition into the numerical binning definition table that was created using CREATE_BIN_NUM.

Definition for each relevant column is computed based on the minimum and maximum values that are computed from the data table. Each of the bin_num (= N) bins bin_1,..., bin_N span ranges of equal width inc = (max – min) / N where bin_I = I when N > 0 or bin_I = N+1–I when N < 0, and bin_0 = bin_(N+1) = NULL.

The values of the val column in the bin definition table are rounded to round_num significant digits. For more information, see the Usage Notes.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_BIN_NUM_EQWIDTH (
     bin_table_name        IN VARCHAR2,
     data_table_name       IN VARCHAR2,
     bin_num               IN PLS_INTEGER DEFAULT 10,
     exclude_list          IN Column_List DEFAULT NULL,
     round_num             IN PLS_INTEGER DEFAULT 6,
     bin_schema_name       IN VARCHAR2 DEFAULT NULL,
     data_schema_name      IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 26–11 INSERT_BIN_EQWIDTH Procedure Parameters

Parameter	Description
bin_table_name	Name of the numerical bin table generated using CREATE_BIN_NUM procedure
data_table_name	Name of the table containing the data
bin_num	Number of bins
exclude_list	List of columns (attributes) to be excluded from this iteration of the binning process
round_num	Number of significant digits. See Usage Notes.
bin_schema_name	Name of the schema hosting the bin definition table
data_schema_name	Name of the schema hosting the table with data

Usage Notes

For a given input table, you can iteratively call this routine with different specifications for number of bins for a given input table. For each iteration, you can selectively exclude attributes (that is, column names) using the exclude_list parameter for a particular binning specification.

Columns with all NULL values or only one unique value are ignored. No bin definitions are populated when bin_num = 0, or bin_num is NULL.

For example, when N=2, col='mycol', min=10, and max = 21, the following three rows are inserted into the definition table (inc = 5.5):

```
COL     VAL  BIN
----- -----  -----
mycol    10  NULL
```

```
 mycol  15.5  1
 mycol    21  2
```

The round_num parameter specifies how to round the number in the VAL column of the definition table. When round_num is positive, it specifies the most significant digits to retain. When round_num is negative, it specifies the least significant digits to remove. In both cases, the result is rounded to the specified number of digits. When round_num is 0, the value is unchanged.

For example, a value of 308.162 would be rounded as follows.

```
For a value of 308.162:
     when round_num = 1       result is 300
     when round_num = 2       result is 310
     when round_num = 3       result is 308
     when round_num =  0      result is 308.162
     when round_num = -1      result is 308.16
     when round_num = -2      result is 308.2
     when round_num = NULL    result is NULL
```

Examples

The simplest invocation of this routine populates bin definitions in the num_bin_table for all the numerical attributes found in build_table.

```
BEGIN
   DBMS_DATA_MINING_TRANSFORM.INSERT_BIN_NUM(
      'num_bin_table', 'build_table');
END;
/
```

INSERT_BIN_NUM_QTILE Procedure

This procedure finds a numerical binning definition for every NUMBER column in the data table that is not specified in the exclusion list and inserts the definition into the binning definition table that was created using CREATE_BIN_NUM.

The definition for each relevant column is computed based on the minimum values for each quantile, where quantiles are computed from the data using NTILE function. Bins bin_1,..., bin_N span the following ranges: bin_1 spans [min_1,min_2]; bin_2,..., bin_i,..., bin_N-1 span (min_i, min_(i+1)] and bin_N spans (min_N, max_N]. Bins with equal left and right boundaries are collapsed.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_BIN_NUM_QTILE (
    bin_table_name       IN VARCHAR2,
    data_table_name      IN VARCHAR2,
    bin_num              IN PLS_INTEGER DEFAULT 10,
    exclude_list         IN Column_List DEFAULT NULL,
    bin_schema_name      IN VARCHAR2 DEFAULT NULL,
    data_schema_name     IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 26–12 INSERT_BIN_NUM_QTILE Procedure Parameters

Parameter	Description
bin_table_name	Name of the numerical binning definition table generated using the CREATE_BIN_NUM procedure
data_table_name	Name of the table containing the data
bin_num	Number of bins
exclude_list	List of columns (attributes) to be excluded from this iteration of the binning process
bin_schema_name	Name of the schema hosting the numerical binning definition table
data_schema_name	Name of the schema hosting the table with data

Usage Notes

For a given input table, you can iteratively call this routine several times with different specifications for bin_num for a given input table. For each iteration, you can selectively exclude attributes (that is, column names) using the exclude_list parameter for a particular specification. Columns with all NULL values are ignored.

Example 1. When N = 4, col='mycol', and data is {1,2,2,2,2,3,4}, the following three rows are inserted into the definition table:

```
COL     VAL   BIN
----- -----  -----
mycol     1   NULL
mycol     2   1
mycol     4   2
```

Here quantities are {1,2}, {2,2}, {2,3}, {4} and min(1) = 1, min(2) = 2, min(3) = 2, min(4) = 4, max(4) = 4, and ranges are [1,2], (2,2], (2,4], (4,4]. After collapsing [1,2] and (2,4].

Examples

The simplest invocation of this routine populates numerical binning definitions in the `num_bin_table` for all the numerical attributes found in `build_table`.

```
BEGIN
    DBMS_DATA_MINING_TRANSFORM.INSERT_BIN_NUM_QTILE(
       'num_bin_table', 'build_table');
END;
```

INSERT_CLIP_TRIM_TAIL Procedure

This procedure finds the trimming definition for every NUMBER column in the data table that is not specified in the exclusion list and inserts the definition into the clipping definition table that was created using CREATE_CLIP.

The definition for each relevant column is computed based on the non-NULL values sorted in ascending order such that val(1) < val(2) <... < val(N), where N is a total number of non-NULL values in a column:

```
lcut = val(1+floor(N*q))
lval = NULL
rcut = val(N-floor(*N*q))
rval = NULL
```

where q = ABS(NVL(tail_frac,0)). Nothing is done when q >= 0.5.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_CLIP_TRIM_TAIL (
    clip_table_name     IN VARCHAR2,
    data_table_name     IN VARCHAR2,
    tail_frac           IN NUMBER DEFAULT 0.025,
    exclude_list        IN Column_List DEFAULT NULL,
    clip_schema_name    IN VARCHAR2 DEFAULT NULL,
    data_schema_name    IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 26–13 INSERT_CLIP_TRIM_TAIL Procedure Parameters

Parameter	Description
clip_table_name	Name of the clipping definition table generated using the CREATE_CLIP procedure
data_table_name	Name of the table containing the data
tail_frac	Tail fraction
exclude_list	List of columns (attributes) to be excluded from this iteration of the clipping process
clip_schema_name	Name of the schema hosting the clipping definition table
data_schema_name	Name of the schema hosting the table with data

Usage Notes

For a given input table, you can iteratively call this routine several times with different specifications for tail_frac for a given input table. For each iteration, you can selectively exclude attributes (that is, column names) using the exclude_list parameter for a particular specification.

Example 1. When q = 0.2, col='mycol', and data is {1,2,2,2,3,4,4}, the following row is inserted into the definition table:

```
COL    LCUT   LVAL   RCUT   RVAL
----- -----  -----  -----  -----
mycol     2    NULL     4   NULL
```

Here 1 + floor(N*q) = 1 + floor(7*0.2) = 2, lcut = val(2) = 2.

N – floor(N*q) = 7 – floor(7*0.2) = 6, rcut = val(6) = 4.

Examples

The simplest invocation of this routine populates clipping definitions in the `clip_table` for all the numerical attributes found in `build_table`.

```
BEGIN
    DBMS_DATA_MINING_TRANSFORM.INSERT_CLIP_TRIM_TAIL(
       'clip_table', 'build_table');
END;
```

INSERT_CLIP_WINSOR_TAIL Procedure

This procedure finds the Winsorizing definition for every NUMBER column in the data table that is not specified in the exclusion list and inserts the definition into the clipping definition table that was created using CREATE_CLIP.

Definition for each relevant column is computed based on the non-NULL values sorted in ascending order such that val(1) < val(2) <... < val(N), where N is a total number of non-NULL values in a column:

```
lcut = val(1+floor(N*q))
lval = lcut
rcut = val(N-floor(N*q))
rval = rcut
```

where q = ABS(NVL(tail_fraq,0)). Nothing is done when q >= 0.5.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_CLIP_WINSOR_TAIL (
    clip_table_name    IN VARCHAR2,
    data_table_name    IN VARCHAR2,
    tail_frac          IN NUMBER DEFAULT 0.025,
    exclude_list       IN Column_List DEFAULT NULL,
    clip_schema_name   IN VARCHAR2 DEFAULT NULL,
    data_schema_name   IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 26–14 INSERT_CLIP_WINSOR_TAIL Procedure Parameters

Parameter	Description
clip_table_name	Name of the clipping definition table generated using CREATE_CLIP procedure
data_table_name	Name of the table containing the data
tail_frac	Tail fraction
exclude_list	List of columns (attributes) to be excluded from this iteration of the clipping process
clip_schema_name	Name of the schema hosting the clipping definition table
data_schema_name	Name of the schema hosting the table with data

Usage Notes

For a given input table, you can iteratively call this routine several times with different specifications for tail_frac for a given input table. For each iteration, you can selectively exclude attribute (that is, column names using the exclude_list parameter for a particular specification. Columns with all NULL values are ignored.

Example 1. When q = 0.2, col='mycol', and data is {1,2,2,2,3,4,4}, the following row is inserted into the definition table:

```
COL    LCUT   LVAL   RCUT  RVAL
----- -----  -----  ----- -----
mycol     2      2      4      4
```

Here 1 + floor(N*q) = 1 + floor(7*0.2) = 2, lcut = val(2) = 2.

N – floor(N*q) = 7 – floor(7*0.2) = 6, rcut = val(6) = 4.

Examples

The simplest invocation of this routine populates clipping definitions in the `clip_table` for all the numerical attributes found in `build_table`.

```
BEGIN
    DBMS_DATA_MINING_TRANSFORM.INSERT_CLIP_WINSOR_TAIL(
       'clip_table', 'build_table');
END;
```

INSERT_MISS_CAT_MODE Procedure

This procedure finds the categorical missing value treatment definition for every VARCHAR2 and CHAR column in the data table that is not specified in the exclusion list and inserts the definition into the definition table that was created using CREATE_MISS_CAT.

The definition for each selected column is computed based on the mode value that is computed from the data table.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_MISS_CAT_MODE (
    miss_table_name    IN VARCHAR2,
    data_table_name    IN VARCHAR2,
    exclude_list       IN COLUMN_LIST DEFAULT NULL,
    miss_schema_name   IN VARCHAR2 DEFAULT NULL,
    data_schema_name   IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 26–15 INSERT_MISS_CAT_MODE Procedure Parameters

Parameter	Description
miss_table_name	Name of the categorical missing value treatment definition table generated using CREATE_MISS_CAT
data_table_name	Name of the table containing the data
exclude_list	List of columns (attributes) to be excluded from this iteration of the missing value treatment. See Table 26–1, " Summary of Data Types" for the definition of COLUMN_LIST.
miss_schema_name	Name of the schema hosting the categorical missing value treatment definition table
data_schema_name	Name of the schema hosting the table containing the data

Usage Notes

You can choose the categorical attributes that will receive missing value treatment by using the exclude_list parameter. NULL values in all the selected attributes will be replaced with the mode (the most commonly occurring value) for the attribute.

If you wish to replace NULLs with some other value, you can edit the definition table.

Example

The simplest invocation of this routine populates missing value definitions (the mode) in miss_table for all categorical attributes found in build_table.

```
BEGIN
    DBMS_DATA_MINING_TRANSFORM.INSERT_MISS_CAT_MODE(
       'miss_table', 'build_table');
END;
```

INSERT_MISS_NUM_MEAN Procedure

This procedure finds the numerical missing value treatment definition for every NUMBER column in the data table that is not specified in the exclusion list and inserts the definition into the definition table that was created using CREATE_MISS_NUM.

The definition for each selected column is computed based on the mean value that is computed from the data table. The value of mean is rounded to round_num significant digits prior to storing it in the definition table.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_MISS_NUM_MEAN (
    miss_table_name    IN VARCHAR2,
    data_table_name    IN VARCHAR2,
    exclude_list       IN COLUMN_LIST DEFAULT NULL,
    round_num          IN PLS_INTEGER DEFAULT 6,
    miss_schema_name   IN VARCHAR2 DEFAULT NULL,
    data_schema_name   IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 26–16 INSERT_MISS_NUM_MEAN Procedure Parameters

Parameter	Description
miss_table_name	Name of the categorical missing value treatment definition table generated using CREATE_MISS_CAT
data_table_name	Name of the table containing the data
exclude_list	List of columns (attributes) to be excluded from this iteration of the miss value treatment. See Table 26–1, " Summary of Data Types" for the definition of COLUMN_LIST.
round_num	The number of significant digits
miss_schema_name	Name of the schema hosting the numerical missing value treatment definition table
data_schema_name	Name of the schema hosting the table containing the data

Usage Notes

You can choose the numerical attributes that will receive missing value treatment by using the exclude_list parameter. NULL values in all the selected attributes will be replaced with the mean (average) value for the attribute.

If you wish to replace NULLs with some other value, you can edit the definition table.

Example

The simplest invocation of this routine populates missing value definitions (the mode) in miss_table for all numerical attributes found in build_table.

```
BEGIN
    DBMS_DATA_MINING_TRANSFORM.INSERT_MISS_CAT_MODE(
       'miss_table', 'build_table');
END;
```

INSERT_NORM_LIN_MINMAX Procedure

This procedure finds the normalization definition for every NUMBER column in the data table that is not specified in the exclusion list and inserts the definition based on min-max normalization into the table that was created using CREATE_NORM_LIN.

Definition for each relevant column is computed based on the mean and standard deviation that are computed from the data table, such that shift = mean and scale = standard deviation. The values of shift and scale are rounded to round_num significant digits prior to storing them in the definition table.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_NORM_LIN_MINMAX (
     norm_table_name     IN VARCHAR2,
     data_table_name     IN VARCHAR2,
     exclude_list        IN Column_List DEFAULT NULL,
     round_num           IN PLS_INTEGER DEFAULT 6,
     norm_schema_name    IN VARCHAR2 DEFAULT NULL,
     data_schema_name    IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 26–17 INSERT_NORM_LIN_MINMAX Procedure Parameters

Parameter	Description
norm_table_name	Name of the normalization table generated using CREATE_NORM_LIN procedure
data_table_name	Name of the table containing the data
exclude_list	List of columns (attributes) to be excluded from this iteration of the normalization process
round_num	Number of significant digits
norm_schema_name	Name of the schema hosting the normalization definition table
data_schema_name	Name of the schema hosting the table with data

Usage Notes

For a given input table, you can iteratively call this routine several times with selective exclusion of attributes (that is, column names) using the exclude_list parameter for a particular normalization specification.

Columns with all NULL values or only one unique value are ignored.

Examples

The simplest invocation of this routine populates normalization definitions in the norm_minmax_table for all the numerical attributes found in build_table.

```
BEGIN
   DBMS_DATA_MINING_TRANSFORM.INSERT_NORM_LIN_MINMAX(
      'norm_minmax_table', 'build_table');
END;
```

INSERT_NORM_LIN_SCALE Procedure

This procedure finds the normalization definition for every NUMBER column in the data table that is not specified in the exclusion list and inserts the definition based on min-max normalization into the table that was created using CREATE_NORM_LIN.

The normalization definition for each attribute is computed based on the minimum and maximum values of the data. The values for shift and scale are shift = 0 and scale = max{abs(max), abs(min)}.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_NORM_LIN_SCALE (
     norm_table_name     IN VARCHAR2,
     data_table_name     IN VARCHAR2,
     exclude_list        IN Column_List DEFAULT NULL,
     round_num           IN PLS_INTEGER DEFAULT 6,
     norm_schema_name    IN VARCHAR2 DEFAULT NULL,
     data_schema_name    IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 26–18 INSERT_NORM_LIN_SCALE Procedure Parameters

Parameter	Description
norm_table_name	Name of the normalization table generated using CREATE_NORM_LIN procedure
data_table_name	Name of the table containing the data
exclude_list	List of columns (attributes) to be excluded from this iteration of the normalization process
round_num	Number of significant digits
norm_schema_name	Name of the schema hosting the normalization definition table
data_schema_name	Name of the schema hosting the table with data

Usage Notes

For a given input table, you can iteratively call this routine several times with selective exclusion of attributes (that is, column names) using the exclude_list parameter for a particular normalization specification.

Columns with all NULL values or only one unique value are ignored.

Examples

The simplest invocation of this routine populates normalization definitions in the norm_minmax_table for all the numerical attributes found in build_table.

```
BEGIN
   DBMS_DATA_MINING_TRANSFORM.INSERT_NORM_LIN_SCALE(
      'norm_scale_table', 'build_table');
END;
```

INSERT_NORM_LIN_ZSCORE Procedure

This procedure finds the normalization definition for every NUMBER column in the data table that is not specified in the exclusion list and inserts the definition based on z-score normalization into the table that was created using CREATE_NORM_LIN.

Definition for each relevant column is computed based on the minimum and maximum values that are computed from the data table, such that shift = min and scale = max – min. The values of shift and scale are rounded to round_num significant digits prior to storing them in the definition table.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_NORM_LIN_ZSCORE (
     norm_table_name     IN VARCHAR2,
     data_table_name     IN VARCHAR2,
     exclude_list        IN Column_List DEFAULT NULL,
     round_num           IN PLS_INTEGER DEFAULT 6,
     norm_schema_name    IN VARCHAR2 DEFAULT NULL,
     data_schema_name    IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 26–19 INSERT_BIN_NORM_LIN_ZSCORE Procedure Parameters

Parameter	Description
norm_table_name	Name of the normalization table generated using CREATE_NORM_LIN procedure
data_table_name	Name of the table containing the data
exclude_list	List of columns (attributes) to be excluded from this iteration of the normalization process
round_num	Number of significant digits
norm_schema_name	Name of the schema hosting the normalization definition table
data_schema_name	Name of the schema hosting the table with data

Usage Notes

For a given input table, you can iteratively call this routine several times with selective exclusion of attributes (that is, column names) using the exclude_list parameter for a particular binning specification.

Columns with all NULL values or only one unique value are ignored.

Examples

The simplest invocation of this routine populates normalization definitions in the norm_zscore_table for all the numerical attributes found in build_table.

```
BEGIN
   DBMS_DATA_MINING_TRANSFORM.INSERT_NORM_LIN_ZSCORE(
      'norm_zscore_table', 'build_table');
END;
/
```

XFORM_BIN_CAT Procedure

This procedure creates the view that performs categorical binning. Only the columns that are specified in the definition table are transformed; the remaining columns do not change.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.XFORM_BIN_CAT (
     bin_table_name       IN VARCHAR2,
     data_table_name      IN VARCHAR2,
     xform_view_name      IN VARCHAR2,
     literal_flag         IN BOOLEAN DEFAULT FALSE,
     bin_schema_name      IN VARCHAR2 DEFAULT NULL,
     data_schema_name     IN VARCHAR2 DEFAULT NULL,
     xform_schema_name    IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 26–20 XFORM_BIN_CAT Procedure Parameters

Parameter	Description
bin_table_name	Name of the categorized binning definition table generated using CREATE_BIN_CAT procedure
data_table_name	Name of the table containing the data
xform_view_name	View representing the transformed output
literal_flag	Literal flag
bin_schema_name	Name of the schema hosting the bin definition table
data_schema_name	Name of the schema hosting the data table
xform_schema_name	Name of the schema hosting the view representing the transformed output

Usage Notes

The bin table created by CREATE_BIN_CAT and populated with bin definitions by INSERT_BIN_CAT_FREQ is used to guide the query generation process to construct categorical binning expressions of the following form:

```
DECODE("col", val_1, bin_1,
              ...
              val_N, bin_N,
              NULL,   NULL,
              bin_(N+1)) "col"
```

This expression maps values val_1,..., val_N into N bins bin_1,..., bin_N, and other values into bin_(N+1), while NULL values remain unchanged. bin_(N+1) is optional. If not specified, it defaults to NULL. To specify bin_(N+1) provide a row with val set to NULL.

The literal_flag parameter indicates whether the values in bin are valid SQL literals. When the flag is set to TRUE, the value of bin is used as is in query generation; otherwise it is converted into a valid text literal (surrounded by quotes and each single quote is replaced by two single quotes). By default, the flag is set to FALSE. One example of when it can be set to TRUE is in cases when all bin are numbers. In that case the transformed column will be numeric as opposed to textual (default behavior).

Set `literal_flag` to TRUE when the data is binned for an O-Cluster model build.

The `col` parameter is case-sensitive since it generates quoted identifiers. In cases when there are multiple entries with the same `col`,`val` combination with different `bin`, the behavior is undefined — any one of the `bin` values might be used.

Examples

Example 1. bin_cat contains four rows with col = 'mycol':

```
{col = 'mycol', val = 'Waltham',        bin = 'MA'}
{col = 'mycol', val = 'Burlington',     bin = 'MA'}
{col = 'mycol', val = 'Redwood Shores', bin = 'CA'}
{col = 'mycol', val = NULL,             bin = 'OTHER'}
```

the following expression is generated:

```
 DECODE("mycol", 'Waltham',        'MA',
                 'Burlington',     'MA',
                 'Redwood Shores', 'CA',
                  NULL,             NULL,
                                   'OTHER') "mycol"
```

Example 2. bin_cat contains three rows with col = 'mycol':

```
{col = 'mycol', val = 'Waltham',        bin = 'MA'}
{col = 'mycol', val = 'Burlington',     bin = 'MA'}
{col = 'mycol', val = 'Redwood Shores', bin = 'CA'}
```

the following expression is generated:

```
DECODE("mycol", 'Waltham',        'MA',
                'Burlington',     'MA',
                'Redwood Shores', 'CA') "mycol"
```

Example 3. For the definition:

```
COL   VAL            BIN
----- ----------     ---
mycol Waltham        1
mycol Burlington     1
mycol Redwood Shores 2
```

the following expression is generated when the literal flag is set to FALSE:

```
DECODE ("mycol", 'Waltham',        '1',
                 'Burlington'      '1',
                 'Redwood Shores', '2') "mycol"
```

and when the flag is set to TRUE:

```
DECODE("mycol", 'Waltham',        1,
                'Burlington',     1,
                'Redwood Shores', 2) "mycol"
```

The simplest invocation of this routine generates a view build_view that represents the transformation query on build_table based on bin definitions in the cat_bin_table.

```
BEGIN
DBMS_DATA_MINING_TRANSFORM.XFORM_BIN_CAT(
'cat_bin_table', 'build_table', 'build_view');
END;
/
```

XFORM_BIN_NUM Procedure

This procedure creates the view that performs numerical binning. Only the columns that are specified in the definition table are transformed; the remaining columns do not change.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.XFORM_BIN_NUM (
     bin_table_name     IN VARCHAR2,
     data_table_name    IN VARCHAR2,
     xform_view_name    IN VARCHAR2,
     literal_flag       IN BOOLEAN DEFAULT FALSE,
     bin_schema_name    IN VARCHAR2 DEFAULT NULL,
     data_schema_name   IN VARCHAR2 DEFAULT NULL,
     xform_schema_name  IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 26–21 XFORM_BIN_NUM Procedure Parameters

Parameter	Description
bin_table_name	Name of the numerical binning definition table generated using CREATE_BIN_NUM procedure
data_table_name	Name of the table containing the data
xform_view_name	View representing the transformed output
literal_flag	Literal flag
bin_schema_name	Name of the schema hosting the bin definition table
data_schema_name	Name of the schema hosting the data table
xform_schema_name	Name of the schema hosting the view representing the transformed output

Usage Notes

The bin table created by CREATE_BIN_NUM and populated with bin definitions by INSERT_BIN_NUM_EQWIDTH or INSERT_BIN_NUM_QTILE is used to guide the query generation process to construct numerical binning expressions of the following form:

```
CASE WHEN "col" <  val_0   THEN 'bin0_0
     WHEN "col" <= val_1   THEN 'bin_1'
     ...
     WHEN "col" <= val_N   THEN 'bin_N'
     WHEN "col" IS NOT NULL THEN 'bin_(N+1)'
END "col"
```

This expression maps values in the range [val_0;val_N] into N bins bin_1,..., bin_N, values outside of this range into bin_0 or bin_(N+1), such that

```
(-inf; val_0)  -> bin_0
[val_0; val_1)  -> bin_1
...
(val_(N-1); val_N] -> bin_N
(val_N; +inf)  -> bin_(N+1)
```

NULL values remain unchanged. bin_(N+1) is optional. If it is not specified, the values ("col" > val_N) are mapped to NULL. To specify bin_(N+1), provide a row with val set to NULL. The order of the WHEN... THEN pairs is based on the ascending order of val for a given col.

The literal_flag parameter indicates whether the values in bin are valid SQL literals. When the flag is set to TRUE, the value of bin is used as is in query generation; otherwise it is converted into a valid text literal (surrounded by quotes and each single quote is replaced by two single quotes). By default, the flag is set to FALSE. One example of when it can be set to TRUE is in cases when all bin are numbers. In that case the transformed column will be numeric as opposed to textual (default behavior).

Note that col is case-sensitive since it generates quoted identifiers. In cases where there are multiple entries with the same col,val combination with different bin, the behavior is undefined — any one of the bin values might be used.

Examples

Example 1. bin_num contains four rows with col = 'mycol':

```
{col = 'mycol', val = 15.5, bin = 'small'}
{col = 'mycol', val = 10,   bin = 'tiny'}
{col = 'mycol', val = 20,   bin = 'large'}
{col = 'mycol', val = NULL, bin = 'huge'}
```

the following expression is generated:

```
CASE WHEN "mycol" <  10       THEN 'tiny'
     WHEN "mycol" <= 15.5     THEN 'small'
     WHEN "mycol" <= 20       THEN 'large'
     WHEN "mycol" IS NOT NULL THEN 'huge'
 END "mycol"
```

Example 2. bin_num contains three rows with col = 'mycol':

```
{col = 'mycol', val = 15.5, bin = NULL}
{col = 'mycol', val = 10,   bin = 'tiny'}
{col = 'mycol', val = 20,   bin = 'large'}
```

the following expression is generated:

```
CASE WHEN "mycol" <  10   THEN NULL
     WHEN "mycol" <= 15.5 THEN 'small'
     WHEN "mycol" <= 20   THEN 'large'
END "mycol"
```

Example 3. For the definition:

```
COL   VAL  BIN
----- ---- ---
mycol   10 NULL
mycol 15.5 1
mycol   21 2
```

the following expression is generated when the literal flag is set to FALSE:

```
CASE WHEN "mycol" <  10   THEN NULL
     WHEN "mycol" <= 15.5 THEN '1'
     WHEN "mycol" <= 20   THEN '2'
 END "mycol"
```

and when the flag is set to TRUE:

```
CASE WHEN "mycol" <   10  THEN NULL
     WHEN "mycol" <= 15.5 THEN 1
     WHEN "mycol" <=  20  THEN 2
     END "mycol"
```

The simplest invocation of this routine generates a view `build_view` that represents the transformation query on `build_table` based on transform definitions in bin definitions in the `num_bin_table`.

```
BEGIN
   DBMS_DATA_MINING_TRANSFORM.XFORM_BIN_NUM(
      'num_bin_table', 'build_table', 'build_view');
END;
/
```

XFORM_CLIP Procedure

This procedure creates the view that performs clipping. Only the columns that are specified in the transform definition are clipped; the remaining columns do not change.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.XFORM_CLIP (
    clip_table_name       IN VARCHAR2,
    data_table_name       IN VARCHAR2,
    xform_view_name       IN VARCHAR2,
    clip_schema_name      IN VARCHAR2 DEFAULT NULL,
    data_schema_name      IN VARCHAR2,DEFAULT NULL,
    xform_schema_name     IN VARCHAR2,DEFAULT NULL;
```

Parameters

Table 26–22 XFORM_CLIP Procedure Parameters

Parameter	Description
clip_table_name	Name of the clipping definition table generated using CREATE_CLIP
data_table_name	Name of the table containing the data
xform_view_name	View representing the transformed output
clip_schema_name	Name of the schema hosting the clipping definition table
data_schema_name	Name of the schema hosting the data table
xform_schema_name	Name of the schema hosting the view representing the transformed output

Usage Notes

The clipping definition table created by CREATE_CLIP and populated with clipping definitions by INSERT_CLIP_WINSOR_TAIL or INSERT_CLIP_TRIM_TAIL is used to guide query generation process to construct clipping expressions of the following form:

```
CASE WHEN "col" < lcut THEN lval
     WHEN "col" > rcut THEN rval
                       ELSE "col"
END "col"
```

Note that col is case-sensitive since it generates quoted identifiers. When there are multiple entries in the transform definition table for the same col, the behavior is undefined. Any one of the definitions may be used in query generation. NULL values remain unchanged.

Example 1 (Winsorizing). When col = 'my_col', lcut = –1.5, lval = –1.5, and rcut = 4.5 and rval = 4.5, the following expression is generated:

```
CASE WHEN "my_col" < -1.5 THEN -1.5
     WHEN "my_col" >  4.5 THEN 4.5
                          ELSE "my_col"
END "my_col"
```

Examples

The simplest invocation of this routine generates a view object `build_view` that represents the transformation query on `build_table` based on transform definitions in clipping definitions in the `clip_table`.

```
BEGIN
    DBMS_DATA_MINING_TRANSFORM.XFORM_CLIP(
    'clip_table', 'build_table', 'build_view');
END;
```

XFORM_MISS_CAT Procedure

This procedure creates a view that performs categorical missing value treatment. Only the columns that are specified in the xform definition are treated; the remaining columns do not change.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.XFORM_MISS_CAT (
    miss_table_name       IN VARCHAR2,
    data_table_name       IN VARCHAR2,
    xform_view_name       IN VARCHAR2,
    miss_schema_name      IN VARCHAR2 DEFAULT NULL,
    data_schema_name      IN VARCHAR2 DEFAULT NULL,
    xform_schema_name     IN VARCHAR2 DEFAULT NULL;
```

Parameters

Table 26–23 XFORM_MISS_CAT Procedure Parameters

Parameter	Description
miss_table_name	Name of the categorical missing value treatment definition table generated using CREATE_MISS_CAT
data_table_name	Name of the table containing the data
xform_view_name	View representing the transformed output
miss_schema_name	Name of the schema hosting the categorical missing value treatment definition table
data_schema_name	Name of the schema hosting the data table
xform_schema_name	Name of the schema hosting the view representing the transformed output

Usage Notes

The data type of the transformed columns is preserved by putting a CAST expression around the NVL function. For example, when col = 'state', val = 'MA' the data type is CHAR(2) the following expression is generated:

```
CAST (NVL("state", 'MA') AS CHAR(2)) "state"
```

Examples

The simplest invocation of this routine generates a view object build_view that represents the transformation query on build_table based on transform definitions in missing value definitions in miss_table.

```
BEGIN
    DBMS_DATA_MINING_TRANSFORM.XFORM_MISS_CAT(
       'miss_table', 'build_table', 'build_view');
END;
```

XFORM_MISS_NUM Procedure

This procedure creates a view that performs numerical missing value treatment of the data table. Only the columns that are specified in the xform definition are treated, the remaining columns do not change.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.XFORM_MISS_NUM (
    miss_table_name       IN VARCHAR2,
    data_table_name       IN VARCHAR2,
    xform_view_name       IN VARCHAR2,
    miss_schema_name      IN VARCHAR2 DEFAULT NULL,
    data_schema_name      IN VARCHAR2 DEFAULT NULL,
    xform_schema_name     IN VARCHAR2 DEFAULT NULL;
```

Parameters

Table 26–24 XFORM_MISS_NUM Procedure Parameters

Parameter	Description
miss_table_name	Name of the numeric missing value treatment definition table generated using CREATE_MISS_NUM
data_table_name	Name of the table containing the data
xform_view_name	View representing the transformed output
miss_schema_name	Name of the schema hosting the numerical missing value treatment definition table
data_schema_name	Name of the schema hosting the data table
xform_schema_name	Name of the schema hosting the view representing the transformed output

Examples

The simplest invocation of this routine generates a view object build_view that represents the transformation query on build_table based on transform definitions in missing value definitions in miss_table.

```
BEGIN
    DBMS_DATA_MINING_TRANSFORM.XFORM_MISS_NUM(
    'miss_table', 'build_table', 'build_view');
END;
```

XFORM_NORM_LIN Procedure

This procedure creates the view that performs linear normalization. Only the columns that are specified in the definition table are transformed; the remaining columns do not change.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.XFORM_NORM_LIN (
     norm_table_name      IN VARCHAR2,
     data_table_name      IN VARCHAR2,
     xform_view_name      IN VARCHAR2,
     norm_schema_name     IN VARCHAR2 DEFAULT NULL,
     data_schema_name     IN VARCHAR2 DEFAULT NULL,
     xform_schema_name    IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 26–25 XFORM_NORM_LIN Procedure Parameters

Parameter	Description
norm_table_name	Name of the normalization definition table generated using CREATE_NORM_LIN procedure
data_table_name	Name of the table containing the data
xform_view_name	View representing the transformed output
norm_schema_name	Name of the schema hosting the normalization definition table
data_schema_name	Name of the schema hosting the data table
xform_schema_name	Name of the schema hosting the view representing the transformed output

Usage Notes

The normalization table created by CREATE_NORM_LIN is populated with definitions by either INSERT_NORM_LIN_ZSCORE or INSERT_NORM_LIN_MINMAX is used to guide the query generation process to construct normalization expressions of the following form:

```
("col" - shift)/scale "col"
```

Note that col is case-sensitive since it generates quoted identifiers. When there are multiple entries in the transform definition table for the same col, the behavior is undefined. Any one of the definitions may be used in query generation. NULL values remain unchanged.

For example, when col = 'my_col', shift = -1.5, and scale = 20. The following expression is generated:

```
("my_col" - (-1.5))/20 "my_col"
```

Examples

The simplest invocation of this routine generates a view build_view that represents the transformation query on build_table based on normalization definitions in the norm_minmax_table.

```
BEGIN
   DBMS_DATA_MINING_TRANSFORM.XFORM_NORM_LIN(
      'norm_minmax_table', 'build_table', 'build_view');
END;
```

27

DBMS_DATAPUMP

The DBMS_DATAPUMP package is used to move all, or part of, a database between databases, including both data and metadata.

> **See Also:** *Oracle Database Utilities* for more information on the concepts behind the DBMS_DATAPUMP API, how it works, and how it is implemented in the Data Pump Export and Import utilities

This chapter contains the following topics:

Using DBMS_DATAPUMP

This section contains topics that relate to using the `DBMS_DATAPUMP` package.

Overview

The support and functionality provided by `DBMS_DATAPUMP` is as follows:

- The source and target databases can have different hardware, operating systems, character sets, and time zones.
- All object types and datatypes existing in Oracle Database 10*g* are supported.
- Data and metadata can be transferred between databases without using any intermediary files.
- A subset of a database can be moved based upon object type and names of objects.
- Schema names, datafile names, and tablespace names can be transformed at import time.
- Previously aborted export and import jobs can be restarted without duplicating or omitting any data or metadata from the original job.
- The resources applied to an export or import job can be modified.
- Data in an Oracle proprietary format can be unloaded and loaded.

Security Model

Security for the DBMS_DATAPUMP package is implemented through roles.

Roles

The existing EXP_FULL_DATABASE and IMP_FULL_DATABASE roles will be used to allow privileged users to take full advantage of the API. The Data Pump API will use these roles to determine whether privileged application roles should be assigned to the processes comprising the job.

EXP_FULL_DATABASE

The EXP_FULL_DATABASE role affects only Export operations. It allows users running these operations to do the following:

- Perform the operation outside of the scope of their schema
- Monitor jobs that were initiated by another user
- Export objects (for example, TABLESPACE definitions) that unprivileged users cannot reference

Although the SYS schema does not have the EXP_FULL_DATABASE role assigned to it, all security checks performed by Data Pump that require the EXP_FULL_DATABASE role will also grant access to the SYS schema.

IMP_FULL_DATABASE

The IMP_FULL_DATABASE role affects only Import and SQL_FILE operations. It allows users running these operations to do the following:

- Perform the operation outside of the scope of their schema
- Monitor jobs that were initiated by another user
- Import objects (for example, DIRECTORY definitions) that unprivileged users cannot create

Although the SYS schema does not have the IMP_FULL_DATABASE role assigned to it, all security checks performed by Data Pump that require the IMP_FULL_DATABASE role will also grant access to the SYS schema.

Constants

There are several public constants defined for use with the DBMS_DATAPUMP.GET_STATUS procedure. All such constants are defined as part of the DBMS_DATAPUMP package. Any references to these constants must be prefixed by DBMS_DATAPUMP. and followed by the symbols in the following lists:

Mask Bit Definitions

The following mask bit definitions are used for controlling the return of data through the DBMS_DATAPUMP.GET_STATUS procedure.

- KU$_STATUS_WIP CONSTANT BINARY_INTEGER := 1;
- KU$_STATUS_JOB_DESC CONSTANT BINARY_INTEGER := 2;
- KU$_STATUS_JOB_STATUS CONSTANT BINARY_INTEGER := 4;
- KU$_STATUS_JOB_ERROR CONSTANT BINARY_INTEGER := 8;

Dump File Type Definitions

The following definitions are used for identifying types of dump files returned through the DBMS_DATAPUMP.GET_STATUS procedure.

- KU$_DUMPFILE_TYPE_DISK CONSTANT BINARY_INTEGER := 0;
- KU$_DUMPFILE_TYPE_TEMPLATE CONSTANT BINARY_INTEGER := 3;

Data Structures

The DBMS_DATAPUMP package defines OBJECT types. The types described in this section are defined in the SYS schema for use by the GET_STATUS function. *The way in which these types are defined and used may be different than what you are accustomed to. Be sure to read this section carefully.*

The collection of types defined for use with the GET_STATUS procedure are version-specific and include version information in the names of the types. Once introduced, these types will always be provided and supported in future versions of Oracle Database and will not change. However, in future releases of Oracle Database, new versions of these types might be created that provide new or different information. The new versions of these types will have different version information embedded in the type names.

For example, in Oracle Database 10*g*, release 1 (10.1), there is a sys.ku$_Status1010 type, and in the next Oracle Database release, there could be a sys.ku$_Status1110 type defined. Both types could be used with the GET_STATUS procedure.

Public synonyms have been defined for each of the types used with the GET_STATUS procedure. This makes it easier to use the types and means that you do not have to be concerned with changes to the actual type names or schemas where they reside. Oracle recommends that you use these synonyms whenever possible.

For each of the types, there is a version-specific synonym and a generic synonym. For example, the version-specific synonym ku$_Status1010 is defined for the sys.ku$_Status1010 type.

The generic synonym always describes the latest version of that type. For example, in Oracle Database 10*g*, release 1, the generic synonym ku$_Status is defined as ku$_Status1010. In a future release, there might be a ku$_Status1110 synonym for sys.ku$Status1110. Because the ku$_Status generic synonym always points to the latest definition, it would now point to ku$_Status1110 rather than to ku$_Status1010.

The choice of whether to use version-specific synonyms or generic synonyms makes a significant difference in how you work. Using version-specific names protects your code from changes in future releases of Oracle Database because those types will continue to exist and be supported. However, access to new information will require code changes to use new synonym names for each of the types. Using the generic names implies that you always want the latest definition of the types and are prepared to deal with changes in different releases of Oracle Database.

When the version of Oracle Database that you are using changes, any C code that accesses types through generic synonym names will need to be recompiled.

Note: Languages other than PL/SQL must ensure that their type definitions are properly aligned with the version-specific definitions.

See Also: GET_STATUS Procedure on page 27-23 for additional information about how types are used

Data Structures - Object Types

The DBMS_DATAPUMP package defines the following kinds of OBJECT types:

- Worker Status Types
- Log Entry and Error Types
- Job Status Types
- Job Description Types
- Status Types

Worker Status Types

The worker status types describe what each worker process in a job is doing. The schema, object name, and object type of an object being processed will be provided. For workers processing user data, the partition name for a partitioned table (if any), the number of bytes processed in the partition, and the number of rows processed in the partition are also returned. Workers processing metadata provide status on the last object that was processed. No status for idle threads is returned.

The percent_done refers to the amount completed for the current data item being processed. It is not updated for metadata objects.

The worker status types are defined as follows:

```
CREATE TYPE sys.ku$_WorkerStatus1010 AS OBJECT (
               worker_number      NUMBER,
               process_name       VARCHAR2(30),
               state              VARCHAR2(30),
               schema             VARCHAR2(30),
               name               VARCHAR2(4000),
               object_type        VARCHAR2(200),
               partition          VARCHAR2(30),
               completed_objects  NUMBER,
               total_objects      NUMBER,
               completed_rows     NUMBER,
               completed_bytes    NUMBER,
               percent_done       NUMBER)

CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatus1010
  FOR sys.ku$_WorkerStatus1010;

CREATE TYPE sys.ku$_WorkerStatus1020 AS OBJECT (
               worker_number     NUMBER,        --  Worker process identifier
               process_name      VARCHAR2(30),  --  Worker process name
               state             VARCHAR2(30),  --  Worker process state
               schema            VARCHAR2(30),  --  Schema name
               name              VARCHAR2(4000),--  Object name
               object_type       VARCHAR2(200), --  Object type
               partition         VARCHAR2(30),  --  Partition name
               completed_objects NUMBER,        --  Completed number of objects
               total_objects     NUMBER,        --  Total number of objects
               completed_rows    NUMBER,        --  Number of rows completed
               completed_bytes   NUMBER,        --  Number of bytes completed
               percent_done      NUMBER,        --  Percent done current object
               degree            NUMBER         --  Degree of parallelism)

CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatus1020
  FOR sys.ku$_WorkerStatus1020;
```

```
CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatus FOR ku$_WorkerStatus1020;

CREATE TYPE sys.ku$_WorkerStatusList1010 AS TABLE OF sys.ku$_WorkerStatus1010

CREATE TYPE sys.ku$_WorkerStatusList1020 AS TABLE OF sys.ku$_WorkerStatus1020

CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatusList1010
  FOR sys.ku$_WorkerStatusList1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatusList1020
  FOR sys.ku$_WorkerStatusList1020;

CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatusList
  FOR ku$_WorkerStatusList1020;
```

Log Entry and Error Types

These types provide informational and error text to attached clients and the log stream. The `ku$LogLine.errorNumber` type is set to `NULL` for informational messages but is specified for error messages. Each log entry may contain several lines of text messages.

The log entry and error types are defined as follows:

```
CREATE TYPE sys.ku$_LogLine1010 AS OBJECT (
              logLineNumber   NUMBER,
              errorNumber     NUMBER,
              LogText         VARCHAR2(2000))

CREATE OR REPLACE PUBLIC SYNONYM ku$_LogLine1010 FOR sys.ku$_LogLine1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$_LogLine1020 FOR sys.ku$_LogLine1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$_LogLine FOR ku$_LogLine1010;
CREATE TYPE sys.ku$_LogEntry1010 AS TABLE OF sys.ku$_LogLine1010

CREATE OR REPLACE PUBLIC SYNONYM ku$_LogEntry1010 FOR sys.ku$_LogEntry1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$_LogEntry1020 FOR sys.ku$_LogEntry1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$_LogEntry FOR ku$_LogEntry1010;
```

Job Status Types

The job status type returns status about a job. Usually, the status concerns a running job but it could also be about a stopped job when a client attaches. It is typically requested at attach time, when the client explicitly requests status from interactive mode and every N seconds when the client has requested status periodically.

The job status types are defined as follows (`percent_done` applies to data only):

```
CREATE TYPE sys.ku$_DumpFile1010 IS OBJECT (
              file_name          VARCHAR2(4000), -- Fully-qualified name
              file_type          NUMBER,         -- 0=Disk, 1=Pipe, etc.
              file_size          NUMBER,         -- Its length in bytes
              file_bytes_written NUMBER          -- Bytes written so far)

CREATE OR REPLACE PUBLIC SYNONYM ku$_DumpFile1010 FOR sys.ku$_DumpFile1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$_DumpFile1020 FOR sys.ku$_DumpFile1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$_DumpFile FOR ku$_DumpFile1010;

CREATE TYPE sys.ku$_DumpFileSet1010 AS TABLE OF sys.ku$_DumpFile1010;

CREATE OR REPLACE PUBLIC SYNONYM ku$_DumpFileSet1010 FOR
  sys.ku$_DumpFileSet1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$_DumpFileSet1020 FOR
```

```
  sys.ku$_DumpFileSet1010;

CREATE OR REPLACE PUBLIC SYNONYM ku$_DumpFileSet FOR ku$_DumpFileSet1010;

CREATE TYPE sys.ku$_JobStatus1010 IS OBJECT (
               job_name            VARCHAR2(30),
               operation           VARCHAR2(30),
               job_mode            VARCHAR2(30),
               bytes_processed     NUMBER,
               percent_done        NUMBER,
               degree              NUMBER,
               error_count         NUMBER,
               state               VARCHAR2(30),
               phase               NUMBER,
               restart_count       NUMBER,
               worker_status_list  ku$_WorkerStatusList1010,
               files               ku$_DumpFileSet1010)

CREATE PUBLIC SYNONYM ku$_JobStatus1010 FOR
   sys.ku$_JobStatus1010;

CREATE TYPE sys.ku$_JobStatus1020 IS OBJECT (
              job_name           VARCHAR2(30),             -- Name of the job
              operation          VARCHAR2(30),             -- Current operation
              job_mode           VARCHAR2(30),             -- Current mode
              bytes_processed    NUMBER,                   -- Bytes so far
              total_bytes        NUMBER,                   -- Total bytes for job
              percent_done       NUMBER,                   -- Percent done
              degree             NUMBER,                   -- Of job parallelism
              error_count        NUMBER,                   -- #errors so far
              state              VARCHAR2(30),             -- Current job state
              phase              NUMBER,                   -- Job phase
              restart_count      NUMBER,                   -- #Job restarts
              worker_status_list ku$_WorkerStatusList1020, -- job worker processes
              files              ku$_DumpFileSet1010       -- Dump file info)

CREATE OR REPLACE PUBLIC SYNONYM ku$_JobStatus1020 FOR   sys.ku$_JobStatus1020;

CREATE OR REPLACE PUBLIC SYNONYM ku$_JobStatus FOR ku$_JobStatus1020;
```

Job Description Types

The job description type holds all the environmental information about the job such as parameter settings and dump file set members. There are a couple of subordinate types required as well.

The job description types are defined as follows:

```
CREATE TYPE sys.ku$_ParamValue1010 AS OBJECT (
               param_name     VARCHAR2(30),
               param_op       VARCHAR2(30),
               param_type     VARCHAR2(30),
               param_length   NUMBER,
               param_value_n  NUMBER,
               param_value_t  VARCHAR2(4000));

CREATE OR REPLACE PUBLIC SYNONYM ku$_ParamValue1010 FOR sys.ku$_ParamValue1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$_ParamValue1020 FOR sys.ku$_ParamValue1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$_ParamValue FOR ku$_ParamValue1010;

CREATE TYPE sys.ku$_ParamValues1010 AS TABLE OF sys.ku$_ParamValue1010;
```

```
CREATE OR REPLACE PUBLIC SYNONYM ku$_ParamValues1010 FOR
  sys.ku$_ParamValues1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$_ParamValues1020 FOR
  sys.ku$_ParamValues1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$_ParamValues FOR ku$_ParamValues1010;

CREATE TYPE sys.ku$_JobDesc1010 AS OBJECT (
               job_name       VARCHAR2(30),
               guid           RAW(16),
               operation      VARCHAR2(30),
               job_mode       VARCHAR2(30),
               remote_link    VARCHAR2(4000),
               owner          VARCHAR2(30),
               instance       VARCHAR2(16),
               db_version     VARCHAR2(30),
               creator_privs  VARCHAR2(30),
               start_time     DATE,
               max_degree     NUMBER,
               log_file       VARCHAR2(4000),
               sql_file       VARCHAR2(4000),
               params         ku$_ParamValues1010)

CREATE OR REPLACE PUBLIC SYNONYM ku$_JobDesc1010 FOR sys.ku$_JobDesc1010;

CREATE TYPE sys.ku$_JobDesc1020 IS OBJECT (
               job_name        VARCHAR2(30),     -- The job name
               guid            RAW(16),          -- The job GUID
               operation       VARCHAR2(30),     -- Current operation
               job_mode        VARCHAR2(30),     -- Current mode
               remote_link     VARCHAR2(4000),   -- DB link, if any
               owner           VARCHAR2(30),     -- Job owner
               platform        VARCHAR2(101),    -- Current job platform
               exp_platform    VARCHAR2(101),    -- Export platform
               global_name     VARCHAR2(4000),   -- Global name of DB
               exp_global_name VARCHAR2(4000),   -- Export global name
               instance        VARCHAR2(16),     -- The instance name
               db_version      VARCHAR2(30),     -- Version of objects
               exp_db_version  VARCHAR2(30),     -- Export version
               scn             NUMBER,           -- Job SCN
               creator_privs   VARCHAR2(30),     -- Privs of job
               start_time      DATE,             -- This job start time
               exp_start_time  DATE,             -- Export start time
               term_reason     NUMBER,           -- Job termination code
               max_degree      NUMBER,           -- Max. parallelism
               log_file        VARCHAR2(4000),   -- Log file name
               sql_file        VARCHAR2(4000),   -- SQL file name
               params          ku$_ParamValues1010  -- Parameter list)

CREATE OR REPLACE PUBLIC SYNONYM ku$_JobDesc1020 FOR sys.ku$_JobDesc1020;
CREATE OR REPLACE PUBLIC SYNONYM ku$_JobDesc FOR ku$_JobDesc1020;
```

Status Types

The status type is an aggregate of some the previous types defined and is the return value for the `GET_STATUS` call. The mask attribute indicates which types of information are being returned to the caller. It is created by a client's shadow process from information it retrieves off the status queue or directly from the master table.

For errors, the ku$_LogEntry that is returned has already had its log lines ordered for proper output. That is, the original ku$_LogEntry objects have been ordered from outermost context to innermost.

The status types are defined as follows:

```
CREATE TYPE sys.ku$_Status1010 AS OBJECT
(
mask NUMBER,         /* Indicates which status types are present*/
wip ku$_LogEntry1010,  /* Work-In-Progress: std. exp/imp msgs */
job_description    ku$_JobDesc1010,   /* Complete job description */
job_status         ku$_JobStatus1010, /* Detailed job status + per-worker sts */
error              ku$_LogEntry1010   /* Multi-level contextual errors */
)

CREATE OR REPLACE PUBLIC SYNONYM ku$_Status1010 FOR sys.ku$_Status1010;

CREATE TYPE sys.ku$_Status1020 IS OBJECT
        (
               mask             NUMBER,            -- Status types present
               wip              ku$_LogEntry1010,  -- Work in progress
               job_description  ku$_JobDesc1020,   -- Complete job description
               job_status       ku$_JobStatus1020, -- Detailed job status
               error            ku$_LogEntry1010   -- Multi-level context errors
        )

CREATE OR REPLACE PUBLIC SYNONYM ku$_Status1020 FOR sys.ku$_Status1020;

CREATE OR REPLACE PUBLIC SYNONYM ku$_Status FOR ku$_Status1020;
```

Summary of DBMS_DATAPUMP Subprograms

Table 27–1 DBMS_DATAPUMP Package Subprograms

Subprogram	Description
ADD_FILE Procedure on page 27-13	Adds dump files to the dump file set for an Export, Import, or SQL_FILE operation. In addition to dump files, other types of files can also be added by using the `FILETYPE` parameter provided with this procedure
ATTACH Function on page 27-16	Used to gain access to a Data Pump job that is in the Defining, Executing, Idling, or Stopped state
DATA_FILTER Procedures on page 27-18	Specifies restrictions on the rows that are to be retrieved
DETACH Procedure on page 27-20	Specifies that the user has no further interest in using the handle
GET_DUMPFILE_INFO Procedure on page 27-21	Monitors the status of a job or waits for the completion of a job.
GET_STATUS Procedure on page 27-23	Monitors the status of a job or waits for the completion of a job or for more details on API errors
LOG_ENTRY Procedure on page 27-26	Inserts a message into the log file
METADATA_FILTER Procedure on page 27-27	Provides filters that allow you to restrict the items that are included in a job
METADATA_REMAP Procedure on page 27-30	Specifies a remapping to be applied to objects as they are processed in the specified job
METADATA_TRANSFORM Procedure on page 27-32	Specifies transformations to be applied to objects as they are processed in the specified job
OPEN Function on page 27-35	Declares a new job using the Data Pump API, the handle returned being used as a parameter for calls to all other procedures except `ATTACH`
SET_PARALLEL Procedure on page 27-38	Adjusts the degree of parallelism within a job
SET_PARAMETER Procedures on page 27-40	Specifies job-processing options
START_JOB Procedure on page 27-44	Begins or resumes execution of a job
STOP_JOB Procedure on page 27-46	Terminates a job, but optionally, preserves the state of the job
WAIT_FOR_JOB Procedure on page 27-48	Runs a job until it either completes normally or stops for some other reason.

ADD_FILE Procedure

This procedure adds files to the dump file set for an Export, Import, or SQL_FILE operation or specifies the log file or the output file for a SQL_FILE operation.

Syntax

```
DBMS_DATAPUMP.ADD_FILE (
   handle     IN NUMBER,
   filename   IN VARCHAR2,
   directory  IN VARCHAR2,
   filesize   IN VARCHAR2 DEFAULT NULL,
   filetype   IN NUMBER DEFAULT DBMS_DATAPUMP.KU$_FILE_TYPE_DUMP_FILE);
```

Parameters

Table 27–2 ADD_FILE Procedure Parameters

Parameter	Description
handle	The handle of a job. The current session must have previously attached to the handle through an OPEN or ATTACH call.
filename	The name of the file being added. filename must be a simple filename without any directory path information. For dump files, the filename can include a substitution variable, %U, which indicates that multiple files may be generated with the specified filename as a template. The %U is expanded in the resulting file names into a two-character, fixed-width, incrementing integer starting at 01. For example, the dump filename of export%U would cause export01, export02, export03, and so on, to be created depending on how many files are needed to perform the export. For filenames containing the % character, the % must be represented as %% to avoid ambiguity. Any % in a filename must be followed by either a % or a U.
directory	The name of a directory object within the database that is used to locate filename. A directory must be specified. See the Data Pump Export chapter in *Oracle Database Utilities* for information about the DIRECTORY command-line parameter.
filesize	The size of the dump file that is being added. It may be specified as the number of bytes, number of kilobytes (if followed by K), number of megabytes (if followed by M) or number of gigabytes (if followed by G). An Export operation will write no more than the specified number of bytes to the file. Once the file is full, it will be closed. If there is insufficient space on the device to write the specified number of bytes, the Export operation will fail, but it can be restarted. If not specified, filesize will default to an unlimited size. For Import and SQL_FILE operations, filesize is ignored. The minimum value for filesize is ten times the default Data Pump block size, which is 4 kilobytes. filesize may only be specified for dump files.
filetype	The type of the file to be added. The legal values are as follows and must be preceded by DBMS_DATAPUMP.: ■ KU$_FILE_TYPE_DUMP_FILE (dump file for a job) ■ KU$_FILE_TYPE_LOG_FILE (log file for a job) ■ KU$_FILE_TYPE_SQL_FILE (output for SQL_FILE job)

Exceptions

- INVALID_HANDLE. The specified handle is not attached to a Data Pump job.

- INVALID_ARGVAL. An invalid value was supplied for an input parameter.
- INVALID_STATE. The job is completing, or the job is past the defining state for an import or SQL_FILE job or is past the defining state for LOG and SQL files.
- FILE_ERROR. Oracle does not have the requested operating system access to the specified file or the file has already been specified for the current operation.
- INVALID_OPERATION. A dump file was specified for a Network Import or ESTIMATE_ONLY export operation.
- SUCCESS_WITH_INFO. The procedure succeeded, but further information is available through the GET_STATUS procedure.
- NO_SUCH_JOB. The specified job does not exist.

Usage Notes

- Adds files to a Data Pump job. Three types of files may be added to jobs: Dump files to contain the data that is being moved, log files to record the messages associated with an operation, and SQL files to record the output of a SQL_FILE operation. Log and SQL files will overwrite previously existing files. Dump files will never overwrite previously existing files. Instead, an error will be generated.
- Import and SQL_FILE operations require that all dump files be specified during the definition phase of the job. For Export operations, dump files can be added at any time. For example, if the user ascertains that the file space is running low during an Export, additional dump files may be added through this API. If the specified dump file already exists for an Export operation, an error will be returned.
- For Export operations, the parallelism setting should be less than or equal to the number of dump files in the dump file set. If there are not enough dump files, the job will not be able to maximize parallelism to the degree specified by the SET_PARALLEL procedure.
- For Import operations, the parallelism setting should also be less than or equal to the number of dump files in the dump file set. If there are not enough dump files, the performance will not be optimal as multiple threads of execution try to access the same dump file.
- If the substitution variable (%U) is included in a filename, multiple dump files may be specified through a single call to ADD_FILE. For Export operations, the new dump files will be created as they are needed. Enough dump files will be created to allow all of the processes specified by the current SET_PARALLEL value to be active. If one of the dump files fills, it will be closed and a new dump file (with a new generated name) will be created to take its place. If multiple ADD_FILEs with substitution variables have been specified for dump files in a job, they will be used to generate dump files in a round robin fashion. For example, if expa%U, expb%U and expc%U were all specified for a job having a parallelism of 6, the initial dump files created would look like: expa01, expb01, expc01, expa02, expb02, and expc02.
- If presented with dump file specifications, expa%U, expb%U and expc%U, an Import or SQL_FILE operation will begin by attempting to open the dump files, expa01, expb01, and expc01. If the dump file containing the master table is not found in this set, the operation will expand its search for dump files by incrementing the substitution variable and looking up the new filenames (for example, expa02, expb02, and expc02). The DataPump API will keep expanding the search until it locates the dump file containing the master table. If the DataPump API determines that the dump file does not exist or is not part of

the current dump set at any iteration, the DataPump API will stop incrementing the substitution variable for the dump file specification that was in error. Once the master table is found, the master table will be used to ascertain when all of dump files in the dump file set have been located.

ATTACH Function

This function gains access to a previously-created job.

Syntax

```
DBMS_DATAPUMP.ATTACH(
   job_name    IN VARCHAR2 DEFAULT NULL,
   job_owner    IN VARCHAR2 DEFAULT NULL)
 RETURN NUMBER;
```

Parameters

Table 27–3 ATTACH Function Parameters

Parameter	Description
job_name	The name of the job. The default is the job name owned by the user who is specified in the job_owner parameter (assuming that user has only one job in the Defining, Executing, or Idling states).
job_owner	The user who originally started the job. If NULL, the value defaults to the owner of the current session. To specify a job owner other than yourself, you must have either the EXP_FULL_DATABASE role (for export operations) or the IMP_FULL_DATABASE role (for import and SQL_FILE operations). Being a privileged user allows you to monitor another user's job, but you cannot restart another user's job.

Return Values

An opaque handle for the job. This handle is used as input to the following procedures: ADD_FILE, DATA_FILTER, DETACH, GET_STATUS, LOG_ENTRY, METADATA_FILTER, METADATA_REMAP, METADATA_TRANSFORM, SET_PARALLEL, SET_PARAMETER, START_JOB, STOP_JOB, and WAIT_FOR_JOB.

Exceptions

- INVALID_ARGVAL. An invalid value was supplied for an input parameter.
- OBJECT_NOT_FOUND. The specified job no longer exists or the user specified a job owned by another schema, but the user did not have the EXP_FULL_DATABASE or IMP_FULL_DATABASE role.
- SUCCESS_WITH_INFO. The procedure succeeded, but further information is available through the GET_STATUS procedure.
- NO_SUCH_JOB. The specified job does not exist.

Usage Notes

- If the job was in the Stopped state, the job is placed into the Idling state. Once the ATTACH succeeds, you can monitor the progress of the job or control the job. The stream of KU$_STATUS_WIP and KU$_STATUS_JOB_ERROR messages returned through the GET_STATUS procedure will be returned to the newly attached job starting at the approximate time of the client's attachment. There will be no repeating of status and error messages that were processed before the client attached to a job.
- If you want to perform a second attach to a job, you must do so from a different session.

- If the ATTACH fails, use a null handle in a subsequent call to GET_STATUS for more information about the failure.

DATA_FILTER Procedures

This procedure specifies restrictions on the rows that are to be retrieved.

Syntax

```
DBMS_DATAPUMP.DATA_FILTER (
   handle      IN NUMBER,
   name        IN VARCHAR2,
   value       IN NUMBER,
   table_name  IN VARCHAR2 DEFAULT NULL,
   schema_name IN VARCHAR2 DEFAULT NULL);

DBMS_DATAPUMP.DATA_FILTER(
   handle      IN NUMBER,
   name        IN VARCHAR2,
   value       IN VARCHAR2,
   table_name  IN VARCHAR2 DEFAULT NULL,
   schema_name IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 27–4 DATA_FILTER Procedure Parameters

Parameter	Description
handle	The handle that is returned from the OPEN procedure.
name	The name of the filter.
value	The value of the filter.
table_name	The name of the table on which the data filter is applied. If no table name is supplied, the filter applies to all tables in the job.
schema_name	The name of the schema that owns the table on which the filter is applied. If no schema name is specified, the filter applies to all schemas in the job. If you supply a schema name you must also supply a table name.

Exceptions

- INVALID_ARGVAL. There can be several reasons for this message:
  - A bad filter name is specified
  - The mode is TRANSPORTABLE, which does not support data filters
  - The specified table does not exist
  - The filter has already been set for the specified values of schema_name and table_name
- INVALID_STATE. The user called DATA_FILTER when the job was not in the Defining state.
- INCONSISTENT_ARGS. The value parameter is missing or its datatype does not match the filter name. Or a schema name was supplied, but not a table name.
- PRIVILEGE_ERROR. A schema name was supplied, but the user did not have the EXP_FULL_DATABASE or IMP_FULL_DATABASE role.
- SUCCESS_WITH_INFO. The procedure succeeded, but further information is available through the GET_STATUS procedure.

- NO_SUCH_JOB. The specified job does not exist.

Usage Notes

- Each data filter can only appear once in each table (for example, you cannot supply multiple SUBQUERY filters to a table) or once in each job. If different filters using the same name are applied to both a particular table and to the whole job, the filter parameter supplied for the specific table will take precedence.

  With the exception of the INCLUDE_ROWS filter, data filters are not supported on tables having nested tables or domain indexes defined upon them. Data filters are not supported in jobs performed in Transportable Tablespace mode.

  The available data filters are described in Table 27–5.

Table 27–5 Data Filters

Name	Datatype	Operations that Support Filter	Description
INCLUDE_ ROWS	NUMBER	EXPORT, IMPORT	If nonzero, this filter specifies that user data for the specified table should be included in the job. The default is 1.
PARTITION_ EXPR PARTITION_ LIST	text	EXPORT, IMPORT	For Export jobs, these filters specify which partitions are unloaded from the database. For Import jobs, they specify which table partitions are loaded into the database. Partition names are included in the job if their names satisfy the specified expression (for PARTITION_EXPR) or are included in the list (for PARTITION_LIST). Whereas the expression version of the filter offers more flexibility, the list version provides for full validation of the partition names. Double quotation marks around partition names are required only if the partition names contain special characters. PARTITION_EXPR is not supported on jobs across a network link. Default=All partitions are processed.
SAMPLE	NUMBER	EXPORT, IMPORT	For Export jobs, specifies a percentage for sampling the data blocks to be moved. This filter allows subsets of large tables to be extracted for testing purposes.
SUBQUERY	text	EXPORT, IMPORT	Specifies a subquery that is added to the end of the SELECT statement for the table. If you specify a WHERE clause in the subquery, you can restrict the rows that are selected. Specifying an ORDER BY clause orders the rows dumped in the export which improves performance when migrating from heap-organized tables to index-organized tables.

DETACH Procedure

This procedure specifies that the user has no further interest in using the handle.

Syntax

```
DBMS_DATAPUMP.DETACH(
   handle  IN NUMBER);
```

Parameters

Table 27–6 DETACH Procedure Parameters

Parameter	Description
handle	The handle of the job. The current session must have previously attached to the handle through an OPEN or ATTACH call.

Exceptions

- INVALID_HANDLE. The specified handle is not attached to a Data Pump job.
- SUCCESS_WITH_INFO. The procedure succeeded, but further information is available through the GET_STATUS procedure.
- NO_SUCH_JOB. The specified job does not exist.

Usage Notes

- Through this call, you specify that you have no further interest in using the handle. Resources associated with a completed job cannot be reclaimed until all users are detached from the job. An implicit detach from a handle is performed when the user's session is exited or aborted. An implicit detach from a handle is also performed upon the expiration of the timeout associated with a STOP_JOB that was applied to the job referenced by the handle. All previously allocated DBMS_DATAPUMP handles are released when an instance is restarted.

GET_DUMPFILE_INFO Procedure

This procedure retrieves information about a specified dump file.

Syntax

```
DBMS_DATAPUMP.GET_DUMPFILE_INFO(
   filename    IN VARCHAR2,
   directory   IN VARCHAR2,
   info_table  OUT ku$_dumpfile_info,
   filetype    OUT NUMBER);
```

Parameters

Table 27–7 GET_DUMPFILE_INFO Procedure Parameters

Parameter	Description
filename	A simple filename with no directory path information.
directory	A directory object that specifies where the file can be found.
info_table	A PL/SQL table for storing information about the dump file.
filetype	The type of file (Data Pump dump file, original Export dump file, or unknown).

Exceptions

The GET_DUMPFILE_INFO procedure is a utility routine that operates outside the context of any Data Pump job. Exceptions are handled differently for this procedure than for procedures associated in some way with a Data Pump job. A full exception stack should be available directly, without the need to call the GET_STATUS procedure to retrieve the detailed information. The exception for this procedure is as follows:

- NO_DUMPFILE_INFO. Unable to retrieve dump file information as specified.

Usage Notes

You can use the GET_DUMPFILE_INFO procedure to request information about a specific file. If the file is not recognized as any type of dump file, then a filetype of zero will be returned and the dump file info_table will remain empty.

A filetype value of one indicates a Data Pump dump file. A file type value of two indicates an original Export dump file. In both cases, the dump file info_table will be populated with information retrieved from the dump file header. Rows of this table consist of item code and value pairs, where the item code indicates the type of information and the value column is a VARCHAR2 containing the actual data (converted to a string in some cases). The table is defined as follows:

```
CREATE TYPE sys.ku$_dumpfile_item IS OBJECT (
               item_code       NUMBER,           -- Identifies header item
               value           VARCHAR2(2048)    -- Text string value)
/

GRANT EXECUTE ON sys.ku$_dumpfile_item TO PUBLIC;
CREATE OR REPLACE PUBLIC SYNONYM ku$_dumpfile_item FOR sys.ku$_dumpfile_item;

CREATE TYPE sys.ku$_dumpfile_info AS TABLE OF sys.ku$_dumpfile_item
/
```

```
GRANT EXECUTE ON sys.ku$_dumpfile_info TO PUBLIC;
CREATE OR REPLACE PUBLIC SYNONYM ku$_dumpfile_info FOR sys.ku$_dumpfile_info;
```

The item codes, which can easily be extended to provide more information as needed, are currently defined as follows (prepended with the package name, `DBMS_DATAPUMP.`):

```
KU$_DFHDR_FILE_VERSION         CONSTANT NUMBER := 1;
KU$_DFHDR_MASTER_PRESENT       CONSTANT NUMBER := 2;
KU$_DFHDR_GUID                 CONSTANT NUMBER := 3;
KU$_DFHDR_FILE_NUMBER          CONSTANT NUMBER := 4;
KU$_DFHDR_CHARSET_ID           CONSTANT NUMBER := 5;
KU$_DFHDR_CREATION_DATE        CONSTANT NUMBER := 6;
KU$_DFHDR_FLAGS                CONSTANT NUMBER := 7;
KU$_DFHDR_JOB_NAME             CONSTANT NUMBER := 8;
KU$_DFHDR_PLATFORM             CONSTANT NUMBER := 9;
KU$_DFHDR_INSTANCE             CONSTANT NUMBER := 10;
KU$_DFHDR_LANGUAGE             CONSTANT NUMBER := 11;
KU$_DFHDR_BLOCKSIZE            CONSTANT NUMBER := 12;
KU$_DFHDR_DIRPATH              CONSTANT NUMBER := 13;
KU$_DFHDR_METADATA_COMPRESSED  CONSTANT NUMBER := 14;
KU$_DFHDR_DB_VERSION           CONSTANT NUMBER := 15;
KU$_DFHDR_MAX_ITEM_CODE        CONSTANT NUMBER := 15;
```

GET_STATUS Procedure

This procedure monitors the status of a job or waits for the completion of a job.

Syntax

```
DBMS_DATAPUMP.GET_STATUS(
   handle    IN NUMBER,
   mask      IN BINARY_INTEGER,
   timeout   IN NUMBER DEFAULT NULL,
   job_state OUT VARCHAR2,
   status    OUT ku$_Status1010);
```

Parameters

Table 27–8 GET_STATUS Procedure Parameters

Parameter	Description
handle	The handle of a job. The current session must have previously attached to the handle through an OPEN or ATTACH call. A null handle can be used to retrieve error information after OPEN and ATTACH failures.
mask	A bit mask that indicates which of four types of information to return: ■ KU$_STATUS_WIP ■ KU$_STATUS_JOB_DESC ■ KU$_STATUS_JOB_STATUS ■ KU$_STATUS_JOB_ERROR Each status has a numerical value. You can request multiple types of information by adding together different combinations of values. See Data Structures - Object Types on page 27-7.
timeout	Maximum number of seconds to wait before returning to the user. A value of 0 requests an immediate return. A value of -1 requests an infinite wait. If KU$_STATUS_WIP or KU$_STATUS_JOB_ERROR information is requested and becomes available during the timeout period, then the procedure returns before the timeout period is over.
job_state	Current state of the job. If only the job state is needed, it is much more efficient to use this parameter than to retrieve the full ku$_Status structure.
status	A ku$_Status is returned. The ku$_Status mask indicates what kind of information is included. This could be none if only KU$_STATUS_WIP or KU$_STATUS_JOB_ERROR information is requested and the timeout period expires. This can be a ku$_Status1010 or ku$_Status1020 object type.

Exceptions

- INVALID_HANDLE. The specified handle is not attached to a Data Pump job.
- INVALID_VALUE. The mask or timeout contains an illegal value.
- SUCCESS_WITH_INFO. The procedure succeeded, but further information is available through the GET_STATUS procedure.
- NO_SUCH_JOB. The specified job does not exist.

Usage Notes

The GET_STATUS procedure is used to monitor the progress of an ongoing job and to receive error notification. You can request various type of information using the mask parameter. The KU$_STATUS_JOB_DESC and KU$_STATUS_JOB_STATUS values are classified as synchronous information because the information resides in the master table. The KU$_STATUS_WIP and KU$_STATUS_JOB_ERROR values are classified as asynchronous because the messages that embody these types of information can be generated at any time by various layers in the Data Pump architecture.

- If synchronous information *only* is requested, the interface will ignore the timeout parameter and simply return the requested information.
- If asynchronous information is requested, the interface will wait a *maximum* of timeout seconds before returning to the client. If a message of the requested asynchronous information type is received, the call will complete prior to timeout seconds. If synchronous information was also requested, it will be returned whenever the procedure returns.
- If the job_state returned by GET_STATUS does not indicate a terminating job, it is possible that the job could still terminate before the next call to GET_STATUS. This would result in an INVALID_HANDLE exception. Alternatively, the job could terminate during the call to GET_STATUS, which would result in a NO_SUCH_JOB exception. Callers should be prepared to handle these cases.

Error Handling

There are two types of error scenarios that need to be handled using the GET_STATUS procedure:

- Errors resulting from other procedure calls: For example, the SET_PARAMETER procedure may produce an INCONSISTENT_ARGS exception. The client should immediately call GET_STATUS with mask=8 (errors) and timeout=0. The returned ku$_Status.error will contain a ku$_LogEntry that describes the inconsistency in more detail.
- Errors resulting from events asynchronous to the client(s): An example might be Table already exists when trying to create a table. The ku$_Status.error will contain a ku$_LogEntry with all error lines (from all processing layers that added context about the error) properly ordered.

After a job has begun, a client's main processing loop will typically consist of a call to GET_STATUS with an infinite timeout (-1) "listening" for KU$_STATUS_WIP and KU$_STATUS_JOB_ERROR messages. If status was requested, then JOB_STATUS information will also be in the request.

When the ku$_Status is interpreted, the following guidelines should be used:

- ku$_Status.ku$_JobStatus.percent_done refers only to the amount of data that has been processed in a job. Metadata is not considered in the calculation. It is determined using the following formulas:
  - EXPORT or network IMPORT--(bytes_processed/estimated_bytes) * 100
  - IMPORT--(bytes_processed/total_expected_bytes) * 100
  - SQL_FILE or estimate-only EXPORT--0.00 if not done or 100.00 if done

  The effects of the QUERY and PARTITION_EXPR data filters are not considered in computing percent_done.

It is expected that the status returned will be transformed by the caller into more user-friendly status. For example, when percent done is not zero, an estimate of completion time could be produced using the following formula:

```
((SYSDATE - start time) / ku$_Status.ku$_JobStatus.percent_done) * 100
```

- The caller should not use `ku$_Status.ku$_JobStatus.percent_done` for determining whether the job has completed. Instead, the caller should only rely on the state of the job as found in `job_state`.

LOG_ENTRY Procedure

This procedure inserts a message into the log file.

Syntax

```
DBMS_DATAPUMP.LOG_ENTRY(
   handle         IN NUMBER,
   message        IN VARCHAR2
   log_file_only  IN NUMBER DEFAULT 0);
```

Parameters

Table 27–9 LOG_ENTRY Procedure Parameters

Parameter	Description
handle	The handle of a job. The current session must have previously attached to the handle through an OPEN or ATTACH call.
message	A text line to be added to the log file.
log_file_only	Specified text should be written only to the log file. It should not be returned in GET_STATUS work-in-progress (KU$_STATUS_WIP) messages.

Exceptions

- INVALID_HANDLE. The specified handle is not attached to a Data Pump job.
- SUCCESS_WITH_INFO. The procedure succeeded, but further information is available through the GET_STATUS procedure.
- NO_SUCH_JOB. The specified job does not exist.

Usage Notes

The message is added to the log file. If log_file_only is zero (the default), the message is also broadcast as a KU$_STATUS_WIP message through the GET_STATUS procedure to all users attached to the job.

The LOG_ENTRY procedure allows applications to tailor the log stream to match the abstractions provided by the application. For example, the command-line interface supports INCLUDE and EXCLUDE parameters defined by the user. Identifying these values as calls to the underlying METADATA_FILTER procedure would be confusing to users. Instead, the command-line interface can enter text into the log describing the settings for the INCLUDE and EXCLUDE parameters.

Lines entered in the log stream from LOG_ENTRY are prefixed by the string, ";;; "

METADATA_FILTER Procedure

This procedure provides filters that allow you to restrict the items that are included in a job.

Syntax

```
DBMS_DATAPUMP.METADATA_FILTER(
   handle       IN NUMBER,
   name         IN VARCHAR2,
   value        IN VARCHAR2,
   object_path  IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 27–10 METADATA_FILTER Procedure Parameters

Parameter	Description
handle	The handle returned from the OPEN procedure.
name	The name of the filter. See Table 27–11 for descriptions of the available filters.
value	The value of the filter.
object_path	The object path to which the filter applies. If the default is used, the filter applies to all applicable objects. Lists of the object paths supported for each mode are contained in the catalog views for DATABASE_EXPORT_OBJECTS, SCHEMA_EXPORT_OBJECTS, and TABLE_EXPORT_OBJECTS. (Note that the TABLE_EXPORT_OBJECTS view is applicable to both Table and Tablespace mode because their object paths are the same.) For an import operation, object paths reference the mode used to create the dump file rather than the mode being used for the import.

Table 27–11 describes the name, the object type, and the meaning of the filters available with the METADATA_FILTER procedure. The datatype for all the filters is a text expression. All operations support all filters.

Table 27–11 Filters Provided by METADATA_FILTER Procedure

Name	Object Type	Meaning
NAME_EXPR NAME_LIST	Named objects	Defines which object names are included in the job. You use the object type parameter to limit the filter to a particular object type. For Table mode, identifies which tables are to be processed.
SCHEMA_EXPR SCHEMA_LIST	Schema objects	Restricts the job to objects whose owning schema name is satisfied by the expression. For Table mode, only a single SCHEMA_EXPR filter is supported. If specified, it must only specify a single schema (for example, 'IN (''SCOTT'')'). For Schema mode, identifies which users are to be processed.

Table 27–11 (Cont.) Filters Provided by METADATA_FILTER Procedure

Name	Object Type	Meaning
TABLESPACE_EXPR	TABLE, CLUSTER, INDEX, ROLLBACK_SEGMENT	Restricts the job to objects stored in a tablespace whose name is satisfied by the expression.
TABLESPACE_LIST		For Tablespace mode, identifies which tablespaces are to be processed. If a partition of an object is stored in the tablespace, the entire object is added to the job. For Transportable mode, identifies which tablespaces are to be processed. If a table has a single partition in the tablespace set, all partitions must be in the tablespace set. An index is not included within the tablespace set unless all of its partitions are in the tablespace set. A domain index is not included in the tablespace set unless all of its secondary objects are included in the tablespace set.
INCLUDE_PATH_EXPR INCLUDE_PATH_LIST EXCLUDE_PATH_EXPR EXCLUDE_PATH_LIST	All	Defines which object paths are included in, or excluded from, the job. You use these filters to select only certain object types from the database or dump file set. Objects of paths satisfying the condition are included (INCLUDE_PATH_*) or excluded (EXCLUDE_PATH_*) from the operation. The object_path parameter is not supported for these filters.

Exceptions

- INVALID_HANDLE. The specified handle is not attached to a Data Pump job.
- INVALID_ARGVAL. This exception can indicate any of the following conditions:
  - An object_path was specified for an INCLUDE_PATH_EXPR or EXCLUDE_PATH_EXPR filter.
  - The specified object_path is not supported for the current mode.
  - The SCHEMA_EXPR filter specified multiple schemas for a Table mode job.
- INVALID_STATE. The user called the METADATA_FILTER procedure after the job left the defining state.
- INCONSISTENT_ARGS. The filter value is of the wrong datatype or is missing.
- SUCCESS_WITH_INFO. The procedure succeeded but further information is available through the GET_STATUS procedure.
- NO_SUCH_JOB. The specified job does not exist.

Usage Notes

- Metadata filters identify a set of objects to be included or excluded from a Data Pump operation. Except for EXCLUDE_PATH_EXPR and INCLUDE_PATH_EXPR, dependent objects of an identified object will be processed along with the identified object. For example, if an index is identified for inclusion by a filter, grants upon that index will also be included by the filter. Likewise, if a table is excluded by a filter, then indexes, constraints, grants and triggers upon the table will also be excluded by the filter.
- Two versions of each filter are supported: SQL expression and List. The SQL expression version of the filters offer maximum flexibility for identifying objects (for example the use of LIKE to support use of wild cards). The names of the expression filters are as follows:

  - NAME_EXPR
  - SCHEMA_EXPR
  - TABLESPACE_EXPR
  - INCLUDE_PATH_EXPR
  - EXCLUDE_PATH_EXPR

  The list version of the filters allow maximum validation of the filter. An error will be reported if one of the elements in the filter is not found within the source database (for Export and network-based jobs) or is not found within the dump file (for file-based Import and SQLFILE jobs). The names of the list filters are as follows:

  - NAME_LIST
  - SCHEMA_LIST
  - TABLESPACE_LIST
  - INCLUDE_PATH_LIST
  - EXCLUDE_PATH_LIST

- Filters allow a user to restrict the items that are included in a job. For example, a user could request a full export, but without Package Specifications or Package Bodies.
- If multiple filters are specified for a object type, they are implicitly 'ANDed' together (that is, objects participating in the job must pass all of the filters applied to their object types).
- The same filter name can be specified multiple times within a job. For example, specifying `NAME_EXPR as '!=''EMP'''` and `NAME_EXPR as '!=''DEPT'''` on a Table mode export would produce a file set containing all of the tables except for `EMP` and `DEPT`.

METADATA_REMAP Procedure

This procedure specifies a remapping to be applied to objects as they are processed in the specified job.

Syntax

```
DBMS_DATAPUMP.METADATA_REMAP (
   handle      IN NUMBER,
   name        IN VARCHAR2,
   old_value   IN VARCHAR2,
   value       IN VARCHAR2,
   object_type IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 27–12 METADATA_REMAP Procedure Parameters

Parameter	Description
handle	The handle for the current job. The current session must have previously attached to the handle through a call to the OPEN procedure.
name	The name of the remap. See Table 27–13 for descriptions of the available remaps.
old_value	Specifies which value in the dump file set should be reset to value.
value	The value of the parameter for the remap. This signifies the new value that old_value should be translated into.
object_type	Designates the object type to which the remap applies. The list of object types supported for each mode are contained in the DATABASE_EXPORT_OBJECTS, SCHEMA_EXPORT_OBJECTS, TABLE_EXPORT_OBJECTS, and TABLESPACE_EXPORT_OBJECTS catalog views. By default, the remap applies to all applicable objects within the job. The object_type parameter allows a caller to specify different parameters for different object types within a job. Remaps that explicitly specify an object type override remaps that apply to all object types.

Table 27–13 describes the remaps provided by the METADATA_REMAP procedure.

Table 27–13 Remaps Provided by the METADATA_REMAP Procedure

Name	Datatype	Object Type	Meaning
REMAP_SCHEMA	Text	Schema objects	Any schema object in the job that matches the object_type parameter and was located in the old_value schema will be moved to the value schema. Privileged users can perform unrestricted schema remaps. Nonprivileged users can perform schema remaps only if their schema is the target schema of the remap. For example, SCOTT can remap his BLAKE's objects to SCOTT, but SCOTT cannot remap SCOTT's objects to BLAKE.

Table 27–13 (Cont.) Remaps Provided by the METADATA_REMAP Procedure

Name	Datatype	Object Type	Meaning
REMAP_TABLESPACE	Text	TABLE, INDEX, ROLLBACK_SEGMENT, MATERIALIZED_VIEW, MATERIALIZED_VIEW_LOG,TABLE_SPACE	Any storage segment in the job that matches the object_type parameter and was located in the old_value tablespace will be relocated to the value tablespace.
REMAP_DATAFILE	Text	LIBRARY, TABLESPACE, DIRECTORY	Any datafile reference in the job that matches the object_type parameter and referenced the old_value datafile will be redefined to use the value datafile.

Exceptions

- INVALID_HANDLE. The specified handle is not attached to a Data Pump job.
- INVALID_ARGVAL. This message can indicate any of the following:
  - The job's mode does not include the specified object_type.
  - The remap has already been specified for the specified old_value and object_type.
- INVALID_OPERATION. Remaps are only supported for SQL_FILE and Import operations. The job's operation was Export, which does not support the use of metadata remaps.
- INVALID_STATE. The user called METADATA_REMAP after the job had started (that is, the job was not in the defining state).
- INCONSISTENT_ARGS. There was no value supplied or it was of the wrong datatype for the remap.
- PRIVILEGE_ERROR. A nonprivileged user attempted to do a REMAP_SCHEMA to a different user's schema or a REMAP_DATAFILE.
- SUCCESS_WITH_INFO. The procedure succeeded, but further information is available through the GET_STATUS procedure.
- NO_SUCH_JOB. The specified job does not exist.

Usage Notes

- The METADATA_REMAP procedure is only supported for Import and SQL_FILE operations. It enables you to apply commonly desired, predefined remappings to the definition of objects as part of the transfer. If you need remaps that are not supported within this procedure, you should do a preliminary SQL_FILE operation to produce a SQL script corresponding to the dump file set. By editing the DDL directly and then executing it, you can produce any remappings that you need.
- Transforms for the DataPump API are a subset of the remaps implemented by the DBMS_METADATA.SET_TRANSFORM_PARAMETER API. Multiple remaps can be defined for a single job. However, each remap defined must be unique according its parameters. That is, two remaps cannot specify conflicting or redundant remaps.

METADATA_TRANSFORM Procedure

This procedure specifies transformations to be applied to objects as they are processed in the specified job.

Syntax

```
DBMS_DATAPUMP.METADATA_TRANSFORM (
   handle      IN NUMBER,
   name        IN VARCHAR2,
   value       IN VARCHAR2,
   object_type IN VARCHAR2 DEFAULT NULL);

DBMS_DATAPUMP.METADATA_TRANSFORM (
   handle      IN NUMBER,
   name        IN VARCHAR2,
   value       IN NUMBER,
   object_type IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 27–14 METADATA_TRANSFORM Procedure Parameters

Parameter	Description
handle	The handle for the current job. The current session must have previously attached to the handle through a call to the OPEN procedure.
name	The name of the transformation. See Table 27–15 for descriptions of the available transforms.
value	The value of the parameter for the transform.
object_type	Designates the object type to which the transform applies. The list of object types supported for each mode are contained in the DATABASE_EXPORT_OBJECTS, SCHEMA_EXPORT_OBJECTS, TABLE_EXPORT_OBJECTS, and TABLESPACE_EXPORT_OBJECTS catalog views. By default, the transform applies to all applicable objects within the job. The object_type parameter allows a caller to specify different transform parameters for different object types within a job. Transforms that explicitly specify an object type override transforms that apply to all object types.

Table 27–15 describes the transforms provided by the METADATA_TRANSFORM procedure.

Table 27–15 Transforms Provided by the METADATA_TRANFORM Procedure

Name	Datatype	Object Type	Meaning
PCTSPACE	NUMBER	TABLE INDEX TABLESPACE	Specifies a percentage multiplier used to alter extent allocations and datafile sizes. Used to shrink large tablespaces for testing purposes. Defaults to 100.
SEGMENT_ATTRIBUTES	NUMBER	TABLE, INDEX	If nonzero (TRUE), emit storage segment parameters. Defaults to 1.

Table 27–15 (Cont.) Transforms Provided by the METADATA_TRANFORM Procedure

Name	Datatype	Object Type	Meaning
STORAGE	NUMBER	TABLE	If nonzero (TRUE), emit storage clause. (Ignored if SEGMENT_ATTRIBUTES is zero.) Defaults to nonzero (TRUE).
OID	NUMBER	TYPE TABLE	If zero, inhibits the assignment of the exported OID during type or table creation. Instead, a new OID will be assigned. Use of this transform on Object Tables will cause breakage in REF columns that point to the table. Defaults to 1.

Exceptions

- INVALID_HANDLE. The specified handle is not attached to a Data Pump job.
- INVALID_ARGVAL. This message can indicate any of the following:
  - The mode is transportable, which doesn't support transforms.
  - The job's mode does not include the specified object_type.
  - The transform has already been specified for the specified value and object_type.
- INVALID_OPERATION. Transforms are only supported for SQL_FILE and Import operations. The job's operation was Export which does not support the use of metadata transforms.
- INVALID_STATE. The user called METADATA_TRANSFORM after the job had started (that is, the job was not in the defining state).
- INCONSISTENT_ARGS. There was no value supplied or it was of the wrong datatype for the transform.
- PRIVILEGE_ERROR. A nonprivileged user attempted to do a REMAP_SCHEMA to a different user's schema or a REMAP_DATAFILE.
- SUCCESS_WITH_INFO. The procedure succeeded, but further information is available through the GET_STATUS procedure.
- NO_SUCH_JOB. The specified job does not exist.

Usage Notes

- The METADATA_TRANSFORM procedure is only supported for Import and SQL_FILE operations. It enables you to apply commonly desired, predefined transformations to the definition of objects as part of the transfer. If you need transforms that are not supported within this procedure, you should do a preliminary SQL_FILE operation to produce a SQL script corresponding to the dump file set. By editing the DDL directly and then executing it, you can produce any transformations that you need.
- Transforms for the DataPump API are a subset of the transforms implemented by the DBMS_METADATA.SET_TRANSFORM_PARAMETER API. Multiple transforms can be defined for a single job. However, each transform defined must be unique

according its parameters. That is, two transforms cannot specify conflicting or redundant transformations.

OPEN Function

This function is used to declare a new job using the Data Pump API. The handle that is returned is used as a parameter for calls to all other procedures except ATTACH.

Syntax

```
DBMS_DATAPUMP.OPEN (
   operation    IN VARCHAR2,
   mode         IN VARCHAR2,
   remote_link  IN VARCHAR2 DEFAULT NULL,
   job_name     IN VARCHAR2 DEFAULT NULL,
   version      IN VARCHAR2 DEFAULT 'COMPATIBLE'
   compression  IN NUMBER DEFAULT KU$_COMPRESS_METADATA)
 RETURN NUMBER;
```

Parameters

Table 27–16 OPEN Function Parameters

Parameter	Meaning
operation	The type of operation to be performed. Table 27–17 contains descriptions of valid operation types.
mode	The scope of the operation to be performed. Table 27–18 contains descriptions of valid modes. Specifying NULL generates an error.
remote_link	If the value of this parameter is non-null, it provides the name of a database link to the remote database that will be the source of data and metadata for the current job.
job_name	The name of the job. The name is limited to 30 characters; it will be truncated if more than 30 characters are used. It may consist of printable characters and spaces. It is implicitly qualified by the schema of the user executing the OPEN procedure and must be unique to that schema (that is, there cannot be other Data Pump jobs using the same name). The name is used to identify the job both within the API and with other database components such as identifying the job in the DBA_RESUMABLE view if the job becomes suspended through lack of resources. If no name is supplied, a system generated name will be provided for the job in the following format: "SYS_<OPERATION>_<MODE>_%N". The default job name is formed where %N expands to a two-digit incrementing integer starting at '01' (for example, "SYS_IMPORT_FULL_03"). The name supplied for the job will also be used to name the master table and other resources associated with the job.
version	The version of database objects to be extracted. This option is only valid for Export, network Import, and SQL_FILE operations. Database objects or attributes that are incompatible with the version will not be extracted. Legal values for this parameter are as follows: ■ COMPATIBLE - (default) the version of the metadata corresponds to the database compatibility level and the compatibility release level for feature (as given in the V$COMPATIBILITY view). Database compatibility must be set to 9.2 or higher. ■ LATEST - the version of the metadata corresponds to the database version. ■ A specific database version, for example, '10.0.0'. In Oracle Database10*g*, this value cannot be lower than 10.0.0.

Table 27–16 (Cont.) OPEN Function Parameters

Parameter	Meaning
compression	The type of compression to use for an export job. The supported compression types are ku$_compress_metadata (which is the default) and ku$_compress_none.

Table 27–17 describes the valid operation types for the OPEN Function.

Table 27–17 Valid Operation Types for the OPEN Function

Operation	Description
EXPORT	Saves data and metadata to a dump file set or obtains an estimate of the size of the data for an operation.
IMPORT	Restores data and metadata from a dump file set or across a database link.
SQL_FILE	Displays the metadata within a dump file set, or from across a network link, as a SQL script. The location of the SQL script is specified through the ADD_FILE procedure.

Table 27–18 describes the valid modes for the OPEN procedure.

Table 27–18 Valid Modes for the OPEN Function

Mode	Description
FULL	Operates on the full database or full dump file set except for the SYS, XDB, ORDSYS, MDSYS, CTXSYS, ORDPLUGINS, and LBACSYS schemas.
SCHEMA	Operates on a set of selected schemas. Defaults to the schema of the current user. All objects in the selected schemas are processed. Users cannot specify SYS, XDB, ORDSYS, MDSYS, CTXSYS, ORDPLUGINS, or LBACSYS schemas for this mode.
TABLE	Operates on a set of selected tables. Defaults to all of the tables in the current user's schema. Only tables and their dependent objects are processed.
TABLESPACE	Operates on a set of selected tablespaces. No defaulting is performed. Tables that have storage in the specified tablespaces are processed in the same manner as in Table mode.
TRANSPORTABLE	Operates on metadata for tables (and their dependent objects) within a set of selected tablespaces to perform a transportable tablespace export/import.

Return Values

- An opaque handle for the job. This handle is used as input to the following procedures: ADD_FILE, DATA_FILTER, DETACH, GET_STATUS, LOG_ENTRY, METADATA_FILTER, METADATA_REMAP, METADATA_TRANSFORM, SET_PARALLEL, SET_PARAMETER, START_JOB, STOP_JOB, and WAIT_FOR_JOB

Exceptions

- INVALID_ARGVAL. An invalid operation or mode was specified. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.

- JOB_EXISTS. A table already exists with the specified job name.
- PRIVILEGE_ERROR. The user does not have the necessary privileges or roles to use the specified mode.
- INTERNAL_ERROR. The job was created under the wrong schema or the master table was of the wrong format.
- SUCCESS_WITH_INFO. The procedure succeeded, but further information is available through the GET_STATUS procedure.
- NO_SUCH_JOB. The specified job does not exist.

Usage Notes

- When the job is created, a master table is created for the job under the caller's schema within the caller's default tablespace. A handle referencing the job is returned that attaches the current session to the job. Once attached, the handle remains valid until either an explicit or implicit detach occurs. The handle is only valid in the caller's session. Other handles can be attached to the same job from a different session by using the ATTACH procedure.
- If the OPEN fails, call GET_STATUS with a null handle to retrieve additional information about the failure.

SET_PARALLEL Procedure

This procedure adjusts the degree of parallelism within a job.

Syntax

```
DBMS_DATAPUMP.SET_PARALLEL(
   handle      IN NUMBER,
   degree      IN NUMBER);
```

Parameters

Table 27–19 SET_PARALLEL Procedure Parameters

Parameter	Description
handle	The handle of a job. The current session must have previously attached to the handle through an OPEN or ATTACH call.
degree	The maximum number of worker processes that can be used for the job. You use this parameter to adjust the amount of resources used for a job.

Exceptions

- INVALID_HANDLE. The specified handle is not attached to a Data Pump job.
- INVALID_OPERATION. The SET_PARALLEL procedure is only valid for export and import operations.
- INVALID_ARGVAL. An invalid value was supplied for an input parameter.
- SUCCESS_WITH_INFO. The procedure succeeded, but further information is available through the GET_STATUS procedure.
- NO_SUCH_JOB. The specified job does not exist.

Usage Notes

- The SET_PARALLEL procedure is only available in the Enterprise Edition of the Oracle database.
- The SET_PARALLEL procedure can be executed by any session attached to a job. The job must be in one of the following states: Defining, Idling, or Executing.
- The effect of decreasing the degree of parallelism may be delayed because ongoing work needs to find an orderly completion point before SET_PARALLEL can take effect.
- Decreasing the parallelism will not result in fewer worker processes associated with the job. It will only decrease the number of worker processes that will be executing at any given time.
- Increasing the parallelism will take effect immediately if there is work that can be performed in parallel.
- The degree of parallelism requested by a user may be decreased based upon settings in the resource manager or through limitations introduced by the PROCESSES or SESSIONS initialization parameters in the init.ora file.
- To parallelize an Export job to a degree of *n*, the user should supply *n* files in the dump file set or specify a substitution variable in a file specification. Otherwise, some of the worker processes will be idle while waiting for files.

- SQL_FILE operations always operate with a degree of 1. Jobs running in the Transportable mode always operate with a degree of 1.

SET_PARAMETER Procedures

This procedure is used to specify job-processing options.

Syntax

```
DBMS_DATAPUMP.SET_PARAMETER(
   handle       IN NUMBER,
   name         IN VARCHAR2,
   value        IN VARCHAR2);

DBMS_DATAPUMP.SET_PARAMETER (
   handle      IN NUMBER,
   name        IN VARCHAR2,
   value       IN NUMBER);
```

Parameters

Table 27–20 SET_PARAMETER Procedure Parameters

Parameter	Description
handle	The handle of a job. The current session must have previously attached to the handle through an OPEN call.
name	The name of the parameter. Table 27–21 describes the valid parameter names.
value	The value for the specified parameter.

Table 27–21 describes the valid options for the name parameter of the SET_PARAMETER procedure.

Table 27–21 Valid Options for the name Parameter in the SET_PARAMETER Procedure

Parameter Name	Datatype	Supported Operations	Meaning
CLIENT_COMMAND	Text	All	An opaque string used to describe the current operation from the client's perspective. The command-line procedures will use this string to store the original command used to invoke the job.
ESTIMATE	Text	Export and Import	Specifies that the estimate method for the size of the tables should be performed before starting the job. If BLOCKS, a size estimate for the user tables is calculated using the count of blocks allocated to the user tables. If STATISTICS, a size estimate for the user tables is calculated using the statistics associated with each table. If no statistics are available for a table, the size of the table is estimated using BLOCKS. The ESTIMATE parameter cannot be used in Transportable Tablespace mode. Default=BLOCKS.
ESTIMATE_ONLY	Number	Export	Specifies that only the estimation portion of an export job should be performed. This option is useful for estimating the size of dump files when the size of the export is unknown.

Table 27–21 (Cont.) Valid Options for the name Parameter in the SET_PARAMETER Procedure

Parameter Name	Datatype	Supported Operations	Meaning
FLASHBACK_SCN	NUMBER	Export and network Import	System change number (SCN) to serve as transactionally consistent point for reading user data. If neither FLASHBACK_SCN nor FLASHBACK_TIME is specified, there will be no transactional consistency between partitions, except for logical standby databases and Streams targets. FLASHBACK_SCN is not supported in Transportable mode.
FLASHBACK_ TIME	Text	Export and network Import	Either the date and time used to determine a consistent point for reading user data or a string of the form TO_ TIMESTAMP(...).
			If neither FLASHBACK_SCN nor FLASHBACK_TIME is specified, there will be no transactional consistency between partitions.
			FLASHBACK_SCN and FLASHBACK_TIME cannot both be specified for the same job. FLASHBACK_TIME is not supported in Transportable mode.
INCLUDE_ METADATA	NUMBER	Export and Import	If nonzero, metadata for objects will be moved in addition to user table data.
			If zero, metadata for objects will not moved. This parameter converts an Export operation into an unload of user data and an Import operation into a load of user data.
			INCLUDE_METADATA is not supported in Transportable mode.
			Default=1.
REUSE_ DATAFILES	NUMBER	Import	If nonzero, tablespace data files can be created using preexisting data files for tablespace creations. REUSE_ DATAFILES is only supported in Full mode.
			Default=0
SKIP_ UNUSABLE_ INDEXES	NUMBER	Import	If nonzero, rows will be inserted into tables having unusable indexes. SKIP_UNUSABLE_INDEXES is not supported in Transportable mode.
			Default=1

Table 27–21 (Cont.) Valid Options for the name Parameter in the SET_PARAMETER Procedure

Parameter Name	Datatype	Supported Operations	Meaning
TABLE_EXISTS_ACTION	Text	Import	Specifies the action to be performed when data is loaded into a preexisting table. The possible actions are: TRUNCATE, REPLACE, APPEND, and SKIP. If INCLUDE_METADATA=0, only TRUNCATE and APPEND are supported. If TRUNCATE, rows are removed from a preexisting table before inserting rows from the Import. Note that if TRUNCATE is specified on tables referenced by foreign key constraints, the TRUNCATE will be modified into a REPLACE. If REPLACE, preexisting tables are replaced with new definitions. Before creating the new table, the old table is dropped. If APPEND, new rows are added to the existing rows in the table. If SKIP, the preexisting table is left unchanged. TABLE_EXISTS_ACTION is not supported in Transportable mode. The default is SKIP if metadata is included in the import. The default is APPEND if INCLUDE_METADATA is set to 0.
TABLESPACE_DATAFILE	Text	Import	Specifies the full file specification for a datafile in the transportable tablespace set. TABLESPACE_DATAFILE is only valid for transportable mode imports. TABLESPACE_DATAFILE can be specified multiple times, but the value specified for each occurrence must be different.
TTS_FULL_CHECK	NUMBER	Export	If nonzero, verifies that a transportable tablespace set has no dependencies (specifically, IN pointers) on objects outside the set, and vice versa. Only valid for Transportable mode Exports. Default=0.
USER_METADATA	NUMBER	Export and network Import	For schema-mode operations, specifies that the metadata to re-create the users' schemas (for example, privilege grants to the exported schemas) should also be part of the operation if set to nonzero. Users must be privileged to explicitly set this parameter. The USER_METADATA parameter cannot be used in Table, Tablespace, or Transportable Tablespace mode. Default=1 if user has EXP_FULL_DATABASE role; 0 otherwise.

Exceptions

- INVALID_HANDLE. The specified handle is not attached to a Data Pump job.
- INVALID_ARGVAL. This exception could be due to any of the following causes:
  - An invalid name was supplied for an input parameter
  - The wrong datatype was used for value
  - A value was not supplied
  - The supplied value was not allowed for the specified parameter name

  - A flashback parameter had been established after a different flashback parameter had already been established
  - A parameter was specified that did not support duplicate definitions
- INVALID_OPERATION. The operation specified is invalid in this context.
- INVALID_STATE. The specified job is not in the Defining state.
- INCONSISTENT_ARGS. Either the specified parameter is not supported for the current operation type or it is not supported for the current mode.
- PRIVILEGE_ERROR. The user does not have the EXP_FULL_DATABASE or IMP_FULL_DATABASE role required for the specified parameter.
- SUCCESS_WITH_INFO. The procedure succeeded, but further information is available through the GET_STATUS procedure.
- NO_SUCH_JOB. The specified job does not exist.

Usage Notes

- The SET_PARAMETER procedure is used to specify optional features for the current job. See Table 27–21 for a list of supported options.

START_JOB Procedure

This procedure begins or resumes execution of a job.

Syntax

```
DBMS_DATAPUMP.START_JOB (
   handle       IN NUMBER,
   skip_current IN NUMBER DEFAULT 0);
```

Parameters

Table 27–22 START_JOB Procedure Parameters

Parameter	Description
handle	The handle of a job. The current session must have previously attached to the handle through either the OPEN or ATTACH procedure.
skip_ current	If nonzero, causes actions that were 'in progress' on a previous execution of the job to be skipped when the job restarts. The skip will only be honored for Import jobs. This mechanism allows the user to skip actions that trigger fatal bugs and cause the premature termination of a job. Multiple actions can be skipped on a restart. The log file will identify which actions are skipped. If a domain index was being processed, all pieces of the domain index are skipped even if the error occurred in only a subcomponent of the domain index. A description of the actions skipped is entered into the log file. skip_ current is ignored for the initial START_JOB in a job. If zero, no data or metadata is lost upon a restart.

Exceptions

- INVALID_HANDLE. The specified handle is not attached to a Data Pump job.
- INVALID_STATE. The causes of this exception can be any of the following:
  - No files have been defined for an Export, non-network Import, or SQL_FILE job
  - An ADD_FILE procedure has not been called to define the output for a SQL_ FILE job
  - A TABLESPACE_DATAFILE parameter has not been defined for a Transportable Import job
  - A TABLESPACE_EXPR metadata filter has not been defined for a Transportable or Tablespace mode Export or Network job
  - The dump file set on an Import of SQL_FILE job was either incomplete or missing a master table specification
- INVALID_OPERATION. Unable to restore master table from a dump file set.
- INTERNAL_ERROR. An inconsistency was detected when the job was started. Additional information may be available through the GET_STATUS procedure.
- SUCCESS_WITH_INFO. The procedure succeeded, but further information is available through the GET_STATUS procedure.
- NO_SUCH_JOB. The specified job does not exist.

Usage Notes

- When this procedure is called to request that the corresponding job be started or restarted, the state of the job is changed from either the Defining or Idling state to the Executing state.
- If the SET_PARALLEL procedure was not called prior to the START_JOB procedure, the initial level of parallelism used in the job will be 1. If SET_PARALLEL was called prior to the job starting, the degree specified by the last SET_PARALLEL call determines the parallelism for the job. On restarts, the parallelism is determined by the previous parallel setting for the job, unless it is overridden by another SET_PARALLEL call.
- To restart a stopped job, an ATTACH must be performed prior to executing the START_JOB procedure.

STOP_JOB Procedure

This procedure terminates a job, but optionally, preserves the state of the job.

Syntax

```
DBMS_DATAPUMP.STOP_JOB (
   handle      IN NUMBER,
   immediate   IN NUMBER DEFAULT 0,
   keep_master IN NUMBER DEFAULT NULL,
   delay       IN NUMBER DEFAULT 0);
```

Parameters

Table 27–23 STOP_JOB Procedure Parameters

Parameter	Description
handle	The handle of a job. The current session must have previously attached to the handle through an OPEN or ATTACH call. At the end of the procedure, the user is detached from the handle.
immediate	If nonzero, the worker processes are aborted immediately. This halts the job quickly, but parts of the job will have to be rerun if the job is ever restarted. If zero, the worker processes are allowed to complete their current work item (either metadata or table data) before they are terminated. The job is placed in a Stop Pending state while the workers finish their current work.
keep_master	If nonzero, the master table is retained when the job is stopped. If zero, the master table is dropped when the job is stopped. If the master table is dropped, the job will not be restartable. If the master table is dropped during an export job, the created dump files are deleted.
delay	The number of seconds to wait until other attached sessions are forcibly detached. The delay allows other sessions attached to the job to be notified that a stop has been performed. The job keeps running until either all clients have detached or the delay has been satisfied. If no delay is specified, then the default delay is 60 seconds. If a shorter delay is used, clients might not be able to retrieve the final messages for the job through the GET_STATUS procedure.

Exceptions

- INVALID_HANDLE. The specified handle is not attached to a Data Pump job.
- INVALID STATE. The job is already in the process of being stopped or completed.
- SUCCESS_WITH_INFO. The procedure succeeded, but further information is available through the GET_STATUS procedure.
- NO_SUCH_JOB. The specified job does not exist.

Usage Notes

- This procedure is used to request that the corresponding job stop executing.
- The termination of a job that is in an Executing state may take several minutes to complete in an orderly fashion.
- For jobs in the Defining, Idling, or Completing states, this procedure is functionally equivalent to the DETACH procedure.

- Once a job is stopped, it can be restarted using the `ATTACH` and `START_JOB` procedures, provided the master table and the dump file set are left intact.
- If the `KEEP_MASTER` parameter is not specified, and the job is in the Defining state or has a mode of Transportable, the master table is dropped. Otherwise, the master table is retained.

WAIT_FOR_JOB Procedure

This procedure runs a job until it either completes normally or stops for some other reason.

Syntax

```
DBMS_DATAPUMP.WAIT_FOR_JOB (
  handle      IN   NUMBER,
  job_state   OUT  VARCHAR2);
```

Parameters

Table 27–24 WAIT_FOR_JOB Procedure Parameters

Parameter	Description
handle	The handle of the job. The current session must have previously attached to the handle through an OPEN or ATTACH call. At the end of the procedure, the user is detached from the handle.
job_state	The state of the job when it has stopped executing. This will be either Stopped or Completed.

Exceptions

- SUCCESS_WITH_INFO. The procedure succeeded, but further information is available through the GET_STATUS API.
- INVALID_HANDLE. The job handle is no longer valid.

Usage Notes

This procedure provides the simplest mechanism for waiting for the completion of a Data Pump job. The job should be started before calling WAIT_FOR_JOB. When WAIT_FOR_JOB returns, the job will no longer be executing. If the job completed normally, the final status will be Completed. If the job stopped executing because of a STOP_JOB request or an internal error, the final status will be Stopped.

28

DBMS_DB_VERSION

The DBMS_DB_VERSION package specifies the Oracle version numbers and other information useful for simple conditional compilation selections based on Oracle versions.

> **See Also:** *PL/SQL Users Guide and Reference* regarding conditional compilation

This package contains the following topics

- Using DBMS_DB_VERSION
  - Overview
  - Constants
  - Examples

Using DBMS_DB_VERSION

Overview

The DBMS_DB_VERSION package specifies the Oracle version numbers and other information useful for simple conditional compilation selections based on Oracle versions.

The package for the Oracle Database 10g Release 2 version is shown below.

```
PACKAGE DBMS_DB_VERSION IS
   VERSION CONSTANT PLS_INTEGER := 10; -- RDBMS version number
   RELEASE CONSTANT PLS_INTEGER := 2;  -- RDBMS release number
   ver_le_9_1    CONSTANT BOOLEAN := FALSE;
   ver_le_9_2    CONSTANT BOOLEAN := FALSE;
   ver_le_9      CONSTANT BOOLEAN := FALSE;
   ver_le_10_1   CONSTANT BOOLEAN := FALSE;
   ver_le_10_2   CONSTANT BOOLEAN := TRUE;
   ver_le_10     CONSTANT BOOLEAN := TRUE;
END DBMS_DB_VERSION;
```

The boolean constants follow a naming convention. Each constant gives a name for a boolean expression. For example:

- VER_LE_9_1 represents version <= 9 and release <= 1
- VER_LE_10_2 represents version <= 10 and release <= 2
- VER_LE_10 represents version <= 10

A typical usage of these boolean constants is:

```
$IF DBMS_DB_VERSION.VER_LE_10 $THEN
   version 10 and earlier code
$ELSIF DBMS_DB_VERSION.VER_LE_11 $THEN
   version 11 code
$ELSE
   version 12 and later code
$END
```

This code structure will protect any reference to the code for hypothetical version 12. It also prevents the controlling package constant DBMS_DB_VERSION.VER_LE_11 from being referenced when the program is compiled under version 10. A similar observation applies to version 11. This scheme works even though the static constant VER_LE_11 is not defined in version 10 database because conditional compilation protects the $ELSIF from evaluation if DBMS_DB_VERSION.VER_LE_10 is TRUE.

Constants

The DBMS_DB_VERSION package contains different constants for different Oracle Database releases. The Oracle Database 10g Release 2 version of the DBMS_DB_VERSION package uses the constants shown in Table 28–1.

Table 28–1 DBMS_DB_VERSION Constants

Name	Type	Value	Description
VERSION	PLS_INTEGER	10	Current version
RELEASE	PLS_INTEGER	2	Current release
VER_LE_9	BOOLEAN	FALSE	Version <= 9
VER_LE_9_1	BOOLEAN	FALSE	Version <= 9 and release <= 1
VER_LE_9_2	BOOLEAN	FALSE	Version <= 9 and release <= 2
VER_LE_10	BOOLEAN	TRUE	Version <= 10
VER_LE_10_1	BOOLEAN	FALSE	Version <= 10 and release <= 1
VER_LE_10_2	BOOLEAN	TRUE	Version <=10 and release <= 2

Examples

This example uses conditional compilation to guard new features.

```
CREATE OR REPLACE PROCEDURE whetstone IS

 -- Notice that conditional compilation constructs
 -- can interrupt a regular PL/SQL statement.
 -- You can locate a conditional compilation directive anywhere
 -- there is whitespace in the regular statement.

 SUBTYPE my_real IS
    $IF DBMS_DB_VERSION.VER_LE_9 $THEN NUMBER
                                 $ELSE BINARY_DOUBLE
    $END;

 t  CONSTANT my_real := $IF DBMS_DB_VERSION.VER_LE_9 $THEN 0.499975
                                                     $ELSE 0.499975d
                        $END;

 t2 CONSTANT my_real := $if DBMS_DB_VERSION.VER_LE_9 $THEN 2.0
                                                     $ELSE 2.0d
                        $END;

 x  CONSTANT my_real := $IF DBMS_DB_VERSION.VER_LE_9 $THEN 1.0
                                                     $ELSE 1.0d
                        $END;

 y  CONSTANT my_real := $IF DBMS_DB_VERSION.VER_LE_9 $THEN 1.0
                                                     $ELSE 1.0d
                        $END;

 z  MY_REAL;

 PROCEDURE P(x IN my_real, y IN my_real, z OUT NOCOPY my_real) IS
   x1 my_real;
   y1 my_real;
 BEGIN
   x1 := x;
   y1 := y;
   x1 := t * (x1 + y1);
   y1 := t * (x1 + y1);
   z := (x1 + y1)/t2;
 END P;
BEGIN
 P(x, y, z);
 DBMS_OUTPUT.PUT_LINE ('z = '|| z);
END whetstone;
/
```

29

DBMS_DDL

This package provides access to some SQL data definition language (DDL) statements from stored procedures. It also provides special administration operations that are not available as Data Definition Language statements (DDLs).

This chapter contains the following topics:

Using DBMS_DDL

This section contains topics which relate to using the `DBMS_DDL` package.

Deprecated Subprograms

Oracle recommends that you do not use deprecated subprograms in new applications. Support for deprecated features is for backward compatibility only

The following subprograms are deprecated with release Release 10gR2:

- ALTER_COMPILE Procedure

Security Model

This package runs with the privileges of the calling user, rather than the package owner `SYS`.

Operational Notes

The `ALTER_COMPILE` procedure commits the current transaction, performs the operation, and then commits again.

Summary of DBMS_DDL Subprograms

Table 29–1 DBMS_DDL Package Subprograms

Subprogram	Description
ALTER_COMPILE Procedure on page 29-7	Compiles the PL/SQL object
ALTER_TABLE_NOT_ REFERENCEABLE Procedure on page 29-8	Reorganizes object tables
ALTER_TABLE_REFERENCEABLE Procedure on page 29-9	Reorganizes object tables
CREATE_WRAPPED Procedures on page 29-10	Takes as input a single `CREATE OR REPLACE` statement that specifies creation of a PL/SQL package specification, package body, function, procedure, type specification or type body, generates a `CREATE OR REPLACE` statement with the PL/SQL source text obfuscated and executes the generated statement
IS_TRIGGER_FIRE_ONCE Function on page 29-12	Returns `TRUE` if the specified DML or DDL trigger is set to fire once. Otherwise, returns `FALSE`
SET_TRIGGER_FIRING_PROPERTY Procedure on page 29-13	Sets the specified DML or DDL trigger's firing property
WRAP Functions on page 29-14	Takes as input a `CREATE OR REPLACE` statement that specifies creation of a PL/SQL package specification, package body, function, procedure, type specification or type body and returns a `CREATE OR REPLACE` statement where the text of the PL/SQL unit has been obfuscated

ALTER_COMPILE Procedure

This procedure is equivalent to the following SQL statement:

```
ALTER PROCEDURE|FUNCTION|PACKAGE [<schema>.] <name> COMPILE [BODY]
```

> **Note:** This procedure is deprecated in Release 10gR2. While the procedure remains available in the package, Oracle recommends using the DDL equivalent in a dynamic SQL statement.

Syntax

```
DBMS_DDL.ALTER_COMPILE (
   type             VARCHAR2,
   schema           VARCHAR2,
   name             VARCHAR2
   reuse_settings   BOOLEAN := FALSE);
```

Parameters

Table 29–2 ALTER_COMPILE Procedure Parameters

Parameter	Description
type	Must be either PROCEDURE, FUNCTION, PACKAGE, PACKAGE BODY or TRIGGER
schema	Schema name If NULL, then use current schema (case-sensitive)
name	Name of the object (case-sensitive)
reuse_settings	Indicates whether the session settings in the objects should be reused, or whether the current session settings should be adopted instead

Exceptions

Table 29–3 ALTER_COMPILE Procedure Exceptions

Exception	Description
ORA-20000:	Insufficient privileges or object does not exist
ORA-20001:	Remote object, cannot compile
ORA-20002:	Bad value for object type: should be either PACKAGE, PACKAGE BODY, PROCEDURE, FUNCTION, or TRIGGER

ALTER_TABLE_NOT_REFERENCEABLE Procedure

This procedure alters the given object table `table_schema.table_name` so it becomes not the default referenceable table for the schema `affected_schema`. This is equivalent to SQL

```
ALTER TABLE [<table_schema>.]<table_name> NOT REFERENCEABLE FOR <affected_schema>
```

which is currently not supported or available as a DDL statement.

Syntax

```
DBMS_DDL.ALTER_TABLE_NOT_REFERENCEABLE (
   table_name        IN           VARCHAR2,
   table_schema      IN  DEFAULT  NULL,
   affected_schema   IN  DEFAULT  NULL);
```

Parameters

Table 29–4 ALTER_TABLE_NOT_REFERENCEABLE Procedure Parameters

Parameter	Description
`table_name`	The name of the table to be altered. Cannot be a synonym. Must not be `NULL`. Case sensitive.
`table_schema`	The name of the schema owning the table to be altered. If `NULL` then the current schema is used. Case sensitive.
`affected_schema`	The name of the schema affected by this alteration. If NULL then the current schema is used. Case sensitive.

Usage Notes

This procedure simply reverts for the affected schema to the default table referenceable for `PUBLIC`; that is., it simply undoes the previous `ALTER_TABLE_REFERENCEABLE` call for this specific schema. The affected schema must a particular schema (cannot be `PUBLIC`).

The user that executes this procedure must own the table (that is, the schema is the same as the user), and the affected schema must be the same as the user.

If the user executing this procedure has `ALTER ANY TABLE` and `SELECT ANY TABLE` and `DROP ANY TABLE` privileges, the user doesn't have to own the table and the affected schema can be any valid schema.

ALTER_TABLE_REFERENCEABLE Procedure

This procedure alters the given object table `table_schema.table_name` so it becomes the referenceable table for the given schema `affected_schema`. This is equivalent to SQL

```
ALTER TABLE [<table_schema>.]<table_name>  REFERENCEABLE FOR <affected_schema>
```

which is currently not supported or available as a DDL statement.

Syntax

```
DBMS_DDL.ALTER_TABLE_REFERENCEABLE
   table_name       IN  VARCHAR2,
   table_schema     IN  DEFAULT  NULL,
   affected_schema  IN  DEFAULT  NULL);
```

Parameters

Table 29–5 ALTER_TABLE_REFERENCEABLE Procedure Parameters

Parameter	Description
`table_name`	The name of the table to be altered. Cannot be a synonym. Must not be `NULL`. Case sensitive.
`table_schema`	The name of the schema owning the table to be altered. If `NULL` then the current schema is used. Case sensitive.
`affected_schema`	The name of the schema affected by this alteration. If NULL then the current schema is used. Case sensitive.

Usage Notes

When you create an object table, it automatically becomes referenceable, unless you use the `OID AS` clause when creating the table. The `OID AS` clause makes it possible for you to create an object table and to assign to the new table the same EOID as another object table of the same type. After you create a new table using the `OID AS` clause, you end up with two object table with the same `EOID`; the new table is not referenceable, the original one is. All references that used to point to the objects in the original table still reference the same objects in the same original table.

If you execute this procedure on the new table, it will make the new table the referenceable table replacing the original one; thus, those references now point to the objects in the new table instead of the original table.

CREATE_WRAPPED Procedures

The procedure takes as input a single CREATE OR REPLACE statement that specifies creation of a PL/SQL package specification, package body, function, procedure, type specification or type body. It then generates a CREATE OR REPLACE statement with the PL/SQL source text obfuscated and executes the generated statement. In effect, this procedure bundles together the operations of wrapping the text and creating the PL/SQL unit.

See Also: WRAP Functions on page 29-14

This procedure has 3 overloads. Each of the three functions provides better performance than using a combination of individual WRAP Functions and DBMS_SQL.PARSE (or EXECUTE IMMEDIATE) calls. The different functionality of each form of syntax is presented with the definition.

Syntax

Is a shortcut for EXECUTE IMMEDIATE SYS.DBMS_DDL.WRAP(*ddl*):

```
DBMS_DDL.CREATE_WRAPPED (
   ddl     VARCHAR2);
```

Is a shortcut for DBMS_SQL.PARSE(cursor, SYS.DBMS_DDL.WRAP (input, lb, ub)):

```
DBMS_DDL.CREATE_WRAPPED(
   ddl     DBMS_SQL.VARCHAR2A,
   lb      PLS_INTEGER,
   ub      PLS_INTEGER);
```

Is a shortcut for DBMS_SQL.PARSE(cursor, SYS.DBMS_DDL.WRAP (input, lb, ub)):

```
DBMS_DDL.CREATE_WRAPPED(
   ddl     DBMS_SQL.VARCHAR2S,
   lb      PLS_INTEGER,
   ub      PLS_INTEGER);
```

Parameters

Table 29–6 CREATE_WRAPPED Procedure Parameters

Parameter	Description
ddl	A CREATE OR REPLACE statement that specifies creation of a PL/SQL package specification, package body, function, procedure, type specification or type body
lb	Lower bound for indices in the string table that specify the CREATE OR REPLACE statement
ub	Upper bound for indices in the string table that specify the CREATE OR REPLACE statement.

Usage Notes

- The CREATE OR REPLACE statement is executed with the privileges of the user invoking DBMS_DDL.CREATE_WRAPPED.
- Any PL/SQL code that attempts to call these interfaces should use the fully qualified package name SYS.DBMS_DDL to avoid the possibility that the name

DBMS_DDL is captured by a locally-defined unit or by redefining the DBMS_DDL public synonym.

- Each invocation of any accepts only a single PL/SQL unit. By contrast, the PL/SQL wrap utility accepts a entire SQL*Plus file and obfuscates the PL/SQL units within the file leaving all other text as-is. These interfaces are intended to be used in conjunction with or as a replacement for PL/SQL's dynamic SQL interfaces (EXECUTE IMMEDIATE and DBMS_SQL.PARSE). Since these dynamic SQL interfaces only accept a single unit at a time (and do not understand the SQL*Plus "/" termination character), both the CREATE_WRAPPED Procedures and the WRAP Functions require input to be a single unit.

Exceptions

ORA-24230: If the input is not a CREATE OR REPLACE statement specifying a PL/SQL unit, exception DBMS_DDL.MALFORMED_WRAP_INPUT is raised.

Examples

```
DECLARE
    ddl VARCHAR2(32767);
BEGIN
    ddl := GENERATE_PACKAGE(...);
    SYS.DBMS_DDL.CREATE_WRAPPED(ddl); -- Instead of EXECUTE IMMEDIATE ddl
END;
```

IS_TRIGGER_FIRE_ONCE Function

This function returns `TRUE` if the specified DML or DDL trigger is set to fire once. Otherwise, it returns `FALSE`.

A fire once trigger fires in a user session but does not fire in the following cases:

- For changes made by a Streams apply process
- For changes made by executing one or more Streams apply errors using the `EXECUTE_ERROR` or `EXECUTE_ALL_ERRORS` procedure in the `DBMS_APPLY_ADM` package

---

Note: Only DML and DDL triggers can be fire once. All other types of triggers always fire.

See Also: "SET_TRIGGER_FIRING_PROPERTY Procedure" on page 29-13

Syntax

```
DBMS_DDL.IS_TRIGGER_FIRE_ONCE
   trig_owner          IN  VARCHAR2,
   trig_name           IN  VARCHAR2)
 RETURN BOOLEAN;
```

Parameters

Table 29–7 IS_TRIGGER_FIRE_ONCE Function Parameters

Parameter	Description
`trig_owner`	Schema of trigger
`trig_name`	Name of trigger

SET_TRIGGER_FIRING_PROPERTY Procedure

This procedure sets the specified DML or DDL trigger's firing property. Use this procedure to control a DML or DDL trigger's firing property for changes:

- Applied by a Streams apply process
- Made by executing one or more Streams apply errors using the `EXECUTE_ERROR` or `EXECUTE_ALL_ERRORS` procedure in the `DBMS_APPLY_ADM` package.

Syntax

```
DBMS_DDL.SET_TRIGGER_FIRING_PROPERTY
   trig_owner         IN  VARCHAR2,
   trig_name          IN  VARCHAR2,
   fire_once          IN  BOOLEAN);
```

Parameters

Table 29–8 SET_TRIGGER_FIRING_PROPERTY Procedure Parameters

Parameter	Description
`trig_owner`	Schema of the trigger to set
`trig_name`	Name of the trigger to set
`fire_once`	If `TRUE`, the trigger is set to fire once. By default, the `fire_once` parameter is set to `TRUE` for DML and DDL triggers. If `FALSE`, the trigger is set to always fire.

Usage Notes

You can specify one of the following settings for a trigger's firing property:

- If the `fire_once` parameter is set to `TRUE` for a trigger, then the trigger does not fire for these types of changes.
- If the `fire_once` parameter is set to `FALSE` for a trigger, then the trigger fires for these types of changes.

Regardless of the firing property set by this procedure, a trigger continues to fire when changes are made by means other than the apply process or apply error execution. For example, if a user session or an application makes a change, then the trigger continues to fire, regardless of the firing property.

Note:

- If you dequeue an error transaction from the error queue and execute it without using the `DBMS_APPLY_ADM` package, then relevant changes resulting from this execution cause a trigger to fire, regardless of the trigger firing property.
- Only DML and DDL triggers can be fire once. All other types of triggers always fire.

See Also: *Oracle Streams Concepts and Administration* for more information about the apply process and controlling a trigger's firing property

WRAP Functions

This function takes as input a single CREATE OR REPLACE statement that specifies creation of a PL/SQL package specification, package body, function, procedure, type specification or type body and returns a CREATE OR REPLACE statement where the text of the PL/SQL unit has been obfuscated.

The function has 3 overloads to allow for the different ways in which DDL statements can be generated dynamically and presented to DBMS_SQL or EXECUTE IMMEDIATE. The different functionality of each form of syntax is presented with the definition.

See Also: CREATE_WRAPPED Procedures on page 29-10

Syntax

Provides basic functionality:

```
DBMS_DDL.WRAP(
   ddl      VARCHAR2)
  RETURN VARCHAR2;
```

Provides the same functionality as the first form, but allows for larger inputs. This function is intended to be used with the PARSE Procedure in the DBMS_SQL package and its argument list follows the convention of DBMS_SQL.PARSE:

```
DBMS_DDL.WRAP(
   ddl      DBMS_SQL.VARCHAR2S,
   lb       PLS_INTEGER,
   ub       PLS_INTEGER)
  RETURN DBMS_SQL.VARCHAR2S;
```

Provides the same functionality as the second form and is provided for compatibility with multiple forms of the PARSE Procedure in the DBMS_SQL package:

```
DBMS_DDL.WRAP(
   ddl      DBMS_SQL.VARCHAR2A,
   lb       PLS_INTEGER,
   ub       PLS_INTEGER)
  RETURN DBMS_SQL.VARCHAR2A;
```

Parameters

Table 29–9 WRAP Function Parameters

Parameter	Description
ddl	A CREATE OR REPLACE statement that specifies creation of a PL/SQL package specification, package body, function, procedure, type specification or type body
lb	Lower bound for indices in the string table that specify the CREATE OR REPLACE statement
ub	Upper bound for indices in the string table that specify the CREATE OR REPLACE statement.

Return Values

A CREATE OR REPLACE statement with the text obfuscated. In the case of the second and third form, the return value is a table of strings that need to be concatenated in order to construct the CREATE OR REPLACE string containing obfuscated source text.

Usage Notes

- Any PL/SQL code that attempts to call these interfaces should use the fully qualified package name SYS.DBMS_DDL to avoid the possibility that the name DBMS_DDL is captured by a locally-defined unit or by redefining the DBMS_DDL public synonym.
- Each invocation of any accepts only a single PL/SQL unit. By contrast, the PL/SQL wrap utility accepts a full SQL file and obfuscates the PL/SQL units within the file leaving all other text as-is. These interfaces are intended to be used in conjunction with or as a replacement for PL/SQL's dynamic SQL interfaces (EXECUTE IMMEDIATE and DBMS_SQL.PARSE). Since these dynamic SQL interfaces only accept a single unit at a time (and do not understand the SQL*Plus "/" termination character), both the CREATE_WRAPPED Procedures and the WRAP Functions require input to be a single unit.

Exceptions

ORA-24230: If the input is not a CREATE OR REPLACE statement specifying a PL/SQL unit, exception DBMS_DDL.MALFORMED_WRAP_INPUT is raised.

Examples

```
DECLARE
   ddl VARCHAR2(32767);
BEGIN
   ddl := GENERATE_PACKAGE(...);
EXECUTE IMMEDIATE SYS.DBMS_DDL.WRAP(ddl); -- Instead of EXECUTE IMMEDIATE ddl
END;
```

30

DBMS_DEBUG

DBMS_DEBUG is a PL/SQL interface to the PL/SQL debugger layer, Probe, in the Oracle server.

This API is primarily intended to implement server-side debuggers and it provides a way to debug server-side PL/SQL program units.

> **Note:** The term *program unit* refers to a PL/SQL program of any type (procedure, function, package, package body, trigger, anonymous block, object type, or object type body).

This chapter contains the following topics:

Using DBMS_DEBUG

Overview

To debug server-side code, you must have two database sessions: one session to run the code in debug mode (the target session), and a second session to supervise the target session (the debug session).

The target session becomes available for debugging by making initializing calls with DBMS_DEBUG. This marks the session so that the PL/SQL interpreter runs in debug mode and generates debug events. As debug events are generated, they are posted from the session. In most cases, debug events require return notification: the interpreter pauses awaiting a reply.

Meanwhile, the debug session must also initialize itself using DBMS_DEBUG: This tells it which target session to supervise. The debug session may then call entry points in DBMS_DEBUG to read events that were posted from the target session and to communicate with the target session.

The following subprograms are run in the target session (the session that is to be debugged):

- SYNCHRONIZE Function
- DEBUG_ON Procedure
- DEBUG_OFF Procedure

DBMS_DEBUG does not provide an interface to the PL/SQL compiler, but it does depend on debug information optionally generated by the compiler. Without debug information, it is not possible to examine or modify the values of parameters or variables.

Constants

A breakpoint status may have the following value:

- `breakpoint_status_unused`—breakpoint is not in use

Otherwise, the status is a mask of the following values:

- `breakpoint_status_active`—a line breakpoint
- `breakpoint_status_disabled`—breakpoint is currently disabled
- `breakpoint_status_remote`—a shadow breakpoint (a local representation of a remote breakpoint)

Variables

The DBMS_DEBUG uses the variables shown in Table 30–1.

Table 30–1 DBMS_DEBUG Variables

Variable	Description
default_timeout	The timeout value (used by both sessions).The smallest possible timeout is 1 second. If this value is set to 0, then a large value (3600) is used.

Exceptions

These values are returned by the various functions called in the debug session (SYNCHRONIZE, CONTINUE, SET_BREAKPOINT, and so on). If PL/SQL exceptions worked across client/server and server/server boundaries, then these would all be exceptions rather than error codes.

Value	Description
success	Normal termination.

Statuses returned by GET_VALUE and SET_VALUE:

Status	Description
error_bogus_frame	No such entrypoint on the stack.
error_no_debug_info	Program was compiled without debug symbols.
error_no_such_object	No such variable or parameter.
error_unknown_type	Debug information is unreadable.
error_indexed_table	Returned by GET_VALUE if the object is a table, but no index was provided.
error_illegal_index	No such element exists in the collection.
error_nullcollection	Table is atomically null.
error_nullvalue	Value is null.

Statuses returned by SET_VALUE:

Status	Description
error_illegal_value	Constraint violation.
error_illegal_null	Constraint violation.
error_value_malformed	Unable to decipher the given value.
error_other	Some other error.
error_name_incomplete	Name did not resolve to a scalar.

Statuses returned by the breakpoint functions:

Status	Description
error_no_such_breakpt	No such breakpoint.
error_idle_breakpt	Cannot enable or disable an unused breakpoint.
error_bad_handle	Unable to set breakpoint in given program (nonexistent or security violation).

General error codes (returned by many of the DBMS_DEBUG subprograms):

Status	Description
error_unimplemented	Functionality is not yet implemented.
error_deferred	No program running; operation deferred.
error_exception	An exception was raised in the DBMS_DEBUG or Probe packages on the server.
error_communication	Some error other than a timeout occurred.
error_timeout	Timout occurred.

Exception	Description
illegal_init	DEBUG_ON was called prior to INITIALIZE.

The following exceptions are raised by procedure SELF_CHECK:

Exception	Description
pipe_creation_failure	Could not create a pipe.
pipe_send_failure	Could not write data to the pipe.
pipe_receive_failure	Could not read data from the pipe.
pipe_datatype_mismatch	Datatype in the pipe was wrong.
pipe_data_error	Data got garbled in the pipe.

Operational Notes

There are two ways to ensure that debug information is generated: through a session switch, or through individual recompilation.

To set the session switch, enter the following statement:

```
ALTER SESSION SET PLSQL_DEBUG = true;
```

This instructs the compiler to generate debug information for the remainder of the session. It does not recompile any existing PL/SQL.

To generate debug information for existing PL/SQL code, use one of the following statements (the second recompiles a package or type body):

```
ALTER [PROCEDURE | FUNCTION | PACKAGE | TRIGGER | TYPE] <name> COMPILE DEBUG;
ALTER [PACKAGE | TYPE] <name> COMPILE DEBUG BODY;
```

Figure 30–1 and Figure 30–2 illustrate the flow of operations in the session to be debugged and in the debugging session.

Figure 30–1 Target Session

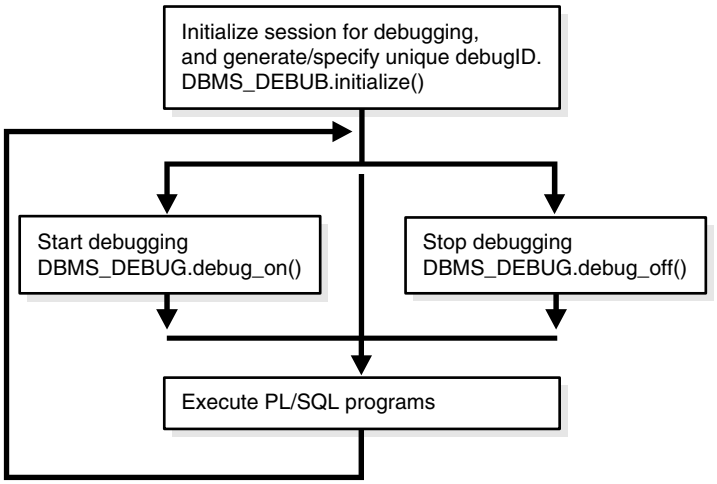


Figure 30–2 Debug Session

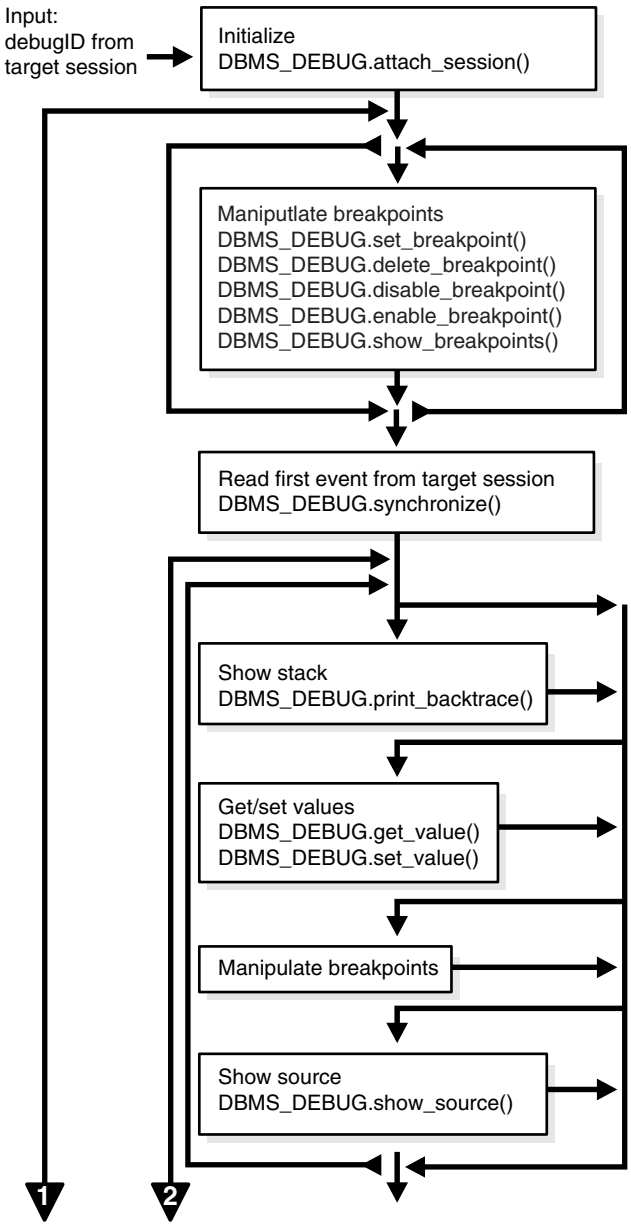
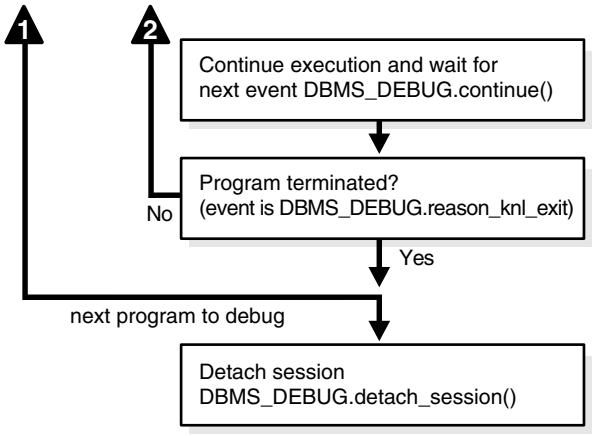


Figure 30–3 Debug Session (Cont.)



Control of the Interpreter

The interpreter pauses execution at the following times:

1. At startup of the interpreter so any deferred breakpoints may be installed prior to execution.
2. At any line containing an enabled breakpoint.
3. At any line where an *interesting* event occurs. The set of interesting events is specified by the flags passed to `DBMS_DEBUG.CONTINUE` in the `breakflags` parameter.

Session Termination

There is no event for session termination. Therefore, it is the responsibility of the debug session to check and make sure that the target session has not ended. A call to `DBMS_DEBUG.SYNCHRONIZE` after the target session has ended causes the debug session to hang until it times out.

Deferred Operations

The diagram suggests that it is possible to set breakpoints prior to having a target session. This is true. In this case, Probe caches the breakpoint request and transmits it to the target session at first synchronization. However, if a breakpoint request is deferred in this fashion, then:

- `SET_BREAKPOINT` does not set the breakpoint number (it can be obtained later from `SHOW_BREAKPOINTS` if necessary).
- `SET_BREAKPOINT` does not validate the breakpoint request. If the requested source line does not exist, then an error silently occurs at synchronization, and no breakpoint is set.

Diagnostic Output

To debug Probe, there are *diagnostics* parameters to some of the calls in `DBMS_DEBUG`. These parameters specify whether to place diagnostic output in the RDBMS tracefile. If output to the RDBMS tracefile is disabled, these parameters have no effect.

Common and Debug Session Sections

- Common Section

- Target Session
- Debug Session Section

Common Section

The following subprograms may be called in either the target or the debug session:

- PROBE_VERSION Procedure
- SELF_CHECK Procedure
- SET_TIMEOUT Function

Target Session

The following subprograms may be called only in the target session:

- INITIALIZE Function
- DEBUG_ON Procedure
- SET_TIMEOUT_BEHAVIOUR Procedure
- GET_TIMEOUT_BEHAVIOUR Function

Debug Session Section

The following subprograms should be run in the debug session only:

- ATTACH_SESSION Procedure
- SYNCHRONIZE Function
- SHOW_FRAME_SOURCE Procedure
- SHOW_SOURCE Procedures
- GET_MORE_SOURCE Procedure
- PRINT_BACKTRACE Procedure
- CONTINUE Function
- SET_BREAKPOINT Function
- DELETE_BREAKPOINT Function
- SET_OER_BREAKPOINT Function
- DELETE_OER_BREAKPOINT Function
- ENABLE_BREAKPOINT Function
- DISABLE_BREAKPOINT Function
- SHOW_BREAKPOINTS Procedures
- SET_VALUE Functionn
- GET_VALUE Function
- TARGET_PROGRAM_RUNNING Procedure
- DETACH_SESSION Procedure
- GET_RUNTIME_INFO Function
- PRINT_INSTANTIATIONS Procedure
- PING Procedure
- GET_LINE_MAP Function

- GET_RUNTIME_INFO Function
- GET_INDEXES Function
- EXECUTE Procedure

OER Breakpoints

Exceptions that are declared in PL/SQL programs are known as user-defined exceptions. In addition, there are Oracle Errors (OERs) that are returned from the Oracle kernel. To tie the two mechanisms together, PL/SQL provides the `exception_init` pragma that turns a user-defined exception into an OER, so that a PL/SQL handler may be used for it, and so that the PL/SQL engine can return OERs to the Oracle kernel. As of the current release, the only information available about an OER is its number. If two user-defined exceptions are exception_init'd to the same OER, they are indistinguishable.

Namespaces

Program units on the server reside in different namespaces. When setting a breakpoint, specify the desired namespace.

1. `Namespace_cursor` contains cursors (anonymous blocks).
2. `Namespace_pgkspec_or_toplevel` contains:
   - Package specifications.
   - Procedures and functions that are not nested inside other packages, procedures, or functions.
   - Object types.
3. `Namespace_pkg_body` contains package bodies and type bodies.
4. `Namespace_trigger` contains triggers.

Libunit Types

These values are used to disambiguate among objects in a given namespace. These constants are used in `PROGRAM_INFO` when Probe is giving a stack backtrace.

- `LibunitType_cursor`
- `LibunitType_procedure`
- `LibunitType_function`
- `LibunitType_package`
- `LibunitType_package_body`
- `LibunitType_trigger`
- `LibunitType_Unknown`

Breakflags

These are values to use for the `breakflags` parameter to `CONTINUE`, in order to tell Probe what events are of interest to the client. These flags may be combined.

Value	Description
`break_next_line`	Break at next source line (step over calls).

Value	Description
break_any_call	Break at next source line (step into calls).
break_any_return	Break after returning from current entrypoint (skip over any entrypoints called from the current routine).
break_return	Break the next time an entrypoint gets ready to return. (This includes entrypoints called from the current one. If interpreter is running Proc1, which calls Proc2, then break_return stops at the end of Proc2.)
break_exception	Break when an exception is raised.
break_handler	Break when an exception handler is executed.
abort_execution	Stop execution and force an 'exit' event as soon as DBMS_DEBUG.CONTINUE is called.

Information Flags

These are flags which may be passed as the info_requested parameter to SYNCHRONIZE, CONTINUE, and GET_RUNTIME_INFO.

Flag	Description
info_getStackDepth	Get the current depth of the stack.
info_getBreakpoint	Get the breakpoint number.
info_getLineinfo	Get program unit information.

Reasons for Suspension

After CONTINUE is run, the program either runs to completion or breaks on some line.

Reason	Description
reason_none	-
reason_interpreter_starting	Interpreter is starting.
reason_breakpoint	Hit a breakpoint.
reason_enter	Procedure entry.
reason_return	Procedure is about to return.
reason_finish	Procedure is finished.
reason_line	Reached a new line.
reason_interrupt	An interrupt occurred.
reason_exception	An exception was raised.
reason_exit	Interpreter is exiting (old form).
reason_knl_exit	Kernel is exiting.
reason_handler	Start exception-handler.
reason_timeout	A timeout occurred.
reason_instantiate	Instantiation block.
reason_abort	Interpreter is aborting.

Data Structures

The DBMS_DEBUG package defines RECORD types and TABLE types.

RECORD Types

- BREAKPOINT_INFO Record Type
- PROGRAM_INFO Record Type
- RUNTIME_INFO Record Type

TABLE Types

- BACKTRACE_TABLE Table Type
- BREAKPOINT_TABLE Table Type
- INDEX_TABLE Table Type
- VC2_TABLE Table Type

BREAKPOINT_INFO Record Type

This type gives information about a breakpoint, such as its current status and the program unit in which it was placed.

Syntax

```
TYPE breakpoint_info IS RECORD (
   name        VARCHAR2(30),
   owner       VARCHAR2(30),
   dblink      VARCHAR2(30),
   line#       BINARY_INTEGER,
   libunittype BINARY_INTEGER,
   status      BINARY_INTEGER);
```

Fields

Table 30–2 RUNTIME_INFO Fields

Field	Description
`name`	Name of the program unit
`owner`	Owner of the program unit
`dblink`	Database link, if remote
`line#`	Line number
`libunittype`	`NULL`, unless this is a nested procedure or function
`status`	See Constants on page 30-4 for values of `breakpoint_status_*`

PROGRAM_INFO Record Type

This type specifies a program location. It is a line number in a program unit. This is used for stack backtraces and for setting and examining breakpoints. The read-only fields are currently ignored by Probe for breakpoint operations. They are set by Probe only for stack backtraces.

Syntax

```
TYPE program_info IS RECORD(
    -- The following fields are used when setting a breakpoint
    namespace        BINARY_INTEGER,
    name             VARCHAR2(30),
    owner            VARCHAR2(30),
    dblink           VARCHAR2(30),
    line#            BINARY_INTEGER,
    -- Read-only fields (set by Probe when doing a stack backtrace)
    libunittype      BINARY_INTEGER,
    entrypointname   VARCHAR2(30));
```

Fields

Table 30–3 PROGRAM_INFO Fields

Field	Description
`namespace`	See Namespaces on page 30-12
`name`	Name of the program unit
`owner`	Owner of the program unit
`dblink`	Database link, if remote
`line#`	Line number
`libunittype`	A read-only field, `NULL`, unless this is a nested procedure or function
`entrypointname`	A read-only field, to disambiguate among objects that share the same namespace (for example, procedure and package specifications). See the Libunit Types on page 30-12 for more information.

RUNTIME_INFO Record Type

This type gives context information about the running program.

Syntax

```
TYPE runtime_info IS RECORD(
    line#            BINARY_INTEGER,
    terminated       binary_integer,
    breakpoint       binary_integer,
    stackdepth       BINARY_INTEGER,
    interpreterdepth BINARY_INTEGER,
    reason           BINARY_INTEGER,
    program          program_info);
```

Fields

Table 30–4 RUNTIME_INFO Fields

Field	Description
`line#`	Duplicate of `program.line#`
`terminated`	Whether the program has terminated
`breakpoint`	Breakpoint number
`stackdepth`	Number of frames on the stack
`interpreterdepth`	[A reserved field]
`reason`	Reason for suspension
`program`	Source location

BACKTRACE_TABLE Table Type

This type is used by `PRINT_BACKTRACE`.

Syntax

```
TYPE backtrace_table IS TABLE OF program_info INDEX BY BINARY_INTEGER;
```

BREAKPOINT_TABLE Table Type

This type is used by SHOW_BREAKPOINTS.

Syntax

```
TYPE breakpoint_table IS TABLE OF breakpoint_info INDEX BY BINARY_INTEGER;
```

INDEX_TABLE Table Type

This type is used by `GET_INDEXES` to return the available indexes for an indexed table.

Syntax

```
TYPE index_table IS table of BINARY_INTEGER INDEX BY BINARY_INTEGER;
```

VC2_TABLE Table Type

This type is used by SHOW_SOURCE.

Syntax

```
TYPE vc2_table IS TABLE OF VARCHAR2(90) INDEX BY BINARY_INTEGER;
```

Summary of DBMS_DEBUG Subprograms

Table 30–5 DBMS_DEBUG Package Subprograms

Subprogram	Description
ATTACH_SESSION Procedure on page 30-24	Notifies the debug session about the target debugID
CONTINUE Function on page 30-25	Continues execution of the target program
DEBUG_OFF Procedure on page 26	Turns debug-mode off
DEBUG_ON Procedure on page 30-27	Turns debug-mode on
DELETE_BREAKPOINT Function on page 30-28	Deletes a breakpoint
DELETE_OER_ BREAKPOINT Function on page 30-29	Deletes an OER breakpoint
DETACH_SESSION Procedure on page 30-30	Stops debugging the target program
DISABLE_BREAKPOINT Function on page 30-31	Disables a breakpoint
ENABLE_BREAKPOINT Function on page 30-32	Activates an existing breakpoint
EXECUTE Procedure on page 30-33	Executes SQL or PL/SQL in the target session
GET_INDEXES Function on page 30-35	Returns the set of indexes for an indexed table
GET_MORE_SOURCE Procedure on page 30-36	Provides additional source in the event of buffer overflow when using SHOW_SOURCE
GET_LINE_MAP Function on page 30-37	Returns information about line numbers in a program unit
GET_RUNTIME_INFO Function on page 30-38	Returns information about the current program
GET_TIMEOUT_ BEHAVIOUR Function on page 30-39	Returns the current timeout behavior
GET_VALUE Function on page 30-40	Gets a value from the currently-running program
INITIALIZE Function on page 30-42	Sets debugID in target session
PING Procedure on page 30-44	Pings the target session to prevent it from timing out
PRINT_BACKTRACE Procedure on page 30-45	Prints a stack backtrace
PRINT_INSTANTIATIONS Procedure on page 30-46	Prints a stack backtrace
PROBE_VERSION Procedure on page 30-47	Returns the version number of `DBMS_DEBUG` on the server

Table 30–5 (Cont.) DBMS_DEBUG Package Subprograms

Subprogram	Description
SELF_CHECK Procedure on page 30-48	Performs an internal consistency check
SET_BREAKPOINT Function on page 30-49	Sets a breakpoint in a program unit
SET_OER_BREAKPOINT Function on page 30-50	Sets an OER breakpoint
SET_TIMEOUT Function on page 30-51	Sets the timeout value
SET_TIMEOUT_ BEHAVIOUR Procedure on page 30-52	Tells Probe what to do with the target session when a timeout occurs
SET_VALUE Function on page 30-53	Sets a value in the currently-running program
SHOW_BREAKPOINTS Procedures on page 30-55	Returns a listing of the current breakpoints
SHOW_FRAME_SOURCE Procedure on page 30-56	Fetches the frame source
SHOW_SOURCE Procedures on page 30-57	Fetches program source
SYNCHRONIZE Function on page 30-59	Waits for program to start running
TARGET_PROGRAM_ RUNNING Procedure on page 30-60	Returns `TRUE` if the target session is currently executing a stored procedure, or `FALSE` if it is not

ATTACH_SESSION Procedure

This procedure notifies the debug session about the target program.

Syntax

```
DBMS_DEBUG.ATTACH_SESSION (
   debug_session_id  IN VARCHAR2,
   diagnostics       IN BINARY_INTEGER := 0);
```

Parameters

Table 30–6 ATTACH_SESSION Procedure Parameters

Parameter	Description
`debug_session_id`	Debug ID from a call to `INITIALIZE` in target session.
`diagnostics`	Generate diagnostic output if nonzero.

CONTINUE Function

This function passes the given breakflags (a mask of the events that are of interest) to Probe in the target process. It tells Probe to continue execution of the target process, and it waits until the target process runs to completion or signals an event.

If `info_requested` is not `NULL`, then calls `GET_RUNTIME_INFO`.

Syntax

```
DBMS_DEBUG.CONTINUE (
   run_info       IN OUT runtime_info,
   breakflags     IN     BINARY_INTEGER,
   info_requested IN     BINARY_INTEGER := NULL)
  RETURN BINARY_INTEGER;
```

Parameters

Table 30–7 CONTINUE Function Parameters

Parameter	Description
`run_info`	Information about the state of the program.
`breakflags`	Mask of events that are of interest. See "Breakflags" on page 30-12.
`info_requested`	Which information should be returned in `run_info` when the program stops. See "Information Flags" on page 30-13.

Return Values

Table 30–8 CONTINUE Function Return Values

Return	Description
`success`	
`error_timeout`	Timed out before the program started running.
`error_communication`	Other communication error.

DEBUG_OFF Procedure

Caution: There must be a debug session waiting if immediate is `TRUE`**.**

This procedure notifies the target session that debugging should no longer take place in that session. It is not necessary to call this function before ending the session.

Syntax

```
DBMS_DEBUG.DEBUG_OFF;
```

Usage Notes

The server does not handle this entrypoint specially. Therefore, it attempts to debug this entrypoint.

DEBUG_ON Procedure

This procedure marks the target session so that all PL/SQL is run in debug mode. This must be done before any debugging can take place.

Syntax

```
DBMS_DEBUG.DEBUG_ON (
   no_client_side_plsql_engine BOOLEAN := TRUE,
   immediate                   BOOLEAN := FALSE);
```

Parameters

Table 30–9 DEBUG_ON Procedure Parameters

Parameter	Description
`no_client_side_ plsql_engine`	Should be left to its default value unless the debugging session is taking place from a client-side PL/SQL engine.
`immediate`	If this is `TRUE`, then the interpreter immediately switches itself into debug-mode, instead of continuing in regular mode for the duration of the call.

DELETE_BREAKPOINT Function

This function deletes a breakpoint.

Syntax

```
DBMS_DEBUG.DELETE_BREAKPOINT (
   breakpoint IN BINARY_INTEGER)
  RETURN BINARY_INTEGER;
```

Parameters

Table 30–10 DELETE_BREAKPOINT Function Parameters

Parameter	Description
`breakpoint`	Breakpoint number from a previous call to `SET_BREAKPOINT`.

Return Values

Table 30–11 DELETE_BREAKPOINT Function Return Values

Return	Description
`success`	
`error_no_such_ breakpt`	No such breakpoint exists.
`error_idle_breakpt`	Cannot delete an unused breakpoint.
`error_stale_breakpt`	The program unit was redefined since the breakpoint was set.

DELETE_OER_BREAKPOINT Function

This function deletes an OER breakpoint.

Syntax

```
DBMS_DEBUG.DELETE_OER_BREAKPOINT (
   oer  IN PLS_INTEGER)
RETURN PLS_INTEGER;
```

Parameters

Table 30–12 DELETE_OER_BREAKPOINT Function Parameters

Parameter	Description
oer	The OER (positive 4-byte number) to delete.

DETACH_SESSION Procedure

This procedure stops debugging the target program. This procedure may be called at any time, but it does not notify the target session that the debug session is detaching itself, and it does not terminate execution of the target session. Therefore, care should be taken to ensure that the target session does not hang itself.

Syntax

```
DBMS_DEBUG.DETACH_SESSION;
```

DISABLE_BREAKPOINT Function

This function makes an existing breakpoint inactive but leaves it in place.

Syntax

```
DBMS_DEBUG.DISABLE_BREAKPOINT (
   breakpoint IN BINARY_INTEGER)
  RETURN BINARY_INTEGER;
```

Parameters

Table 30–13 DISABLE_BREAKPOINT Function Parameters

Parameter	Description
breakpoint	Breakpoint number from a previous call to SET_BREAKPOINT.

Return Values

Table 30–14 DISABLE_BREAKPOINT Function Return Values

Returns	Description
success	
error_no_such_ breakpt	No such breakpoint exists.
error_idle_breakpt	Cannot disable an unused breakpoint.

ENABLE_BREAKPOINT Function

This function is the reverse of disabling. This enables a previously disabled breakpoint.

Syntax

```
DBMS_DEBUG.ENABLE_BREAKPOINT (
   breakpoint IN BINARY_INTEGER)
  RETURN BINARY_INTEGER;
```

Parameters

Table 30–15 ENABLE_BREAKPOINT Function Parameters

Parameter	Description
`breakpoint`	Breakpoint number from a previous call to `SET_BREAKPOINT`.

Return Values

Table 30–16 ENABLE_BREAKPOINT Function Return Values

Return	Description
`success`	Success.
`error_no_such_ breakpt`	No such breakpoint exists.
`error_idle_breakpt`	Cannot enable an unused breakpoint.

EXECUTE Procedure

This procedure executes SQL or PL/SQL code in the target session. The target session is assumed to be waiting at a breakpoint (or other event). The call to DBMS_DEBUG.EXECUTE occurs in the debug session, which then asks the target session to execute the code.

Syntax

```
DBMS_DEBUG.EXECUTE (
   what         IN VARCHAR2,
   frame#       IN BINARY_INTEGER,
   bind_results IN BINARY_INTEGER,
   results      IN OUT NOCOPY dbms_debug_vc2coll,
   errm         IN OUT NOCOPY VARCHAR2);
```

Parameters

Table 30–17 EXECUTE Procedure Parameters

Parameter	Description
what	SQL or PL/SQL source to execute.
frame#	The context in which to execute the code. Only -1 (global context) is supported at this time.
bind_results	Whether the source wants to bind to results in order to return values from the target session. 0 = No 1 = Yes
results	Collection in which to place results, if bind_results is not 0.
errm	Error message, if an error occurred; otherwise, NULL.

Examples

Example 1

This example executes a SQL statement. It returns no results.

```
DECLARE
   coll sys.dbms_debug_vc2coll; -- results (unused)
   errm VARCHAR2(100);
BEGIN
   dbms_debug.execute('insert into emp(ename,empno,deptno) ' ||
                      'values(''LJE'', 1, 1)',
                      -1, 0, coll, errm);
END;
```

Example 2

This example executes a PL/SQL block, and it returns no results. The block is an autonomous transaction, which means that the value inserted into the table becomes visible in the debug session.

```
DECLARE
   coll sys.dbms_debug_vc2coll;
   errm VARCHAR2(100);
BEGIN
```

```
   dbms_debug.execute(
       'DECLARE PRAGMA autonomous_transaction; ' ||
       'BEGIN ' ||
       '   insert into emp(ename, empno, deptno) ' ||
       '   values(''LJE'', 1, 1); ' ||
       ' COMMIT; ' ||
       'END;',
       -1, 0, coll, errm);
END;
```

Example 3

This example executes a PL/SQL block, and it returns some results.

```
DECLARE
   coll sys.dbms_debug_vc2coll;
   errm VARCHAR2(100);
BEGIN
   dbms_debug.execute(
      'DECLARE ' ||
      '   pp SYS.dbms_debug_vc2coll := SYS.dbms_debug_vc2coll(); ' ||
      '   x  PLS_INTEGER; ' ||
      '   i  PLS_INTEGER := 1; ' ||
      'BEGIN ' ||
      '   SELECT COUNT(*) INTO x FROM emp; ' ||
      '   pp.EXTEND(x * 6); ' ||
      '   FOR c IN (SELECT * FROM emp) LOOP ' ||
      '      pp(i) := ''Ename: '' || c.ename; i := i+1; ' ||
      '      pp(i) := ''Empno: '' || c.empno; i := i+1; ' ||
      '      pp(i) := ''Job:   '' || c.job;   i := i+1; ' ||
      '      pp(i) := ''Mgr:   '' || c.mgr;   i := i+1; ' ||
      '      pp(i) := ''Sal:   '' || c.sal;   i := i+1; ' ||
      '      pp(i) := null;                   i := i+1; ' ||
      '   END LOOP; ' ||
      '   :1 := pp;' ||
      'END;',
       -1, 1, coll, errm);
   each := coll.FIRST;
   WHILE (each IS NOT NULL) LOOP
      dosomething(coll(each));
      each := coll.NEXT(each);
   END LOOP;
END;
```

GET_INDEXES Function

Given a name of a variable or parameter, this function returns the set of its indexes, if it is an indexed table. An error is returned if it is not an indexed table.

Syntax

```
DBMS_DEBUG.GET_INDEXES (
   varname   IN  VARCHAR2,
   frame#    IN  BINARY_INTEGER,
   handle    IN  program_info,
   entries   OUT index_table)
RETURN BINARY_INTEGER;
```

Parameters

Table 30–18 GET_INDEXES Function Parameters

Parameter	Description
varname	Name of the variable to get index information about.
frame#	Number of frame in which the variable or parameter resides; NULL for a package variable.
handle	Package description, if object is a package variable.
entries	1-based table of the indexes. If non-NULL, then entries(1) contains the first index of the table, entries(2) contains the second index, and so on.

Return Values

Table 30–19 GET_INDEXES Function Return Values

Return	Description
error_no_such_object	Either: - The package does not exist. - The package is not instantiated. - The user does not have privileges to debug the package. - The object does not exist in the package.

GET_MORE_SOURCE Procedure

When source does not fit in the buffer provided by that version of the SHOW_SOURCE Procedures which produce a formatted buffer, this procedure provides additional source.

Syntax

```
DBMS_DEBUG.GET_MORE_SOURCE (
   buffer          IN OUT VARCHAR2,
   buflen          IN BINARY_INTEGER,
   piece#          IN BINARY_INTEGER);
```

Parameters

Table 30–20 GET_MORE_SOURCE Procedure Parameters

Parameter	Description
buffer	The buffer.
buflen	The length of the buffer.
piece#	A value between 2 and the value returned in the parameter pieces from the call to the relevant version of the SHOW_SOURCE Procedures.

Usage Notes

This procedure should be called only after the version of SHOW_SOURCE that returns a formatted buffer.

GET_LINE_MAP Function

This function finds line and entrypoint information about a program so that a debugger can determine the source lines at which it is possible to place breakpoints.

Syntax

```
DBMS_DEBUG.GET_LINE_MAP (
   program                IN   program_info,
   maxline                OUT  BINARY_INTEGER,
   number_of_entry_points OUT  BINARY_INTEGER,
   linemap                OUT  RAW)
  RETURN BINARY_INTEGER;
```

Parameters

Table 30–21 GET_LINE_MAP Function Parameters

Parameter	Description
program	A top-level program unit (procedure / package / function / package body, and so on). Its Namespace, Name, and Owner fields must be initialized, the remaining fields are ignored.
maxline	The largest source code line number in 'program'.
number_of_entry_points	The number of subprograms in 'program'
linemap	A bitmap representing the executable lines of 'program'. If line number N is executable, bit number N MOD 8 will be set to 1 at linemap position N / 8. The length of returned linemap is either maxline divided by 8 (plus one if maxline MOD 8 is not zero) or 32767 in the unlikely case of maxline being larger than 32767 * 8.

Return Values

Table 30–22 GET_LINE_MAP Function Return Values

Return	Description
success	A successful completion.
error_no_debug_info	The program unit exists, but has no debug info.
error_bad_handle	No such program unit exists.

GET_RUNTIME_INFO Function

This function returns information about the current program. It is only needed if the `info_requested` parameter to `SYNCHRONIZE` or `CONTINUE` was set to `0`.

Note: This is currently only used by client-side PL/SQL.

Syntax

```
DBMS_DEBUG.GET_RUNTIME_INFO (
   info_requested  IN  BINARY_INTEGER,
   run_info        OUT runtime_info)
  RETURN BINARY_INTEGER;
```

Parameters

Table 30–23 GET_RUNTIME_INFO Function Parameters

Parameter	Description
`info_requested`	Which information should be returned in `run_info` when the program stops. See "Information Flags" on page 30-13.
`run_info`	Information about the state of the program.

GET_TIMEOUT_BEHAVIOUR Function

This procedure returns the current timeout behavior. This call is made in the target session.

Syntax

```
DBMS_DEBUG.GET_TIMEOUT_BEHAVIOUR
 RETURN BINARY_INTEGER;
```

Parameters

Table 30–24 GET_TIMEOUT_BEHAVIOUR Function Parameters

Parameter	Description
`oer`	The OER (a 4-byte positive number).

Return Values

Table 30–25 GET_TIMEOUT_BEHAVIOUR Function Return Values

Return	Description
`success`	A successful completion.

Information Flags

```
info_getOerInfo CONSTANT PLS_INTEGER:= 32;
```

Usage Notes

Less functionality is supported on OER breakpoints than on code breakpoints. In particular, note that:

- No "breakpoint number" is returned - the number of the OER is used instead. Thus it is impossible to set duplicate breakpoints on a given OER (it is a no-op).
- It is not possible to disable an OER breakpoint (although clients are free to simulate this by deleting it).
- OER breakpoints are deleted using `delete_oer_breakpoint`.

GET_VALUE Function

This function gets a value from the currently-running program. There are two overloaded GET_VALUE functions.

Syntax

```
DBMS_DEBUG.GET_VALUE (
   variable_name  IN  VARCHAR2,
   frame#         IN  BINARY_INTEGER,
   scalar_value   OUT VARCHAR2,
   format         IN  VARCHAR2 := NULL)
RETURN BINARY_INTEGER;
```

Parameters

Table 30–26 GET_VALUE Function Parameters

Parameter	Description
variable_name	Name of the variable or parameter.
frame#	Frame in which it lives; 0 means the current procedure.
scalar_value	Value.
format	Optional date format to use, if meaningful.

Return Values

Table 30–27 GET_VALUE Function Return Values

Return	Description
success	A successful completion.
error_bogus_frame	Frame does not exist.
error_no_debug_info	Entrypoint has no debug information.
error_no_such_object	variable_name does not exist in frame#.
error_unknown_type	The type information in the debug information is illegible.
error_nullvalue	Value is NULL.
error_indexed_table	The object is a table, but no index was provided.

This form of GET_VALUE is for fetching package variables. Instead of a frame#, it takes a handle, which describes the package containing the variable.

Syntax

```
DBMS_DEBUG.GET_VALUE (
   variable_name  IN  VARCHAR2,
   handle         IN  program_info,
   scalar_value   OUT VARCHAR2,
   format         IN  VARCHAR2 := NULL)
RETURN BINARY_INTEGER;
```

Parameters

Table 30–28 GET_VALUE Function Parameters

Parameter	Description
variable_name	Name of the variable or parameter.
handle	Description of the package containing the variable.
scalar_value	Value.
format	Optional date format to use, if meaningful.

Return Values

Table 30–29 GET_VALUE Function Return Values

Return	Description
error_no_such_object	Either: - Package does not exist. - Package is not instantiated. - User does not have privileges to debug the package. - Object does not exist in the package.
error_indexed_table	The object is a table, but no index was provided.

Examples

This example illustrates how to get the value with a given package PACK in schema SCOTT, containing variable VAR:

```
DECLARE
   handle     dbms_debug.program_info;
   resultbuf  VARCHAR2(500);
   retval     BINARY_INTEGER;
BEGIN
   handle.Owner     := 'SCOTT';
   handle.Name      := 'PACK';
   handle.namespace := dbms_debug.namespace_pkgspec_or_toplevel;
   retval           := dbms_debug.get_value('VAR', handle, resultbuf, NULL);
END;
```

INITIALIZE Function

This function initializes the target session for debugging.

Syntax

```
DBMS_DEBUG.INITIALIZE (
   debug_session_id  IN VARCHAR2       := NULL,
   diagnostics       IN BINARY_INTEGER := 0)
  RETURN VARCHAR2;
```

Parameters

Table 30–30 INITIALIZE Function Parameters

Parameter	Description
debug_session_id	Name of session ID. If NULL, then a unique ID is generated.
diagnostics	Indicates whether to dump diagnostic output to the tracefile. 0 = (default) no diagnostics 1 = print diagnostics

Return Values

The newly-registered debug session ID (debugID)

Usage Notes

You cannot use DBMS_DEBUG and the JDWP-based debugging interface simultaneously. This call will either fail with an ORA-30677 error if the session is currently being debugged with the JDWP-based debugging interface or, if the call succeeds, any further use of the JDWP-based interface to debug this session will be disallowed.

Calls to DBMS_DEBUG will succeed only if either the caller or the specified debug role carries the DEBUG CONNECT SESSION privilege. Failing that, an ORA-1031 error will be raised. Other exceptions are also possible if a debug role is specified but the password does not match, or if the calling user has not been granted the role, or the role is application-enabled and this call does not originate from within the role-enabling package.

The CREATE ANY PROCEDURE privilege does not affect the visibility of routines through the debugger. A privilege DEBUG for each object has been introduced with a corresponding DEBUG ANY PROCEDURE variant. These are required in order to see routines owned by users other than the session's login user.

Authentication of the debug role and the check for DEBUG CONNECT SESSION privilege will be done in the context of the caller to this routine. If the caller is a definer's rights routine or has been called from one, only privileges granted to the defining user, the debug role, or PUBLIC will be used to check for DEBUG CONNECT SESSION. If this call is from within a definer's rights routine, the debug role, if specified, must be one that has been granted to that definer, but it need not also have been granted to the session login user or be enabled in the calling session at the time the call is made.

The checks made by the debugger after this call is made looking for the DEBUG privilege on individual procedures will be done in the context of the session's login user, the roles that were enabled at session level at the moment this call was made

(even if those roles were not available within a definer's rights environment of the call), and the debug role.

PING Procedure

This procedure pings the target session to prevent it from timing out. Use this procedure when execution is suspended in the target session, for example at a breakpoint.

If the `timeout_behaviour` is set to `retry_on_timeout` then this procedure is not necessary.

Syntax

```
DBMS_DEBUG.PING;
```

Exceptions

Oracle will display the `no_target_program` exception if there is no target program or if the target session is not currently waiting for input from the debug session.

Usage Notes

Timeout options for the target session are registered with the target session by calling `set_timeout_behaviour`.

- `retry_on_timeout` - Retry. Timeout has no effect. This is like setting the timeout to an infinitely large value.
- `continue_on_timeout` - Continue execution, using same event flags.
- `nodebug_on_timeout` - Turn debug-mode OFF (in other words, call debug_off) and then continue execution. No more events will be generated by this target session unless it is re-initialized by calling debug_on.
- `abort_on_timeout` - Continue execution, using the abort_execution flag, which should cause the program to terminate immediately. The session remains in debug-mode.

```
retry_on_timeout CONSTANT BINARY_INTEGER:= 0;

continue_on_timeout CONSTANT BINARY_INTEGER:= 1;

nodebug_on_timeout CONSTANT BINARY_INTEGER:= 2;

abort_on_timeout CONSTANT BINARY_INTEGER:= 3;
```

PRINT_BACKTRACE Procedure

This procedure prints a backtrace listing of the current execution stack. This should only be called if a program is currently running.

There are two overloaded PRINT_BACKTRACE procedures.

Syntax

```
DBMS_DEBUG.PRINT_BACKTRACE (
  listing IN OUT VARCHAR2);

DBMS_DEBUG.PRINT_BACKTRACE (
  backtrace OUT backtrace_table);
```

Parameters

Table 30–31 PRINT_BACKTRACE Procedure Parameters

Parameter	Description
listing	A formatted character buffer with embedded newlines.
backtrace	1-based indexed table of backtrace entries. The currently-running procedure is the last entry in the table (that is, the frame numbering is the same as that used by GET_VALUE). Entry 1 is the oldest procedure on the stack.

PRINT_INSTANTIATIONS Procedure

This procedure returns a list of the packages that have been instantiated in the current session.

Syntax

```
DBMS_DEBUG.PRINT_INSTANTIATIONS (
   pkgs   IN OUT NOCOPY backtrace_table,
   flags  IN BINARY_INTEGER);
```

Parameters

Table 30–32 PRINT_INSTANTIATIONS Procedure Parameters

Parameter	Description
`pkgs`	The instantiated packages
`flags`	Bitmask of options: ■ 1 - show specs ■ 2 - show bodies ■ 4 - show local instantiations ■ 8 - show remote instantiations (NYI) ■ 16 - do a fast job. The routine does not test whether debug information exists or whether the libunit is shrink-wrapped.

Exceptions

`no_target_program` - target session is not currently executing

Usage Notes

On return, `pkgs` contains a `program_info` for each instantiation. The valid fields are: `Namespace, Name, Owner,` and `LibunitType.`

In addition, Line# contains a bitmask of:

- 1 - the libunit contains debug info
- 2 - the libunit is shrink-wrapped

PROBE_VERSION Procedure

This procedure returns the version number of DBMS_DEBUG on the server.

Syntax

```
DBMS_DEBUG.PROBE_VERSION (
   major out BINARY_INTEGER,
   minor out BINARY_INTEGER);
```

Parameters

Table 30–33 PROBE_VERSION Procedure Parameters

Parameter	Description
major	Major version number.
minor	Minor version number: increments as functionality is added.

SELF_CHECK Procedure

This procedure performs an internal consistency check. SELF_CHECK also runs a communications test to ensure that the Probe processes are able to communicate.

If SELF_CHECK does not return successfully, then an incorrect version of DBMS_DEBUG was probably installed on this server. The solution is to install the correct version (pbload.sql loads DBMS_DEBUG and the other relevant packages).

Syntax

```
DBMS_DEBUG.SELF_CHECK (
   timeout IN binary_integer := 60);
```

Parameters

Table 30–34 SELF_CHECK Procedure Parameters

Parameter	Description
timeout	The timeout to use for the communication test. Default is 60 seconds.

Exceptions

Table 30–35 SELF_CHECK Procedure Exceptions

Exception	Description
OER-6516	Probe version is inconsistent.
pipe_creation_ failure	Could not create a pipe.
pipe_send_failure	Could not write data to the pipe.
pipe_receive_failure	Could not read data from the pipe.
pipe_datatype_ mismatch	Datatype in the pipe was wrong.
pipe_data_error	Data got garbled in the pipe.

All of these exceptions are fatal. They indicate a serious problem with Probe that prevents it from working correctly.

SET_BREAKPOINT Function

This function sets a breakpoint in a program unit, which persists for the current session. Execution pauses if the target program reaches the breakpoint.

Syntax

```
DBMS_DEBUG.SET_BREAKPOINT (
   program     IN  program_info,
   line#       IN  BINARY_INTEGER,
   breakpoint# OUT BINARY_INTEGER,
   fuzzy       IN  BINARY_INTEGER := 0,
   iterations  IN  BINARY_INTEGER := 0)
  RETURN BINARY_INTEGER;
```

Parameters

Table 30–36 SET_BREAKPOINT Function Parameters

Parameter	Description
program	Information about the program unit in which the breakpoint is to be set. (In version 2.1 and later, the namespace, name, owner, and dblink may be set to NULL, in which case the breakpoint is placed in the currently-running program unit.)
line#	Line at which the breakpoint is to be set.
breakpoint#	On successful completion, contains the unique breakpoint number by which to refer to the breakpoint.
fuzzy	Only applicable if there is no executable code at the specified line: 0 means return error_illegal_line. 1 means search forward for an adjacent line at which to place the breakpoint. -1 means search backward for an adjacent line at which to place the breakpoint.
iterations	Number of times to wait before signalling this breakpoint.

Return Values

Note: The fuzzy and iterations parameters are not yet implemented.

Table 30–37 SET_BREAKPOINT Function Return Values

Return	Description
success	A successful completion.
error_illegal_line	Cannot set a breakpoint at that line.
error_bad_handle	No such program unit exists.

SET_OER_BREAKPOINT Function

This function sets an OER breakpoint.

Syntax

```
DBMS_DEBUG.SET_OER_BREAKPOINT (
   oer  IN PLS_INTEGER)
RETURN PLS_INTEGER;
```

Parameters

Table 30–38 SET_OER_BREAKPOINT Function Parameters

Parameter	Description
`oer`	The OER (positive 4-byte number) to delete.

Return Values

Table 30–39 SET_OER_BREAKPOINT Function Return Values

Return	Description
`success`	A successful completion.
`error_no_such_ breakpt`	No such OER breakpoint exists.

SET_TIMEOUT Function

This function sets the timeout value and returns the new timeout value.

Syntax

```
DBMS_DEBUG.SET_TIMEOUT (
   timeout BINARY_INTEGER)
  RETURN BINARY_INTEGER;
```

Parameters

Table 30–40 SET_TIMEOUT Function Parameters

Parameter	Description
`timeout`	The timeout to use for communication between the target and debug sessions.

SET_TIMEOUT_BEHAVIOUR Procedure

This procedure tells Probe what to do with the target session when a timeout occurs. This call is made in the target session.

Syntax

```
DBMS_DEBUG.SET_TIMEOUT_BEHAVIOUR (
   behaviour IN PLS_INTEGER);
```

Parameters

Table 30–41 SET_TIMEOUT_BEHAVIOUR Procedure Parameters

Parameter	Description
`behaviour - One of the following:`	
`retry_on_timeout`	Retry. Timeout has no effect. This is like setting the timeout to an infinitely large value.
`continue_on_timeout`	Continue execution, using same event flags.
`nodebug_on_timeout`	Turn debug-mode OFF (in other words, call `debug_off`) and continue execution. No more events will be generated by this target session unless it is re-initialized by calling `debug_on`.
`abort_on_timeout`	Continue execution, using the `abort_execution` flag, which should cause the program to terminate immediately. The session remains in debug-mode.

Exceptions

unimplemented - the requested behavior is not recognized

Usage Notes

The default behavior (if this procedure is not called) is `continue_on_timeout`, since it allows a debugger client to reestablish control (at the next event) but does not cause the target session to hang indefinitely.

SET_VALUE Function

This function sets a value in the currently-running program. There are two overloaded SET_VALUE functions.

Syntax

```
DBMS_DEBUG.SET_VALUE (
   frame#               IN binary_integer,
   assignment_statement IN varchar2)
  RETURN BINARY_INTEGER;

DBMS_DEBUG.SET_VALUE (
   handle               IN program_info,
   assignment_statement IN VARCHAR2)
  RETURN BINARY_INTEGER;
```

Parameters

Table 30–42 SET_VALUE Function Parameters

Parameter	Description
frame#	Frame in which the value is to be set; 0 means the currently executing frame.
handle	Description of the package containing the variable.
assignment_statement	An assignment statement (which must be legal PL/SQL) to run in order to set the value. For example, 'x := 3;'. Only scalar values are supported in this release. The right side of the assignment statement must be a scalar.

Return Values

Table 30–43 SET_VALUE Function Return Values

Return	Description
success	-
error_illegal_value	Not possible to set it to that value.
error_illegal_null	Cannot set to NULL because object type specifies it as 'not null'.
error_value_ malformed	Value is not a scalar.
error_name_ incomplete	The assignment statement does not resolve to a scalar. For example, 'x := 3;', if x is a record.
error_no_such_object	Either: - Package does not exist. - Package is not instantiated. - User does not have privileges to debug the package. - Object does not exist in the package.

Usage Notes

In some cases, the PL/SQL compiler uses temporaries to access package variables, and does not guarantee to update such temporaries. It is possible, although unlikely, that

modification to a package variable using SET_VALUE might not take effect for a line or two.

Examples

To set the value of SCOTT.PACK.var to 6:

```
DECLARE
   handle  dbms_debug.program_info;
   retval  BINARY_INTEGER;
BEGIN
   handle.Owner     := 'SCOTT';
   handle.Name      := 'PACK';
   handle.namespace := dbms_debug.namespace_pkgspec_or_toplevel;
   retval           := dbms_debug.set_value(handle, 'var := 6;');
END;
```

SHOW_BREAKPOINTS Procedures

There are two overloaded procedures that return a listing of the current breakpoints. There are three overloaded SHOW_BREAKPOINTS procedures.

Syntax

```
DBMS_DEBUG.SHOW_BREAKPOINTS (
   listing    IN OUT VARCHAR2);

DBMS_DEBUG.SHOW_BREAKPOINTS (
   listing    OUT breakpoint_table);

DBMS_DEBUG.SHOW_BREAKPOINTS (
   code_breakpoints  OUT breakpoint_table,
   oer_breakpoints   OUT oer_table);
```

Parameters

Table 30–44 SHOW_BREAKPOINTS Procedure Parameters

Parameter	Description
listing	A formatted buffer (including newlines) of the breakpoints.
	Indexed table of breakpoint entries. The breakpoint number is indicated by the index into the table. Breakpoint numbers start at 1 and are reused when deleted.
code_breakpoints	The indexed table of breakpoint entries, indexed by breakpoint number.
oer_breakpoints	The indexed table of OER breakpoints, indexed by OER.

SHOW_FRAME_SOURCE Procedure

The procedure gets the source code. There are two overloaded SHOW_SOURCE procedures.

Syntax

```
DBMS_DEBUG.SHOW_FRAME_SOURCE (
   first_line  IN            BINARY_INTEGER,
   last_line   IN            BINARY_INTEGER,
   source      IN OUT NOCOPY vc2_table,
   frame_num   IN            BINARY_INTEGER);
```

Parameters

Table 30–45 SHOW_FRAME_SOURCE Procedure Parameters

Parameter	Description
first_line	Line number of first line to fetch. (PL/SQL programs always start at line 1 and have no holes.)
last_line	Line number of last line to fetch. No lines are fetched past the end of the program.
source	The resulting table, which may be indexed by line#.
frame_num	1-based frame number.

Usage Notes

- You use this function only when backtrace shows an anonymous unit is executing at a given frame position and you need to view the source n order to set a breakpoint.
- If frame number is top of the stack and it's an anonymous block then SHOW_SOURCE can also be used.
- If it's a stored PLSQL package/function/procedure then use SQL as described in the Usage Notes to SHOW_SOURCE Procedures.

SHOW_SOURCE Procedures

The procedure gets the source code. There are two overloaded `SHOW_SOURCE` procedures.

Syntax

```
DBMS_DEBUG.SHOW_SOURCE (
   first_line  IN   BINARY_INTEGER,
   last_line   IN   BINARY_INTEGER,
   source      OUT  vc2_table);

DBMS_DEBUG.SHOW_SOURCE (
   first_line   IN     BINARY_INTEGER,
   last_line    IN     BINARY_INTEGER,
   window       IN     BINARY_INTEGER,
   print_arrow  IN     BINARY_INTEGER,
   buffer       IN OUT VARCHAR2,
   buflen       IN     BINARY_INTEGER,
   pieces       OUT    BINARY_INTEGER);
```

Parameters

Table 30–46 SHOW_SOURCE Procedure Parameters

Parameter	Description
`first_line`	Line number of first line to fetch. (PL/SQL programs always start at line 1 and have no holes.)
`last_line`	Line number of last line to fetch. No lines are fetched past the end of the program.
`source`	The resulting table, which may be indexed by line#.
`window`	'Window' of lines (the number of lines around the current source line).
`print_arrow`	Nonzero means to print an arrow before the current line.
`buffer`	Buffer in which to place the source listing.
`buflen`	Length of buffer.
`pieces`	Set to nonzero if not all the source could be placed into the given buffer.

Return Values

An indexed table of source-lines. The source lines are stored starting at `first_line`. If any error occurs, then the table is empty.

Usage Notes

The best way to get the source code (for a program that is being run) is to use SQL. For example:

```
DECLARE
    info DBMS_DEBUG.runtime_info;
BEGIN
   -- call DBMS_DEBUG.SYNCHRONIZE, CONTINUE,
   -- or GET_RUNTIME_INFO to fill in 'info'
   SELECT text INTO <buffer> FROM all_source
```

```
   WHERE owner = info.Program.Owner
     AND name  = info.Program.Name
     AND line  = info.Line#;
END;
```

However, this does not work for nonpersistent programs (for example, anonymous blocks and trigger invocation blocks). For nonpersistent programs, call `SHOW_SOURCE`. There are two flavors: one returns an indexed table of source lines, and the other returns a packed (and formatted) buffer.

The second overloading of `SHOW_SOURCE` returns the source in a formatted buffer, complete with line-numbers. It is faster than the indexed table version, but it does not guarantee to fetch all the source.

If the source does not fit in bufferlength (`buflen`), then additional pieces can be retrieved using the `GET_MORE_SOURCE` procedure (`pieces` returns the number of additional pieces that need to be retrieved).

SYNCHRONIZE Function

This function waits until the target program signals an event. If `info_requested` is not `NULL`, then it calls `GET_RUNTIME_INFO`.

Syntax

```
DBMS_DEBUG.SYNCHRONIZE (
   run_info       OUT  runtime_info,
   info_requested IN   BINARY_INTEGER := NULL)
  RETURN BINARY_INTEGER;
```

Parameters

Table 30–47 SYNCHRONIZE Function Parameters

Parameter	Description
`run_info`	Structure in which to write information about the program. By default, this includes information about what program is running and at which line execution has paused.
`info_requested`	Optional bit-field in which to request information other than the default (which is `info_getStackDepth` + `info_getLineInfo`). 0 means that no information is requested at all. See "Information Flags" on page 30-13.

Return Values

Table 30–48 SYNCHRONIZE Function Return Values

Return	Description
`success`	A successful completion.
`error_timeout`	Timed out before the program started execution.
`error_communication`	Other communication error.

TARGET_PROGRAM_RUNNING Procedure

This procedure returns `TRUE` if the target session is currently executing a stored procedure, or `FALSE` if it is not.

Syntax

```
DBMS_DEBUG.TARGET_PROGRAM_RUNNING
  RETURN BOOLEAN;
```

31

DBMS_DEFER

DBMS_DEFER is the user interface to a replicated transactional deferred remote procedure call facility. Replicated applications use the calls in this interface to queue procedure calls for later transactional execution at remote nodes.

These procedures are typically called from either after row triggers or application specified update procedures.

Documentation of DBMS_DEFER

For a complete description of this package within the context of Replication, see DBMS_DEFER in the *Oracle Database Advanced Replication Management API Reference.*

32

DBMS_DEFER_QUERY

DBMS_DEFER_QUERY enables you to query the deferred transactions queue data that is not exposed through views.

- Documentation of DBMS_DEFER_QUERY

Documentation of DBMS_DEFER_QUERY

For a complete description of this package within the context of Replication, see DBMS_DEFER_QUERY in the *Oracle Database Advanced Replication Management API Reference*.

33

DBMS_DEFER_SYS

DBMS_DEFER_SYS subprograms manage default replication node lists. This package is the system administrator interface to a replicated transactional deferred remote procedure call facility. Administrators and replication daemons can execute transactions queued for remote nodes using this facility, and administrators can control the nodes to which remote calls are destined.

Documentation of DBMS_DEFER_SYS

For a complete description of this package within the context of Replication, see `DBMS_DEFER_SYS` in the *Oracle Database Advanced Replication Management API Reference*.

34

DBMS_DESCRIBE

You can use the DBMS_DESCRIBE package to get information about a PL/SQL object. When you specify an object name, DBMS_DESCRIBE returns a set of indexed tables with the results. Full name translation is performed and security checking is also checked on the final object.

This chapter contains the following topics:

Using DBMS_DESCRIBE

Overview

This package provides the same functionality as the Oracle Call Interface `OCIDescribeAny` call.

See Also: *Oracle Call Interface Programmer's Guide*

Security Model

This package is available to `PUBLIC` and performs its own security checking based on the schema object being described.

Types

The DBMS_DESCRIBE package declares two PL/SQL table types, which are used to hold data returned by DESCRIBE_PROCEDURE in its OUT parameters. The types are:

```
TYPE VARCHAR2_TABLE IS TABLE OF VARCHAR2(30)
    INDEX BY BINARY_INTEGER;

TYPE NUMBER_TABLE IS TABLE OF NUMBER
    INDEX BY BINARY_INTEGER;
```

Exceptions

DBMS_DESCRIBE can raise application errors in the range -20000 to -20004.

Table 34–1 DBMS_DESCRIBE Errors

Error	Description
ORA-20000	ORU 10035: cannot describe a package ('X') only a procedure within a package.
ORA-20001	ORU-10032: procedure 'X' within package 'Y' does not exist.
ORA-20002	ORU-10033: object 'X' is remote, cannot describe; expanded name 'Y'.
ORA-20003	ORU-10036: object 'X' is invalid and cannot be described.
ORA-20004	Syntax error attempting to parse 'X'.

Examples

One use of the DESCRIBE_PROCEDURE procedure is as an external service interface.

For example, consider a client that provides an OBJECT_NAME of SCOTT.ACCOUNT_UPDATE, where ACCOUNT_UPDATE is an overloaded function with specification:

```
TABLE account (account_no NUMBER, person_id NUMBER,
               balance NUMBER(7,2))
TABLE person  (person_id number(4), person_nm varchar2(10))

FUNCTION ACCOUNT_UPDATE (account_no   NUMBER,
                         person       person%rowtype,
                         amounts      DBMS_DESCRIBE.NUMBER_TABLE,
                         trans_date   DATE)
                         return       account.balance%type;

FUNCTION ACCOUNT_UPDATE (account_no   NUMBER,
                         person       person%rowtype,
                         amounts      DBMS_DESCRIBE.NUMBER_TABLE,
                         trans_no     NUMBER)
                         return       account.balance%type;
```

This procedure might look similar to the following output:

```
overload position  argument level  datatype length prec scale rad
-------- --------- -------- ------ -------- ------ ---- ----- ---
       1         0             0        2     22    7     2  10
       1         1   ACCOUNT   0        2      0    0     0   0
       1         2   PERSON    0      250      0    0     0   0
       1         1   PERSON_ID 1        2     22    4     0  10
       1         2   PERSON_NM 1        1     10    0     0   0
       1         3   AMOUNTS   0      251      0    0     0   0
       1         1             1        2     22    0     0   0
       1         4   TRANS_DATE 0      12      0    0     0   0
       2         0             0        2     22    7     2  10
       2         1   ACCOUNT_NO 0       2     22    0     0   0
       2         2   PERSON    0        2     22    4     0  10
       2         3   AMOUNTS   0      251     22    4     0  10
       2         1             1        2      0    0     0   0
       2         4   TRANS_NO  0        2      0    0     0   0
```

The following PL/SQL procedure has as its parameters all of the PL/SQL datatypes:

```
CREATE OR REPLACE PROCEDURE p1 (
        pvc2    IN     VARCHAR2,
        pvc     OUT    VARCHAR,
        pstr    IN OUT STRING,
        plong   IN     LONG,
        prowid  IN     ROWID,
        pchara  IN     CHARACTER,
        pchar   IN     CHAR,
        praw    IN     RAW,
        plraw   IN     LONG RAW,
        pbinint IN     BINARY_INTEGER,
        pplsint IN     PLS_INTEGER,
        pbool   IN     BOOLEAN,
        pnat    IN     NATURAL,
        ppos    IN     POSITIVE,
        pposn   IN     POSITIVEN,
```

```
        pnatn   IN     NATURALN,
        pnum    IN     NUMBER,
        pintgr  IN     INTEGER,
        pint    IN     INT,
        psmall  IN     SMALLINT,
        pdec    IN     DECIMAL,
        preal   IN     REAL,
        pfloat  IN     FLOAT,
        pnumer  IN     NUMERIC,
        pdp     IN     DOUBLE PRECISION,
        pdate   IN     DATE,
        pmls    IN     MLSLABEL) AS

BEGIN
    NULL;
END;
```

If you describe this procedure using the following:

```
CREATE OR REPLACE PACKAGE describe_it AS

    PROCEDURE desc_proc (name VARCHAR2);

END describe_it;

CREATE OR REPLACE PACKAGE BODY describe_it AS

  PROCEDURE prt_value(val VARCHAR2, isize INTEGER) IS
    n INTEGER;
  BEGIN
    n := isize - LENGTHB(val);
    IF n < 0 THEN
      n := 0;
    END IF;
    DBMS_OUTPUT.PUT(val);
    FOR i in 1..n LOOP
      DBMS_OUTPUT.PUT(' ');
    END LOOP;
  END prt_value;

  PROCEDURE desc_proc (name VARCHAR2) IS

      overload     DBMS_DESCRIBE.NUMBER_TABLE;
      position     DBMS_DESCRIBE.NUMBER_TABLE;
      c_level      DBMS_DESCRIBE.NUMBER_TABLE;
      arg_name     DBMS_DESCRIBE.VARCHAR2_TABLE;
      dty          DBMS_DESCRIBE.NUMBER_TABLE;
      def_val      DBMS_DESCRIBE.NUMBER_TABLE;
      p_mode       DBMS_DESCRIBE.NUMBER_TABLE;
      length       DBMS_DESCRIBE.NUMBER_TABLE;
      precision    DBMS_DESCRIBE.NUMBER_TABLE;
      scale        DBMS_DESCRIBE.NUMBER_TABLE;
      radix        DBMS_DESCRIBE.NUMBER_TABLE;
      spare        DBMS_DESCRIBE.NUMBER_TABLE;
      idx          INTEGER := 0;

  BEGIN
      DBMS_DESCRIBE.DESCRIBE_PROCEDURE(
              name,
              null,
              null,
```

```
                overload,
                position,
                c_level,
                arg_name,
                dty,
                def_val,
                p_mode,
                length,
                precision,
                scale,
                radix,
                spare);

        DBMS_OUTPUT.PUT_LINE('Position    Name        DTY  Mode');
        LOOP
            idx := idx + 1;
            prt_value(TO_CHAR(position(idx)), 12);
            prt_value(arg_name(idx), 12);
            prt_value(TO_CHAR(dty(idx)), 5);
            prt_value(TO_CHAR(p_mode(idx)), 5);
            DBMS_OUTPUT.NEW_LINE;
        END LOOP;
    EXCEPTION
       WHEN NO_DATA_FOUND THEN
          DBMS_OUTPUT.NEW_LINE;
          DBMS_OUTPUT.NEW_LINE;

    END desc_proc;
END describe_it;
```

Then the results list all the numeric codes for the PL/SQL datatypes:

```
Position  Name    Datatype_Code  Mode
1         PVC2      1              0
2         PVC       1              1
3         PSTR      1              2
4         PLONG     8              0
5         PROWID    11             0
6         PCHARA    96             0
7         PCHAR     96             0
8         PRAW      23             0
9         PLRAW     24             0
10        PBININT   3              0
11        PPLSINT   3              0
12        PBOOL     252            0
13        PNAT      3              0
14        PPOS      3              0
15        PPOSN     3              0
16        PNATN     3              0
17        PNUM      2              0
18        PINTGR    2              0
19        PINT      2              0
20        PSMALL    2              0
21        PDEC      2              0
22        PREAL     2              0
23        PFLOAT    2              0
24        PNUMER    2              0
25        PDP       2              0
26        PDATE     12             0
27        PMLS      106            0
```

Summary of DBMS_DESCRIBE Subprograms

Table 34–2 DBMS_DESCRIBE Package Subprograms

Subprogram	Description
DESCRIBE_PROCEDURE Procedure on page 34-11	Provides a brief description of a PL/SQL stored procedure

DESCRIBE_PROCEDURE Procedure

The procedure DESCRIBE_PROCEDURE provides a brief description of a PL/SQL stored procedure. It takes the name of a stored procedure and returns information about each parameter of that procedure.

Syntax

```
DBMS_DESCRIBE.DESCRIBE_PROCEDURE(
   object_name                   IN  VARCHAR2,
   reserved1                     IN  VARCHAR2,
   reserved2                     IN  VARCHAR2,
   overload                      OUT NUMBER_TABLE,
   position                      OUT NUMBER_TABLE,
   level                         OUT NUMBER_TABLE,
   argument_name                 OUT VARCHAR2_TABLE,
   datatype                      OUT NUMBER_TABLE,
   default_value                 OUT NUMBER_TABLE,
   in_out                        OUT NUMBER_TABLE,
   length                        OUT NUMBER_TABLE,
   precision                     OUT NUMBER_TABLE,
   scale                         OUT NUMBER_TABLE,
   radix                         OUT NUMBER_TABLE,
   spare                         OUT NUMBER_TABLE
   include_string_constraints    OUT BOOLEAN DEFAULT FALSE);
```

Parameters

Table 34–3 DBMS_DESCRIBE.DESCRIBE_PROCEDURE Parameters

Parameter	Description
object_name	Name of the procedure being described. The syntax for this parameter follows the rules used for identifiers in SQL. The name can be a synonym. This parameter is required and may not be null. The total length of the name cannot exceed 197 bytes. An incorrectly specified OBJECT_NAME can result in one of the following exceptions: ORA-20000 - A package was specified. You can only specify a stored procedure, stored function, packaged procedure, or packaged function. ORA-20001 - The procedure or function that you specified does not exist within the given package. ORA-20002 - The object that you specified is a remote object. This procedure cannot currently describe remote objects. ORA-20003 - The object that you specified is invalid and cannot be described. ORA-20004 - The object was specified with a syntax error.
reserved1 reserved2	Reserved for future use -- must be set to NULL or the empty string.
overload	A unique number assigned to the procedure's signature. If a procedure is overloaded, then this field holds a different value for each version of the procedure.
position	Position of the argument in the parameter list. Position 0 returns the values for the return type of a function.

Table 34–3 (Cont.) DBMS_DESCRIBE.DESCRIBE_PROCEDURE Parameters

Parameter	Description
level	If the argument is a composite type, such as record, then this parameter returns the level of the datatype. See the *Oracle Call Interface Programmer's Guide* for a description of the ODESSP call for an example.
argument_name	Name of the argument associated with the procedure that you are describing.
datatype	Oracle datatype of the argument being described. The datatypes and their numeric type codes are: 0 placeholder for procedures with no arguments 1 VARCHAR, VARCHAR, STRING 2 NUMBER, INTEGER, SMALLINT, REAL, FLOAT, DECIMAL 3 BINARY_INTEGER, PLS_INTEGER, POSITIVE, NATURAL 8 LONG 11 ROWID 12 DATE 23 RAW 24 LONG RAW 58 OPAQUE TYPE 96 CHAR (ANSI FIXED CHAR), CHARACTER 106 MLSLABEL 121 OBJECT 122 NESTED TABLE 123 VARRAY 178 TIME 179 TIME WITH TIME ZONE 180 TIMESTAMP 181 TIMESTAMP WITH TIME ZONE 231 TIMESTAMP WITH LOCAL TIME ZONE 250 PL/SQL RECORD 251 PL/SQL TABLE 252 PL/SQL BOOLEAN
default_value	1 if the argument being described has a default value; otherwise, the value is 0.
in_out	Describes the mode of the parameter: 0 IN 1 OUT 2 IN OUT
length	For %rowtype formal arguments, the length constraint is returned, otherwise 0 is returned.If the include_string_constraints parameter is set to TRUE, the argument's formal length constraint is passed back if it is of the appropriate type. Those are the string types: 1;8;23;24;96
precision	If the argument being described is of datatype 2 (NUMBER), then this parameter is the precision of that number.
scale	If the argument being described is of datatype 2 (NUMBER), then this parameter is the scale of that number.
radix	If the argument being described is of datatype 2 (NUMBER), then this parameter is the radix of that number.
spare	Reserved for future functionality.
include_ string_ constraints	The default is FALSE. If the parameter is set to TRUE, the arguments' formal type constraints is passed back if it is of the appropriate type.Those are the string types: 1;8;23;24;96

Return Values

All values from DESCRIBE_PROCEDURE are returned in its OUT parameters. The datatypes for these are PL/SQL tables, to accommodate a variable number of parameters.

35

DBMS_DIMENSION

DBMS_DIMENSION enables you to verify dimension relationships and provides an alternative to the Enterprise Manager Dimension Wizard for displaying a dimension definition.

> **See Also:** *Oracle Database Data Warehousing Guide* for detailed conceptual and usage information about the DBMS_DIMENSION package

This chapter contains the following topics:

Using DBMS_DIMENSION

This section contains topics which relate to using the `DBMS_DIMENSION` package.

Security Model

Security on this package can be controlled by granting `EXECUTE` to selected users or roles.

A user can validate or describe all the dimensions in his own schema. To validate or describe a dimension in another schema, you must have either an object privilege on the dimension or one of the following system privileges: `CREATE ANY DIMENSION`, `ALTER ANY DIMENSION`, and `DROP ANY DIMENSION`.

Summary of DBMS_DIMENSION Subprograms

Table 35–1 DBMS_DIMENSION Package Subprograms

Subprogram	Description
DESCRIBE_DIMENSION Procedure on page 35-5	Prints out the definition of the input dimension, including dimension owner and name, levels, hierarchies, and attributes
VALIDATE_DIMENSION Procedure on page 35-6	Verifies that the relationships specified in a dimension are correct

DESCRIBE_DIMENSION Procedure

This procedure displays the definition of the dimension, including dimension name, levels, hierarchies, and attributes. It displays the output using the DBMS_OUTPUT package.

Syntax

```
DBMS_DIMENSION.DESCRIBE_DIMENSION (
   dimension  IN VARCHAR2);
```

Parameters

Table 35–2 DESCRIBE_DIMENSION Procedure Parameter

Parameter	Description
dimension	The owner and name of the dimension in the format of owner.name.

VALIDATE_DIMENSION Procedure

This procedure verifies that the relationships specified in a dimension are valid. The rowid for any row that is found to be invalid will be stored in the table DIMENSION_EXCEPTIONS in the user's schema.

Syntax

```
DBMS_DIMENSION.VALIDATE_DIMENSION (
   dimension               IN VARCHAR2,
   incremental             IN BOOLEAN := TRUE,
   check_nulls             IN BOOLEAN := FALSE,
   statement_id            IN VARCHAR2 := NULL );
```

Parameters

Table 35–3 VALIDATE_DIMENSION Procedure Parameters

Parameter	Description
dimension	The owner and name of the dimension in the format of owner.name.
incremental	If TRUE, check only the new rows for tables of this dimension. If FALSE, check all the rows.
check_nulls	If TRUE, then all level columns are verified to be non-null. If FALSE, this check is omitted. Specify FALSE when non-NULLness is guaranteed by other means, such as NOT NULL constraints.
statement_id	A client-supplied unique identifier to associate output rows with specific invocations of the procedure.

36

DBMS_DISTRIBUTED_TRUST_ADMIN

DBMS_DISTRIBUTED_TRUST_ADMIN procedures maintain the Trusted Servers List. Use these procedures to define whether a server is trusted. If a database is not trusted, Oracle refuses current user database links from the database.

This chapter contains the following topics:

Using DBMS_DISTRIBUTED_TRUST_ADMIN

Overview

Oracle uses local Trusted Servers Lists, along with enterprise domain membership lists stored in the enterprise LDAP directory service, to determine if another database is trusted. The LDAP directory service entries are managed with the Enterprise Security Manager Tool in Oracle Enterprise Manager.

Oracle considers another database to be "trusted" if it meets the following criteria:

1. It is in the same enterprise domain in the directory service as the local database.
2. The enterprise domain is marked as trusted in the directory service.
3. It is not listed as untrusted in the local Trusted Servers List. Current user database links will only be accepted from another database if both databases involved trust each other.

You can list a database server locally in the Trusted Servers List regardless of what is listed in the directory service. However, if you list a database that is not in the same domain as the local database, or if that domain is untrusted, the entry will have no effect.

This functionality is part of the Enterprise User Security feature of the Oracle Advanced Security Option.

Security Model

To execute DBMS_DISTRIBUTED_TRUST_ADMIN, the EXECUTE_CATALOG_ROLE role must be granted to the DBA. To select from the view TRUSTED_SERVERS, the SELECT_CATALOG_ROLE role must be granted to the DBA.

It is important to know whether all servers are trusted or not trusted. Trusting a particular server with the ALLOW_SERVER procedure does not have any effect if the database already trusts all databases, or if that database is already trusted. Similarly, denying a particular server with the DENY_SERVER procedure does not have any effect if the database already does not trust any database or if that database is already untrusted.

The procedures DENY_ALL and ALLOW_ALL delete all entries (in other words, server names) that are explicitly allowed or denied using the ALLOW_SERVER procedure or DENY_SERVER procedure respectively.

Examples

If you have not yet used the package DBMS_DISTRIBUTED_TRUST_ADMIN to change the trust listing, by default you trust all databases in the same enterprise domain if that domain it listed as trusted in the directory service:

```
SELECT * FROM TRUSTED_SERVERS;
TRUST     NAME
--------- --------------------
Trusted   All
```

Because all servers are currently trusted, you can execute the DENY_SERVER Procedure and specify that a particular server is not trusted:

```
EXECUTE DBMS_DISTRIBUTED_TRUST_ADMIN.DENY_SERVER
                           ('SALES.US.AMERICAS.ACME_AUTO.COM');
PL/SQL procedure successfully completed.

SELECT * FROM TRUSTED_SERVERS;
TRUST      NAME
--------- ----------------------------------------------
Untrusted SALES.US.AMERICAS.ACME_AUTO.COM
```

By executing the DENY_ALL Procedure, you can choose to not trust any database server:

```
EXECUTE DBMS_DISTRIBUTED_TRUST_ADMIN.DENY_ALL;

PL/SQL procedure successfully completed.

SELECT * FROM TRUSTED_SERVERS;

TRUST      NAME
--------- ----------------------------------------------
Untrusted All
```

The ALLOW_SERVER Procedure can be used to specify that one particular database is to be trusted:

```
EXECUTE DBMS_DISTRIBUTED_TRUST_ADMIN.ALLOW_SERVER
                             ('SALES.US.AMERICAS.ACME_AUTO.COM');
PL/SQL procedure successfully completed.

SELECT * FROM TRUSTED_SERVERS;
TRUST      NAME
--------- ----------------------------------------------
Trusted   SALES.US.AMERICAS.ACME_AUTO.COM
```

Summary of DBMS_DISTRIBUTED_TRUST_ADMIN Subprograms

Table 36–1 DBMS_DISTRIBUTED_TRUST_ADMIN Package Subprograms

Subprogram	Description
ALLOW_ALL Procedure on page 36-7	Empties the list and inserts a row indicating that all servers should be trusted
ALLOW_SERVER Procedure on page 36-8	Enables a specific server to be allowed access even though `deny` all is indicated in the list
DENY_ALL Procedure on page 36-9	Empties the list and inserts a row indicating that all servers should be untrusted
on page 36-10DENY_ SERVER Procedure	Enables a specific server to be denied access even though allow all is indicated in the list

ALLOW_ALL Procedure

This procedure empties the Trusted Servers List and specifies that all servers that are members of a trusted domain in an enterprise directory service and that are in the same domain are allowed access.

The view TRUSTED_SERVERS will show "TRUSTED ALL" indicating that the database trusts all servers that are currently trusted by the enterprise directory service.

Syntax

```
DBMS_DISTRIBUTED_TRUST_ADMIN.ALLOW_ALL;
```

Usage Notes

ALLOW_ALL only applies to servers listed as trusted in the enterprise directory service and in the same enterprise domain.

ALLOW_SERVER Procedure

This procedure ensures that the specified server is considered trusted (even if you have previously specified "`deny all`").

Syntax

```
DBMS_DISTRIBUTED_TRUST_ADMIN.ALLOW_SERVER (
   server IN VARCHAR2);
```

Parameters

Table 36–2 ALLOW_SERVER Procedure Parameters

Parameter	Description
`server`	Unique, fully-qualified name of the server to be trusted.

Usage Notes

If the Trusted Servers List contains the entry "`deny all`", then this procedure adds a specification indicating that a specific database (for example, `DBx`) is to be trusted.

If the Trusted Servers List contains the entry "`allow all`", and if there is no "`deny DBx`" entry in the list, then executing this procedure causes no change.

If the Trusted Servers List contains the entry "`allow all`", and if there is a "`deny DBx`" entry in the list, then that entry is deleted.

DENY_ALL Procedure

This procedure empties the Trusted Servers List and specifies that all servers are denied access.

The view TRUSTED_SERVERS will show "UNTRUSTED ALL" indicating that no servers are currently trusted.

Syntax

```
DBMS_DISTRIBUTED_TRUST_ADMIN.DENY_ALL;
```

DENY_SERVER Procedure

This procedure ensures that the specified server is considered untrusted (even if you have previously specified `allow all`).

Syntax

```
DBMS_DISTRIBUTED_TRUST_ADMIN.DENY_SERVER (
   server IN VARCHAR2);
```

Parameters

Table 36–3 DENY_SERVER Procedure Parameters

Parameter	Description
`server`	Unique, fully-qualified name of the server to be untrusted.

Usage Notes

If the Trusted Servers List contains the entry `allow all`, then this procedure adds an entry indicating that the specified database (for example, `DBx`) is not to be trusted.

If the Trusted Servers List contains the entry "`deny all`", and if there is no "`allow DBx`" entry in the list, then this procedure causes no change.

If the Trusted Servers List contains the entry "`deny all`", and if there is an "`allow DBx`" entry, then this procedure causes that entry to be deleted.

37

DBMS_EPG

The DBMS_EPG package implements the embedded PL/SQL gateway that enables a web browser to invoke a PL/SQL stored procedure through an HTTP listener.

This chapter contains the following topics:

Using DBMS_EPG

Overview

The DBMS_EPG package is a platform on which PL/SQL users develop and deploy PL/SQL web applications. The embedded PL/SQL gateway is an embedded version of the gateway that runs in the XML database HTTP server in the Oracle database. It provides the core features of mod_plsql in the database but does not require the Oracle HTTP server powered by Apache.

In order to make a PL/SQL application accessible from a browser via HTTP, a Database Access Descriptor (DAD) must be created and mapped to a virtual path. A DAD is a set of configuration values used for database access and the virtual path mapping makes the application accessible under a virtual path of the XML DB HTTP Server. A DAD is represented as a servlet in XML DB HTTP Server.

Security Model

The XDBADMIN role is required to invoke the configuration interface. It may invoked by the database user "XDB".

The authorization interface can be invoked by any user.

Exceptions

The following table lists the exceptions raised by the DBMS_EPG package.

Table 37–1 DBMS_EPG Exceptions

Exception	Error Code	Description
DAD_NOT_FOUND	20000	Database Access Descriptor (DAD) %s not found. Ensure that the name of the DAD is correct and that it exists.

Data Structures

The DBMS_EPG package defines a TABLE type.

VARCHAR2_TABLE Table Type

This type is used by the procedures GET_ALL_GLOBAL_ATTRIBUTES, GET_ALL_DAD_ATTRIBUTES, GET_ALL_DAD_MAPPINGS, and GET_DAD_LIST to return lists of attribute names, attribute values, virtual paths, and database access descriptors (DAD).

```
TYPE VARCHAR2_TABLE IS TABLE OF VARCHAR2(4000) INDEX BY BINARY_INTEGER;
```

Subprogram Groups

The DBMS_EPG consists of two interfaces:

- Configuration Subprograms
- Authorization Subprograms

Configuration Subprograms

The Configuration subprogram group contain the subprogram interfaces to examine and modify the global and database access descriptor (DAD) specific settings of the embedded PL/SQL gateway.

Table 37–2 Configuration Subprogram Group

Subprogram	Description
CREATE_DAD Procedure on page 37-12	Creates a new DAD
DELETE_DAD_ATTRIBUTE Procedure on page 37-14	Deletes a DAD attribute
DELETE_GLOBAL_ATTRIBUTE Procedure on page 37-15	Deletes a global attribute
DROP_DAD Procedure on page 37-16	Drops a DAD
GET_ALL_DAD_ATTRIBUTES Procedure on page 37-17	Retrieves all the attributes of a DAD.
GET_ALL_DAD_MAPPINGS Procedure on page 37-18	Retrieves all virtual paths to which the specified DAD is mapped.
GET_ALL_GLOBAL_ATTRIBUTES Procedure on page 37-19	Retrieves all global attributes and values
GET_DAD_ATTRIBUTE Function on page 37-20	Retrieves the value of a DAD attribute
GET_DAD_LIST Procedure on page 37-21	Retrieves a list of all DADs for an Embedded Gateway instance.
GET_GLOBAL_ATTRIBUTE Function on page 37-22	Retrieves the value of a global attribute
MAP_DAD Procedure on page 37-23	Maps a DAD to the specified virtual path.
SET_DAD_ATTRIBUTE Procedure on page 37-24	Sets the value for a DAD
SET_GLOBAL_ATTRIBUTE Procedure on page 37-27	Sets the value of a global attribute
UNMAP_DAD Procedure on page 37-28	Unmaps a DAD from the specified virtual path

Authorization Subprograms

The Authorization subprogram group contains the subprogram interfaces to authorize and deauthorize the use of a database user's privileges by the embedded PL/SQL gateway through a specific database access descriptor (DAD)

Table 37–3 Authorization Subprogram Group

Subprogram	Description
AUTHORIZE_DAD Procedure on page 37-11	Authorizes a DAD to invoke procedures and access document tables with a database user's privileges
DEAUTHORIZE_DAD Procedure on page 37-13	Deauthorizes a DAD with regard to invoking procedures and accessing document tables with a database user's privileges

Summary of DBMS_EPG Subprograms

Table 37–4 DBMS_ALERT Package Subprograms

Subprogram	Description
AUTHORIZE_DAD Procedure on page 37-11	authorizes a DAD to invoke procedures and access document tables with a database user's privileges
CREATE_DAD Procedure on page 37-12	Creates a new DAD
DEAUTHORIZE_DAD Procedure on page 37-13	Deauthorizes a DAD with regard to invoking procedures and accessing document tables with a database user's privileges
DELETE_DAD_ATTRIBUTE Procedure on page 37-14	Deletes a DAD attribute
DELETE_GLOBAL_ATTRIBUTE Procedure on page 37-15	Deletes a global attribute
DROP_DAD Procedure on page 37-16	Drops a DAD
GET_ALL_DAD_ATTRIBUTES Procedure on page 37-17	Retrieves all the attributes of a DAD.
GET_ALL_DAD_MAPPINGS Procedure on page 37-18	Retrieves all virtual paths to which the specified DAD is mapped.
GET_ALL_GLOBAL_ATTRIBUTES Procedure on page 37-19	Retrieves all global attributes and values
GET_DAD_ATTRIBUTE Function on page 37-20	Retrieves the value of a DAD attribute
GET_DAD_LIST Procedure on page 37-21	Retrieves a list of all DADs for an Embedded Gateway instance.
GET_GLOBAL_ATTRIBUTE Function on page 37-22	Retrieves the value of a global attribute
MAP_DAD Procedure on page 37-23	Maps a DAD to the specified virtual path.
SET_DAD_ATTRIBUTE Procedure on page 37-24	Sets the value for a DAD
SET_GLOBAL_ATTRIBUTE Procedure on page 37-27	Sets the value of a global attribute
UNMAP_DAD Procedure on page 37-28	Unmaps a DAD from the specified virtual path

AUTHORIZE_DAD Procedure

This procedure authorizes a DAD to invoke procedures and access document tables with a database user's privileges. The invoker can always authorize the use of her/his own privileges.

> **See Also:** Authorization Subprograms on page 37-9 for other subprograms in this group

Syntax

```
DBMS_EPG.AUTHORIZE_DAD (
   dad_name  IN  VARCHAR2,
   path      IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 37–5 AUTHORIZE_DAD Procedure Parameters

Parameter	Description
`dad_name`	The name of the DAD to create
`user`	The user whose privileges to deauthorize. If use, the invoker is assumed.

Usage Notes

- To authorize the use of another user's privileges, the invoker must have the `ALTER USER` system privilege.
- The DAD must exist but its "database-username" DAD attribute does not have to be set to user to authorize.
- Multiple users can authorize the same DAD and it is up to the DAD's "database-username" setting to decide which user's privileges to use.

Exceptions

Raises an error if the DAD or user does not exist, or the invoker does not have the needed system privilege.

Examples

```
DBMS_EPG.AUTHORIZE_DAD('HR');
```

CREATE_DAD Procedure

This procedure creates a new DAD.

> **See Also:** Configuration Subprograms on page 37-8 for other subprograms in this group

Syntax

```
DBMS_EPG.CREATE_DAD (
   dad_name  IN  VARCHAR2,
   path      IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 37–6 CREATE_DAD Procedure Parameters

Parameter	Description
dad_name	The name of the DAD to create
path	The virtual path to which to map the DAD

DEAUTHORIZE_DAD Procedure

This procedure deauthorizes a DAD with regard to invoking procedures and accessing document tables with a database user's privileges. The invoker can always deauthorize the use of his own privileges.

> **See Also:** Authorization Subprograms on page 37-9 for other subprograms in this group

Syntax

```
DBMS_EPG.DEAUTHORIZE_DAD (
   dad_name  IN  VARCHAR2,
   path      IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 37–7 DEAUTHORIZE_DAD Procedure Parameters

Parameter	Description
dad_name	The name of the DAD for which to deauthorize use
user	The user whose privileges to deauthorize. If use, the invoker is assumed.

Usage Notes

To deauthorize the use of another user's privileges, the invoker must have the ALTER USER system privilege.

Exceptions

Raises an error if the DAD or user does not exist, or the invoker does not have the needed system privilege.

Examples

```
DBMS_EPG.DEAUTHORIZE_DAD('HR');
```

DELETE_DAD_ATTRIBUTE Procedure

This procedure deletes a DAD attribute.

> **See Also:** Configuration Subprograms on page 37-8 for other subprograms in this group

Syntax

```
DBMS_EPG.DELETE_DAD_ATTRIBUTE (
   dad_name      IN  VARCHAR2,
   attr_name     IN  VARCHAR2);
```

Parameters

Table 37–8 DELETE_DAD_ATTRIBUTE Procedure Parameters

Parameter	Description
dad_name	The name of the DAD for which to delete a DAD attribute
attr_name	The name of the DAD attribute to delete

Exceptions

Raises an error if DAD does not exist

DELETE_GLOBAL_ATTRIBUTE Procedure

This procedure deletes a global attribute.

> **See Also:** Configuration Subprograms on page 37-8 for other subprograms in this group

Syntax

```
DBMS_EPG.DELETE_GLOBAL_ATTRIBUTE (
   attr_name     IN  VARCHAR2);
```

Parameters

Table 37–9 DELETE_GLOBAL_ATTRIBUTE Procedure Parameters

Parameter	Description
attr_name	The global attribute to delete

DROP_DAD Procedure

This procedure drops a DAD. All the virtual-path mappings of the DAD will be dropped also

> **See Also:** Configuration Subprograms on page 37-8 for other subprograms in this group

Syntax

```
DBMS_EPG.DROP_DAD (
   dadname  IN  VARCHAR2);
```

Parameters

Table 37–10 DROP_DAD Procedure Parameters

Parameter	Description
dad_name	The DAD to drop

Exceptions

Raises an error if the DAD does not exist.

GET_ALL_DAD_ATTRIBUTES Procedure

This procedure retrieves all the attributes of a DAD. The outputs are 2 correlated index-by tables of the name/value pairs.

> **See Also:** Configuration Subprograms on page 37-8 for other subprograms in this group

Syntax

```
DBMS_EPG.GET_ALL_DAD_ATTRIBUTES (
   dad_name      IN          VARCHAR2,
   attr_names    OUT NOCOPY  VARCHAR2_TABLE,
   attr_values   OUT NOCOPY  VARCHAR2_TABLE);
```

Parameters

Table 37–11 GET_ALL_DAD_ATTRIBUTES Procedure Parameters

Parameter	Description
dad_names	The name of the DAD
attr_names	The attribute names
attr_values	The attribute values

Exceptions

Raises an error if DAD does not exist.

Usage Notes

If the DAD has no attributes set, then attr_names and attr_values will be set to empty arrays.

GET_ALL_DAD_MAPPINGS Procedure

This procedure retrieves all virtual paths to which the specified DAD is mapped.

> **See Also:** Configuration Subprograms on page 37-8 for other subprograms in this group

Syntax

```
DBMS_EPG.GET_ALL_DAD_MAPPINGS (
   dad_name      IN          VARCHAR2,
   paths         OUT NOCOPY  VARCHAR2_TABLE);
```

Parameters

Table 37–12 GET_ALL_DAD_MAPPINGS Procedure Parameters

Parameter	Description
dad_names	The name of the DAD
paths	The virtual paths to which h the DAD is mapped

Exceptions

Raises an error if DAD does not exist.

Usage Notes

If the DAD is not mapped to any virtual path, paths will be set to empty arrays.

GET_ALL_GLOBAL_ATTRIBUTES Procedure

This procedure retrieves all global attributes and values. The outputs are 2 correlated index-by tables of the name/value pairs.

> **See Also:** Configuration Subprograms on page 37-8 for other subprograms in this group

Syntax

```
DBMS_EPG.GET_ALL_GLOBAL_ATTRIBUTES (
   attr_names     OUT   NOCOPY  VARCHAR2_TABLE,
   attr_values    OUT   NOCOPY  VARCHAR2_TABLE);
```

Parameters

Table 37–13 GET_ALL_GLOBAL_ATTRIBUTES Procedure Parameters

Parameter	Description
attr_names	The global attribute names
attr_values	The values of the global attributes

Usage Notes

If the gateway instance has no global attributes set, then attr_names and attr_values will be set to empty arrays.

GET_DAD_ATTRIBUTE Function

This procedure retrieves the value of a DAD attribute.

> **See Also:** Configuration Subprograms on page 37-8 for other subprograms in this group

Syntax

```
DBMS_EPG.GET_DAD_ATTRIBUTE (
   dad_name     IN  VARCHAR2,
   attr_name    IN  VARCHAR2)
 RETURN VARCHAR2;
```

Parameters

Table 37–14 GET_DAD_ATTRIBUTE Procedure Parameters

Parameter	Description
dad_names	The name of the DAD for which to delete an attribute
attr_name	The name of the attribute to delete

Return values

Returns the DAD attribute value. Returns NULL if attribute is unknown or has not been set.

Exceptions

Raises an error if DAD does not exist.

GET_DAD_LIST Procedure

This procedure retrieves a list of all DADs for an Embedded Gateway instance.

> **See Also:** Configuration Subprograms on page 37-8 for other subprograms in this group

Syntax

```
DBMS_EPG.GET_DAD_LIST (
   dad_names     OUT NOCOPY  VARCHAR2_TABLE);
```

Parameters

Table 37–15 GET_DAD_LIST Procedure Parameters

Parameter	Description
dad_names	The list of all DADs

Usage Notes

If no DADs exist then dad_names will be set to an empty array.

GET_GLOBAL_ATTRIBUTE Function

This function retrieves the value of a global attribute.

> **See Also:** Configuration Subprograms on page 37-8 for other subprograms in this group

Syntax

```
DBMS_EPG.GET_GLOBAL_ATTRIBUTE (
   attr_name  IN  VARCHAR2)
 RETURN VARCHAR2;
```

Parameters

Table 37–16 GET_GLOBAL_ATTRIBUTE Procedure Parameters

Parameter	Description
attr_name	The global attribute to retrieve

Return Values

Returns the global attribute value. Returns NULL if attribute has not been set or is not a valid attribute.

MAP_DAD Procedure

This procedure maps a DAD to the specified virtual path. If the virtual path exists already, the old virtual-path mapping will be overridden.

> **See Also:** Configuration Subprograms on page 37-8 for other subprograms in this group

Syntax

```
DBMS_EPG.MAP_DAD (
   dad_name  IN  VARCHAR2,
   path      IN  VARCHAR2);
```

Parameters

Table 37–17 MAP_DAD Procedure Parameters

Parameter	Description
dad_name	The name of the DAD to map
path	The virtual path to map

Exceptions

Raises and error if the DAD does not exist.

SET_DAD_ATTRIBUTE Procedure

This procedure sets the value for a DAD.

See Also: Configuration Subprograms on page 37-8 for other subprograms in this group

Syntax

```
DBMS_EPG.SET_DAD_ATTRIBUTE (
   dad_name    IN  VARCHAR2,
   attr_name   IN  VARCHAR2,
   attr_value  IN  VARCHAR2);
```

Parameters

Table 37–18 SET_DAD_ATTRIBUTE Procedure Parameters

Parameter	Description
dad_name	The name of the DAD for which to set the attribute
attr_name	The name of the attribute to set
attr_value	The attribute value to set

Table 37–19 Mapping Between mod_plsql and Embedded PL/SQL Gateway DAD Attributes

mod_plsql DAD Attribute	Embedded PL/SQL Gateway DAD Attribute	Allows Multiple Occurr-ences	Legal Values
PlsqlAfterProcedure	after-procedure	No	String
PlsqlAlwaysDescribeProcedure	always-describe-procedure	No	Enumeration of On, Off
PlsqlAuthenticationMode	authentication-mode	No	Enumeration of Basic, SingleSignOn, GlobalOwa, CustomOwa, PerPackageOwa
PlsqlBeforeProcedure	before-procedure	No	String
PlsqlBindBucketLengths	bind-bucket-lengths	Yes	Unsigned integer
PlsqlBindBucketWidths	bind-bucket-widths	Yes	Unsigned integer
PlsqlCGIEnvironmentList	cgi-environment-list	Yes	String
PlsqlCompatibilityMode	compatibility-mode	No	Unsigned integer
PlsqlDatabaseUsername	database-username	No	String
PlsqlDefaultPage	default-page	No	String
PlsqlDocumentPath	document-path	No	String
PlsqlDocumentProcedure	document-procedure	No	String
PlsqlDocumentTablename	document-table-name	No	String

Table 37–19 (Cont.) Mapping Between mod_plsql and Embedded PL/SQL Gateway DAD Attributes

mod_plsql DAD Attribute	Embedded PL/SQL Gateway DAD Attribute	Allows Multiple Occurr-ences	Legal Values
PlsqlErrorStyle	error-style	No	Enumeration of ApacheStyle, ModplsqlStyle, DebugStyle
PlsqlExclusionList	exclusion-list	Yes	String
PlsqlFetchBufferSize	fetch-buffer-size	No	Unsigned integer
PlsqlInputFilterEnable	input-filter-enable	No	Enumeration of On, Off
PlsqlInfoLogging	info-logging	No	Enumeration of InfoDebug
PlsqlOWADebugEnable	owa-debug-enable	No	Enumeration of On, Off
PlsqlMaxRequestsPerSession	max-requests-per-session	No	Unsigned integer
PlsqlNLSLanguage	nls-language	No	String
PlsqlPathAlias	path-alias	No	String
PlsqlPathAliasProcedure	path-alias-procedure	No	String
PlsqlRequestValidationFunction	request-validation-function	No	String
PlsqlSessionCookieName	session-cookie-name	No	String
PlsqlSessionStateManagement	session-state-management	No	Enumeration of StatelessWithResetPackageState, StatelessWithFastRestPackageState, StatelessWithPreservePackageState
PlsqlTransferMode	transfer-mode	No	Enumeration of Char, Raw
PlsqlUploadAsLongRaw	upload-as-long-raw	No	String

Exceptions

Raises an error if DAD does not exist or the attribute is unknown.

Usage Notes

- If `attr_name` attribute has been set before, then the old value will be overwritten with the new `attr_value` argument.
- The embedded gateway assumes default values when the attributes are not set. The default values of the DAD attributes should be sufficient for most users of the embedded gateway. mod_plsql users should note the following
  - The `PlsqlDatabasePassword` attribute is not needed.
  - The `PlsqlDatabaseConnectString` attribute is not needed because the embedded gateway does not support logon to external databases.

Examples

```
DBMS_EPG.SET_DAD_ATTRIBUTE('HR', 'default-page', 'HRApp.home');
```

SET_GLOBAL_ATTRIBUTE Procedure

This procedure sets the value of a global attribute.

> **See Also:** Configuration Subprograms on page 37-8 for other subprograms in this group

Syntax

```
DBMS_EPG.SET_GLOBAL_ATTRIBUTE (
   attr_name    IN VARCHAR2,
   attr_value   IN VARCHAR2);
```

Parameters

Table 37–20 SET_GLOBAL_ATTRIBUTE Procedure Parameters

Parameter	Description
attr_name	The global attribute to set
attr_value	The attribute value to set

Table 37–21 Mapping Between mod_plsql and Embedded PL/SQL Gateway Global Attributes

mod_plsql DAD Attribute	Embedded PL/SQL Gateway DAD Attribute	Allows Multiple Occurr-ences	Legal Values
PlsqlLogLevel	log-level	No	Unsigned integer
PlsqlMaxParameters	max-parameters	No	Unsigned integer

Usage Notes

- The attribute name is case sensitive. The value may or may not be case-sensitive depending on the attribute.
- If attr_name attribute has been set before, then the old value will be overwritten with the new attr_value argument.

Exceptions

Raises an error if the attribute is unknown.

Examples

```
dbms_epg.set_global_attribute('max-parameters', '100');
```

UNMAP_DAD Procedure

This procedure unmaps a DAD from the specified virtual path. If path is `NULL`, the procedure removes all virtual-path mappings for the DAD but keeps the DAD.

> **See Also:** Configuration Subprograms on page 37-8 for other subprograms in this group

Syntax

```
DBMS_EPG.UNMAP_DAD (
   dad_name IN VARCHAR2,
   path     IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 37–22 UNMAP_DAD Procedure Parameters

Parameter	Description
`dad_name`	The name of the DAD to unmap
`path`	The virtual path to unmap

Usage Notes

Raises and error if the DAD does not exist.

38

DBMS_ERRLOG

The DBMS_ERRLOG package provides a procedure that enables you to create an error logging table so that DML operations can continue after encountering errors rather than abort and roll back. This enables you to save time and system resources.

> **See Also:** *Oracle Database Data Warehousing Guide* for more information regarding how to use DBMS_ERRLOG and *Oracle Database SQL Reference* for error_logging_clause syntax

This chapter contains the following topics:

Using DBMS_ERRLOG

This section contains topics which relate to using the `DBMS_ERRLOG` package.

Security Model

Security on this package can be controlled by granting `EXECUTE` on this package to selected users or roles. The `EXECUTE` privilege is granted publicly. However, to create an error logging table, you need `SELECT` access on the base table or view, the `CREATE TABLE` privilege, as well as tablespace quota for the target tablespace.

Summary of DBMS_ERRLOG Subprograms

Table 38–1 DBMS_ERRLOG Package Subprograms

Subprogram	Description
CREATE_ERROR_LOG Procedure on page 38-5	Creates the error logging table used in DML error logging

CREATE_ERROR_LOG Procedure

This procedure creates the error logging table needed to use the DML error logging capability.

LONG, CLOB, BLOB, BFILE, and ADT datatypes are not supported in the columns.

Syntax

```
DBMS_ERRLOG.CREATE_ERROR_LOG (
   dml_table_name            IN VARCHAR2,
   err_log_table_name        IN VARCHAR2 := NULL,
   err_log_table_owner       IN VARCHAR2 := NULL,
   err_log_table_space       IN VARCHAR2 := NULL,
   skip_unsupported          IN BOOLEAN := FALSE);
```

Parameters

Table 38–2 CREATE_ERROR_LOG Procedure Parameters

Parameter	Description
dml_table_name	The name of the DML table to base the error logging table on. The name can be fully qualified (for example, emp, scott.emp, "EMP", "SCOTT"."EMP"). If a name component is enclosed in double quotes, it will not be upper cased.
err_log_table_name	The name of the error logging table you will create. The default is the first 25 characters in the name of the DML table prefixed with 'ERR$_'. Examples are the following: dml_table_name: 'EMP',err_log_table_name: 'ERR$_EMP' dml_table_name: '"Emp2"',err_log_table_name: 'ERR$_Emp2'
err_log_table_owner	The name of the owner of the error logging table. You can specify the owner in dml_table_name. Otherwise, the schema of the current connected user is used.
err_log_table_space	The tablespace the error logging table will be created in. If not specified, the default tablespace for the user owning the DML error logging table will be used.
skip_unsupported	When set to TRUE, column types that are not supported by error logging will be skipped over and not added to the error logging table. When set to FALSE, an unsupported column type will cause the procedure to terminate. The default is FALSE.

Examples

First, create an error log table for the channels table in the SH schema, using the default name generation.

Then, see all columns of the table channels:

```
SQL> DESC channels
Name                                      Null?    Type
----------------------------              -------  -----
CHANNEL_ID                                NOT NULL CHAR(1)
```

```
CHANNEL_DESC                            NOT NULL    VARCHAR2(20)
CHANNEL_CLASS                                       VARCHAR2(20)
```

Finally, see all columns of the generated error log table. Note the mandatory control columns that are created by the package:

```
SQL> DESC ERR$_CHANNELS
Name                            Null?    Type
------------------              ----     ----
ORA_ERR_NUMBER$                          NUMBER
ORA_ERR_MESG$                            VARCHAR2(2000)
ORA_ERR_ROWID$                           ROWID
ORA_ERR_OPTYP$                           VARCHAR2(2)
ORA_ERR_TAG$                             VARCHAR2(2000)
CHANNEL_ID                               VARCHAR2(4000)
CHANNEL_DESC                             VARCHAR2(4000)
CHANNEL_CLASS                            VARCHAR2(4000)
```

See *Oracle Database Administrator's Guide* for more information regarding control columns.

39

DBMS_EXPFIL

The DBMS_EXPFIL package contains all the procedures used to manage attribute sets, expression sets, expression indexes, optimizer statistics, and privileges by Expression Filter.

> **See Also:** *Oracle Database Application Developer's Guide - Rules Manager and Expression Filter* for more information.

This chapter contains the following topics:

- Summary of Expression Filter Subprograms

Summary of Expression Filter Subprograms

Table 39–1 describes the subprograms in the DBMS_EXPFIL package.

All the values and names passed to the procedures defined in the DBMS_EXPFIL package are not case sensitive, unless otherwise mentioned. To preserve the case, you use double quotation marks around the values.

Table 39–1 DBMS_EXPFIL Package Subprograms

Subprogram	Description
ADD_ELEMENTARY_ATTRIBUTE Procedures	Adds the specified attribute to the attribute set
ADD_FUNCTIONS Procedure	Adds a function, type, or package to the approved list of functions with an attribute set
ASSIGN_ATTRIBUTE_SET Procedure	Assigns an attribute set to a column storing expressions
BUILD_EXCEPTIONS_TABLE Procedure	Creates an exception table to hold references to invalid expressions
CLEAR_EXPRSET_STATS Procedure	Clears the predicate statistics for an expression set
COPY_ATTRIBUTE_SET Procedure	Makes a copy of the attribute set
CREATE_ATTRIBUTE_SET Procedure	Creates an attribute set
DEFAULT_INDEX_PARAMETERS Procedure	Assigns default index parameters to an attribute set
DEFAULT_XPINDEX_PARAMETERS Procedure	Assigns default XPath index parameters to an attribute set
DEFRAG_INDEX Procedure	Rebuilds the bitmap indexes online to reduce fragmentation
DROP_ATTRIBUTE_SET Procedure	Drops an unused attribute set
GET_EXPRSET_STATS Procedure	Collects predicate statistics for an expression set
GRANT_PRIVILEGE Procedure	Grants an expression DML privilege to a user
INDEX_PARAMETERS Procedure	Assigns index parameters to an expression set
MODIFY_OPERATOR_LIST Procedure	Modifies the list of common operators used in predicates with a certain attribute
REVOKE_PRIVILEGE Procedure	Revokes an expression DML privilege from a user
UNASSIGN_ATTRIBUTE_SET Procedure	Breaks the association between a column storing expressions and the attribute set
VALIDATE_EXPRESSIONS Procedure	Validates expression metadata and the expressions stored in a column
XPINDEX_PARAMETERS Procedure	Assigns XPath index parameters to an expression set

ADD_ELEMENTARY_ATTRIBUTE Procedures

This procedure adds the specified attribute to the attribute set. The procedure is overloaded. The different functionality of each form of syntax is presented along with the definitions.

Syntax

Adds the specified elementary attribute to the attribute set:

```
DBMS_EXPFIL.ADD_ELEMENTARY_ATTRIBUTE (
   attr_set   IN   VARCHAR2,
   attr_name  IN   VARCHAR2,
   attr_type  IN   VARCHAR2,
   attr_defv1 IN   VARCHAR2 DEFAULT NULL);
```

Identifies the elementary attributes that are table aliases and adds them to the attribute set:

```
DBMS_EXPFIL.ADD_ELEMENTARY_ATTRIBUTE (
   attr_set   IN   VARCHAR2,
   attr_name  IN   VARCHAR2,
   tab_alias  IN   exf$table_alias);
```

Parameters

Table 39–2 ADD_ELEMENTARY_ATTRIBUTE Procedure Parameters

Parameter	Description
attr_set	Name of the attribute set to which this attribute is added
attr_name	Name of the elementary attribute to be added. No two attributes in a set can have the same name.
attr_type	Datatype of the attribute. This argument accepts any standard SQL datatype or the name of an object type that is accessible to the current user.
attr_defv1	Default value for the elementary attribute
tab_alias	The type that identifies the database table to which the attribute is aliased

Usage Notes

- This procedure adds an elementary attribute to an attribute set. If the attribute set was originally created from an existing object type, then additional attributes cannot be added.
- One or more, or all elementary attributes in an attribute set can be table aliases. If an elementary attribute is a table alias, then the value assigned to the elementary attribute is a ROWID from the corresponding table. An attribute set with one or more table alias attributes cannot be created from an existing object type. For more information about table aliases, see Appendix A in *Oracle Database Application Developer's Guide - Rules Manager and Expression Filter*.
- Elementary attributes cannot be added to an attribute set that is already assigned to a column storing expressions.
- The default value specification for an attribute is similar to a default value specification for a table column. The resulting default values should agree with the

datatype of the attribute. For example, valid default values for an attribute of `DATE` datatype are `SYSDATE` and `to_date('01-01-2004','DD-MM-YYYY')`.

- See "Defining Attribute Sets" in *Oracle Database Application Developer's Guide - Rules Manager and Expression Filter* for more information about adding elementary attributes.
- Related views: `USER_EXPFIL_ATTRIBUTE_SETS` and `USER_EXPFIL_ATTRIBUTES`.

Examples

The following commands add two elementary attributes to an attribute set:

```
BEGIN
  DBMS_EXPFIL.ADD_ELEMENTARY_ATTRIBUTE (
                              attr_set   => 'HRAttrSet',
                              attr_name  => 'HRREP',
                              attr_type  => 'VARCHAR2(30)'
                              attr_defv1 => 'Betty Smith');
  DBMS_EXPFIL.ADD_ELEMENTARY_ATTRIBUTE (
                              attr_set  => 'HRAttrSet',
                              attr_name => 'DEPT',
                              tab_alias => exf$table_alias('DEPT'));
END;
```

The following commands define a CreationTime elementary attribute that takes the database time as the default value.

```
BEGIN
  DBMS_EXPFIL.ADD_ELEMENTARY_ATTRIBUTE (
                         attr_set   => 'PurchaseOrder',
                         attr_name  => 'CreationTime',
                         attr_type  => 'DATE',
                         attr_defvl => 'SYSDATE');
END;
```

Alternately, the following commands initialize the CreationTime attribute to a specific value when it is not explicitly specified in the data item passed to the EVALUATE operator.

```
BEGIN
  DBMS_EXPFIL.ADD_ELEMENTARY_ATTRIBUTE (
                         attr_set   => 'PurchaseOrder',
                         attr_name  => 'CreationTime',
                         attr_type  => 'DATE',
                         attr_defvl => 'to_date(''01-01-2004'',''DD-MM-YYYY'')');
END;
```

ADD_FUNCTIONS Procedure

This procedure adds a user-defined function, package, or type representing a set of functions to the attribute set.

Syntax

```
DBMS_EXPFIL.ADD_FUNCTIONS (
   attr_set   IN   VARCHAR2,
   funcs_name IN   VARCHAR2);
```

Parameters

Table 39–3 ADD_FUNCTIONS Procedure Parameters

Parameter	Description
attr_set	Name of the attribute set to which the functions are added
funcs_name	Name of a function, package, or type (representing a function set) or its synonyms

Usage Notes

- By default, an attribute set implicitly allows references to all Oracle supplied SQL functions for use by the expression set. If the expression set refers to a user-defined function, the function must be explicitly added to the attribute set.
- The ADD_FUNCTIONS procedure adds a user-defined function or a package (or type) representing a set of functions to the attribute set. Any new or modified expressions are validated using this list. The function added to the attribute set, and thus used in the stored expressions, should not perform any DML or DDL (database state changing) operations. Any violations to this rule will only be caught at run-time while evaluating the expressions (this implies that this will not be checked during the ADD_FUNCTIONS procedure call).
- The function or the package name can be specified with a schema extension. If a function name is specified without a schema extension, only such references in the expression set are considered valid. The expressions in a set can be restricted to use a synonym to a function or a package by adding the corresponding synonym to the attribute set. This preserves the portability of the expression set to other schemas.
- See "Defining Attribute Sets"in *Oracle Database Application Developer's Guide - Rules Manager and Expression Filter* for more information about adding functions to an attribute set.
- Related views: USER_EXPFIL_ATTRIBUTE_SETS and USER_EXPFIL_EXPRESSION_SETS

Examples

The following commands add two functions to the attribute set:

```
BEGIN
  DBMS_EXPFIL.ADD_FUNCTIONS (
   attr_set   => 'Car4Sale',
   funcs_name => 'HorsePower');
  DBMS_EXPFIL.ADD_FUNCTIONS (
   attr_set   => 'Car4Sale',
```

```
   funcs_name => 'Scott.CrashTestRating');
END;
```

ASSIGN_ATTRIBUTE_SET Procedure

This procedure assigns an attribute set to a VARCHAR2 column in a user table to create an Expression column.

Syntax

```
DBMS_EXPFIL.ASSIGN_ATTRIBUTE_SET (
   attr_set   IN   VARCHAR2,
   expr_tab   IN   VARCHAR2,
   expr_col   IN   VARCHAR2,
   force      IN   VARCHAR2 DEFAULT 'FALSE');
```

Parameters

Table 39–4 ASSIGN_ATTRIBUTE_SET Procedure Parameters

Parameter	Description
attr_set	The name of the attribute set
expr_tab	The table storing the expression set
expr_col	The column in the table that stores the expressions
force	Argument used to trust the existing expressions in a table (and skip validation)

Usage Notes

- The ASSIGN_ATTRIBUTE_SET procedure assigns an attribute set to a VARCHAR2 column in a user table to create an Expression column. The attribute set contains the elementary attribute names and their datatypes and any functions used in the expressions. The attribute set is used by the Expression column to validate changes and additions to the expression set.
- An attribute set can be assigned only to a table column in the same schema as the attribute set. An attribute set can be assigned to one or more table columns. Assigning an attribute set to a column storing expressions implicitly creates methods for the associated object type. For this operation to succeed, the object type cannot have any dependent objects before the attribute set is assigned.
- By default, the column should not have any expressions at the time of association. However, if the values in the column are known to be valid expressions, you can use a value of 'TRUE' for the force argument to assign the attribute set to a column containing expressions.
- See "Defining Expression Columns" in *Oracle Database Application Developer's Guide - Rules Manager and Expression Filter* for more information about adding elementary attributes.
- Related views: USER_EXPFIL_ATTRIBUTE_SETS and USER_EXPFIL_EXPRESSION_SETS

Examples

The following command assigns the attribute set to a column storing expressions. The expression set should be empty at the time of association.

```
BEGIN
  DBMS_EXPFIL.ASSIGN_ATTRIBUTE_SET (attr_set => 'Car4Sale',
```

```
                                        expr_tab => 'Consumer',
                                        expr_col => 'Interest');
END;
```

BUILD_EXCEPTIONS_TABLE Procedure

This procedure creates the exception table, used in validation, in the current schema.

Syntax

```
DBMS_EXPFIL.BUILD_EXCEPTIONS_TABLE (
   exception_tab IN VARCHAR2);
```

Parameters

Table 39–5 BUILD_EXCEPTIONS_TABLE Procedure Parameter

Parameter	Description
exception_tab	The name of the exception table

Usage Notes

- The expressions stored in a table column can be validated using the VALIDATE_EXPRESSIONS procedure. During expression validation, you can optionally provide the name of the exception table in which the references to the invalid expressions are stored. The BUILD_EXCEPTIONS_TABLE procedure creates the exception table in the current schema.
- See "Evaluation Semantics" in *Oracle Database Application Developer's Guide - Rules Manager and Expression Filter* and VALIDATE_EXPRESSIONS Procedure in this chapter for more information.
- Related view: USER_TABLES

Examples

The following command creates the exception table, InterestExceptions, in the current schema:

```
BEGIN
  DBMS_EXPFIL.BUILD_EXCEPTIONS_TABLE (
                              exception_tab => 'InterestExceptions');
END;
```

CLEAR_EXPRSET_STATS Procedure

This procedure clears the predicate statistics for the expression set stored in a table column.

Syntax

```
DBMS_EXPFIL.CLEAR_EXPRSET_STATS (
   expr_tab   IN   VARCHAR2,
   expr_col   IN   VARCHAR2);
```

Parameters

Table 39–6 CLEAR_EXPRSET_STATS Procedure Parameters

Parameter	Description
expr_tab	The table storing the expression set
expr_col	The column in the table that stores the expressions

Usage Notes

- This procedure clears the predicate statistics for the expression set stored in a table column. See also GET_EXPRSET_STATS Procedure in this chapter for information about gathering the statistics.
- Related views: USER_EXPFIL_EXPRESSION_SETS and USER_EXPFIL_EXPRSET_STATS

Examples

The following command clears the predicate statistics for the expression set stored in Interest column of the Consumer table:

```
BEGIN
  DBMS_EXPFIL.CLEAR_EXPRSET_STATS (expr_tab => 'Consumer',
                                   expr_col => 'Interest');
END;
```

COPY_ATTRIBUTE_SET Procedure

This procedure copies an attribute set along with its user-defined function list and default index parameters to another set.

Syntax

```
DBMS_EXPFIL.COPY_ATTRIBUTE_SET (
   from_set   IN   VARCHAR2,
   to_set     IN   VARCHAR2);
```

Parameters

Table 39–7 COPY_ATTRIBUTE_SET Procedure Parameters

Parameter	Description
from_set	Name of an existing attribute set to be copied
to_set	Name of the new attribute set

Usage Notes

- A schema-extended name can be used for the from_set argument to copy an attribute set across schemas. The user issuing the command must have EXECUTE privileges for the object type associated with the original attribute set. The user must ensure that any references to schema objects (user-defined functions, tables, and embedded objects) are valid in the new schema.
- The default index parameters and the user-defined function list of the new set can be changed independent of the original set.
- Related views: ALL_EXPFIL_ATTRIBUTE_SETS and ALL_EXPFIL_ATTRIBUTES.

Examples

The following command makes a copy of the Car4Sale attribute set:

```
BEGIN
  DBMS_EXPFIL.COPY_ATTRIBUTE_SET (from_set => 'Car4Sale',
                                  to_set   => 'Vehicle');
END;
```

CREATE_ATTRIBUTE_SET Procedure

This procedure creates an empty attribute set or an attribute set with a complete set of elementary attributes derived from an object type with a matching name.

Syntax

```
DBMS_EXPFIL.CREATE_ATTRIBUTE_SET (
   attr_set   IN   VARCHAR2,
   from_type  IN   VARCHAR2 DEFAULT 'NO');
```

Parameters

Table 39–8 CREATE_ATTRIBUTE_SET Procedure Parameters

Parameter	Description
attr_set	The name of the attribute set to be created
from_type	YES, if the attributes for the attribute set should be derived from an existing object type

Usage Notes

- The object type used for an attribute set cannot contain any user methods, and it should not be an evolved type (with the use of ALTER TYPE command). This object type should not have any dependent objects at the time of the attribute set creation. If the attribute set is not derived from an existing object type, this procedure creates an object type with a matching name.
- An attribute set with one or more table alias attributes cannot be derived from an object type. For this purpose, create an empty attribute set and add one elementary attribute at a time using the DBMS_EXPFIL.ADD_ELEMENTARY_ATTRIBUTE procedure. (See Appendix A in *Oracle Database Application Developer's Guide - Rules Manager and Expression Filter* for more information.)
- See "Defining Attribute Sets" in *Oracle Database Application Developer's Guide - Rules Manager and Expression Filter* and ADD_ELEMENTARY_ATTRIBUTE Procedures in this chapter for more information.
- Related views: USER_EXPFIL_ATTRIBUTE_SETS and USER_EXPFIL_ATTRIBUTES.

Examples

The following commands create an attribute set with all the required elementary attributes derived from the Car4Sale type:

```
CREATE OR REPLACE TYPE Car4Sale AS OBJECT
                                  (Model   VARCHAR2(20),
                                   Year    NUMBER,
                                   Price   NUMBER,
                                   Mileage NUMBER);


BEGIN
  DBMS_EXPFIL.CREATE_ATTRIBUTE_SET(attr_set  => 'Car4Sale',
                                  from_type => 'YES');
END;
```

Assuming that the `Car4Sale` type does not exist, the attribute set can be created from scratch as shown in the following example:

```
BEGIN
  DBMS_EXPFIL.CREATE_ATTRIBUTE_SET(attr_set => 'Car4Sale');
  DBMS_EXPFIL.ADD_ELEMENTARY_ATTRIBUTE(
                               attr_set  => 'Car4Sale',
                               attr_name => 'Model',
                               attr_type => 'VARCHAR2(20)');
  DBMS_EXPFIL.ADD_ELEMENTARY_ATTRIBUTE(
                               attr_set  => 'Car4Sale',
                               attr_name => 'Year',
                               attr_type => 'NUMBER');
  DBMS_EXPFIL.ADD_ELEMENTARY_ATTRIBUTE(
                               attr_set  => 'Car4Sale',
                               attr_name => 'Price',
                               attr_type => 'NUMBER');
  DBMS_EXPFIL.ADD_ELEMENTARY_ATTRIBUTE(
                               attr_set  => 'Car4Sale',
                               attr_name => 'Mileage',
                               attr_type => 'NUMBER');
END;
```

DEFAULT_INDEX_PARAMETERS Procedure

This procedure assigns default index parameters to an attribute set. It also adds or drops a partial list of stored and indexed attributes to or from the default list associated with the attribute list.

Syntax

```
DBMS_EXPFIL.DEFAULT_INDEX_PARAMETERS (
   attr_set   IN   VARCHAR2,
   attr_list  IN   EXF$ATTRIBUTE_LIST,
   operation  IN   VARCHAR2 DEFAULT 'ADD');
```

Parameters

Table 39–9 DEFAULT_INDEX_PARAMETERS Procedure Parameters

Parameter	Description
attr_set	The name of the attribute set
attr_list	An instance of EXF$ATTRIBUTE_LIST with a partial list of (default) stored and indexed attributes for an Expression Filter index
operation	The operation to be performed on the list of index parameters. Default value: ADD. Valid values: ADD and DROP.

Usage Notes

- Existing Expression Filter indexes are not modified when the default parameters for the corresponding attribute set are changed. The new index defaults are used when a new Expression Filter index is created and when an existing index is rebuilt. (See "Alter Index Rebuild" in *Oracle Database Application Developer's Guide - Rules Manager and Expression Filter* for more information about rebuilding indexes.)
- See "Creating an Index from Default Parameters" in *Oracle Database Application Developer's Guide - Rules Manager and Expression Filter* for more information about assigning default index parameters to an attribute set.
- Related views: USER_EXPFIL_ATTRIBUTE_SETS and USER_EXPFIL_DEF_INDEX_PARAMS

Examples

The following command adds the specified stored and indexed attributes to the attribute set's default index parameters list:

```
BEGIN
 DBMS_EXPFIL.DEFAULT_INDEX_PARAMETERs(
    attr_set  => 'Car4Sale',
    attr_list => exf$attribute_list (
       exf$attribute (attr_name => 'Model',
                      attr_oper => exf$indexoper('='),
                      attr_indexed => 'TRUE'),
       exf$attribute (attr_name => 'Price',
                      attr_oper => exf$indexoper('all'),
                      attr_indexed => 'TRUE'),
       exf$attribute (attr_name => 'HorsePower(Model, Year)',
                      attr_oper => exf$indexoper('=','<','>','>=','<='),
                      attr_indexed => 'FALSE'),
       exf$attribute (attr_name => 'CrashTestRating(Model, Year)',
```

```
                    attr_oper => exf$indexoper('=','<','>','>=','<='),
                    attr_indexed => 'FALSE')),
    operation => 'ADD');
END;
```

The following command drops the `CrashTestRating(Model, Year)` attribute (stored or indexed) from the previous list.

```
BEGIN
  DBMS_EXPFIL.DEFAULT_INDEX_PARAMETERS(
      attr_set  => 'Car4Sale',
      attr_list => exf$attribute_list (
         exf$attribute (attr_name => 'CrashTestRating(Model, Year)')),
      operation => 'DROP');
END;
```

DEFAULT_XPINDEX_PARAMETERS Procedure

This procedure adds (or drops) a partial list of XPath parameters to the default index parameters associated with the attribute set.

Syntax

```
DBMS_EXPFIL.DEFAULT_XPINDEX_PARAMETERS (
   attr_set   IN   VARCHAR2,
   xmlt_attr  IN   VARCHAR2,
   xptag_list IN   EXF$XPATH_TAGS,
   operation  IN   VARCHAR2 DEFAULT 'ADD');
```

Parameters

Table 39–10 DEFAULT_XPINDEX_PARAMETERS Procedure Parameters

Parameter	Description
attr_set	The name of the attribute set
xmlt_attr	The name of the attribute with the XMLType datatype
xptag_list	An instance of EXF$XPATH_TAGS type with a partial list of XML elements and attributes to be configured for the Expression Filter index
operation	The operation to be performed on the list of index parameters. Default value: ADD. Valid values: ADD and DROP.

Usage Notes

- The attribute set used for an expression set may have one or more XML type attributes (defined with XMLType datatype) and the corresponding expressions may contain XPath predicates on these attributes. The Expression Filter index created for the expression set can be tuned to process these XPath predicates efficiently by using some XPath-specific index parameters (in addition to some non-XPath index parameters).
- The DEFAULT_XPINDEX_PARAMETERS procedure adds (or drops) a partial list of XPath parameters to the default index parameters associated with the attribute set. The XPath parameters are assigned to a specific XMLType attribute in the attribute set and this information can be viewed using the USER_EXPFIL_DEF_INDEX_PARAMS view. The DEFAULT_INDEX_PARAMETERS procedure and the DEFAULT_XPINDEX_PARAMETERS procedure can be used independent of each other. They maintain a common list of default index parameters for the attribute set.
- See "Index Tuning for XPath Predicates"in *Oracle Database Application Developer's Guide - Rules Manager and Expression Filter* for more information about XPath parameters to the default index parameters of an attribute set. See also DEFAULT_INDEX_PARAMETERS Procedure in this chapter for more information about default index parameters.
- Related views: USER_EXPFIL_ATTRIBUTES and USER_EXPFIL_DEF_INDEX_PARAMS.

> **Note:** The values assigned to the tag_name argument of exf$xpath_tag type are case sensitive.

Examples

The following command adds the specified XML tags to the default index parameters list along with their preferences such as positional or value filter and indexed or stored predicate group:

```
BEGIN
  DBMS_EXPFIL.DEFAULT_XPINDEX_PARAMETERS(
       attr_set   => 'Car4Sale',
       xmlt_attr  => 'Details',
       xptag_list =>                                    --- XPath tag list
         exf$xpath_tags(
           exf$xpath_tag(tag_name    => 'stereo@make',  --- XML attribute
                         tag_indexed => 'TRUE',
                         tag_type    => 'VARCHAR(15)'), --- value filter
           exf$xpath_tag(tag_name    => 'stereo',       --- XML element
                         tag_indexed => 'FALSE',
                         tag_type    => null),          --- positional filter
           exf$xpath_tag(tag_name    => 'memory',       --- XML element
                         tag_indexed => 'TRUE',
                         tag_type    => 'VARCHAR(10)'), --- value filter
           exf$xpath_tag(tag_name    => 'GPS',
                         tag_indexed => 'TRUE',
                         tag_type    => null)
         )
        );
END;
```

The following command drops the `stereo@make` tag from the default index parameters:

```
BEGIN
  DBMS_EXPFIL.DEFAULT_XPINDEX_PARAMETERS(
       attr_set   => 'Car4Sale',
       xmlt_attr  => 'Details',
       xptag_list =>                                    --- XPath tag list
         exf$xpath_tags(
           exf$xpath_tag(tag_name    => 'stereo@make')
         ),
       operation => 'DROP'
       );
END;
```

DEFRAG_INDEX Procedure

This procedure rebuilds the bitmap indexes online and thus reduces the fragmentation.

Syntax

```
DBMS_EXPFIL.DEFRAG_INDEX (
   idx_name   IN  VARCHAR2);
```

Parameters

Table 39–11 DEFRAG_INDEX Procedure Parameter

Parameter	Description
idx_name	The name of the Expression Filter index

Usage Notes

- The bitmap indexes defined for the indexed attributes of an Expression Filter index become fragmented as additions and updates are made to the expression set. The DEFRAG_INDEX procedure rebuilds the bitmap indexes online and thus reduces the fragmentation.
- Indexes can be defragmented when the expression set is being modified. However, you should schedule defragmentation when the workload is relatively light.
- See "Index Storage and Maintenance" in *Oracle Database Application Developer's Guide - Rules Manager and Expression Filter* for more information about rebuilding indexes.
- Related views: USER_EXPFIL_INDEXES and USER_INDEXES.

Examples

The following command is issued to defragment the bitmap indexes associated with the Expression Filter index:

```
BEGIN
  DBMS_EXPFIL.DEFRAG_INDEX (idx_name => 'InterestIndex');
END;
```

DROP_ATTRIBUTE_SET Procedure

This procedure drops an attribute set not being used for any expression set.

Syntax

```
DBMS_EXPFIL.DROP_ATTRIBUTE_SET (
   attr_set   IN   VARCHAR2);
```

Parameters

Table 39–12 DROP_ATTRIBUTE_SET Procedure Parameter

Parameter	Description
attr_set	The name of the attribute set to be dropped

Usage Notes

- The DROP_ATTRIBUTE_SET procedure drops an attribute set not being used for any expression set. If the attribute set was initially created from an existing object type, the object type remains after dropping the attribute set. Otherwise, the object type is dropped with the attribute set.
- Related views: USER_EXPFIL_ATTRIBUTE_SETS and USER_EXPFIL_EXPRESSION_SETS.

Examples

Assuming that the attribute set is not used by an Expression column, the following command drops the attribute set:

```
BEGIN
  DBMS_EXPFIL.DROP_ATTRIBUTE_SET(attr_set => 'Car4Sale');
END;
```

GET_EXPRSET_STATS Procedure

This procedure computes the predicate statistics for an expression set and stores them in the expression filter dictionary.

Syntax

```
DBMS_EXPFIL.GET_EXPRSET_STATS (
   expr_tab   IN   VARCHAR2,
   expr_col   IN   VARCHAR2);
```

Parameters

Table 39–13 GET_EXPRSET_STATS Procedure Parameters

Parameter	Description
`expr_tab`	The table storing the expression set
`expr_col`	The column in the table that stores the expressions

Usage Notes

- When a representative set of expressions are stored in a table column, you can use predicate statistics for those expressions to configure the corresponding Expression Filter index (using the `TOP` parameters clause). The `GET_EXPRSET_STATS` procedure computes the predicate statistics for an expression set and stores them in the expression filter dictionary.
- See "Creating an Index from Statistics"in *Oracle Database Application Developer's Guide - Rules Manager and Expression Filter* for more information about using predicate statistics.
- Related views: `USER_EXPFIL_EXPRESSION_SETS` and `USER_EXPFIL_EXPRSET_STATS`.

Examples

The following command computes the predicate statistics for the expressions stored in the `Interest` column of the `Consumer` table:

```
BEGIN
  DBMS_EXPFIL.GET_EXPRSET_STATS (expr_tab => 'Consumer',
                                 expr_col => 'Interest');
END;
```

GRANT_PRIVILEGE Procedure

This procedure grants privileges on one or more Expression columns to other users.

Syntax

```
DBMS_EXPFIL.GRANT_PRIVILEGE (
   expr_tab   IN  VARCHAR2,
   expr_col   IN  VARCHAR2,
   priv_type  IN  VARCHAR2,
   to_user    IN  VARCHAR2);
```

Parameters

Table 39–14 GRANT_PRIVILEGE Procedure Parameters

Parameter	Description
expr_tab	The table storing the expression set
expr_col	The column in the table that stores the expressions
priv_type	Type of the privilege to be granted. Valid values: INSERT EXPRESSION, UPDATE EXPRESSION, ALL.
to_user	The user to whom the privilege is to be granted

Usage Notes

- The SQL EVALUATE operator evaluates expressions with the privileges of the owner of the table that stores the expressions. The privileges of the user issuing the query are not considered. The owner of the table can insert, update, and delete expressions. Other users must have INSERT and UPDATE privileges for the table and INSERT EXPRESSION and UPDATE EXPRESSION privilege for a specific Expression column in the table.
- Using the GRANT_PRIVILEGE procedure, the owner of the table can grant INSERT EXPRESSION or UPDATE EXPRESSION privileges on one or more Expression columns to other users. Both the privileges can be granted to a user by specifying ALL for the privilege type.
- See REVOKE_PRIVILEGE Procedure in this chapter and "Granting and Revoking Privileges" in *Oracle Database Application Developer's Guide - Rules Manager and Expression Filter* for more information about granting and revoking privileges.
- Related views: USER_EXPFIL_EXPRESSION_SETS and USER_EXPFIL_PRIVILEGES.

Examples

The owner of Consumer table can grant INSERT EXPRESSION privileges to user SCOTT with the following command. User SCOTT should also have INSERT privileges on the table so that he can add new expressions to the set.

```
BEGIN
  DBMS_EXPFIL.GRANT_PRIVILEGE (expr_tab  => 'Consumer',
                               expr_col  => 'Interest',
                               priv_type => 'INSERT EXPRESSION',
                               to_user   => 'SCOTT');
END;
```

INDEX_PARAMETERS Procedure

This procedure fine-tunes the index parameters for each expression set before index creation.

Syntax

```
DBMS_EXPFIL.INDEX_PARAMETERS (
   expr_tab   IN   VARCHAR2,
   expr_col   IN   VARCHAR2,
   attr_list  IN   EXF$ATTRIBUTE_LIST,
   operation  IN   VARCHAR2 DEFAULT 'ADD');
```

Parameters

Table 39–15 INDEX_PARAMETERS Procedure Parameters

Parameter	Description
expr_tab	The table storing the expression set
expr_col	The column in the table that stores the expressions.
attr_list	An instance of EXF$ATTRIBUTE_LIST with a partial list of stored and indexed attributes
operation	The operation to be performed on the list of index parameters. Default value: ADD. Valid values: ADD and DROP.

Usage Notes

- An attribute set can be used by multiple expression sets stored in different columns of user tables. By default, the index parameters associated with the attribute set are used to define an Expression Filter index on an expression set. If you need to fine-tune the index for each expression set, you can specify a small list of the index parameters in the PARAMETERS clause of the CREATE INDEX statement. However, when an Expression Filter index uses a large number of index parameters or if the index is configured for XPath predicates, fine-tuning the parameters with the CREATE INDEX statement is not possible.
- The INDEX_PARAMETERS procedure fine-tunes the index parameters for each expression set before index creation. This procedure can be used to copy the defaults from the corresponding attribute set and selectively add (or drop) additional index parameters for the expression set. (You use the XPINDEX_PARAMETERS procedure to add and drop XPath index parameters.) The Expression Filter index defined for an expression set with a non-empty list of index parameters always uses these parameters. The INDEX_PARAMETERS procedure cannot be used when the Expression Filter index is already defined for the column storing expressions.
- The operations allowed with this procedure include:
  - Deriving the current list of default index parameters (including any XPath-specific parameters) from the corresponding attribute set and assigning them to the specified expression set (a value of DEFAULT for the operation argument).
  - Adding (or dropping) one or more attributes to (or from) the current list of parameters assigned to the expression set (values of ADD or DROP for the operation argument).

– Clearing the index parameters assigned to the expression set. This enables the user to start using default parameters or tune the parameters from scratch (a value of CLEAR for the operation argument).

Note: This procedure is useful only when an attribute set is shared across multiple expression sets. In all other cases, the defaults assigned to the attribute set can be tuned for the expression set using it.

- See "Creating an Index from Exact Parameters" in *Oracle Database Application Developer's Guide - Rules Manager and Expression Filter* and XPINDEX_PARAMETERS Procedure in this chapter for more information.
- Related views: USER_EXPFIL_EXPRESSION_SETS, USER_EXPFIL_DEF_INDEX_PARAMS and USER_EXPFIL_INDEX_PARAMS.

Examples

The following command synchronizes the expression set's index parameters with the defaults associated with the corresponding attribute set:

```
BEGIN
  DBMS_EXPFIL.INDEX_PARAMETERS(expr_tab  => 'Consumer',
                               expr_col  => 'Interest',
                               attr_list => null,
                               operation => 'DEFAULT');
END;
```

The following command adds a stored attribute to the expression set's index parameters.

```
BEGIN
  DBMS_EXPFIL.INDEX_PARAMETERS(expr_tab  => 'Consumer',
                               expr_col  => 'Interest',
                               attr_list =>
                                exf$attribute_list (
                                 exf$attribute (
                                   attr_name => 'CrashTestRating(Model, Year)',
                                   attr_oper => exf$indexoper('all'),
                                   attr_indexed => 'FALSE')),
                               operation => 'ADD');
END;
```

The following command clears the index parameters associated with the expression set:

```
BEGIN
  DBMS_EXPFIL.INDEX_PARAMETERS(expr_tab  => 'Consumer',
                               expr_col  => 'Interest',
                               attr_list => null,
                               operation => 'CLEAR');
END;
```

A subsequent index creation will use the default index parameters assigned to the corresponding attribute set.

MODIFY_OPERATOR_LIST Procedure

This procedure modifies the list of common operators associated with a certain attribute in the attribute set.

Syntax

```
DBMS_EXPFIL.MODIFY_OPERATOR_LIST (
   attr_set   IN   VARCHAR2,
   attr_name  IN   VARCHAR2,
   attr_oper  IN   EXF$INDEXOPER);
```

Parameters

Table 39–16 MODIFY_OPERATOR_LIST Procedure Parameters

Parameter	Description
attr_set	The name of the attribute set
attr_name	The name of the stored or indexed attribute being modified
attr_oper	The new list of operators that are frequently used in the predicates with the attribute

Usage Notes

- The MODIFY_OPERATOR_LIST procedure modifies the operator list for the stored and indexed attributes defined in the attribute set's default index parameters. Existing Expression Filter indexes are not affected when an attribute's operator list is modified. The updated index defaults are used when a new Expression Filter index is created or when an existing index is rebuilt.
- Related views: USER_EXPFIL_DEF_INDEX_PARAMS

Examples

The following command modifies the operator list associated with the HorsePower(Model,Year) attribute defined in the Car4Sale attribute set.

```
BEGIN
  DBMS_EXPFIL.MODIFY_OPERATOR_LIST (
    attr_set => 'Car4Sale',
    attr_name => 'HorsePower(Model, Year)',
    attr_oper => exf$indexoper('=','<','>', 'between'));
END;
```

REVOKE_PRIVILEGE Procedure

This procedure revokes an expression privilege previously granted by the owner.

Syntax

```
DBMS_EXPFIL.REVOKE_PRIVILEGE (
   expr_tab   IN  VARCHAR2,
   expr_col   IN  VARCHAR2,
   priv_type  IN  VARCHAR2,
   from_user  IN  VARCHAR2);
```

Parameters

Table 39–17 REVOKE_PRIVILEGE Procedure Parameters

Parameter	Description
expr_tab	The table storing the expression set
expr_col	The column in the table that stores the expressions
priv_type	Type of privilege to be revoked
from_user	The user from whom the privilege is to be revoked

Usage Notes

- The REVOKE_PRIVILEGE procedure revokes an expression privilege previously granted by the owner.
- See GRANT_PRIVILEGE Procedure in this chapter and "Granting and Revoking Privileges" in *Oracle Database Application Developer's Guide - Rules Manager and Expression Filter* for more information about granting and revoking privileges.
- Related views: USER_EXPFIL_EXPRESSION_SETS and USER_EXPFIL_PRIVILEGES.

Examples

The following command revokes the INSERT EXPRESSION privilege on the Interest column of the Consumer table from user SCOTT:

```
BEGIN
  DBMS_EXPFIL.REVOKE_PRIVILEGE (expr_tab  => 'Consumer',
                                 expr_col  => 'Interest',
                                 priv_type => 'INSERT EXPRESSION',
                                  from_user => 'SCOTT');
END;
```

UNASSIGN_ATTRIBUTE_SET Procedure

This procedure unassigns an attribute set from a column storing expressions.

Syntax

```
DBMS_EXPFIL.UNASSIGN_ATTRIBUTE_SET (
   expr_tab   IN   VARCHAR2,
   expr_col   IN   VARCHAR2);
```

Parameters

Table 39–18 UNASSIGN_ATTRIBUTE_SET Procedure Parameters

Parameter	Description
`expr_tab`	The table storing the expression set
`expr_col`	The column in the table that stores the expressions

Usage Notes

- A column of an expression datatype can be converted back to a `VARCHAR2` type by unassigning the attribute set. You can unassign an attribute set from a column storing expressions if an Expression Filter index is not defined on the column.
- See ASSIGN_ATTRIBUTE_SET Procedure in this chapter for information about assigning attribute sets.
- Related views: `USER_EXPFIL_EXPRESSION_SETS` and `USER_EXPFIL_INDEXES`.

Examples

The following command unassigns the attribute set previously assigned to the `Interest` column of the `Consumer` table. (See "Bulk Loading of Expression Data" in *Oracle Database Application Developer's Guide - Rules Manager and Expression Filter*.)

```
BEGIN
  DBMS_EXPFIL.UNASSIGN_ATTRIBUTE_SET (expr_tab => 'Consumer',
                                      expr_col => 'Interest');
END;
```

VALIDATE_EXPRESSIONS Procedure

This procedure validates all the expressions in a set.

Syntax

```
DBMS_EXPFIL.VALIDATE_EXPRESSIONS (
   expr_tab      IN  VARCHAR2,
   expr_col      IN  VARCHAR2,
   exception_tab IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 39–19 VALIDATE_EXPRESSIONS Procedure Parameters

Parameter	Description
expr_tab	The table storing the expression set
expr_col	The column in the table that stores the expressions
exception_tab	The name of the exception table. This table is created using the BUILD_EXCEPTIONS_TABLE procedure.

Usage Notes

- The expressions stored in a table may have references to schema objects like user-defined functions and tables. When these schema objects are dropped or modified, the expressions could become invalid and the subsequent evaluation (query with EVALUATE operator) could fail.
- The VALIDATE_EXPRESSIONS procedure validates all the expressions in a set. By default, the expression validation utility fails on the first expression that is invalid. Optionally, the caller can pass an exception table to store references to all the invalid expressions. In addition to validating expressions in the set, this procedure validates the parameters (stored and indexed attributes) of the associated index and the approved list of user-defined functions. Any errors in the index parameters or the user-defined function list are immediately reported to the caller.
- See "Evaluation Semantics" in *Oracle Database Application Developer's Guide - Rules Manager and Expression Filter* and BUILD_EXCEPTIONS_TABLE Procedure in this chapter for more information.
- Related views: USER_EXPFIL_EXPRESSION_SETS, USER_EXPFIL_ASET_FUNCTIONS, and USER_EXPFIL_PREDTAB_ATTRIBUTES.

Examples

The following command validates the expressions stored in the Interest column of the Consumer table.

```
BEGIN
  DBMS_EXPFIL.VALIDATE_EXPRESSIONS (expr_tab => 'Consumer',
                                    expr_col => 'Interest');
END;
```

XPINDEX_PARAMETERS Procedure

This procedure is used in conjunction with the INDEX_PARAMETERS procedure to fine-tune the XPath-specific index parameters for each expression set.

Syntax

```
DBMS_EXPFIL.XPINDEX_PARAMETERS (
   expr_tab   IN   VARCHAR2,
   expr_col   IN   VARCHAR2,
   xmlt_attr  IN   VARCHAR2,
   xptag_list IN   EXF$XPATH_TAGS,
   operation  IN   VARCHAR2 DEFAULT 'ADD');
```

Parameters

Table 39–20 XPINDEX_PARAMETERS Procedure Parameters

Parameter	Description
exp_tab	The table storing the expression set
expr_col	The column in the table that stores the expressions
xmlt_attr	The name of the attribute with the XMLType datatype
xptag_list	An instance of EXF$XPATH_TAGS type with a partial list of XML elements and attributes
operation	The operation to be performed on the list of index parameters. Default value: ADD. Valid values: ADD and DROP.

Usage Notes

- When an attribute set is shared by multiple expression sets, the INDEX_PARAMETERS procedure can be used to tune the simple (non-XPath) index parameters for each expression set. The XPINDEX_PARAMETERS procedure is used in conjunction with the INDEX_PARAMETERS procedure to fine-tune the XPath-specific index parameters for each expression set.
- See also INDEX_PARAMETERS Procedure in this chapter and "Index Tuning for XPath Predicates" in *Oracle Database Application Developer's Guide - Rules Manager and Expression Filter* for more information.
- Related views: USER_EXPFIL_ATTRIBUTES, USER_EXPFIL_DEF_INDEX_PARAMS, and USER_EXPFIL_INDEX_PARAMS.

> **Note:** The values assigned to the tag_name argument of exf$xpath_tag type are case-sensitive.

Examples

The following command synchronizes the expression set's index parameters (XPath and non-XPath) with the defaults associated with the corresponding attribute set:

```
BEGIN
  DBMS_EXPFIL.INDEX_PARAMETERS(expr_tab  => 'Consumer',
                               expr_col  => 'Interest',
                               attr_list => null,
                               operation => 'DEFAULT');
```

```
END;
```

The following command adds an XPath-specific index parameter to the expression set:

```
BEGIN
  DBMS_EXPFIL.XPINDEX_PARAMETERS(expr_tab   => 'Consumer',
                                 expr_col   => 'Interest',
                                 xmlt_attr  => 'Details',
                                 xptag_list =>
                                  exf$xpath_tags(
                                    exf$xpath_tag(tag_name    => 'GPS',
                                                  tag_indexed => 'TRUE',
                                                  tag_type    => NULL)),
                                 operation  => 'ADD');
END;
```

40

DBMS_FGA

The `DBMS_FGA` package provides fine-grained security functions.

This chapter contains the following topics:

Using DBMS_FGA

Security Model

Execute privilege on `DBMS_FGA` is needed for administering audit policies. Because the audit function can potentially capture all user environment and application context values, policy administration should be executable by privileged users only. The policy event handler module will be executed with the module owner's privilege.

Operational Notes

This package is available for only cost-based optimization. The rule-based optimizer may generate unnecessary audit records since audit monitoring can occur before row filtering. For both the rule-based optimizer and the cost-based optimizer, you can refer to `DBA_FGA_AUDIT_TRAIL` to analyze the SQL text and corresponding bind variables that are issued.

Summary of DBMS_FGA Subprograms

Table 40–1 DBMS_FGA Package Subprograms

Subprogram	Description
ADD_POLICY Procedure on page 40-6	Creates an audit policy using the supplied predicate as the audit condition
DISABLE_POLICY Procedure on page 40-11	Disables an audit policy
DROP_POLICY Procedure on page 40-12	Drops an audit policy
ENABLE_POLICY Procedure on page 40-13	Enables an audit policy

ADD_POLICY Procedure

This procedure creates an audit policy using the supplied predicate as the audit condition. The maximum number of FGA policies on any table or view object is 256.

Syntax

```
DBMS_FGA.ADD_POLICY(
   object_schema      VARCHAR2,
   object_name        VARCHAR2,
   policy_name        VARCHAR2,
   audit_condition    VARCHAR2,
   audit_column       VARCHAR2,
   handler_schema     VARCHAR2,
   handler_module     VARCHAR2,
   enable             BOOLEAN,
   statement_types    VARCHAR2,
   audit_trail        BINARY_INTEGER IN DEFAULT,
   audit_column_opts  BINARY_INTEGER IN DEFAULT);
```

Parameters

Table 40–2 ADD_POLICY Procedure Parameters

Parameter	Description	Default Value
object_schema	The schema of the object to be audited. (If NULL, the current log-on user schema is assumed.)	NULL
object_name	The name of the object to be audited.	-
policy_name	The unique name of the policy.	-
audit_condition	A condition in a row that indicates a monitoring condition. NULL is allowed and acts as TRUE.	NULL
audit_column	The columns to be checked for access. These can include hidden columns. The default, NULL, causes audit if any column is accessed or affected.	NULL
handler_schema	The schema that contains the event handler. The default, NULL, causes the current schema to be used.	NULL
handler_module	The function name of the event handler; includes the package name if necessary. This function is invoked only after the first row that matches the audit condition in the query is processed. If the procedure fails with an exception, the user SQL statement will fail as well.	NULL
enable	Enables the policy if TRUE, which is the default.	TRUE
statement_types	The SQL statement types to which this policy is applicable: INSERT, UPDATE, DELETE, or SELECT only.	SELECT
audit_trail	Destination (DB or XML) of fine grained audit records. Also specifies whether to populate LSQLTEXT and LSQLBIND in fga_log$.	DB+EXTENDED
audit_column_opts	Establishes whether a statement is audited when the query references *any* column specified in the audit_column parameter or only when *all* such columns are referenced.	ANY_COLUMNS

Usage Notes

- If object_schema is not specified, the current log-on user schema is assumed.

- An FGA policy should not be applied to out-of-line columns such as LOB columns.
- Each audit policy is applied to the query individually. However, at most one audit record may be generated for each policy, no matter how many rows being returned satisfy that policy's `audit_condition`. In other words, whenever any number of rows being returned satisfy an audit condition defined on the table, a single audit record will be generated for each such policy.
- If a table with an FGA policy defined on it receives a Fast Path insert or a vectored update, the hint is automatically disabled before any such operations. Disabling the hint allows auditing to occur according to the policy's terms. (One example of a Fast Path insert is the statement `INSERT-WITH-APPEND-hint`.)
- The `audit_condition` must be a boolean expression that can be evaluated using the values in the row being inserted, updated, or deleted. This condition can be NULL (or omitted), which is interpreted as `TRUE`, but it cannot contain the following elements:
  - Subqueries or sequences
  - Any direct use of `SYSDATE, UID, USER or USERENV` functions. However, a user-defined function and other SQL functions can use these functions to return the desired information.
  - Any use of the pseudo columns `LEVEL, PRIOR, or ROWNUM`.

  Specifying an audit condition of "1=1" to force auditing of all specified statements ("`statement_types`") affecting the specified column ("`audit_column`") is no longer needed to achieve this purpose. `NULL` will cause audit even if no rows were processed, so that all actions on a table with this policy are audited.
- The audit function (`handler_module`) is an alerting mechanism for the administrator. The required interface for such a function is as follows:

  ```
  PROCEDURE <fname> ( object_schema VARCHAR2, object_name VARCHAR2, policy_name
  VARCHAR2 )  AS ...
  ```

  where `fname` is the name of the procedure, `object_schema` is the name of the schema of the table audited, `object_name` is the name of the table to be audited, and `policy_name` is the name of the policy being enforced. The audit function will be executed with the function owner's privilege.
- The `audit_trail` parameter specifies both where the fine-grained audit trail will be written and whether it is to include the query's SQL Text and SQL Bind variable information (typically in columns named `LSQLTEXT` and `LSQLBIND`):
  - If audit_trail includes XML, then fine-grained audit records are written to XML-format operating system files stored in the directory specified by an `AUDIT_FILE_DEST` statement in SQL. (The default `AUDIT_FILE_DEST` is `$ORACLE_BASE/admin/$DB_UNIQUE_NAME/adump` on Unix-based systems, and `$ORACLE_BASE\admin\$DB_UNIQUE_NAME\adump` on Windows systems.)
  - If audit_trail includes DB instead, then the audit records are written to the `SYS.FGA_LOG$` table in the database.
  - If audit_trail includes `EXTENDED`, then the query's SQL Text and SQL Bind variable information are included in the audit trail.
  - For example:

  * Setting audit_trail to DBMS_FGA.DB sends the audit trail to the SYS.FGA_LOG$ table in the database and omits SQL Text and SQL Bind.
  * Setting audit_trail to DBMS_FGA.DB + DBMS_FGA.EXTENDED sends the audit trail to the SYS.FGA_LOG$ table in the database and includes SQL Text and SQL Bind.
  * Setting audit_trail to DBMS_FGA.XML writes the audit trail in XML files sent to the operating system and omits SQL Text and SQL Bind.
  * Setting audit_trail to DBMS_FGA.XML + DBMS_FGA.EXTENDED writes the audit trail in XML files sent to the operating system and includes SQL Text and SQL Bind.

  The audit_trail parameter appears in the ALL_AUDIT_POLICIES view.

- You can change the operating system destination using the following command:

  ```
  ALTER SYSTEM SET AUDIT_FILE_DEST = '<New Directory>' DEFERRED
  ```

- On many platforms, XML audit files are named <process_name>_<processId>.xml, for example, ora_2111.xml, or s002_11.xml. On Windows, the XML audit files are named <process_name>_<ThreadId>.xml (or <process_name>_ProcessId>.xml if the process is not running as a thread).
- The audit_column_opts parameter establishes whether a statement is audited
  - when the query references *any* column specified in the audit_column parameter (audit_column_opts = DBMS_FGA.ANY_COLUMNS), or
  - only when *all* such columns are referenced (audit_column_opts = DBMS_FGA.ALL_COLUMNS).

  The default is DBMS_FGA.ANY_COLUMNS.

  The ALL_AUDIT_POLICIES view also shows audit_column_opts.
- When audit_column_opts is set to DBMS_FGA.ALL_COLUMNS, a SQL statement is audited only when all the columns mentioned in audit_column have been explicitly referenced in the statement. And these columns must be referenced in the same SQL-statement or in the sub-select.

  Also, all these columns must refer to a single table/view or alias.

  Thus, if a SQL statement selects the columns from different table aliases, the statement will not be audited.

V$XML_AUDIT_TRAIL View

The new values for the audit_trail parameter (XML and XML+EXTENDED) cause fine-grained auditing records to be written to operating system files in XML format.

Audit records stored in operating system files can be more secure than database-stored audit records because access can require file permissions that DBAs do not have. Operating system storage for audit records also offers higher availability, since such records remain available even if the database is temporarily inaccessible.

A new dynamic view, V$XML_AUDIT_TRAIL, makes such audit records from XML files available to DBAs through SQL query, providing enhanced usability. Querying this view causes all XML files (all files with an .xml extension) in the AUDIT_FILE_DEST directory to be parsed and presented in relational table format.

The DBA_COMMON_AUDIT_TRAIL view includes the contents of the V$XML_AUDIT_TRAIL dynamic view for standard and fine-grained audit records.

Since the audit XML files are stored in files with extension .xml on all platforms, the dynamic view presents audit information similarly on all platforms, using the following schema:

Table 40–3 Elements in the V$XML_AUDIT_TRAIL Dynamic View

Element	Type
AUDIT_TYPE	VARCHAR2(18)
SESSION_ID	NUMBER
PROXY_SESSIONID	NUMBER
STATEMENTID	NUMBER
ENTRYID	NUMBER
EXTENDED_TIMESTAMP	TIMESTAMP(6) WITH TIME ZONE
GLOBAL_UID	VARCHAR2(32)
DB_USER	VARCHAR2(30)
CLIENT_ID	VARCHAR2(64)
EXT_NAME	VARCHAR2(4000)
OS_USER	VARCHAR2(255)
USERHOST	VARCHAR2(128)
OS_PROCESS	VARCHAR2(16)
TERMINAL	VARCHAR2(255)
INSTANCE_NUMBER	NUMBER
OBJECT_SCHEMA	VARCHAR2(30)
OBJECT_NAME	VARCHAR2(128)
POLICY_NAME	VARCHAR2(30)
STATEMENT_TYPE	VARCHAR2(28)
TRANSACTIONID	RAW(8)
SCN	NUMBER
COMMENT_TEXT	VARCHAR2(4000)
SQL_BIND	VARCHAR2(4000)
SQL_TEXT	VARCHAR2(4000)

Usage Notes

- Every XML audit record contains the elements AUDIT_TYPE and EXTENDED_TIMESTAMP, with the latter printed in UTC zone (with no timezone information). Values retrieved using V$XML_AUDIT_TRAIL view are converted to session timezone and printed.
- For SQL_TEXT and SQL_BIND element values (CLOB type columns), the dynamic view shows only the first 4000 characters. The underlying XML file may have more than 4000 characters for such SQL_TEXT and SQL_BIND values.
- For large numbers of XML audit files, querying V$XML_AUDIT_TRAIL is faster when they are loaded into a database table using SQL*Loader or a similar tool. XML audit files are larger than the equivalent written to OS files when AUDIT_TRAIL=OS.

- Error handling is the same as when AUDIT_TRAIL=OS. If any error occurs in writing an audit record to disk, including the directory identified by AUDIT_FILE_DEST being full, the auditing operation fails. An alert message is logged.
- The policy event handler module will be executed with the module owner's privilege.

Examples

```
DBMS_FGA.ADD_POLICY (
   object_schema      =>  'scott',
   object_name        =>  'emp',
   policy_name        =>  'mypolicy1',
   audit_condition    =>  'sal < 100',
   audit_column       =>  'comm,sal',
   handler_schema     =>   NULL,
   handler_module     =>   NULL,
   enable             =>   TRUE,
   statement_types    =>  'INSERT, UPDATE',
   audit_trail        =>   DBMS_FGA.XML + DBMS_FGA.EXTENDED,
   audit_column_opts  =>   DBMS_FGA.ANY_COLUMNS);
```

DISABLE_POLICY Procedure

This procedure disables an audit policy.

Syntax

```
DBMS_FGA.DISABLE_POLICY(
   object_schema  VARCHAR2,
   object_name    VARCHAR2,
   policy_name    VARCHAR2 );
```

Parameters

Table 40–4 DISABLE_POLICY Procedure Parameters

Parameter	Description
object_schema	The schema of the object to be audited. (If NULL, the current log-on user schema is assumed.)
object_name	The name of the object to be audited.
policy_name	The unique name of the policy.

The default value for object_schema is NULL. (If NULL, the current log-on user schema is assumed.)

Examples

```
DBMS_FGA.DISABLE_POLICY (
object_schema   =>  'scott',
object_name     =>  'emp',
policy_name     =>  'mypolicy1');
```

DROP_POLICY Procedure

This procedure drops an audit policy.

Syntax

```
DBMS_FGA.DROP_POLICY(
   object_schema  VARCHAR2,
   object_name    VARCHAR2,
   policy_name    VARCHAR2 );
```

Parameters

Table 40–5 DROP_POLICY Procedure Parameters

Parameter	Description
`object_schema`	The schema of the object to be audited. (If `NULL`, the current log-on user schema is assumed.)
`object_name`	The name of the object to be audited.
`policy_name`	The unique name of the policy.

Usage Notes

The `DBMS_FGA` procedures cause current DML transactions, if any, to commit before the operation unless they are inside a DDL event trigger. With DDL transactions, the `DBMS_FGA` procedures are part of the DDL transaction. The default value for object_schema is `NULL`. (If `NULL`, the current log-on user schema is assumed.)

Examples

```
DBMS_FGA.DROP_POLICY (
object_schema   =>  'scott',
object_name     =>  'emp',
policy_name     =>  'mypolicy1');
```

ENABLE_POLICY Procedure

This procedure enables an audit policy.

Syntax

```
DBMS_FGA.ENABLE_POLICY(
   object_schema  VARCHAR2,
   object_name    VARCHAR2,
   policy_name    VARCHAR2,
   enable         BOOLEAN);
```

Parameters

Table 40–6 ENABLE_POLICY Procedure Parameters

Parameter	Description
`object_schema`	The schema of the object to be audited. (If `NULL`, the current log-on user schema is assumed.)
`object_name`	The name of the object to be audited.
`policy_name`	The unique name of the policy.
`enable`	Defaults to `TRUE` to enable the policy.

Examples

```
DBMS_FGA.ENABLE_POLICY (
object_schema    =>  'scott',
object_name      =>  'emp',
policy_name      =>  'mypolicy1',
enable           =>   TRUE);
```

41

DBMS_FILE_GROUP

The DBMS_FILE_GROUP package, one of a set of Streams packages, provides administrative interfaces for managing file groups, file group versions, and files. A file group repository is a collection of all of the file groups in a database and can contain multiple versions of a particular file group. This package can be used to create and manage file group repositories.

See Also: *Oracle Streams Concepts and Administration*

This chapter contains the following topics:

Using DBMS_FILE_GROUP

This section contains topics which relate to using the `DBMS_FILE_GROUP` package.

Overview

The following terms pertain to the DBMS_FILE_GROUP package:

File

A **file** is a reference to a file stored on hard disk. A file is composed of a file name, a directory object, and a file type. The directory object references the directory in which the file is stored on hard disk. For example, a file might have the following components:

- The file name is expdat.dmp.
- The directory object that contains the file is db_files.
- The file type is DBMS_FILE_GROUP.EXPORT_DUMP_FILE.

Version

A **version** is a collection of related files. For example, a version might consist of a set of datafiles and a Data Pump export dump file generated by a Data Pump transportable tablespace export. Only one Data Pump export dump file is allowed in a version.

File Group

A **file group** is a collection of versions. A file group can be used to logically group a set of versions. For example, a file group named financial_quarters can keep track of quarterly financial data by logically grouping versions of files related to a tablespace set. The tablespaces containing the data can be exported at the end of each quarter and versioned under names such as Q1FY04, Q2FY04, and so on.

Constants

The DBMS_FILE_GROUP package defines several enumerated constants that should be used for specifying parameter values. Enumerated constants must be prefixed with the package name. For example, specify DBMS_FILE_GROUP.EXPORT_DUMP_FILE for an export dump file.

Table 41–1 DBMS_FILE_GROUP Parameters with Enumerated Constants

Parameter	Enumerated Constant	Type	Description
file_type new_file_type	■ DATAFILE ■ EXPORT_DUMP_FILE ■ DATAPUMP_LOG_FILE	VARCHAR2(30)	DATAFILE is a data file for a database. This constant can be specified as 'DATAFILE'. EXPORT_DUMP_FILE is a Data Pump export dump file. This constant can be specified as 'DUMPSET'. DATAPUMP_LOG_FILE is a Data Pump export log file. This constant can be specified as 'DATAPUMPLOG'.
max_versions retention_days	■ INFINITE	NUMBER	INFINITE specifies no limit. The max_versions or retention_days can increase without reaching a limit.
privilege	System privilege specified in the GRANT_SYSTEM_PRIVILEGE procedure: ■ READ_ANY_FILE_GROUP ■ MANAGE_ANY_FILE_GROUP ■ MANAGE_FILE_GROUP Object privilege specified in the GRANT_OBJECT_PRIVILEGE procedure: ■ READ_ON_FILE_GROUP ■ MANAGE_ON_FILE_GROUP	BINARY_INTEGER	READ_ANY_FILE_GROUP grants the privilege to view information about any file group in any schema in the data dictionary. MANAGE_ANY_FILE_GROUP grants the privilege to create, manage, and drop any file group in any schema. MANAGE_FILE_GROUP grants the privilege to create, manage, and drop file groups in the user's schema. READ_ON_FILE_GROUP grants the privilege to view information about a specific file group in the data dictionary. MANAGE_ON_FILE_GROUP grants the privilege to manage a specific file group in a schema other than the user's schema.

Summary of DBMS_FILE_GROUP Subprograms

Table 41–2 DBMS_FILE_GROUP Package Subprograms

Subprogram	Description
ADD_FILE Procedure on page 41-6	Adds a file to a version of a file group
ALTER_FILE Procedure on page 41-8	Alters a file in a version of a file group
ALTER_FILE_GROUP Procedure on page 41-10	Alters a file group
ALTER_VERSION Procedure on page 41-12	Alters a version of a file group
CREATE_FILE_GROUP Procedure on page 41-14	Creates a file group
CREATE_VERSION Procedure on page 41-16	Creates a version of a file group
DROP_FILE_GROUP Procedure on page 41-17	Drops a file group
DROP_VERSION Procedure on page 41-18	Drops a version of a file group
GRANT_OBJECT_PRIVILEGE Procedure on page 41-19	Grants object privileges on a file group to a user
GRANT_SYSTEM_PRIVILEGE Procedure on page 41-20	Grants system privileges for file group operations to a user
PURGE_FILE_GROUP Procedure on page 41-21	Purges a file group using the file group's retention policy
REMOVE_FILE Procedure on page 41-22	Removes a file from a version of a file group
REVOKE_OBJECT_PRIVILEGE Procedure on page 41-23	Revokes object privileges on a file group from a user
REVOKE_SYSTEM_PRIVILEGE Procedure on page 41-24	Revokes system privileges for file group operations from a user

Note: All subprograms commit unless specified otherwise.

ADD_FILE Procedure

This procedure adds a file to a version of a file group.

Syntax

```
DBMS_FILE_GROUP.ADD_FILE(
  file_group_name  IN  VARCHAR2,
  file_name        IN  VARCHAR2,
  file_type        IN  VARCHAR2  DEFAULT NULL,
  file_directory   IN  VARCHAR2  DEFAULT NULL,
  version_name     IN  VARCHAR2  DEFAULT NULL,
  comments         IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 41–3 ADD_FILE Procedure Parameters

Parameter	Description
`file_group_name`	The name of the file group that contains the version, specified as *`[schema_name.]file_group_name`*. For example, if the schema is `hq_dba` and the file group name is `sales_tbs`, then specify `hq_dba.sales_tbs`. If the schema is not specified, then the current user is the default.
`file_name`	The name of the file being added to the version. Each file name in a version must be unique.

`file_type`	The file type. The following are reserved file types:
	■ If the file is a datafile, then enter the following:
	`'DATAFILE'`
	■ If the file is a Data Pump export dump file, then enter the following:
	`'DUMPSET'`
	Data Pump metadata is populated when a Data Pump export dump file is imported.
	■ If the file is a Data Pump export log file, then enter the following:
	`'DATAPUMPLOG'`
	If the file type is not one of the reserved file types, then either enter a text description of the file type, or specify `NULL` to omit a file type description.
	See "Constants" on page 41-4 for more information about the reserved file types.

`file_directory`	The name of the directory object that corresponds to the directory containing the file. If `NULL`, then the procedure uses the default directory object for the version. If `NULL` and no default directory object exists for the version, then the procedure uses the default directory object for the file group. If `NULL` and no default directory object exists for the version or file group, then the procedure raises an error.

Table 41–3 (Cont.) ADD_FILE Procedure Parameters

Parameter	Description
version_name	The name of the version to which the file is added. If a positive integer is specified as a VARCHAR2 value, then the integer is interpreted as a version number. For example, if '1' is specified, then the file is added to version 1 of the file group. If NULL, then the procedure uses the version with the latest creation time for the file group.
comments	Comments about the file being added

Usage Notes

To run this procedure with either DBMS_FILE_GROUP.EXPORT_DUMP_FILE or 'DUMPSET' specified for the file_type parameter, a user must meet the following requirements:

- Have the appropriate privileges to import the Data Pump export dump file
- Have READ privilege on the directory object that contains the Data Pump export dump file

> **See Also:** *Oracle Database Utilities* for more information about Data Pump privileges

ALTER_FILE Procedure

This procedure alters a file in a version of a file group.

Syntax

```
DBMS_FILE_GROUP.ALTER_FILE(
  file_group_name     IN  VARCHAR2,
  file_name           IN  VARCHAR2,
  version_name        IN  VARCHAR2  DEFAULT NULL,
  new_file_name       IN  VARCHAR2  DEFAULT NULL,
  new_file_directory  IN  VARCHAR2  DEFAULT NULL,
  new_file_type       IN  VARCHAR2  DEFAULT NULL,
  remove_file_type    IN  VARCHAR2  DEFAULT 'N',
  new_comments        IN  VARCHAR2  DEFAULT NULL,
  remove_comments     IN  VARCHAR2  DEFAULT 'N');
```

Parameters

Table 41–4 ALTER_FILE Procedure Parameters

Parameter	Description
file_group_name	The name of the file group that contains the version, specified as *[schema_name.]file_group_name*. For example, if the schema is hq_dba and the file group name is sales_tbs, then specify hq_dba.sales_tbs. If the schema is not specified, then the current user is the default.
file_name	The name of the file being altered in the version
version_name	The name of the version that contains the file being altered. If a positive integer is specified as a VARCHAR2 value, then the integer is interpreted as a version number. For example, if '1' is specified, then the file in version 1 of the file group is altered. If NULL, then the procedure uses the version with the latest creation time for the file group.
new_file_name	The new name of the file if the file name is being changed. Each file name in a version must be unique. If NULL, then the procedure does not change the file name. **Note:** When a non-NULL new file name is specified, this procedure changes the metadata for the file name in the data dictionary, but it does not change the file name on the hard disk.
new_file_directory	The new name of the directory object that corresponds to the directory containing the file, if the directory object is being changed. If NULL, then the procedure does not change the directory object name. **Note:** When a non-NULL new file directory is specified, this procedure changes the metadata for the file directory in the data dictionary, but it does not change the file directory on the hard disk.

Table 41–4 (Cont.) ALTER_FILE Procedure Parameters

Parameter	Description
new_file_type	The file type. The following are reserved file types: ■ If the file is a datafile, then enter the following: 'DATAFILE' ■ If the file is a Data Pump export dump file, then enter the following: 'DUMPSET' ■ If the file is a Data Pump export log file, then enter the following: 'DATAPUMPLOG' If the file type is not one of the reserved file types, then enter a text description of the file type. If NULL, then the procedure does not change the file type. **See Also:** "Constants" on page 41-4 for more information about the reserved file types.
remove_file_type	If Y, then the procedure removes the file type. If Y and the new_file_type parameter is non-NULL, then the procedure raises an error. If N, then the procedure does not remove the file type.
new_comments	New comments about the file being altered. If non-NULL, then the procedure replaces the existing comments with the specified comments. If NULL, then the procedure does not change the existing comments.
remove_comments	If Y, then the procedure removes the comments for the file. If Y and the new_comments parameter is non-NULL, then the procedure raises an error. If N, then the procedure does not change the existing comments.

Usage Notes

If the file type is changed to DBMS_FILE_GROUP.EXPORT_DUMP_FILE or 'DUMPSET', then Data Pump metadata for the file is populated. If the file type is changed from DBMS_FILE_GROUP.EXPORT_DUMP_FILE or 'DUMPSET', then Data Pump metadata for the file is purged.

To run this procedure with DBMS_FILE_GROUP.EXPORT_DUMP_FILE or 'DUMPSET' specified for the new_file_type parameter, a user must meet the following requirements:

- Have the appropriate privileges to import the Data Pump export dump file
- Have READ privilege on the directory object that contains the Data Pump export dump file

> **See Also:** *Oracle Database Utilities* for more information about Data Pump privileges

ALTER_FILE_GROUP Procedure

This procedure alters a file group.

Syntax

```
DBMS_FILE_GROUP.ALTER_FILE_GROUP(
  file_group_name           IN  VARCHAR2,
  keep_files                IN  VARCHAR2  DEFAULT NULL,
  min_versions              IN  NUMBER    DEFAULT NULL,
  max_versions              IN  NUMBER    DEFAULT NULL,
  retention_days            IN  NUMBER    DEFAULT NULL,
  new_default_directory     IN  VARCHAR2  DEFAULT NULL,
  remove_default_directory  IN  VARCHAR2  DEFAULT 'N',
  new_comments              IN  VARCHAR2  DEFAULT NULL,
  remove_comments           IN  VARCHAR2  DEFAULT 'N');
```

Parameters

Table 41–5 ALTER_FILE_GROUP Procedure Parameters

Parameter	Description
file_group_name	The name of the file group being altered, specified as *[schema_name.]file_group_name*. For example, if the schema is hq_dba and the file group name is sales_tbs, then specify hq_dba.sales_tbs. If the schema is not specified, then the current user is the default.
keep_files	If Y, then the files in the file group are retained on hard disk if the file group or a version of the file group is dropped or purged. If N, then the files in the file group are deleted from hard disk if the file group or a version of the file group is dropped or purged. If NULL, then this parameter is not changed. **Note:** If the file group is dropped as a result of a DROP USER CASCADE statement, then the setting of this parameter determines whether the files are dropped from the hard disk.
min_versions	The minimum number of versions to retain. The specified value must be greater than or equal to 1. If NULL, then the procedure does not change the min_versions setting for the file group.
max_versions	The maximum number of versions to retain. The specified value must be greater than or equal to the value specified for min_versions. When the number of versions exceeds the specified max_versions, the oldest version is purged. Specify DBMS_FILE_GROUP.INFINITE for no limit to the number of versions. If NULL, then the procedure does not change the max_versions setting for the file group.

Table 41–5 (Cont.) ALTER_FILE_GROUP Procedure Parameters

Parameter	Description
retention_days	The maximum number of days to retain a version. The specified value must be greater than or equal to 0 (zero). When the age of a version exceeds the specified retention_days and there are more versions than the number specified in min_versions, the version is purged. The age of a version is calculated by subtracting the creation time from the current time. A decimal value can be used to specify a fraction of a day. For example, 1.25 specifies one day and six hours. Specify DBMS_FILE_GROUP.INFINITE for no limit to the number of days a version can exist. If NULL, then the procedure does not change the retention_days setting for the file group.
new_default_directory	The default directory object used when files are added to a file group if no directory is specified when the files are added, and no default directory object is specified for the version. If NULL, then the procedure does not change the default directory.
remove_default_directory	If Y, then the procedure removes the default directory for the file group. If Y and the new_default_directory parameter is set to a non-NULL value, then the procedure raises an error. If N, then the procedure does not remove the default directory for the file group.
new_comments	Comments about the file group. If non-NULL, then the new comments replace the existing comments for the file group. If NULL, then the procedure does not change the existing comments.
remove_comments	If Y, then the comments for the file group are removed. If Y and the new_comments parameter is set to a non-NULL value, then the procedure raises an error. If N, then the procedure does not change the comments for the file group.

Usage Notes

If min_versions is set to 1, then the only version of the file group can be purged when a new version is added. If the addition of the new version is not complete when the existing version is purged, then there can be a period of time when no version of the file group is available. Therefore, set min_versions to at least 2 if a version of the file group must be available at all times.

ALTER_VERSION Procedure

This procedure alters a version of a file group.

Syntax

```
DBMS_FILE_GROUP.ALTER_VERSION(
  file_group_name           IN  VARCHAR2,
  version_name              IN  VARCHAR2  DEFAULT NULL,
  new_version_name          IN  VARCHAR2  DEFAULT NULL,
  remove_version_name       IN  VARCHAR2  DEFAULT 'N',
  new_default_directory     IN  VARCHAR2  DEFAULT NULL,
  remove_default_directory  IN  VARCHAR2  DEFAULT 'N',
  new_comments              IN  VARCHAR2  DEFAULT NULL,
  remove_comments           IN  VARCHAR2  DEFAULT 'N');
```

Parameters

Table 41–6 ALTER_VERSION Procedure Parameters

Parameter	Description
file_group_name	The name of the file group that contains the version, specified as *[schema_name.]file_group_name*. For example, if the schema is hq_dba and the file group name is sales_tbs, then specify hq_dba.sales_tbs. If the schema is not specified, then the current user is the default.
version_name	The name of the version being altered. If a positive integer is specified as a VARCHAR2 value, then the integer is interpreted as a version number. For example, if '1' is specified, then version 1 of the file group is altered. If '*' is specified, then the procedure alters all versions, and the new_version_name parameter must be NULL. If NULL, then the procedure uses the version with the latest creation time for the file group.
new_version_name	The new name of the version. Do not specify a schema. The specified version name cannot be a positive integer or an asterisk ('*'). If NULL, then the procedure does not change the version name.
remove_version_name	If Y, then the procedure removes the version name. If the version name is removed, then the version number must be used to manage the version. If Y and the new_version_name parameter is set to a non-NULL value, then the procedure raises an error. If N, then the procedure does not remove the version name.
new_default_directory	The default directory object used when files are added to a version if no directory is specified when the files are added. If NULL, then the procedure does not change the default directory.
remove_default_directory	If Y, then the procedure removes the default directory. If Y and the new_default_directory parameter is set to a non-NULL value, then the procedure raises an error. If N, then the procedure does not remove the default directory.

Table 41–6 (Cont.) ALTER_VERSION Procedure Parameters

Parameter	Description
new_comments	Comments about the version. If non-NULL, then the new comments replace the existing comments for the version. If NULL, then the procedure does not change the comments.
remove_comments	If Y, then the procedure removes the comments for the version. If Y and the new_comments parameter is set to a non-NULL value, then the procedure raises an error. If N, then the procedure does not remove the comments for the version.

CREATE_FILE_GROUP Procedure

This procedure creates a file group.

Syntax

```
DBMS_FILE_GROUP.CREATE_FILE_GROUP(
  file_group_name    IN  VARCHAR2,
  keep_files         IN  VARCHAR2  DEFAULT 'Y',
  min_versions       IN  NUMBER    DEFAULT 2,
  max_versions       IN  NUMBER    DEFAULT DBMS_FILE_GROUP.INFINITE,
  retention_days     IN  NUMBER    DEFAULT DBMS_FILE_GROUP.INFINITE,
  default_directory  IN  VARCHAR2  DEFAULT NULL,
  comments           IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 41–7 CREATE_FILE_GROUP Procedure Parameters

Parameter	Description
file_group_name	The name of the file group, specified as *[schema_name.]file_group_name*. For example, if the schema is hq_dba and the file group name is sales_tbs, then specify hq_dba.sales_tbs. If the schema is not specified, then the current user is the default.
keep_files	If Y, then the files in the file group are retained on hard disk if the file group or a version of the file group is dropped or purged. If N, then the files in the file group are deleted from hard disk if the file group or a version of the file group is dropped or purged. **Note:** If the file group is dropped as a result of a DROP USER CASCADE statement, then the setting of this parameter determines whether the files are dropped from the hard disk.
min_versions	The minimum number of versions to retain. The specified value must be greater than or equal to 1.
max_versions	The maximum number of versions to retain. The specified value must be greater than or equal to the value specified for min_versions. When the number of versions exceeds the specified max_versions, the oldest version is purged. Specify DBMS_FILE_GROUP.INFINITE for no limit to the number of versions.
retention_days	The maximum number of days to retain a version. The specified value must be greater than or equal to 0 (zero). When the age of a version exceeds the specified retention_days and there are more versions than the number specified in min_versions, the version is purged. The age of a version is calculated by subtracting the creation time from the current time. A decimal value can be used to specify a fraction of a day. For example, 1.25 specifies one day and six hours. Specify DBMS_FILE_GROUP.INFINITE for no limit to the number of days a version can exist.
default_directory	The default directory object used when files are added to a file group if no directory is specified when the files are added, and no default directory object is specified for the version.
comments	Comments about the file group being created.

Usage Notes

If `min_versions` is set to `1`, then the only version of the file group can be purged when a new version is added. If the addition of the new version is not complete when the existing version is purged, then there can be a period of time when no version of the file group is available. Therefore, set `min_versions` to at least `2` if a version of the file group must be available at all times.

CREATE_VERSION Procedure

This procedure creates a version of a file group.

This procedure automatically runs the PURGE_FILE_GROUP procedure. Therefore, versions can be purged based on the file group's retention policy.

This procedure is overloaded. One version of the procedure contains the OUT parameter version_out, and the other does not.

See Also: "PURGE_FILE_GROUP Procedure" on page 41-21

Syntax

```
DBMS_FILE_GROUP.CREATE_VERSION(
  file_group_name    IN  VARCHAR2,
  version_name       IN  VARCHAR2 DEFAULT NULL,
  default_directory  IN  VARCHAR2 DEFAULT NULL,
  comments           IN  VARCHAR2 DEFAULT NULL);

DBMS_FILE_GROUP.CREATE_VERSION(
  file_group_name    IN   VARCHAR2,
  version_name       IN   VARCHAR2 DEFAULT NULL,
  default_directory  IN   VARCHAR2 DEFAULT NULL,
  comments           IN   VARCHAR2 DEFAULT NULL,
  version_out        OUT  VARCHAR2);
```

Parameters

Table 41–8 CREATE_VERSION Procedure Parameters

Parameter	Description
file_group_name	The name of the file group to which the new version is added, specified as *[schema_name.]file_group_name*. For example, if the schema is hq_dba and the file group name is sales_tbs, then specify hq_dba.sales_tbs. If the schema is not specified, then the current user is the default.
version_name	The name of the version being created. Do not specify a schema. The specified version name cannot be a positive integer because, when a version is created, a version number is generated automatically. The specified version name cannot be an asterisk ('*').
default_directory	The default directory object used when files are added to a version if no directory is specified when the files are added.
comments	Comments about the version being created
version_out	If the version_name parameter is set to a non-NULL value, then this parameter contains the specified version name. If the version_name parameter is set to NULL, then this parameter contains the generated version number.

DROP_FILE_GROUP Procedure

This procedure drops a file group.

Syntax

```
DBMS_FILE_GROUP.DROP_FILE_GROUP(
  file_group_name  IN  VARCHAR2,
  keep_files       IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 41–9 DROP_FILE_GROUP Procedure Parameters

Parameter	Description
`file_group_name`	The name of the file group being dropped, specified as *`[schema_name.]file_group_name`*. For example, if the schema is `hq_dba` and the file group name is `sales_tbs`, then specify `hq_dba.sales_tbs`. If the schema is not specified, then the current user is the default.
`keep_files`	If `Y`, then the procedure retains the files in the file group on hard disk. If `N`, then the procedure deletes the files in the file group from hard disk. If `NULL`, then the procedure uses the default keep files property of the file group.

Usage Notes

If this procedure deletes files on hard disk, then the user who runs the procedure must have `WRITE` privilege on the directory object that contains the files.

DROP_VERSION Procedure

This procedure drops a version of a file group.

Syntax

```
DBMS_FILE_GROUP.DROP_VERSION(
  file_group_name  IN  VARCHAR2,
  version_name     IN  VARCHAR2 DEFAULT NULL,
  keep_files       IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 41–10 DROP_VERSION Procedure Parameters

Parameter	Description
file_group_name	The name of the file group that contains the version, specified as *[schema_name.]file_group_name*. For example, if the schema is hq_dba and the file group name is sales_tbs, then specify hq_dba.sales_tbs. If the schema is not specified, then the current user is the default.
version_name	The name of the version being dropped. If a positive integer is specified as a VARCHAR2 value, then the integer is interpreted as a version number. For example, if '1' is specified, then version 1 of the file group is dropped. If NULL, then the procedure uses the version with the oldest creation time for the file group. If '*', then the procedure drops all versions.
keep_files	If Y, then the procedure retains the files in the version on hard disk. If N, then the procedure deletes the files in the version from hard disk. If NULL, then the procedure uses the default keep files property of the file group.

Usage Notes

If this procedure deletes files on hard disk, then the user who runs the procedure must have WRITE privilege on the directory object that contains the files.

GRANT_OBJECT_PRIVILEGE Procedure

This procedure grants object privileges on a file group to a user.

Syntax

```
DBMS_FILE_GROUP.GRANT_OBJECT_PRIVILEGE(
  object_name   IN  VARCHAR2,
  privilege     IN  BINARY_INTEGER,
  grantee       IN  VARCHAR2,
  grant_option  IN  BOOLEAN DEFAULT FALSE);
```

Parameters

Table 41–11 GRANT_OBJECT_PRIVILEGE Procedure Parameters

Parameter	Description
`object_name`	The name of the file group on which the privilege is granted, specified as *`[schema_name.]file_group_name`*. For example, if the schema is `hq_dba` and the file group name is `sales_tbs`, then specify `hq_dba.sales_tbs`. If the schema is not specified, then the current user is the default.
`privilege`	The constant that specifies the privilege. See "Constants" on page 41-4 for valid privileges.
`grantee`	The name of the user or role for which the privilege is granted. The specified user cannot be the owner of the object.
`grant_option`	If `TRUE`, then the specified user granted the specified privilege can grant this privilege to others. If `FALSE`, then the specified user granted the specified privilege cannot grant this privilege to others.

Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Be the owner of the object on which the privilege is granted
- Have the same privilege as the privilege being granted with the grant option

GRANT_SYSTEM_PRIVILEGE Procedure

This procedure grants system privileges for file group operations to a user.

> **Note:** When you grant a privilege on "ANY" object (for example, ALTER_ANY_RULE), and the initialization parameter O7_DICTIONARY_ACCESSIBILITY is set to FALSE, you give the user access to that type of object in all schemas, except the SYS schema. By default, the initialization parameter O7_DICTIONARY_ACCESSIBILITY is set to FALSE.
>
> If you want to grant access to an object in the SYS schema, then you can grant object privileges explicitly on the object. Alternatively, you can set the O7_DICTIONARY_ACCESSIBILITY initialization parameter to TRUE. Then privileges granted on "ANY" object will allow access to any schema, including SYS. Set the O7_DICTIONARY_ACCESSIBILITY initialization parameter with caution.

Syntax

```
DBMS_FILE_GROUP.GRANT_SYSTEM_PRIVILEGE(
  privilege     IN  BINARY_INTEGER,
  grantee       IN  VARCHAR2,
  grant_option  IN  BOOLEAN  DEFAULT FALSE);
```

Parameters

Table 41–12 GRANT_SYSTEM_PRIVILEGE Procedure Parameters

Parameter	Description
privilege	The constant that specifies the privilege. See "Constants" on page 41-4 for valid privileges.
grantee	The name of the user or role for which the privilege is granted. The user who runs the procedure cannot be specified.
grant_option	If TRUE, then the specified user granted the specified privilege can grant this privilege to others. If FALSE, then the specified user granted the specified privilege cannot grant this privilege to others.

PURGE_FILE_GROUP Procedure

This procedure purges a file group using the file group's retention policy.

A file group's retention policy is determined by its settings for the `max_versions`, `min_versions`, and `retention_days` parameters. The following versions of a file group are removed when a file group is purged:

- All versions greater than the `max_versions` setting for the file group when versions are ordered in descending order by creation time. Therefore, the older versions are purged before the newer versions.
- All versions older than the `retention_days` setting for the file group except when purging a version will cause the number of versions to drop below the `min_versions` setting for the file group.

A job named `SYS.FGR$AUTOPURGE_JOB` automatically purges all file groups in a database periodically according to the job's schedule. You can adjust this job's schedule using the `DBMS_SCHEDULER` package. Alternatively, you can create a job that runs the `PURGE_FILE_GROUP` procedure periodically.

Syntax

```
DBMS_FILE_GROUP.PURGE_FILE_GROUP(
  file_group_name  IN  VARCHAR2);
```

Parameter

Table 41–13 PURGE_FILE_GROUP Procedure Parameter

Parameter	Description
`file_group_name`	The name of the file group, specified as *`[schema_name.]file_group_name`*. For example, if the schema is `hq_dba` and the file group name is `sales_tbs`, then specify `hq_dba.sales_tbs`. If the schema is not specified, then the current user is the default. If `NULL` and this procedure is run by `SYS` user, then the procedure purges all file groups.

Usage Notes

If this procedure deletes files on hard disk, then the user who runs the procedure must have `WRITE` privilege on the directory object that contains the files. Files are deleted when a version is purged and the `keep_files` parameter is set to `N` for the version's file group.

REMOVE_FILE Procedure

This procedure removes a file from a version of a file group.

Syntax

```
DBMS_FILE_GROUP.REMOVE_FILE(
  file_group_name  IN  VARCHAR2,
  file_name        IN  VARCHAR2,
  version_name     IN  VARCHAR2 DEFAULT NULL,
  keep_file        IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 41–14 REMOVE_FILE Procedure Parameters

Parameter	Description
`file_group_name`	The name of the file group that contains the version, specified as *`[schema_name.]file_group_name`*. For example, if the schema is `hq_dba` and the file group name is `sales_tbs`, then specify `hq_dba.sales_tbs`. If the schema is not specified, then the current user is the default.
`file_name`	The name of the file being removed from the version
`version_name`	The name of the version from which the file is removed. If a positive integer is specified as a `VARCHAR2` value, then the integer is interpreted as a version number. For example, if `'1'` is specified, then the file is removed from version 1 of the file group. If `NULL`, then the procedure uses the version with the latest creation time for the file group. If `'*'`, then the procedure removes the file from all versions.
`keep_file`	If `Y`, then the procedure retains the file on hard disk. If `N`, then the procedure deletes the file from hard disk. If `NULL`, then the procedure uses the default keep files property of the file group.

Usage Notes

If this procedure deletes files on hard disk, then the user who runs the procedure must have `WRITE` privilege on the directory object that contains the files.

REVOKE_OBJECT_PRIVILEGE Procedure

This procedure revokes object privileges on a file group from a user.

Syntax

```
DBMS_FILE_GROUP.REVOKE_OBJECT_PRIVILEGE(
  object_name  IN  VARCHAR2,
  privilege    IN  BINARY_INTEGER,
  revokee      IN  VARCHAR2);
```

Parameters

Table 41–15 REVOKE_OBJECT_PRIVILEGE Procedure Parameters

Parameter	Description
`object_name`	The name of the file group on which the privilege is revoked, specified as *`[schema_name.]file_group_name`*. For example, if the schema is `hq_dba` and the file group name is `sales_tbs`, then specify `hq_dba.sales_tbs`. If the schema is not specified, then the current user is the default.
`privilege`	The constant that specifies the privilege. See "Constants" on page 41-4 for valid privileges.
`revokee`	The name of the user or role from which the privilege is revoked. The user who owns the object cannot be specified.

REVOKE_SYSTEM_PRIVILEGE Procedure

This procedure revokes system privileges for file group operations from a user.

Syntax

```
DBMS_FILE_GROUP.REVOKE_SYSTEM_PRIVILEGE(
  privilege  IN  BINARY_INTEGER,
  revokee    IN  VARCHAR2);
```

Parameters

Table 41–16 REVOKE_SYSTEM_PRIVILEGE Procedure Parameters

Parameter	Description
`privilege`	The constant that specifies the privilege. See "Constants" on page 41-4 for valid privileges.
`revokee`	The name of the user or role from which the privilege is revoked. The user who runs the procedure cannot be specified.

42

DBMS_FILE_TRANSFER

The DBMS_FILE_TRANSFER package provides procedures to copy a binary file within a database or to transfer a binary file between databases.

See Also:

- *Oracle Database Concepts* for conceptual information about file transfer
- *Oracle Database Administrator's Guide* for instructions about using file transfer
- *Oracle Streams Concepts and Administration* for applications of file transfer.

This chapter contains the following topic:

Using DBMS_FILE_TRANSFER

Operating Notes

Caution: `DBMS_FILE_TRANSFER` supports online backup. You should therefore be careful in copying or transferring a file that is being modified by the database because this can result in an inconsistent file, and require recovery. To guarantee consistency, bring files offline when the database is in use.

Summary of DBMS_FILE_TRANSFER Subprograms

Table 42–1 DBMS_FILE_TRANSFER Package Subprograms

Subprogram	Description
COPY_FILE Procedure on page 42-5	Reads a file from a source directory and creates a copy of it in a destination directory. The source and destination directories can both be in a local file system, or both be in an Automatic Storage Management (ASM) disk group, or between local file system and ASM with copying in either direction.
GET_FILE Procedure on page 42-7	Contacts a remote database to read a remote file and then creates a copy of the file in the local file system or ASM
PUT_FILE Procedure on page 42-9	Reads a local file or ASM and contacts a remote database to create a copy of the file in the remote file system

COPY_FILE Procedure

This procedure reads a file from a source directory and creates a copy of it in a destination directory. The source and destination directories can both be in a local file system, or both be in an Automatic Storage Management (ASM) disk group, or between local file system and ASM with copying in either direction.

You can copy any type of file to and from a local file system. However, you can copy only database files (such as datafiles, tempfiles, controlfiles, and so on) to and from an ASM disk group.

The destination file is not closed until the procedure completes successfully.

Syntax

```
DBMS_FILE_TRANSFER.COPY_FILE(
   source_directory_object       IN  VARCHAR2,
   source_file_name              IN  VARCHAR2,
   destination_directory_object  IN  VARCHAR2,
   destination_file_name         IN  VARCHAR2);
```

Parameters

Table 42–2 COPY_FILE Procedure Parameters

Parameter	Description
`source_directory_object`	The directory object that designates the source directory. The directory object must already exist. (You create directory objects with the `CREATE DIRECTORY` command).
`source_file_name`	The name of the file to copy. This file must exist in the source directory.
`destination_directory_object`	The directory object that designates the destination directory. The directory object must already exist. If the destination is ASM, the directory object must designate either a disk group name (for example, `+diskgroup1`) or a directory created for alias names. In the case of a directory, the full path to the directory must be specified (for example: `+diskgroup1/dbs/control`).
`destination_file_name`	The name to assign to the file in the destination directory. A file with the same name must not exist in the destination directory. If the destination is ASM: ■ The file is given a fully qualified ASM filename and created in the appropriate directory (depending on the database name and file type) ■ The file type tag assigned to the file is `COPY_FILE` ■ The value of the `destination_file_name` argument becomes the file's alias name in the designated destination directory The file name can be followed by an ASM template name in parentheses. The file is then given the attributes specified by the template.

Usage Notes

To run this procedure successfully, the current user must have the following privileges:

- `READ` privilege on the directory object specified in the `source_directory_object` parameter
- `WRITE` privilege on directory object specified in the `destination_directory_object` parameter

This procedure converts directory object parameters to uppercase unless they are surrounded by double quotation marks, but this procedure does not convert file names to uppercase.

Also, the copied file must meet the following requirements:

- The size of the copied file must be a multiple of 512 bytes.
- The size of the copied file must be less than or equal to two terabytes.

Transferring the file is not transactional. The copied file is treated as a binary file, and no character set conversion is performed. To monitor the progress of a long file copy, query the `V$SESSION_LONGOPS` dynamic performance view.

> **See Also:** *Oracle Database Administrator's Guide* for instructions about using file transfer

Examples

```
SQL> create directory DGROUP as '+diskgroup1/dbs/backup';

Directory created.

SQL>  BEGIN
   2    DBMS_FILE_TRANSFER.COPY_FILE('SOURCEDIR','t_xdbtmp.f', 'DGROUP',
                                     't_xdbtmp.f');
   3  END;
   4  /

PL/SQL procedure successfully completed.

SQL> EXIT
$ASMCMD
ASMCMD> ls
DISKGROUP1/
ASMCMD> cd diskgroup1/dbs/backup
ASMCMD> ls
t_xdbtmp.f => +DISKGROUP1/ORCL/TEMPFILE/COPY_FILE.267.546546525
```

GET_FILE Procedure

This procedure contacts a remote database to read a remote file and then creates a copy of the file in the local file system or ASM. The file that is copied is the source file, and the new file that results from the copy is the destination file. The destination file is not closed until the procedure completes successfully.

Syntax

```
DBMS_FILE_TRANSFER.GET_FILE
   source_directory_object      IN  VARCHAR2,
   source_file_name             IN  VARCHAR2,
   source_database              IN  VARCHAR2,
   destination_directory_object IN  VARCHAR2,
   destination_file_name        IN  VARCHAR2);
```

Parameters

Table 42–3 GET_FILE Procedure Parameters

Parameter	Description
`source_directory_object`	The directory object from which the file is copied at the source site. This directory object must exist at the source site.
`source_file_name`	The name of the file that is copied in the remote file system. This file must exist in the remote file system in the directory associated with the source directory object.
`source_database`	The name of a database link to the remote database where the file is located.
`destination_directory_object`	The directory object into which the file is placed at the destination site. This directory object must exist in the local file system.
`destination_file_name`	The name of the file copied to the local file system. A file with the same name must not exist in the destination directory in the local file system.

Usage Notes

To run this procedure successfully, the following users must have the following privileges:

- The connected user at the source database must have read privilege on the directory object specified in the `source_directory_object` parameter.
- The current user at the local database must have write privilege on the directory object specified in the `destination_directory_object` parameter.

This procedure converts directory object parameters to uppercase unless they are surrounded by double quotation marks, but this procedure does not convert file names to uppercase.

Also, the copied file must meet the following requirements:

- The size of the copied file must be a multiple of 512 bytes.
- The size of the copied file must be less than or equal to two terabytes.

Transferring the file is not transactional. The copied file is treated as a binary file, and no character set conversion is performed. To monitor the progress of a long file transfer, query the V$SESSION_LONGOPS dynamic performance view.

Examples

```
CREATE OR REPLACE DIRECTORY df AS '+datafile' ;
GRANT WRITE ON DIRECTORY df TO "user";
CREATE DIRECTORY DSK_FILES AS ''^t_work^'';
GRANT WRITE ON DIRECTORY dsk_files TO "user";

-- asumes that dbs2 link has been created and we are connected to the instance.
-- dbs2 could be a loopback or point to another instance.

BEGIN
-- asm file to an os file
-- get an asm file from dbs1.asm/a1 to dbs2.^t_work^/oa5.dat
  DBMS_FILE_TRANSFER.GET_FILE ( 'df' , 'a1' , 'dbs1', 'dsk_files' , 'oa5.dat' );

-- os file to an os file
-- get an os file from dbs1.^t_work^/a2.dat to dbs2.^t_work^/a2back.dat
  DBMS_FILE_TRANSFER.GET_FILE ( 'dsk_files' , 'a2.dat' , 'dbs1', 'dsk_files' ,
'a2back.dat' );

END ;
/
```

PUT_FILE Procedure

This procedure reads a local file or ASM and contacts a remote database to create a copy of the file in the remote file system. The file that is copied is the source file, and the new file that results from the copy is the destination file. The destination file is not closed until the procedure completes successfully.

Syntax

```
DBMS_FILE_TRANSFER.PUT_FILE(
   source_directory_object       IN  VARCHAR2,
   source_file_name              IN  VARCHAR2,
   destination_directory_object  IN  VARCHAR2,
   destination_file_name         IN  VARCHAR2,
   destination_database          IN  VARCHAR2);
```

Parameters

Table 42–4 PUT_FILE Procedure Parameters

Parameter	Description
source_directory_object	The directory object from which the file is copied at the local source site. This directory object must exist at the source site.
source_file_name	The name of the file that is copied from the local file system. This file must exist in the local file system in the directory associated with the source directory object.
destination_directory_object	The directory object into which the file is placed at the destination site. This directory object must exist in the remote file system.
destination_file_name	The name of the file placed in the remote file system. A file with the same name must not exist in the destination directory in the remote file system.
destination_database	The name of a database link to the remote database to which the file is copied.

Usage Notes

To run this procedure successfully, the following users must have the following privileges:

- The current user at the local database must have read privilege on the directory object specified in the source_directory_object parameter.
- The connected user at the destination database must have write privilege to the directory object specified in the destination_directory_object parameter.

This procedure converts directory object parameters to uppercase unless they are surrounded by double quotation marks, but this procedure does not convert file names to uppercase.

Also, the copied file must meet the following requirements:

- The size of the copied file must be a multiple of 512 bytes.
- The size of the copied file must be less than or equal to two terabytes.

Transferring the file is not transactional. The copied file is treated as a binary file, and no character set conversion is performed. To monitor the progress of a long file transfer, query the V$SESSION_LONGOPS dynamic performance view.

Examples

```
CREATE OR REPLACE DIRECTORY df AS '+datafile' ;
GRANT WRITE ON DIRECTORY df TO "user";
CREATE OR REPLACE DIRECTORY ft1 AS '+datafile/ft1' ;
GRANT READ,WRITE ON DIRECTORY ft1 TO "user";
CREATE OR REPLACE DIRECTORY ft1_1 AS '+datafile/ft1/ft1_1' ;

CONNECT user/password@inst1

-- - put a1.dat to a4.dat (using dbs2 dblink)
-- - level 2 sub dir to parent dir
-- - user has read privs on ft1_1 at dbs1 and write on df in dbs2
BEGIN
 DBMS_FILE_TRANSFER.PUT_FILE ( 'ft1_1' , 'a2.dat' , 'df' , 'a4.dat' ,
                               'dbs2' ) ;
END ;
```

43

DBMS_FLASHBACK

Using DBMS_FLASHBACK, you can flash back to a version of the database at a specified wall-clock time or a specified system change number (SCN)

> **See Also:** For detailed information about DBMS_FLASHBACK:
>
> - *Oracle Database Application Developer's Guide - Fundamentals*
> - *Oracle Database SQL Reference* .

This chapter contains the following topics:

Using DBMS_FLASHBACK

Overview

When DBMS_FLASHBACK is enabled, the user session uses the Flashback version of the database, and applications can execute against the Flashback version of the database.

You may want to use DBMS_FLASHBACK for the following reasons:

- Self-service repair: If you accidentally delete rows from a table, you can recover the deleted rows.
- Packaged applications such as e-mail and voicemail: You can use Flashback to restore deleted e-mail by re-inserting the deleted message into the current message box.
- Decision support system (DSS) and online analytical processing (OLAP) applications: You can perform data analysis or data modeling to track seasonal demand.

Security Model

To use this package, a database administrator must grant `EXECUTE` privileges for `DBMS_FLASHBACK`.

Exceptions

Table 43–1 DBMS_FLASHBACK Error Messages

Error	Description
ORA-08180	Time specified is too old.
ORA-08181	Invalid system change number specified.
ORA-08182	User cannot begin read-only or serializable transactions in Flashback mode.
ORA-08183	User cannot enable Flashback within an uncommitted transaction.
ORA-08184	User cannot enable Flashback within another Flashback session.
ORA-08185	SYS cannot enable Flashback mode.

Operational Notes

DBMS_FLASHBACK is automatically turned off when the session ends, either by disconnection or by starting another connection.

PL/SQL cursors opened in Flashback mode return rows as of the flashback time or SCN. Different concurrent sessions (connections) in the database can perform Flashback to different wall-clock times or SCNs. DML and DDL operations and distributed operations are not allowed while a session is running in Flashback mode. You can use PL/SQL cursors opened before disabling Flashback to perform DML.

Under Automatic Undo Management (AUM) mode, you can use retention control to control how far back in time to go for the version of the database you need.

If you need to perform a Flashback over a 24-hour period, the DBA should set the undo_retention parameter to 24 hours. This way, the system retains enough undo information to regenerate the older versions of the data.

You can set the RETENTION GUARANTEE clause for the undo tablespace to ensure that unexpired undo is not discarded.UNDO_RETENTION is not in itself a complete guarantee because, if the system is under space pressure, unexpired undo may be overwritten with freshly generated undo. In such cases, RETENTION GUARANTEE prevents this. For more information, see the *Oracle Database Administrator's Guide*

In a Flashback-enabled session, SYSDATE will not be affected; it will continue to provide the current time.

DBMS_FLASHBACK can be used within logon triggers to enable Flashback without changing the application code.

Examples

The following example illustrates how Flashback can be used when the deletion of a senior employee triggers the deletion of all the personnel reporting to him. Using the Flashback feature, you can recover and re-insert the missing employees.

```
DROP TABLE employee;
DROP TABLE keep_scn;

REM -- Keep_scn is a temporary table to store scns that we are interested in

CREATE TABLE keep_scn (scn number);
SET ECHO ON
CREATE TABLE employee (
   employee_no   number(5) PRIMARY KEY,
   employee_name varchar2(20),
   employee_mgr  number(5)
      CONSTRAINT mgr_fkey REFERENCES EMPLOYEE ON DELETE CASCADE,
   salary        number,
   hiredate      date
);

REM -- Populate the company with employees
INSERT INTO employee VALUES (1, 'John Doe', null, 1000000, '5-jul-81');
INSERT INTO employee VALUES (10, 'Joe Johnson', 1, 500000, '12-aug-84');
INSERT INTO employee VALUES (20, 'Susie Tiger', 10, 250000, '13-dec-90');
INSERT INTO employee VALUES (100, 'Scott Tiger', 20, 200000, '3-feb-86');
INSERT INTO employee VALUES (200, 'Charles Smith', 100, 150000, '22-mar-88');
INSERT INTO employee VALUES (210, 'Jane Johnson', 100, 100000, '11-apr-87');
INSERT INTO employee VALUES (220, 'Nancy Doe', 100, 100000, '18-sep-93');
INSERT INTO employee VALUES (300, 'Gary Smith', 210, 75000, '4-nov-96');
INSERT INTO employee VALUES (310, 'Bob Smith', 210, 65000, '3-may-95');
COMMIT;

REM -- Show the entire org
SELECT lpad(' ', 2*(level-1)) || employee_name Name
FROM employee
CONNECT BY PRIOR employee_no = employee_mgr
START WITH employee_no = 1
ORDER BY LEVEL;

REM -- Sleep for a short time (approximately 10 to 20  seconds) to avoid
REM -- querying close to table creation

EXECUTE DBMS_LOCK.SLEEP(10);

REM -- Store this snapshot for later access through Flashback
DECLARE
I NUMBER;
BEGIN
I := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER;
INSERT INTO keep_scn VALUES (I);
COMMIT;
END;
/

REM -- Scott decides to retire but the transaction is done incorrectly
DELETE FROM EMPLOYEE WHERE employee_name = 'Scott Tiger';
COMMIT;
```

```
REM -- notice that all of scott's employees are gone
SELECT lpad(' ', 2*(level-1)) || employee_name Name
FROM EMPLOYEE
CONNECT BY PRIOR employee_no = employee_mgr
START WITH employee_no = 1
ORDER BY LEVEL;

REM -- Flashback to see Scott's organization
DECLARE
   restore_scn number;
BEGIN
   SELECT  scn INTO restore_scn FROM keep_scn;
   DBMS_FLASHBACK.ENABLE_AT_SYSTEM_CHANGE_NUMBER (restore_scn);
END;
/

REM -- Show Scott's org.
SELECT lpad(' ', 2*(level-1)) || employee_name Name
FROM employee
CONNECT BY PRIOR employee_no = employee_mgr
START WITH employee_no =
   (SELECT employee_no FROM employee WHERE employee_name = 'Scott Tiger')
ORDER BY LEVEL;

REM -- Restore scott's organization.
DECLARE
   scotts_emp NUMBER;
   scotts_mgr NUMBER;
   CURSOR c1 IS
      SELECT employee_no, employee_name, employee_mgr, salary, hiredate
      FROM employee
      CONNECT BY PRIOR employee_no = employee_mgr
      START WITH employee_no =
         (SELECT employee_no FROM employee WHERE employee_name = 'Scott Tiger');
   c1_rec c1 % ROWTYPE;
BEGIN
   SELECT employee_no, employee_mgr INTO scotts_emp, scotts_mgr FROM employee
   WHERE employee_name = 'Scott Tiger';
   /* Open c1 in flashback mode */
   OPEN c1;
   /* Disable Flashback */
   DBMS_FLASHBACK.DISABLE;
 LOOP
   FETCH c1 INTO c1_rec;
   EXIT WHEN c1%NOTFOUND;
   /*
     Note that all the DML operations inside the loop are performed
     with Flashback disabled
   */
   IF (c1_rec.employee_mgr = scotts_emp) then
      INSERT INTO employee VALUES (c1_rec.employee_no,
         c1_rec.employee_name,
         scotts_mgr,
         c1_rec.salary,
         c1_rec.hiredate);
   ELSE
   IF (c1_rec.employee_no != scotts_emp) THEN
   INSERT INTO employee VALUES (c1_rec.employee_no,
         c1_rec.employee_name,
```

```
        c1_rec.employee_mgr,
        c1_rec.salary,
        c1_rec.hiredate);
      END IF;
    END IF;
 END LOOP;
END;
/

REM -- Show the restored organization.
select lpad(' ', 2*(level-1)) || employee_name Name
FROM employee
CONNECT BY PRIOR employee_no = employee_mgr
START WITH employee_no = 1
ORDER BY LEVEL;
```

Summary of DBMS_FLASHBACK Subprograms

Table 43–2 DBMS_FLASHBACK Package Subprograms

Subprogram	Description
DISABLE Procedure on page 43-11	Disables the Flashback mode for the entire session
ENABLE_AT_SYSTEM_CHANGE_NUMBER Procedure on page 43-12	Enables Flashback for the entire session. Takes an SCN as an Oracle number and sets the session snapshot to the specified number. Inside the Flashback mode, all queries will return data consistent as of the specified wall-clock time or SCN
ENABLE_AT_TIME Procedure on page 43-13	Enables Flashback for the entire session. The snapshot time is set to the SCN that most closely matches the time specified in `query_time`
GET_SYSTEM_CHANGE_NUMBER Function on page 43-14	Returns the current SCN as an Oracle number. You can use the SCN to store specific snapshots
SCN_TO_TIMESTAMP Function on page 43-15	Takes the current SCN as an Oracle number datatype and returns a `TIMESTAMP`.
TIMESTAMP_TO_SCN Function on page 43-16	Takes a `TIMESTAMP` as input and returns the current SCN as an Oracle number datatype

DISABLE Procedure

This procedure disables the Flashback mode for the entire session.

Syntax

```
DBMS_FLASHBACK.DISABLE;
```

Examples

The following example queries the salary of an employee, Joe, on August 30, 2000:

```
EXECUTE dbms_flashback.enable_at_time('30-AUG-2000');
SELECT salary FROM emp where name = 'Joe'
EXECUTE dbms_flashback.disable;
```

ENABLE_AT_SYSTEM_CHANGE_NUMBER Procedure

This procedure takes an SCN as an input parameter and sets the session snapshot to the specified number. In the Flashback mode, all queries return data consistent as of the specified wall-clock time or SCN. It enables Flashback for the entire session.

Syntax

```
DBMS_FLASHBACK.ENABLE_AT_SYSTEM_CHANGE_NUMBER (
   query_scn IN NUMBER);
```

Parameters

Table 43–3 ENABLE_AT_SYSTEM_CHANGE_NUMBER Procedure Parameters

Parameter	Description
query_scn	The system change number (SCN), a version number for the database that is incremented on every transaction commit.

ENABLE_AT_TIME Procedure

This procedure enables Flashback for the entire session. The snapshot time is set to the SCN that most closely matches the time specified in query_time.It enables Flashback for the entire session.

Syntax

```
DBMS_FLASHBACK.ENABLE_AT_TIME (
   query_time   IN TIMESTAMP);
```

Parameters

Table 43–4 ENABLE_AT_TIME Procedure Parameters

Parameter	Description
query_time	This is an input parameter of type TIMESTAMP. A time stamp can be specified in the following ways: ■ Using the TIMESTAMP constructor EXECUTE DBMS_FLASHBACK.ENABLE_AT_TIME(TIMESTAMP '2001-01-09 12:31:00'). Use the Globalization Support (NLS) format and supply a string. The format depends on the Globalization Support settings. ■ Using the TO_TIMESTAMP function: EXECUTE DBMS_FLASHBACK.ENABLE_AT_TIME(TO_TIMESTAMP('12-02-2001 14:35:00', 'DD-MM-YYYY HH24:MI:SS')) You provide the format you want to use. This example shows the TO_TIMESTAMP function for February 12, 2001, 2:35 PM. ■ If the time is omitted from query time, it defaults to the beginning of the day, that is, 12:00 A.M. ■ Note that if the query time contains a time zone, the time zone information is truncated.

GET_SYSTEM_CHANGE_NUMBER Function

This function returns the current SCN as an Oracle number datatype. You can obtain the current change number and store it for later use. This helps you retain specific snapshots.

Syntax

```
DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER
 RETURN NUMBER;
```

SCN_TO_TIMESTAMP Function

This function takes the SCN as an Oracle number datatype and returns the corresponding `TIMESTAMP`.

Syntax

```
DBMS_FLASHBACK.SCN_TO_TIMESTAMP
  query_scn  IN        NUMBER)
RETURN TIMESTAMP;
```

Parameters

Table 43–5 SCN_TO_TIMESTAMP Procedure Parameters

Parameter	Description
`query_scn`	The system change number (SCN), a version number for the database that is incremented on every transaction commit.

TIMESTAMP_TO_SCN Function

This function takes a TIMESTAMP as input and returns the corresponding SCN as an Oracle number datatype.

Syntax

```
DBMS_FLASHBACK.TIMESTAMP_TO_SCN
  query_time    IN        TIMESTAMP
RETURN NUMBER);
```

Parameters

Table 43–6 TIMESTAMP_TO_SCN Procedure Parameters

Parameter	Description

query_time	This is an input parameter of type TIMESTAMP. A time stamp can be specified in the following ways:
	■ Using the TIMESTAMP constructor
	`EXECUTE DBMS_FLASHBACK.ENABLE_AT_TIME(TIMESTAMP '2001-01-09 12:31:00')`
	Use the Globalization Support (NLS) format and supply a string. The format depends on the Globalization Support settings.
	■ Using the TO_TIMESTAMP function:
	`EXECUTE DBMS_FLASHBACK.ENABLE_AT_TIME(TO_TIMESTAMP('12-02-2001 14:35:00', 'DD-MM-YYYY HH24:MI:SS'))`
	You provide the format you want to use. This example shows the TO_TIMESTAMP function for February 12, 2001, 2:35 PM.
	■ If the time is omitted from query time, it defaults to the beginning of the day, that is, 12:00 A.M.
	■ Note that if the query time contains a time zone, the time zone information is truncated.

44

DBMS_FREQUENT_ITEMSET

The DBMS_FREQUENT_ITEMSET package enables frequent itemset counting. The two functions are identical except in the input cursor format difference.

This chapter contains the following topics:

Summary of DBMS_FREQUENT_ITEMSET Subprograms

Table 44–1 DBMS_FREQUENT_ITEMSET Package Subprograms

Subprogram	Description
FI_HORIZONTAL Function on page 44-3	Counts all frequent itemsets given a cursor for input data which is in 'HORIZONTAL' row format, support threshold, minimum itemset length, maximum itemset length, items to be included, items to be excluded
FI_TRANSACTIONAL Function on page 44-5	Counts all frequent itemsets given a cursor for input data which is in 'TRANSACTIONAL' row format, support threshold, minimum itemset length, maximum itemset length, items to be included, items to be excluded

FI_HORIZONTAL Function

The purpose of this table function is to count all frequent itemsets given a cursor for input data which is in 'HORIZONTAL' row format, support threshold, minimum itemset length, maximum itemset length, items to be included, items to be excluded. The result will be a table of rows in form of itemset, support, length, total transactions counted.

In 'HORIZONTAL' row format, each row contains all of the item ids for a single transaction. Since all of the items come together, no transaction id is necessary.

The benefit of this table function is that if an application already has data in horizontal format, the database can skip the step of transforming rows that are in transactional format into horizontal format.

Syntax

```
DBMS_FREQUENT_ITEMSET.FI_HORIZONTAL(
   tranx_cursor         IN    SYSREFCURSOR,
   support_threshold    IN    NUMBER,
   itemset_length_min   IN    NUMBER,
   itemset_length_max   IN    NUMBER,
   including_items      IN    SYS_REFCURSOR DEFAULT NULL,
   excluding_items      IN    SYS_REFCURSOR DEFAULT NULL)
  RETURN TABLE OF ROW (
     itemset [Nested Table of Item Type DERIVED FROM tranx_cursor],
     support        NUMBER,
     length         NUMBER,
     total_tranx    NUMBER);
```

Parameters

Table 44–2 FI_HORIZONTAL Procedure Parameters

Parameter	Description
tranx_cursor	The cursor parameter that the user will supply when calling the function. There is no limits on the number of returning columns.Each column of cursor represents an item. All columns of the cursor must be of the same data type. The item id must be number or character type (for example, VARCHAR2(n)).
support_threshold	A fraction number of total transaction count. An itemset is termed "frequent" if [the number of transactions it occurs in] divided by [the total number of transactions] exceed the fraction. The parameter must be a NUMBER.
itemset_length_min	The minimum length for interested frequent itemset. The parameter must be a NUMBER between 1 and 20, inclusive.
itemset_length_max	The maximum length for interested frequent itemset. This parameter must be a NUMBER between 1 and 20, inclusive, and must not be less than itemset_length_min.
including_items	A cursor from which a list of items can be fetched. At least one item from the list must appear in frequent itemsets that are returned. The default is NULL.
excluding_items	A cursor from which a list of items can be fetched. No item from the list can appear in frequent itemsets that are returned.The default is NULL.

Return Values

Table 44–3 FI_HORIZONTAL Procedure Parameters

Parameter	Description
support	The number of transactions in which a frequent itemset occurs. This will be returned as a NUMBER.
itemset	A collection of items which is computed as frequent itemset. This will be returned as a nested table of item type which is the item column type of the input cursor.
length	Number of items in a frequent itemset. This will be returned as a NUMBER.
total_tranx	The total transaction count. This will be returned as a NUMBER.

Example

Suppose you have a table horiz_table_in.

```
horiz_table_in(iid1 VARCHAR2(30), iid2 VARCHAR2(30), iid3 VARCHAR2(30), iid4
VARCHAR2(30), iid5 VARCHAR2(30));
```

and the data in horiz_table_in looks as follows:

```
('apple', 'banana', NULL, NULL, NULL)
('apple', 'milk', 'banana', NULL, NULL)
('orange', NULL, NULL, NULL, NULL)
```

Suppose you want to find out what combinations of items is frequent with a given support threshold of 30%, requiring itemset containing at least one of ('apple','banana','orange'), but excluding any of ('milk') in any itemset. You use the following query:

```
CREATE TYPE fi_varchar_nt AS TABLE OF VARCHAR2(30);
SELECT CAST(itemset as FI_VARCHAR_NT)itemset, support, length, total_tranx
   FROM table(DBMS_FREQUENT_ITEMSET.FI_HORIZONTAL(
                CURSOR(SELECT iid1, iid2, iid3, iid4, iid5
                         FROM horiz_table_in),
                0.3,
                2,
                5,
                CURSOR(SELECT * FROM table(FI_VARCHAR_NT
                                    ('apple','banana','orange'))),
                CURSOR(SELECT * FROM table(FI_VARCHAR_NT('milk')))));
```

FI_TRANSACTIONAL Function

This procedure counts all frequent itemsets given a cursor for input data which is in 'TRANSACTIONAL' row format, support threshold, minimum itemset length, maximum itemset length, items to be included, items to be excluded. The result will be a table of rows in form of itemset, support, length, total number of transactions.

In 'TRANSACTIONAL' row format, each transaction is spread across multiple rows. All the rows of a given transaction have the same transaction id, and each row has a different item id. Combining all of the item ids which share a given transaction id results in a single transaction.

Syntax

```
DBMS_FREQUENT_ITEMSET.FI_TRANSACTIONAL (
   tranx_cursor         IN    SYSREFCURSOR,
   support_threshold    IN    NUMBER,
   itemset_length_min   IN    NUMBER,
   itemset_length_max   IN    NUMBER,
   including_items      IN    SYS_REFCURSOR DEFAULT NULL,
   excluding_items      IN    SYS_REFCURSOR DEFAULT NULL)
  RETURN TABLE OF ROW (
     itemset [Nested Table of Item Type DERIVED FROM tranx_cursor],
     support        NUMBER,
     length         NUMBER,
     total_tranx    NUMBER);
```

Parameters

Table 44–4 FI_TRANSACTIONAL Procedure Parameters

Parameter	Description
tranx_cursor	The cursor parameter that the user will supply when calling the function. It should return two columns in its returning row, the first column being the transaction id, the second column being the item id. The item id must be number or character type (for example, VARCHAR2(n)).
support_threshold	A fraction number of total transaction count. An itemset is termed "frequent" if [the number of transactions it occurs in] divided by [the total number of transactions] exceed the fraction. The parameter must be a NUMBER.
itemset_length_min	The minimum length for interested frequent itemset. The parameter must be a NUMBER between 1 and 20, inclusive.
itemset_length_max	The maximum length for interested frequent itemset. This parameter must be a NUMBER between 1 and 20, inclusive, and must not be less than itemset_length_min.
including_items	A cursor from which a list of items can be fetched. At least one item from the list must appear in frequent itemsets that will be returned. The default is NULL.
excluding_items	A cursor from which a list of items can be fetched. No item from the list can appear in frequent itemsets that will returned. The default is NULL.

Return Values

Table 44–5 FI_TRANSACTIONAL Procedure Parameters

Parameter	Description
support	The number of transactions in which a frequent itemset occurs. This will be returned as a NUMBER.
itemset	A collection of items which is computed as frequent itemset. This will be returned as a nested table of item type which is the item column type of the input cursor.
length	Number of items in a frequent itemset. This will be returned as a NUMBER.
total_tranx	The total transaction count. This will be returned as a NUMBER, and will be the same for all returned rows, similar to a reporting aggregate.

Usage Notes

Applications must predefine a nested table type of the input item type and cast the output itemset into this predefined nested table type before further processing, such as loading into a table.

Examples

Suppose that the input table tranx_table_in looks as follows:

```
(1, 'apple')
(1, 'banana')
(2, 'apple')
(2, 'milk')
(2, 'banana')
(3, 'orange')
```

and the user is trying to find itemsets that satisfy a support-threshold of 60% and have the itemset-length greater than 1 (namely, (apple, banana)).

The output of this function would contain the following output row:

```
itemset=('apple','banana'), support=2, length=2, total_tranx=3
```

You need to create a nested table of item type before you submit a query to perform the frequent itemset counting. In this example, since item is of VARCHAR2(30), you must create a nested table of VARCHAR2(30):

```
CREATE TYPE fi_varchar_nt AS TABLE OF VARCHAR2(30);
SELECT CAST(itemset as FI_VARCHAR_NT) itemset, support, length, total_tranx
   FROM table(DBMS_FREQUENT_ITEMSET.FI_TRANSACTIONAL(
                cursor(SELECT tid, iid FROM tranx_table_in),
                0.6,
                2,
                5,
                NULL,
                NULL));
```

Here is another example to illustrate how to include certain items and exclude certain items in the counting.

```
SELECT CAST(itemset as FI_VARCHAR_NT)itemset, support, length, total_tranx
   FROM table(DBMS_FREQUENT_ITEMSET.FI_TRANSACTIONAL(
                CURSOR(SELECT tid, iid FROM tranx_table_in),
```

```
              0.6,
              2,
              5,
             CURSOR(SELECT * FROM table(FI_VARCHAR_NT
                                        ('apple','banana','orange'))),
             CURSOR(SELECT * FROM table(FI_VARCHAR_NT('milk'))));
```

Using the including/excluding items parameter, you are able to further optimize the execution by ignoring itemsets that are not expected by application.

You can also use transactional output through collection unnesting:

```
SELECT
  bt.setid, nt.*
FROM
(SELECT cast(Itemset as FI_VARCHAR_NT) itemset, rownum setid
 FROM table(
   DBMS_FREQUENT_ITEMSET.FI_TRANSACTIONAL(
     CURSOR(SELECT tid, iid FROM tranx_table_in), 0.6, 2, 5,
     NULL, NULL))) bt,
table(bt.itemset) nt;
```

If you want to use an insert statement to load frequent itemsets into a nested table, it is better to use the `NESTED_TABLE_FAST_INSERT` hint for performance:

```
CREATE TABLE fq_nt (coll FI_VARCHAR_NT) NESTED TABLE coll STORE AS
  coll_nest;
INSERT /*+ NESTED_TABLE_FAST_INSERT */  INTO fq_nt
 SELECT cast(itemset as FI_VARCHAR_NT)
 FROM table(DBMS_FREQUENT_ITEMSET.FI_TRANSACTIONAL(
        cursor(SELECT tid, iid FROM tranx_table_in), 0.6, 2, 5,
        NULL, NULL));
```

Note that if you want to use the package inside a PL/SQL cursor, you must cast the return type of the table function:

```
CREATE TYPE fi_res AS OBJECT (
  itemset      FI_VARCHAR_NT,
  support      NUMBER,
  length       NUMBER,
  total_tranx  NUMBER
);
/
CREATE TYPE fi_coll AS TABLE OF fi_res;
/

DECLARE
  cursor freqC is
    SELECT Itemset
    FROM table(
      CAST(DBMS_FREQUENT_ITEMSET.FI_TRANSACTIONAL(
        cursor(SELECT tid, iid FROM tranx_table_in), 0.6, 2, 5,
        NULL, NULL) AS fi_coll));
  coll_nt  FI_VARCHAR_NT;
  num_rows int;
  num_itms int;
BEGIN
  num_rows := 0;
  num_itms := 0;
  OPEN freqC;
  LOOP
```

```
      FETCH freqC INTO coll_nt;
      EXIT WHEN freqC%NOTFOUND;
      num_rows := num_rows + 1;
      num_itms := num_itms + coll_nt.count;
    END LOOP;
  CLOSE freqC;
  DBMS_OUTPUT.PUT_LINE('Totally ' || num_rows || ' rows ' || num_itms || '
items were produced.');
END;
/
```

45

DBMS_HS_PASSTHROUGH

The pass-through SQL feature allows an application developer to send a statement directly to a non-Oracle system without being interpreted by the Oracle server. This can be useful if the non-Oracle system allows for operations in statements for which there is no equivalent in Oracle.

You can run these statements directly at the non-Oracle system using the PL/SQL package `DBMS_HS_PASSTHROUGH`. Any statement executed with this package is run in the same transaction as regular "transparent" SQL statements.

This chapter discusses the following topic:

- Summary of DBMS_HS_PASSTHROUGH Subprograms

> **See Also:** *Oracle Database Heterogeneous Connectivity Administrator's Guide*

Summary of DBMS_HS_PASSTHROUGH Subprograms

Table 45–1 DBMS_HS_PASSTHROUGH Package Subprograms

Subprogram	Description
BIND_INOUT_VARIABLE Procedure on page 45-3	Binds `IN OUT` bind variables
BIND_INOUT_VARIABLE_ RAW Procedure on page 45-4	Binds `IN OUT` bind variables of datatype `RAW`
BIND_OUT_VARIABLE Procedure on page 45-5	Binds an `OUT` variable with a PL/SQL program variable
BIND_OUT_VARIABLE_RAW Procedure on page 45-6	Binds an `OUT` variable of datatype `RAW` with a PL/SQL program variable
BIND_VARIABLE Procedure on page 45-7	Binds an `IN` variable positionally with a PL/SQL program variable
BIND_VARIABLE_RAW Procedure on page 45-8	Binds `IN` variables of type `RAW`
CLOSE_CURSOR Procedure on page 45-9	Closes the cursor and releases associated memory after the SQL statement has been run at the non-Oracle system
EXECUTE_IMMEDIATE Procedure on page 45-10	Runs a (non-`SELECT`) SQL statement immediately, without bind variables
EXECUTE_NON_QUERY Function on page 45-11	Runs a (non-`SELECT`) SQL statement
FETCH_ROW Function on page 45-12	Fetches rows from a query
GET_VALUE Procedure on page 45-13	Retrieves column value from `SELECT` statement, or retrieves `OUT` bind parameters
GET_VALUE_RAW Procedure on page 45-14	Similar to `GET_VALUE`, but for datatype `RAW`
OPEN_CURSOR Function on page 45-15	Opens a cursor for running a passthrough SQL statement at the non-Oracle system
PARSE Procedure on page 45-16	Parses SQL statement at non-Oracle system

BIND_INOUT_VARIABLE Procedure

This procedure binds IN OUT bind variables.

Syntax

```
DBMS_HS_PASSTHROUGH.BIND_INOUT_VARIABLE (
   c     IN     BINARY_INTEGER NOT NULL,
   p     IN     BINARY_INTEGER NOT NULL,
   v     IN OUT <dty>,
   n     IN     VARCHAR2);
```

<dty> is either DATE, NUMBER, or VARCHAR2.

> **See Also:** For binding IN OUT variables of datatype RAW see BIND_INOUT_VARIABLE_RAW Procedure on page 45-4.

Parameters

Table 45–2 BIND_INOUT_VARIABLE Procedure Parameters

Parameter	Description
c	Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.
p	Position of the bind variable in the SQL statement: Starts at 1.
v	This value is used for two purposes: - To provide the IN value before the SQL statement is run. - To determine the size of the out value.
n	(Optional) Name of the bind variable. For example, in SELECT * FROM emp WHERE ename=:ename, the position of the bind variable :ename is 1, the name is :ename. This parameter can be used if the non-Oracle system supports "named binds" instead of positional binds. Passing the position is still required.

Exceptions

Table 45–3 BIND_INOUT_VARIABLE Procedure Exceptions

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

```
Purity level defined : WNDS, RNDS
```

BIND_INOUT_VARIABLE_RAW Procedure

This procedure binds IN OUT bind variables of datatype RAW.

Syntax

```
DBMS_HS_PASSTHROUGH.BIND_INOUT_VARIABLE_RAW (
   c      IN     BINARY_INTEGER NOT NULL,
   p      IN     BINARY_INTEGER NOT NULL,
   v      IN OUT RAW,
   n      IN     VARCHAR2);
```

Parameters

Table 45–4 BIND_INOUT_VARIABLE_RAW Procedure Parameters

Parameter	Description
c	Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed using the routines OPEN_CURSOR and PARSE respectively.
p	Position of the bind variable in the SQL statement: Starts at 1.
v	This value is used for two purposes: - To provide the IN value before the SQL statement is run. - To determine the size of the out value.
n	(Optional) Name the bind variable. For example, in SELECT * FROM emp WHERE ename=:ename, the position of the bind variable :ename is 1, the name is :ename. This parameter can be used if the non-Oracle system supports "named binds" instead of positional binds. Passing the position is still required.

Exceptions

Table 45–5 BIND_INOUT_VARIABLE_RAW Procedure Exceptions

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

```
Purity level defined : WNDS, RNDS
```

BIND_OUT_VARIABLE Procedure

This procedure binds an OUT variable with a PL/SQL program variable.

Syntax

```
DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE (
   c      IN  BINARY_INTEGER NOT NULL,
   p      IN  BINARY_INTEGER NULL,
   v      OUT <dty>,
   n      IN  VARCHAR2);
```

<dty> is either DATE, NUMBER, or VARCHAR2.

> **See Also:** For binding OUT variables of datatype RAW, see BIND_OUT_VARIABLE_RAW Procedure on page 45-6.

Parameters

Table 45–6 BIND_OUT_VARIABLE Procedure Parameters

Parameter	Description
c	Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.
p	Position of the bind variable in the SQL statement: Starts at 1.
v	Variable in which the OUT bind variable stores its value. The package remembers only the "size" of the variable. After the SQL statement is run, you can use GET_VALUE to retrieve the value of the OUT parameter. The size of the retrieved value should not exceed the size of the parameter that was passed using BIND_OUT_VARIABLE.
n	(Optional) Name of the bind variable. For example, in SELECT * FROM emp WHERE ename=:ename, the position of the bind variable :ename is 1, the name is :ename. This parameter can be used if the non-Oracle system supports "named binds" instead of positional binds. Passing the position is still required.

Exceptions

Table 45–7 BIND_OUT_VARIABLE Procedure Exceptions

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

```
Purity level defined : WNDS, RNDS
```

BIND_OUT_VARIABLE_RAW Procedure

This procedure binds an OUT variable of datatype RAW with a PL/SQL program variable.

Syntax

```
DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE_RAW (
   c     IN  BINARY_INTEGER NOT NULL,
   p     IN  BINARY_INTEGER NOT NULL,
   v     OUT RAW,
   n     IN  VARCHAR2);
```

Parameters

Table 45–8 BIND_OUT_VARIABLE_RAW Procedure Parameters

Parameter	Description
c	Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.
p	Position of the bind variable in the SQL statement: Starts at 1.
v	Variable in which the OUT bind variable stores its value. The package remembers only the "size" of the variable. After the SQL statement is run, you can use GET_VALUE to retrieve the value of the OUT parameter. The size of the retrieved value should not exceed the size of the parameter that was passed using BIND_OUT_VARIABLE_RAW.
n	(Optional) Name of the bind variable. For example, in SELECT * FROM emp WHERE ename=:ename, the position of the bind variable :ename is 1, the name is :ename. This parameter can be used if the non-Oracle system supports "named binds" instead of positional binds. Passing the position is still required.

Exceptions

Table 45–9 BIND_OUT_VARIABLE_RAW Procedure Exceptions

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

```
Purity level defined : WNDS, RNDS
```

BIND_VARIABLE Procedure

This procedure binds an IN variable positionally with a PL/SQL program variable.

Syntax

```
DBMS_HS_PASSTHROUGH.BIND_VARIABLE (
   c      IN BINARY_INTEGER NOT NULL,
   p      IN BINARY_INTEGER NOT NULL,
   v      IN <dty>,
   n      IN VARCHAR2);
```

<dty> is either DATE, NUMBER, or VARCHAR2.

> **See Also:** To bind RAW variables use BIND_VARIABLE_RAW Procedure on page 45-8.

Parameters

Table 45–10 BIND_VARIABLE Procedure Parameters

Parameter	Description
c	Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed using the routines OPEN_CURSOR and PARSE respectively.
p	Position of the bind variable in the SQL statement: Starts at 1.
v	Value that must be passed to the bind variable name.
n	(Optional) Name of the bind variable. For example, in SELECT * FROM emp WHERE ename=:ename, the position of the bind variable :ename is 1, the name is :ename. This parameter can be used if the non-Oracle system supports "named binds" instead of positional binds. Passing the position is still required.

Exceptions

Table 45–11 BIND_VARIABLE Procedure Exceptions

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

```
Purity level defined: WNDS, RNDS
```

BIND_VARIABLE_RAW Procedure

This procedure binds IN variables of type RAW.

Syntax

```
DBMS_HS_PASSTHROUGH.BIND_VARIABLE_RAW (
   c    IN BINARY_INTEGER NOT NULL,
   p    IN BINARY_INTEGER NOT NULL,
   v    IN RAW,
   n    IN VARCHAR2);
```

Parameters

Table 45–12 BIND_VARIABLE_RAW Procedure Parameters

Parameter	Description
c	Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.
p	Position of the bind variable in the SQL statement: Starts at 1.
v	Value that must be passed to the bind variable.
n	(Optional) Name of the bind variable. For example, in SELECT * FROM emp WHERE ename=:ename, the position of the bind variable :ename is 1, the name is :ename. This parameter can be used if the non-Oracle system supports "named binds" instead of positional binds. Passing the position is still required.

Exceptions

Table 45–13 BIND_VARIABLE_RAW Procedure Exceptions

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

```
Purity level defined : WNDS, RNDS
```

CLOSE_CURSOR Procedure

This function closes the cursor and releases associated memory after the SQL statement has been run at the non-Oracle system. If the cursor was not open, then the operation is a "no operation".

Syntax

```
DBMS_HS_PASSTHROUGH.CLOSE_CURSOR (
   c IN BINARY_INTEGER NOT NULL);
```

Parameters

Table 45–14 CLOSE_CURSOR Procedure Parameters

Parameter	Description
c	Cursor to be released.

Exceptions

Table 45–15 CLOSE_CURSOR Procedure Exceptions

Exception	Description
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

```
Purity level defined : WNDS, RNDS
```

EXECUTE_IMMEDIATE Procedure

This function runs a SQL statement immediately. Any valid SQL command except SELECT can be run immediately. The statement must not contain any bind variables. The statement is passed in as a VARCHAR2 in the argument. Internally the SQL statement is run using the PASSTHROUGH SQL protocol sequence of OPEN_CURSOR, PARSE, EXECUTE_NON_QUERY, CLOSE_CURSOR.

Syntax

```
DBMS_HS_PASSTHROUGH.EXECUTE_IMMEDIATE (
   s IN VARCHAR2 NOT NULL)
RETURN BINARY_INTEGER;
```

Parameters

Table 45–16 EXECUTE_IMMEDIATE Procedure Parameters

Parameter	Description
s	VARCHAR2 variable with the statement to be executed immediately.

Return Values

The number of rows affected by the execution of the SQL statement.

Exceptions

Table 45–17 EXECUTE_IMMEDIATE Procedure Exceptions

Exception	Description
ORA-28551	SQL statement is invalid.
ORA-28554	Max open cursors.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

EXECUTE_NON_QUERY Function

This function runs a SQL statement. The SQL statement cannot be a SELECT statement. A cursor has to be open and the SQL statement has to be parsed before the SQL statement can be run.

Syntax

```
DBMS_HS_PASSTHROUGH.EXECUTE_NON_QUERY (
   c IN BINARY_INTEGER NOT NULL)
  RETURN BINARY_INTEGER;
```

Parameters

Table 45–18 EXECUTE_NON_QUERY Function Parameters

Parameter	Description
c	Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.

Return Values

The number of rows affected by the SQL statement in the non-Oracle system

Exceptions

Table 45–19 EXECUTE_NON_QUERY Procedure Exceptions

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28552	BIND_VARIABLE procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28555	A NULL value was passed for a NOT NULL parameter.

FETCH_ROW Function

This function fetches rows from a result set. The result set is defined with a SQL SELECT statement. When there are no more rows to be fetched, the exception NO_DATA_FOUND is raised. Before the rows can be fetched, a cursor has to be opened, and the SQL statement has to be parsed.

Syntax

```
DBMS_HS_PASSTHROUGH.FETCH_ROW (
   c   IN BINARY_INTEGER NOT NULL,
   f   IN BOOLEAN)
  RETURN BINARY_INTEGER;
```

Parameters

Table 45–20 FETCH_ROW Function Parameters

Parameter	Description
c	Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.
first	(Optional) Reexecutes SELECT statement. Possible values: – TRUE: reexecute SELECT statement. – FALSE: fetch the next row, or if run for the first time, then execute and fetch rows (default).

Return Values

The returns the number of rows fetched. The function returns "0" if the last row was already fetched.

Exceptions

Table 45–21 FETCH_ROW Procedure Exceptions

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

```
Purity level defined : WNDS
```

GET_VALUE Procedure

This procedure has two purposes:

- It retrieves the select list items of SELECT statements, after a row has been fetched.
- It retrieves the OUT bind values, after the SQL statement has been run.

Syntax

```
DBMS_HS_PASSTHROUGH.GET_VALUE (
   c      IN  BINARY_INTEGER NOT NULL,
   p    IN  BINARY_INTEGER NOT NULL,
   v    OUT <dty>);
```

<dty> is either DATE, NUMBER, or VARCHAR2.

> **See Also:** For retrieving values of datatype RAW, see GET_VALUE_RAW Procedure on page 45-14.

Parameters

Table 45–22 GET_VALUE Procedure Parameters

Parameter	Description
c	Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.
p	Position of the bind variable or select list item in the SQL statement: Starts at 1.
v	Variable in which the OUT bind variable or select list item stores its value.

Exceptions

Table 45–23 GET_VALUE Procedure Exceptions

Exception	Description
ORA-1403	Returns NO_DATA_FOUND exception when running the GET_VALUE after the last row was fetched (that is, FETCH_ROW returned "0").
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

```
Purity level defined : WNDS
```

GET_VALUE_RAW Procedure

This procedure is similar to GET_VALUE, but for datatype RAW.

Syntax

```
DBMS_HS_PASSTHROUGH.GET_VALUE_RAW (
   c    IN  BINARY_INTEGER NOT NULL,
   p    IN  BINARY_INTEGER NOT NULL,
   v    OUT RAW);
```

Parameters

Table 45–24 GET_VALUE_RAW Procedure Parameters

Parameter	Description
c	Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.
p	Position of the bind variable or select list item in the SQL statement: Starts at 1.
v	Variable in which the OUT bind variable or select list item stores its value.

Exceptions

Table 45–25 GET_VALUE_RAW Procedure Exceptions

Exception	Description
ORA-1403	Returns NO_DATA_FOUND exception when running the GET_VALUE after the last row was fetched (that is, FETCH_ROW returned "0").
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

```
Purity level defined : WNDS
```

OPEN_CURSOR Function

This function opens a cursor for running a pass-through SQL statement at the non-Oracle system. This function must be called for any type of SQL statement.

The function returns a cursor, which must be used in subsequent calls. This call allocates memory. To deallocate the associated memory, call the procedure `CLOSE_CURSOR`.

Syntax

```
DBMS_HS_PASSTHROUGH.OPEN_CURSOR
  RETURN BINARY_INTEGER;
```

Return Values

The cursor to be used on subsequent procedure and function calls.

Exceptions

Table 45–26 OPEN_CURSOR Function Exceptions

Exception	Description
`ORA-28554`	Maximum number of open cursor has been exceeded. Increase Heterogeneous Services' `OPEN_CURSORS` initialization parameter.

Pragmas

```
Purity level defined : WNDS, RNDS
```

PARSE Procedure

This procedure parses SQL statement at non-Oracle system.

Syntax

```
DBMS_HS_PASSTHROUGH.PARSE (
   c       IN  BINARY_INTEGER NOT NULL,
   stmt    IN  VARCHAR2 NOT NULL);
```

Parameters

Table 45–27 PARSE Procedure Parameters

Parameter	Description
c	Cursor associated with the pass-through SQL statement. Cursor must be opened using function OPEN_CURSOR.
stmt	Statement to be parsed.

Exceptions

Table 45–28 PARSE Procedure Exceptions

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28551	SQL statement is illegal.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

```
Purity level defined : WNDS, RNDS
```

46

DBMS_IOT

The DBMS_IOT package creates a table into which references to the chained rows for an index-organized table can be placed using the ANALYZE command. DBMS_IOT can also create an exception table into which references to the rows of an index-organized table that violate a constraint can be placed during the enable_constraint operation.

DBMS_IOT is not loaded during database installation. To install DBMS_IOT, run dbmsiotc.sql, available in the ADMIN directory.

This chapter contains the following topics:

> **Note:** With the introduction of logical-rowids for IOTs with Oracle Database Release 8.1, you no longer need to use the procedures contained in this package which is retained for backward compatibility only. It is however required for servers running with Oracle Database Release 8.0.

Summary of DBMS_IOT Subprograms

Table 46–1 DBMS_IOT Package Subprograms

Subprogram	Description
BUILD_CHAIN_ROWS_TABLE Procedure on page 46-3	Creates a table into which references to the chained rows for an index-organized table can be placed using the `ANALYZE` command
BUILD_EXCEPTIONS_TABLE Procedure on page 46-4	Creates an exception table into which rows of an index-organized table that violate a constraint can be placed

BUILD_CHAIN_ROWS_TABLE Procedure

This procedure creates a table into which references to the chained rows for an index-organized table can be placed using the ANALYZE command.

Syntax

```
DBMS_IOT.BUILD_CHAIN_ROWS_TABLE (
   owner               IN VARCHAR2,
   iot_name            IN VARCHAR2,
   chainrow_table_name IN VARCHAR2 default 'IOT_CHAINED_ROWS');
```

Parameters

Table 46–2 BUILD_CHAIN_ROWS_TABLE Procedure Parameters

Parameter	Description
owner	Owner of the index-organized table.
iot_name	Index-organized table name.
chainrow_table_name	Intended name for the chained-rows table.

Usage Notes

You should create a separate chained-rows table for each index-organized table to accommodate its primary key.

Examples

```
CREATE TABLE l(a char(16),b char(16), c char(16), d char(240),
PRIMARY KEY(a,b,c)) ORGANIZATION INDEX pctthreshold 10 overflow;
EXECUTE DBMS_IOT.BUILD_CHAIN_ROWS_TABLE('SYS','L','LC');
```

A chained-row table is created with the following columns:

```
Column Name                    Null?    Type
------------------------------ -------- ----
OWNER_NAME                              VARCHAR2(30)
TABLE_NAME                              VARCHAR2(30)
CLUSTER_NAME                            VARCHAR2(30)
PARTITION_NAME                          VARCHAR2(30)
SUBPARTITION_NAME                       VARCHAR2(30)
HEAD_ROWID                              ROWID
TIMESTAMP                               DATE
A                                       CHAR(16)
B                                       CHAR(16)
C                                       CHAR(16)
```

BUILD_EXCEPTIONS_TABLE Procedure

This procedure creates an exception table into which rows of an index-organized table that violate a constraint can be placed during the execution of the following SQL statements:

- ALTER TABLE ... ENABLE CONSTRAINT ... EXCEPTIONS INTO
- ALTER TABLE ... ADD CONSTRAINT ... EXCEPTIONS INTO

Syntax

```
DBMS_IOT.BUILD_EXCEPTIONS_TABLE (
   owner                 IN VARCHAR2,
   iot_name              IN VARCHAR2,
   exceptions_table_name IN VARCHAR2 default 'IOT_EXCEPTIONS');
```

Parameters

Table 46–3 BUILD_EXCEPTIONS_TABLE Procedure Parameters

Parameter	Description
owner	Owner of the index-organized table.
iot_name	Index-organized table name.
exceptions_table_name	Intended name for exception-table.

Usage Notes

You should create a separate exception table for each index-organized table to accommodate its primary key.

Examples

```
EXECUTE DBMS_IOT.BUILD_EXCEPTIONS_TABLE('SYS','L','LE');
```

An exception table for the preceding index-organized table with the following columns:

```
Column Name                    Null?    Type
------------------------------ -------- ----
ROW_ID                                  VARCHAR2(30)
OWNER                                   VARCHAR2(30)
TABLE_NAME                              VARCHAR2(30)
CONSTRAINT                              VARCHAR2(30)
A                                       CHAR(16)
B                                       CHAR(16)
C                                       CHAR(16)
```

47

DBMS_JAVA

The DBMS_JAVA package provides a PL/SQL interface for accessing database functionality from Java.

- Documentation of DBMS_JAVA

Documentation of DBMS_JAVA

For a complete description of this package within the context of DBMS_JAVA, see DBMS_JAVA in the *Oracle Database Java Developer's Guide.*

48

DBMS_JOB

The DBMS_JOB package schedules and manages jobs in the job queue.

> **Note:** The DBMS_JOB package has been superseded by the DBMS_SCHEDULER package. In particular, if you are administering jobs to manage system load, you should consider disabling DBMS_JOB by revoking the package execution privilege for users.
>
> For more information, see Chapter 93, "DBMS_SCHEDULER" and "Moving from DBMS_JOB to DBMS_SCHEDULER" in *Oracle Database Administrator's Guide*.

This chapter contains the following topics:

Using DBMS_JOB

Security Model

No specific system privileges are required to use DBMS_JOB. No system privileges are available to manage DBMS_JOB. Jobs cannot be altered or deleted other than jobs owned by the user. This is true for all users including those users granted DBA privileges.

You can execute procedures that are owned by the user or for which the user is explicitly granted EXECUTE. However, procedures for which the user is granted the execute privilege through roles cannot be executed.

Note that, once a job is started and running, there is no easy way to stop the job.

Operational Notes

Working with Real Application Clusters

DBMS_JOB supports multi-instance execution of jobs. By default jobs can be executed on any instance, but only one single instance will execute the job. In addition, you can force instance binding by binding the job to a particular instance. You implement instance binding by specifying an instance number to the instance affinity parameter. Note, however, that in Oracle Database 10g Release 1 (10.1) instance binding is not recommended. Service affinity is preferred. This concept is implemented in the DBMS_SCHEDULER package.

The following procedures can be used to create, alter or run jobs with instance affinity. Note that not specifying affinity means any instance can run the job.

DBMS_JOB.SUBMIT

To submit a job to the job queue, use the following syntax:

```
DBMS_JOB.SUBMIT(
   job       OUT    BINARY_INTEGER,
   what      IN     VARCHAR2, NEXT_DATE IN DATE DEFAULTSYSDATE,
   interval  IN     VARCHAR2 DEFAULT 'NULL',
   no_parse  IN     BOOLEAN DEFAULT FALSE,
   instance  IN     BINARY_INTEGER DEFAULT ANY_INSTANCE,
   force     IN     BOOLEAN DEFAULT FALSE);
```

Use the parameters instance and force to control job and instance affinity. The default value of instance is 0 (zero) to indicate that any instance can execute the job. To run the job on a certain instance, specify the instance value. Oracle displays error ORA-23319 if the instance value is a negative number or NULL.

The force parameter defaults to false. If force is TRUE, any positive integer is acceptable as the job instance. If force is FALSE, the specified instance must be running, or Oracle displays error number ORA-23428.

DBMS_JOB.INSTANCE

To assign a particular instance to execute a job, use the following syntax:

```
DBMS_JOB.INSTANCE(  JOB IN BINARY_INTEGER,
  instance                IN BINARY_INTEGER,
  force                   IN BOOLEAN DEFAULT FALSE);
```

The FORCE parameter in this example defaults to FALSE. If the instance value is 0 (zero), job affinity is altered and any available instance can execute the job despite the value of force. If the INSTANCE value is positive and the FORCE parameter is FALSE, job affinity is altered only if the specified instance is running, or Oracle displays error ORA-23428.

If the force parameter is TRUE, any positive integer is acceptable as the job instance and the job affinity is altered. Oracle displays error ORA-23319 if the instance value is negative or NULL.

DBMS_JOB.CHANGE

To alter user-definable parameters associated with a job, use the following syntax:

```
DBMS_JOB.CHANGE(  JOB IN BINARY_INTEGER,
```

```
   what                 IN VARCHAR2 DEFAULT NULL,
next_date             IN DATE DEFAULT NULL,
interval              IN VARCHAR2 DEFAULT NULL,
instance              IN BINARY_INTEGER DEFAULT NULL,
force                 IN BOOLEAN DEFAULT FALSE );
```

Two parameters, `instance` and `force`, appear in this example. The default value of `instance` is `null` indicating that job affinity will not change.

The default value of `force` is `FALSE`. Oracle displays error `ORA-23428` if the specified instance is not running and error `ORA-23319` if the `instance` number is negative.

DBMS_JOB.RUN

The `force` parameter for `DBMS_JOB.RUN` defaults to `FALSE`. If `force` is `TRUE`, instance affinity is irrelevant for running jobs in the foreground process. If force is `FALSE`, the job can run in the foreground only in the specified instance. Oracle displays error `ORA-23428` if force is `FALSE` and the connected instance is the incorrect instance.

```
DBMS_JOB.RUN(
   job    IN BINARY_INTEGER,
   force  IN BOOLEAN DEFAULT FALSE);
```

Stopping a Job

Note that, once a job is started and running, there is no easy way to stop the job.

Summary of DBMS_JOB Subprograms

Table 48–1 DBMS_JOB Package Subprograms

Subprogram	Description
BROKEN Procedure on page 48-7	Disables job execution
CHANGE Procedure on page 48-8	Alters any of the user-definable parameters associated with a job
INSTANCE Procedure on page 48-9	Assigns a job to be run by a instance
INTERVAL Procedure on page 48-10	Alters the interval between executions for a specified job
NEXT_DATE Procedure on page 48-11	Alters the next execution time for a specified job
REMOVE Procedure on page 48-12	Removes specified job from the job queue
RUN Procedure on page 48-13	Forces a specified job to run
SUBMIT Procedure on page 48-14	Submits a new job to the job queue
USER_EXPORT Procedures on page 48-16	Re-creates a given job for export, or re-creates a given job for export with instance affinity
WHAT Procedure on page 48-17	Alters the job description for a specified job

BROKEN Procedure

This procedure sets the broken flag. Broken jobs are never run.

Syntax

```
DBMS_JOB.BROKEN (
   job       IN  BINARY_INTEGER,
   broken    IN  BOOLEAN,
   next_date IN  DATE DEFAULT SYSDATE);
```

Parameters

Table 48–2 BROKEN Procedure Parameters

Parameter	Description
job	Number of the job being run.
broken	Job broken: IN value is FALSE.
next_data	Date of the next refresh.

> **Note:** If you set job as broken while it is running, Oracle resets the job's status to normal after the job completes. Therefore, only execute this procedure for jobs that are not running.

Usage Notes

You must issue a COMMIT statement immediately after the statement.

CHANGE Procedure

This procedure changes any of the fields a user can set in a job.

Syntax

```
DBMS_JOB.CHANGE (
   job       IN  BINARY_INTEGER,
   what      IN  VARCHAR2,
   next_date IN  DATE,
   interval  IN  VARCHAR2,
   instance  IN  BINARY_INTEGER DEFAULT NULL,
   force     IN  BOOLEAN DEFAULT FALSE);
```

Parameters

Table 48–3 CHANGE Procedure Parameters

Parameter	Description
job	Number of the job being run.
what	PL/SQL procedure to run.
next_date	Date of the next refresh.
interval	Date function; evaluated immediately before the job starts running.
instance	When a job is submitted, specifies which instance can run the job. This defaults to NULL, which indicates that instance affinity is not changed.
force	If this is FALSE, then the specified instance (to which the instance number change) must be running. Otherwise, the routine raises an exception. If this is TRUE, then any positive integer is acceptable as the job instance.

Usage Notes

- You must issue a COMMIT statement immediately after the statement.
- The parameters instance and force are added for job queue affinity. Job queue affinity gives users the ability to indicate whether a particular instance or any instance can run a submitted job.
- If the parameters what, next_date, or interval are NULL, then leave that value as it is.

Example

```
BEGIN
   DBMS_JOB.CHANGE(14144, null, null, 'sysdate+3');
   COMMIT;
END;
```

INSTANCE Procedure

This procedure changes job instance affinity.

Syntax

```
DBMS_JOB.INSTANCE (
   job        IN BINARY_INTEGER,
   instance   IN BINARY_INTEGER,
   force      IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 48–4 INSTANCE Procedure Parameters

Parameter	Description
job	Number of the job being run.
instance	When a job is submitted, a user can specify which instance can run the job.
force	If this is TRUE, then any positive integer is acceptable as the job instance. If this is FALSE (the default), then the specified instance must be running; otherwise the routine raises an exception.

Usage Notes

You must issue a COMMIT statement immediately after the statement.

INTERVAL Procedure

This procedure changes how often a job runs.

Syntax

```
DBMS_JOB.INTERVAL (
   job       IN  BINARY_INTEGER,
   interval  IN  VARCHAR2);
```

Parameters

Table 48–5 INTERVAL Procedure Parameters

Parameter	Description
`job`	Number of the job being run.
`interval`	Date function, evaluated immediately before the job starts running.

Usage Notes

- If the job completes successfully, then this new date is placed in `next_date`. `interval` is evaluated by plugging it into the statement select `interval` into `next_date` from dual;
- The `interval` parameter must evaluate to a time in the future. Legal intervals include:

Interval	Description
`'sysdate + 7'`	Run once a week.
`'next_day(sysdate,''TUESDAY'')'`	Run once every Tuesday.
`'null'`	Run only once.

- If `interval` evaluates to `NULL` and if a job completes successfully, then the job is automatically deleted from the queue.
- You must issue a `COMMIT` statement immediately after the statement.

NEXT_DATE Procedure

This procedure changes when an existing job next runs.

Syntax

```
DBMS_JOB.NEXT_DATE (
   job       IN  BINARY_INTEGER,
   next_date IN  DATE);
```

Parameters

Table 48–6 NEXT_DATE Procedure Parameters

Parameter	Description
job	Number of the job being run.
next_date	Date of the next refresh: it is when the job will be automatically run, assuming there are background processes attempting to run it.

Usage Notes

You must issue a COMMIT statement immediately after the statement.

REMOVE Procedure

This procedure removes an existing job from the job queue. This currently does not stop a running job.

Syntax

```
DBMS_JOB.REMOVE (
   job       IN  BINARY_INTEGER );
```

Parameters

Table 48–7 REMOVE Procedure Parameters

Parameter	Description
job	Number of the job being run.

Usage Notes

You must issue a COMMIT statement immediately after the statement.

Example

```
BEGIN
   DBMS_JOB.REMOVE(14144);
   COMMIT;
END;
```

RUN Procedure

This procedure runs job `JOB` now. It runs it even if it is broken.

Running the job recomputes `next_date`. See view `user_jobs`.

Syntax

```
DBMS_JOB.RUN (
   job       IN  BINARY_INTEGER,
   force     IN  BOOLEAN DEFAULT FALSE);
```

Parameters

Table 48–8 RUN Procedure Parameters

Parameter	Description
`job`	Number of the job being run.
`force`	If this is `TRUE`, then instance affinity is irrelevant for running jobs in the foreground process. If this is `FALSE`, then the job can be run in the foreground only in the specified instance.

Example

```
EXECUTE DBMS_JOB.RUN(14144);
```

Caution: This re-initializes the current session's packages.

Exceptions

An exception is raised if `force` is `FALSE`, and if the connected instance is the wrong one.

SUBMIT Procedure

This procedure submits a new job. It chooses the job from the sequence `sys.jobseq`.

Syntax

```
DBMS_JOB.SUBMIT (
   job       OUT BINARY_INTEGER,
   what      IN  VARCHAR2,
   next_date IN  DATE DEFAULT sysdate,
   interval  IN  VARCHAR2 DEFAULT 'null',
   no_parse  IN  BOOLEAN DEFAULT FALSE,
   instance  IN  BINARY_INTEGER DEFAULT any_instance,
   force     IN  BOOLEAN DEFAULT FALSE);
```

Parameters

Table 48–9 SUBMIT Procedure Parameters

Parameter	Description
`job`	Number of the job being run.
`what`	PL/SQL procedure to run.
`next_date`	Next date when the job will be run.
`interval`	Date function that calculates the next time to run the job. The default is `NULL`. This must evaluate to a either a future point in time or `NULL`.
`no_parse`	A flag. The default is `FALSE`. If this is set to `FALSE`, then Oracle parses the procedure associated with the job. If this is set to `TRUE`, then Oracle parses the procedure associated with the job the first time that the job is run. For example, if you want to submit a job before you have created the tables associated with the job, then set this to `TRUE`.
`instance`	When a job is submitted, specifies which instance can run the job.
`force`	If this is `TRUE`, then any positive integer is acceptable as the job instance. If this is `FALSE` (the default), then the specified instance must be running; otherwise the routine raises an exception.

Usage Notes

- You must issue a `COMMIT` statement immediately after the statement.
- The parameters `instance` and `force` are added for job queue affinity. Job queue affinity gives users the ability to indicate whether a particular instance or any instance can run a submitted job.

Example

This submits a new job to the job queue. The job calls the procedure `DBMS_DDL.ANALYZE_OBJECT` to generate optimizer statistics for the table `DQUON.ACCOUNTS`. The statistics are based on a sample of half the rows of the `ACCOUNTS` table. The job is run every 24 hours:

```
VARIABLE jobno number;
BEGIN
```

```
   DBMS_JOB.SUBMIT(:jobno,
      'dbms_ddl.analyze_object(''TABLE'',
      ''DQUON'', ''ACCOUNTS'',
      ''ESTIMATE'', NULL, 50);'
      SYSDATE, 'SYSDATE + 1');
   COMMIT;
END;
/
Statement processed.
print jobno
JOBNO
----------
14144
```

USER_EXPORT Procedures

There are two overloaded procedures. The first produces the text of a call to re-create the given job. The second alters instance affinity (8*i* and after) and preserves the compatibility.

Syntax

```
DBMS_JOB.USER_EXPORT (
   job    IN     BINARY_INTEGER,
   mycall IN OUT VARCHAR2);

DBMS_JOB.USER_EXPORT (
   job      IN     BINARY_INTEGER,
   mycall   IN OUT VARCHAR2,
   myinst   IN OUT VARCHAR2);
```

Parameters

Table 48–10 USER_EXPORT Procedure Parameter

Parameter	Description
`job`	Number of the job being run.
`mycall`	Text of a call to re-create the given job.
`myinst`	Text of a call to alter instance affinity.

WHAT Procedure

This procedure changes what an existing job does, and replaces its environment.

Syntax

```
DBMS_JOB.WHAT (
   job       IN  BINARY_INTEGER,
   what      IN  VARCHAR2);
```

Parameters

Table 48–11 WHAT Procedure Parameters

Parameter	Description
job	Number of the job being run.
what	PL/SQL procedure to run.

Usage Notes

- You must issue a COMMIT statement immediately after the statement.
- Some legal values of what (assuming the routines exist) are:
  - 'myproc(''10-JAN-82'', next_date, broken);'
  - 'scott.emppackage.give_raise(''JENKINS'', 30000.00);'
  - 'dbms_job.remove(job);'

49

DBMS_LDAP

The DBMS_LDAP package lets you access data from LDAP servers.

- Documentation of DBMS_LDAP

Documentation of DBMS_LDAP

For a complete description of this package within the context of Oracle Internet Directory, see `DBMS_LDAP` in the *Oracle Internet Directory Application Developer's Guide.*

50

DBMS_LDAP_UTL

The DBMS_LDAP_UTL package contains the Oracle Extension utility functions.

- Documentation of DBMS_LDAP_UTL

Documentation of DBMS_LDAP_UTL

For a complete description of this package within the context of Oracle Internet Directory, see `DBMS_LDAP_UTL` in the *Oracle Internet Directory Application Developer's Guide.*

51

DBMS_LIBCACHE

The DBMS_LIBCACHE package consists of one subprogram that prepares the library cache on an Oracle instance by extracting SQL and PL/SQL from a remote instance and compiling this SQL locally without execution. The value of compiling the cache of an instance is to prepare the information the application requires to execute in advance of failover or switchover.

This chapter contains the following topics:

Using DBMS_LIBCACHE

Overview

Compiling a shared cursor consists of open, parse, and bind operations, plus the type-checking and execution plan functions performed at the first execution. All of these steps are executed in advance by the package `DBMS_LIBCACHE` for `SELECT` statements. The open and parse functions are executed in advance for PL/SQL and DML. For PL/SQL, executing the parse phase has the effect of loading all library cache heaps other than the `MCODE`.

Security Model

To execute DBMS_LIBCACHE you must directly access the same objects as do SQL statements. You can best accomplish this by utilizing the same user id as the original system on the remote system.

When there are multiple schema users, DBMS_LIBCACHE should be called for each.

Alternatively, DBMS_LIBCACHE may be called with the generic user PARSER. However, this user cannot parse the SQL that uses objects with access granted though roles. This is a standard PL/SQL security limitation.

Summary of DBMS_LIBCACHE Subprograms

Table 51–1 DBMS_LIBCACHE Package Subprograms

Subprogram	Description
COMPILE_FROM_REMOTE Procedure on page 51-6	Extracts SQL in batch from the source instance and compiles the SQL at the target instance

COMPILE_FROM_REMOTE Procedure

This procedure extracts SQL in batch from the source instance and compiles the SQL at the target instance.

Syntax

```
DBMS_LIBCACHE.COMPILE_FROM_REMOTE (
   p_db_link                IN     dbms_libcache$def.db_link%type,
   p_username               IN     VARCHAR2 default null,
   p_threshold_executions   IN     NATURAL  default 3,
   p_threshold_sharable_mem IN     NATURAL  default 1000,
   p_parallel_degree        IN     NATURAL  default 1);
```

Parameters

Table 51–2 COMPILE_FROM_REMOTE Procedure Parameters

Parameter	Description
p_db_link	Database link to the source name (mandatory). The database link pointing to the instance that will be used for extracting the SQL statements. The user must have the role SELECT_ON_CATALOG at the source instance. For improved security, the connection may use a password file or LDAP authentication. The database link is mandatory only for releases with dbms_libcache$def.ACCESS_METHOD = DB_LINK_METHOD
p_instance_name	(Reserved for future use). The name of the instance that will be used for extracting the SQL statements. The instance name must be unique for all instances excluding the local instance. The name is not case sensitive.
p_username	Source username (default is all users). The name of the username that will be used for extracting the SQL statements. The username is an optional parameter that is used to ensure the parsing user id is the same as that on the source instance. For an application where users connect as a single user_id, for example APPS, APPS is the parsing user_id that is recorded in the shared pool. To select only SQL statements parsed by APPS, enter the string 'APPS' in this field. To also select statements executed by batch, repeat the executing the procedure with the schema owner, for example GL. If the username is supplied, it must be valid. The name is not case sensitive.
p_threshold_executions	The lower bound for the number of executions, below which a SQL statement will not be selected for parsing. This parameter is optional. It allows the application to extract and compile statements with executions, for example, greater than 3. The default value is 1. This means SQL statements that have never executed, including invalid SQL statements, will not be extracted.

Table 51–2 (Cont.) COMPILE_FROM_REMOTE Procedure Parameters

Parameter	Description
p_threshold_sharable_mem	The lower bound for the size of the shared memory consumed by the cursors on the source instance. Below this value a SQL statement will not be selected for parsing. This parameter is optional. It allows the application to extract and compile statements with shared memory for example, greater than 10000 bytes.
p_parallel_degree	The number of parallel jobs that execute to complete the parse operation. These tasks are spawned as parallel jobs against a sub-range of the SQL statements selected for parsing. This parameter is reserved for parallel compile jobs which are currently not implemented.

52

DBMS_LOB

The DBMS_LOB package provides subprograms to operate on BLOBs, CLOBs, NCLOBs, BFILEs, and temporary LOBs. You can use DBMS_LOB to access and manipulation specific parts of a LOB or complete LOBs.

> **See Also:** *Oracle Database Application Developer's Guide - Large Objects*

This chapter contains the following topics:

Using DBMS_LOB

Overview

DBMS_LOB can read and modify BLOBs, CLOBs, and NCLOBs; it provides read-only operations for BFILEs. The bulk of the LOB operations are provided by this package.

Security Model

This package must be created under SYS. Operations provided by this package are performed under the current calling user, not under the package owner SYS.

Any DBMS_LOB subprogram called from an anonymous PL/SQL block is executed using the privileges of the current user. Any DBMS_LOB subprogram called from a stored procedure is executed using the privileges of the owner of the stored procedure.

When creating the procedure, users can set the AUTHID to indicate whether they want definer's rights or invoker's rights. For example:

```
CREATE PROCEDURE proc1 authid definer ...
```

or

```
CREATE PROCEDURE proc1 authid current_user ...
```

> **See Also:** For more information on AUTHID and privileges, see *Oracle Database PL/SQL User's Guide and Reference*

You can provide secure access to BFILEs using the DIRECTORY feature discussed in BFILENAME function in the *Oracle Database Application Developer's Guide - Large Objects* and the *Oracle Database SQL Reference*.

For information about the security model pertaining to temporary LOBs, see Operational Notes.

Constants

DBMS_LOB defines the following constants:

```
file_readonly CONSTANT BINARY_INTEGER := 0;
lob_readonly  CONSTANT BINARY_INTEGER := 0;
lob_readwrite CONSTANT BINARY_INTEGER := 1;
lobmaxsize    CONSTANT INTEGER        := 18446744073709551615;
call          CONSTANT PLS_INTEGER    := 12;
session       CONSTANT PLS_INTEGER    := 10;
```

Datatypes

The DBMS_LOB package uses the datatypes shown in Table 52–1.

Table 52–1 Datatypes Used by DBMS_LOB

Type	Description
BLOB	Source or destination binary LOB.
RAW	Source or destination RAW buffer (used with BLOB).
CLOB	Source or destination character LOB (including NCLOB).
VARCHAR2	Source or destination character buffer (used with CLOB and NCLOB).
INTEGER	Specifies the size of a buffer or LOB, the offset into a LOB, or the amount to access.
BFILE	Large, binary object stored outside the database.

The DBMS_LOB package defines no special types.

An NCLOB is a CLOB for holding fixed-width and varying-width, multibyte national character sets.

The clause ANY_CS in the specification of DBMS_LOB subprograms for CLOBs enables the CLOB type to accept a CLOB or NCLOB locator variable as input.

Rules and Limits

General Rules and Limits

- The following rules apply in the specification of subprograms in this package:
  - `length`, `offset`, and `amount` parameters for subprograms operating on `BLOB`s and `BFILE`s must be specified in terms of *bytes*.
  - `length`, `offset`, and `amount` parameters for subprograms operating on `CLOB`s must be specified in terms of *characters*.
- A subprogram raises an `INVALID_ARGVAL` exception if the following restrictions are not followed in specifying values for parameters (unless otherwise specified):
  1. Only positive, absolute offsets from the beginning of `LOB` data are permitted: Negative offsets from the tail of the `LOB` are not permitted.
  2. Only positive, nonzero values are permitted for the parameters that represent size and positional quantities, such as `amount`, `offset`, `newlen`, `nth`, and so on. Negative offsets and ranges observed in SQL string functions and operators are not permitted.
  3. The value of `offset`, `amount`, `newlen`, `nth` must not exceed the value `lobmaxsize` (4GB-1) in any `DBMS_LOB` subprogram.
  4. For `CLOB`s consisting of fixed-width multibyte characters, the maximum value for these parameters must not exceed (`lobmaxsize/character_width_in_bytes`) characters .

     For example, if the `CLOB` consists of 2-byte characters, such as:

     ```
     JA16SJISFIXED
     ```

     Then, the maximum `amount` value should not exceed:

     ```
     4294967295/2 = 2147483647 characters.
     ```
- PL/SQL language specifications stipulate an upper limit of 32767 bytes (not characters) for `RAW` and `VARCHAR2` parameters used in `DBMS_LOB` subprograms. For example, if you declare a variable to be:

  ```
  charbuf VARCHAR2(3000)
  ```

  Then, `charbuf` can hold 3000 single byte characters or 1500 2-byte fixed width characters. This has an important consequence for `DBMS_LOB` subprograms for `CLOB`s and `NCLOB`s.
- The `%CHARSET` clause indicates that the form of the parameter with `%CHARSET` must match the form of the `ANY_CS` parameter to which it refers.

  For example, in `DBMS_LOB` subprograms that take a `VARCHAR2` buffer parameter, the form of the `VARCHAR2` buffer must match the form of the `CLOB` parameter. If the input `LOB` parameter is of type `NCLOB`, then the buffer must contain `NCHAR`

data. Conversely, if the input LOB parameter is of type CLOB, then the buffer must contain CHAR data.

For DBMS_LOB subprograms that take two CLOB parameters, both CLOB parameters must have the same form; that is, they must both be NCLOBs, or they must both be CLOBs.

- If the value of amount plus the offset exceeds the maximum LOB size allowed by the database, then access exceptions are raised.

  Under these input conditions, read subprograms, such as READ, COMPARE, INSTR, and SUBSTR, read until End of Lob/File is reached. For example, for a READ operation on a BLOB or BFILE, if the user specifies offset value of 3 GB and an amount value of 2 GB, then READ reads only ((4GB-1)-3GB) bytes.

- Functions with NULL or invalid input values for parameters return a NULL. Procedures with NULL values for destination LOB parameters raise exceptions.
- Operations involving patterns as parameters, such as COMPARE, INSTR, and SUBSTR do not support regular expressions or special matching characters (such as % in the LIKE operator in SQL) in the pattern parameter or substrings.
- The End Of LOB condition is indicated by the READ procedure using a NO_DATA_FOUND exception. This exception is raised only upon an attempt by the user to read beyond the end of the LOB. The READ buffer for the last read contains 0 bytes.
- For consistent LOB updates, you must lock the row containing the destination LOB before making a call to any of the procedures (mutators) that modify LOB data.
- Unless otherwise stated, the default value for an offset parameter is 1, which indicates the first byte in the BLOB or BFILE data, and the first character in the CLOB or NCLOB value. No default values are specified for the amount parameter — you must input the values explicitly.
- You must lock the row containing the destination internal LOB before calling any subprograms that modify the LOB, such as APPEND, COPY, ERASE, TRIM, or WRITE. These subprograms do not implicitly lock the row containing the LOB.

Rules and Limits Specific to External Files (BFILEs)

- The subprograms COMPARE, INSTR, READ, SUBSTR, FILECLOSE, FILECLOSEALL and LOADFROMFILE operate only on an *opened* BFILE locator; that is, a successful FILEOPEN call must precede a call to any of these subprograms.
- For the functions FILEEXISTS, FILEGETNAME and GETLENGTH, a file's open/close status is unimportant; however, the file must exist physically, and you must have adequate privileges on the DIRECTORY object and the file.
- DBMS_LOB does not support any concurrency control mechanism for BFILE operations.
- In the event of several open files in the session whose closure has not been handled properly, you can use the FILECLOSEALL subprogram to close all files opened in the session and resume file operations from the beginning.
- If you are the creator of a DIRECTORY, or if you have system privileges, then use the CREATE OR REPLACE, DROP, and REVOKE statements in SQL with extreme caution.

  If you, or other grantees of a particular directory object, have several open files in a session, then any of the preceding commands can adversely affect file operations. In the event of such abnormal termination, your only choice is to

invoke a program or anonymous block that calls FILECLOSEALL, reopen your files, and restart your file operations.

- All files opened during a user session are implicitly closed at the end of the session. However, Oracle strongly recommends that you close the files after *both* normal and abnormal termination of operations on the BFILE.

  In the event of normal program termination, proper file closure ensures that the number of files that are open simultaneously in the session remains less than SESSION_MAX_OPEN_FILES.

  In the event of abnormal program termination from a PL/SQL program, it is imperative that you provide an exception handler that ensures closure of all files opened in that PL/SQL program. This is necessary because after an exception occurs, only the exception handler has access to the BFILE variable in its most current state.

  After the exception transfers program control outside the PL/SQL program block, all references to the open BFILEs are lost. The result is a larger open file count which may or may not exceed the SESSION_MAX_OPEN_FILES value.

  For example, consider a READ operation past the end of the BFILE value, which generates a NO_DATA_FOUND exception:

```
DECLARE
       fil BFILE;
       pos INTEGER;
       amt BINARY_INTEGER;
       buf RAW(40);
BEGIN
       SELECT ad_graphic INTO fil FROM print_media WHERE product_id = 3106;
       dbms_lob.open(fil, dbms_lob.lob_readonly);
       amt := 40; pos := 1 + dbms_lob.getlength(fil); buf := '';
       dbms_lob.read(fil, amt, pos, buf);
       dbms_output.put_line('Read F1 past EOF: '||
           utl_raw.cast_to_varchar2(buf));
       dbms_lob.close(fil);
END;

ORA-01403: no data found
ORA-06512: at "SYS.DBMS_LOB", line 373
ORA-06512: at line 10
```

  After the exception has occurred, the BFILE locator variable file goes out of scope, and no further operations on the file can be done using that variable. Therefore, the solution is to use an exception handler:

```
DECLARE
     fil BFILE;
     pos INTEGER;
     amt BINARY_INTEGER;
     buf RAW(40);
BEGIN
     SELECT ad_graphic INTO fil FROM print_media WHERE product_id = 3106;
     dbms_lob.open(fil, dbms_lob.lob_readonly);
     amt := 40; pos := 1 + dbms_lob.getlength(fil); buf := '';
     dbms_lob.read(fil, amt, pos, buf);
     dbms_output.put_line('Read F1 past EOF: '||
          utl_raw.cast_to_varchar2(buf));
     dbms_lob.close(fil);
     exception
     WHEN no_data_found
```

```
      THEN
        BEGIN
          dbms_output.put_line('End of File reached. Closing file');
          dbms_lob.fileclose(fil);
          -- or dbms_lob.filecloseall if appropriate
        END;
END;
      /

Statement processed.
End of File reached. Closing file
```

In general, you should ensure that files opened in a PL/SQL block using DBMS_LOB are closed before normal or abnormal termination of the block.

Maximum LOB Size

The maximum size of a LOB supported by the database is equal to the value of the db_block_size initialization parameter times the value 4294967295. This allows for a maximum LOB size ranging from 8 terabytes to 128 terabytes.

Maximum Buffer Size

The maximum buffer size, 32767 bytes, is represented by maxbufsize.

Operational Notes

All DBMS_LOB subprograms work based on LOB locators. For the successful completion of DBMS_LOB subprograms, you must provide an input locator that represents a LOB that already exists in the database tablespaces or external file system. See also Chapter 1 of *Oracle Database Application Developer's Guide - Large Objects.*

To use LOBs in your database, you must first use SQL data definition language (DDL) to define the tables that contain LOB columns.

- Internal LOBs
- External LOBs
- Temporary LOBs

Internal LOBs

To populate your table with internal LOBs after LOB columns are defined in a table, you use the SQL data manipulation language (DML) to initialize or populate the locators in the LOB columns.

External LOBs

For an external LOB (BFILE) to be represented by a LOB locator, you must:

- Ensure that a DIRECTORY object representing a valid, existing physical directory has been defined, and that physical files (the LOBs you plan to add) exist with read permission for the database. If your operating system uses case-sensitive path names, then be sure you specify the directory in the correct format.
- Pass the DIRECTORY object and the filename of the external LOB you are adding to the BFILENAME function to create a LOB locator for your external LOB.

Once you have completed these tasks, you can insert or update a row containing a LOB column using the given LOB locator.

After the LOBs are defined and created, you can then SELECT from a LOB locator into a local PL/SQL LOB variable and use this variable as an input parameter to DBMS_LOB for access to the LOB value.

For details on the different ways to do this, you must refer to the section of the *Oracle Database Application Developer's Guide - Large Objects* that describes "Accessing External LOBs (BFILEs)."

Temporary LOBs

The database supports the definition, creation, deletion, access, and update of temporary LOBs. Your temporary tablespace stores the temporary LOB data. Temporary LOBs are not permanently stored in the database. Their purpose is mainly to perform transformations on LOB data.

For temporary LOBs, you must use the OCI, PL/SQL, or another programmatic interface to create or manipulate them. Temporary LOBs can be either BLOBs, CLOBs, or NCLOBs.

A temporary LOB is empty when it is created. By default, all temporary LOBs are deleted at the end of the session in which they were created. If a process dies unexpectedly or if the database crashes, then temporary LOBs are deleted, and the space for temporary LOBs is freed.

There is also an interface to let you group temporary LOBs together into a logical bucket. The duration represents this logical store for temporary LOBs. Each temporary LOB can have separate storage characteristics, such as CACHE/ NOCACHE. There is a default store for every session into which temporary LOBs are placed if you don't specify a specific duration. Additionally, you are able to perform a free operation on durations, which causes all contents in a duration to be freed.

There is no support for consistent read (CR), undo, backup, parallel processing, or transaction management for temporary LOBs. Because CR and rollbacks are not supported for temporary LOBs, you must free the temporary LOB and start over again if you encounter an error.

Because CR, undo, and versions are not generated for temporary LOBs, there is potentially a performance impact if you assign multiple locators to the same temporary LOB. Semantically, each locator should have its own copy of the temporary LOB.

A copy of a temporary LOB is created if the user modifies the temporary LOB while another locator is also pointing to it. The locator on which a modification was performed now points to a new copy of the temporary LOB. Other locators no longer see the same data as the locator through which the modification was made. A deep copy was not incurred by permanent LOBs in these types of situations, because CR snapshots and version pages enable users to see their own versions of the LOB cheaply.

You can gain pseudo-REF semantics by using pointers to locators in OCI and by having multiple pointers to locators point to the same temporary LOB locator, if necessary. In PL/SQL, you must avoid using more than one locator for each temporary LOB. The temporary LOB locator can be passed by reference to other procedures.

Because temporary LOBs are not associated with any table schema, there are no meanings to the terms in-row and out-of-row temporary LOBs. Creation of a temporary LOB instance by a user causes the engine to create and return a locator to the LOB data. The PL/SQL DBMS_LOB package, PRO*C, OCI, and other programmatic interfaces operate on temporary LOBs through these locators just as they do for permanent LOBs.

There is no support for client side temporary LOBs. All temporary LOBs reside in the server.

Temporary LOBs do not support the EMPTY_BLOB or EMPTY_CLOB functions that are supported for permanent LOBs. The EMPTY_BLOB function specifies the fact that the LOB is initialized, but not populated with any data.

A temporary LOB instance can only be destroyed by using OCI or the DBMS_LOB package by using the appropriate FREETEMPORARY or OCIDurationEnd statement.

A temporary LOB instance can be accessed and modified using appropriate OCI and DBMS_LOB statements, just as for regular permanent internal LOBs. To make a temporary LOB permanent, you must explicitly use the OCI or DBMS_LOB COPY command, and copy the temporary LOB into a permanent one.

Security is provided through the LOB locator. Only the user who created the temporary LOB is able to see it. Locators are not expected to be able to pass from one user's session to another. Even if someone did pass a locator from one session to another, they would not access the temporary LOBs from the original session. Temporary LOB lookup is localized to each user's own session. Someone using a locator from somewhere else is only able to access LOBs within his own session that have the same LOB ID. Users should not try to do this, but if they do, they are not able to affect anyone else's data.

The database keeps track of temporary LOBs for each session in a v$ view called V$TEMPORARY_LOBS, which contains information about how many temporary LOBs exist for each session. V$ views are for DBA use. From the session, the database can determine which user owns the temporary LOBs. By using V$TEMPORARY_LOBS in conjunction with DBA_SEGMENTS, a DBA can see how much space is being used by a session for temporary LOBs. These tables can be used by DBAs to monitor and guide any emergency cleanup of temporary space used by temporary LOBs.

The following notes are specific to temporary LOBs:

1. All functions in DBMS_LOB return NULL if any of the input parameters are NULL. All procedures in DBMS_LOB raise an exception if the LOB locator is input as NULL.
2. Operations based on CLOBs do not verify if the character set IDs of the parameters (CLOB parameters, VARCHAR2 buffers and patterns, and so on) match. It is the user's responsibility to ensure this.
3. Data storage resources are controlled by the DBA by creating different temporary tablespaces. DBAs can define separate temporary tablespaces for different users, if necessary.
4. Temporary LOBs still adhere to value semantics in order to be consistent with permanent LOBs and to try to conform to the ANSI standard for LOBs. As a result, each time a user does an OCILobLocatatorAssign, or the equivalent assignment in PL/SQL, the database makes a copy of the temporary LOB.

   Each locator points to its own LOB value. If one locator is used to create a temporary LOB, and then is assigned to another LOB locator using OCILobLOcatorAssign in OCI or through an assignment operation in PL/SQL, then the database copies the original temporary LOB and causes the second locator to point to the copy.

   In order for users to modify the same LOB, they must go through the same locator. In OCI, this can be accomplished fairly easily by using pointers to locators and assigning the pointers to point to the same locator. In PL/SQL, the same LOB variable must be used to update the LOB to get this effect.

   The following example shows a place where a user incurs a copy, or at least an extra round-trip to the server.

```
DECLARE
  a blob;
  b blob;
BEGIN
  dbms_lob.createtemporary(b, TRUE);
  -- the following assignment results in a deep copy
  a := b;
END;
```

   The PL/SQL compiler makes temporary copies of actual arguments bound to OUT or IN OUT parameters. If the actual parameter is a temporary LOB, then the temporary copy is a deep (value) copy.

   The following PL/SQL block illustrates the case where the user incurs a deep copy by passing a temporary LOB as an IN OUT parameter.

```
DECLARE
  a blob;
  procedure foo(parm IN OUT blob) is
  BEGIN
   ...
  END;
```

```
BEGIN
  dbms_lob.createtemporary(a, TRUE);
  -- the following call results in a deep copy of the blob a
  foo(a);
END;
```

To minimize deep copies on PL/SQL parameter passing, use the `NOCOPY` compiler hint where possible.

The duration parameter passed to `dbms_lob.createtemporary()` is a hint. The duration of the new temp `LOB` is the same as the duration of the locator variable in PL/SQL. For example, in the preceding program block, the program variable `a` has the duration of the residing frame. Therefore at the end of the block, memory of `a` will be freed at the end of the function.

If a PL/SQL package variable is used to create a temp `LOB`, it will have the duration of the package variable, which has a duration of `SESSION`.

```
BEGIN
   y clob;
  END;
/
BEGIN
   dbms_lob.createtemporary(package.y, TRUE);
END;
```

See Also: *Oracle Database PL/SQL User's Guide and Reference* for more information on `NOCOPY` syntax

Exceptions

Table 52–2 DBMS_LOB Exceptions

Exception	Code	Description
INVALID_ARGVAL	21560	The argument is expecting a nonNULL, valid value but the argument value passed in is NULL, invalid, or out of range.
ACCESS_ERROR	22925	You are trying to write too much data to the LOB: LOB size is limited to 4 gigabytes.
NOEXIST_ DIRECTORY	22285	The directory leading to the file does not exist.
NOPRIV_ DIRECTORY	22286	The user does not have the necessary access privileges on the directory or the file for the operation.
INVALID_ DIRECTORY	22287	The directory used for the current operation is not valid if being accessed for the first time, or if it has been modified by the DBA since the last access.
OPERATION_ FAILED	22288	The operation attempted on the file failed.
UNOPENED_FILE	22289	The file is not open for the required operation to be performed.
OPEN_TOOMANY	22290	The number of open files has reached the maximum limit.
NO_DATA_FOUND		EndofLob indicator for looping read operations. This is not a hard error.
VALUE_ERROR	6502	PL/SQL error for invalid values to subprogram's parameters.

Summary of DBMS_LOB Subprograms

Table 52–3 DBMS_LOB Package Subprograms

Subprogram	Description
APPEND Procedures on page 52-18	Appends the contents of the source LOB to the destination LOB
CLOSE Procedure on page 52-19	Closes a previously opened internal or external LOB
COMPARE Functions on page 52-20	Compares two entire LOBs or parts of two LOBs
CONVERTTOBLOB Procedure on page 52-22	Reads character data from a source CLOB or NCLOB instance, converts the character data to the specified character, writes the converted data to a destination BLOB instance in binary format, and returns the new offsets
CONVERTTOCLOB Procedure on page 52-25	Takes a source BLOB instance, converts the binary data in the source instance to character data using the specified character, writes the character data to a destination CLOB or NCLOB instance, and returns the new offsets
COPY Procedures on page 52-28	Copies all, or part, of the source LOB to the destination LOB
CREATETEMPORARY Procedures on page 52-30	Creates a temporary BLOB or CLOB and its corresponding index in the user's default temporary tablespace
ERASE Procedures on page 52-31	Erases all or part of a LOB
FILECLOSE Procedure on page 52-33	Closes the file
FILECLOSEALL Procedure on page 52-34	Closes all previously opened files
FILEEXISTS Function on page 52-35	Checks if the file exists on the server
FILEGETNAME Procedure on page 52-36	Gets the directory object name and file name
FILEISOPEN Function on page 52-37	Checks if the file was opened using the input BFILE locators
FILEOPEN Procedure on page 52-38	Opens a file
FREETEMPORARY Procedures on page 52-39	Frees the temporary BLOB or CLOB in the user's default temporary tablespace
GETCHUNKSIZE Functions on page 52-41	Returns the amount of space used in the LOB chunk to store the LOB value
GETLENGTH Functions on page 52-42	Gets the length of the LOB value
GET_STORAGE_LIMIT on page 52-40	Returns the storage limit for LOBs in your database configuration
INSTR Functions on page 52-43	Returns the matching position of the *nth* occurrence of the pattern in the LOB
ISOPEN Functions on page 52-45	Checks to see if the LOB was already opened using the input locator

Table 52–3 (Cont.) DBMS_LOB Package Subprograms

Subprogram	Description
ISTEMPORARY Functions on page 52-46	Checks if the locator is pointing to a temporary `LOB`
LOADBLOBFROMFILE Procedure on page 52-47	Loads `BFILE` data into an internal `BLOB`
LOADCLOBFROMFILE Procedure on page 52-49	Loads `BFILE` data into an internal `CLOB`
LOADFROMFILE Procedure on page 52-52	Loads `BFILE` data into an internal `LOB`
OPEN Procedures on page 52-54	Opens a `LOB` (internal, external, or temporary) in the indicated mode
READ Procedures on page 52-56	Reads data from the `LOB` starting at the specified offset
SUBSTR Functions on page 52-58	Returns part of the `LOB` value starting at the specified offset
TRIM Procedures on page 52-60	Trims the `LOB` value to the specified shorter length
WRITE Procedures on page 52-62	Writes data to the `LOB` from a specified offset
WRITEAPPEND Procedures on page 52-64	Writes a buffer to the end of a `LOB`

APPEND Procedures

This procedure appends the contents of a source internal LOB to a destination LOB. It appends the complete source LOB.

There are two overloaded APPEND procedures.

Syntax

```
DBMS_LOB.APPEND (
   dest_lob IN OUT  NOCOPY BLOB,
   src_lob  IN             BLOB);

DBMS_LOB.APPEND (
   dest_lob IN OUT  NOCOPY CLOB  CHARACTER SET ANY_CS,
   src_lob  IN             CLOB  CHARACTER SET dest_lob%CHARSET);
```

Parameters

Table 52–4 APPEND Procedure Parameters

Parameter	Description
dest_lob	Locator for the internal LOB to which the data is to be appended.
src_lob	Locator for the internal LOB from which the data is to be read.

Exceptions

Table 52–5 APPEND Procedure Exceptions

Exception	Description
VALUE_ERROR	Either the source or the destination LOB is NULL.

Usage Notes

It is not mandatory that you wrap the LOB operation inside the Open/Close APIs. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit or rollback the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

If you do not wrap the LOB operation inside the Open/Close API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the OPEN or CLOSE statement.

> **See Also:** *Oracle Database Application Developer's Guide - Large Objects* for additional details on usage of this procedure

CLOSE Procedure

This procedure closes a previously opened internal or external LOB.

Syntax

```
DBMS_LOB.CLOSE (
   lob_loc    IN OUT NOCOPY BLOB);

DBMS_LOB.CLOSE (
   lob_loc    IN OUT NOCOPY CLOB CHARACTER SET ANY_CS);

DBMS_LOB.CLOSE (
   file_loc   IN OUT NOCOPY BFILE);
```

Parameters

Table 52–6 CLOSE Procedure Parameters

Parameter	Description
lob_loc	LOB locator. For more information, see Operational Notes.

Exceptions

No error is returned if the BFILE exists but is not opened. An error is returned if the LOB is not open.

Usage Notes

CLOSE requires a round-trip to the server for both internal and external LOBs. For internal LOBs, CLOSE triggers other code that relies on the close call, and for external LOBs (BFILEs), CLOSE actually closes the server-side operating system file.

It is not mandatory that you wrap all LOB operations inside the Open/Close APIs. However, if you open a LOB, you must close it before you commit or rollback the transaction; an error is produced if you do not. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

It is an error to commit the transaction before closing all opened LOBs that were opened by the transaction. When the error is returned, the openness of the open LOBs is discarded, but the transaction is successfully committed. Hence, all the changes made to the LOB and non-LOB data in the transaction are committed, but the domain and function-based indexes are not updated. If this happens, you should rebuild the functional and domain indexes on the LOB column.

> **See Also:** *Oracle Database Application Developer's Guide - Large Objects* for additional details on usage of this procedure

COMPARE Functions

This function compares two entire LOBs or parts of two LOBs.

Syntax

```
DBMS_LOB.COMPARE (
   lob_1            IN BLOB,
   lob_2            IN BLOB,
   amount           IN INTEGER := 4294967295,
   offset_1         IN INTEGER := 1,
   offset_2         IN INTEGER := 1)
  RETURN INTEGER;

DBMS_LOB.COMPARE (
   lob_1            IN CLOB  CHARACTER SET ANY_CS,
   lob_2            IN CLOB  CHARACTER SET lob_1%CHARSET,
   amount           IN INTEGER := 4294967295,
   offset_1         IN INTEGER := 1,
   offset_2         IN INTEGER := 1)
  RETURN INTEGER;

DBMS_LOB.COMPARE (
   lob_1            IN BFILE,
   lob_2            IN BFILE,
   amount           IN INTEGER,
   offset_1         IN INTEGER := 1,
   offset_2         IN INTEGER := 1)
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(COMPARE, WNDS, WNPS, RNDS, RNPS);
```

Parameters

Table 52–7 COMPARE Function Parameters

Parameter	Description
lob_1	LOB locator of first target for comparison.
lob_2	LOB locator of second target for comparison.
amount	Number of bytes (for BLOBs) or characters (for CLOBs) to compare.
offset_1	Offset in bytes or characters on the first LOB (origin: 1) for the comparison.
offset_2	Offset in bytes or characters on the first LOB (origin: 1) for the comparison.

Return Values

- INTEGER: Zero if the comparison succeeds, nonzero if not.
- NULL, if
  - amount < 1
  - amount > LOBMAXSIZE
  - offset_1 or offset_2 < 1

* offset_1 or offset_2 > LOBMAXSIZE

Usage Notes

You can only compare LOBs of the same datatype (LOBs of BLOB type with other BLOBs, and CLOBs with CLOBs, and BFILEs with BFILEs). For BFILEs, the file must be already opened using a successful FILEOPEN operation for this operation to succeed.

COMPARE returns zero if the data exactly matches over the range specified by the offset and amount parameters. Otherwise, a nonzero INTEGER is returned.

For fixed-width *n*-byte CLOBs, if the input amount for COMPARE is specified to be greater than (4294967295/*n*), then COMPARE matches characters in a range of size (4294967295/*n*), or Max(length(clob1), length(clob2)), whichever is lesser.

Exceptions

Table 52–8 COMPARE Function Exceptions for BFILE operations

Exception	Description
UNOPENED_FILE	File was not opened using the input locator.
NOEXIST_ DIRECTORY	Directory does not exist.
NOPRIV_DIRECTORY	You do not have privileges for the directory.
INVALID_ DIRECTORY	Directory has been invalidated after the file was opened.
INVALID_ OPERATION	File does not exist, or you do not have access privileges on the file.

See Also: *Oracle Database Application Developer's Guide - Large Objects* for additional details on usage of this procedure

CONVERTTOBLOB Procedure

This procedure reads character data from a source CLOB or NCLOB instance, converts the character data to the character set you specify, writes the converted data to a destination BLOB instance in binary format, and returns the new offsets. You can use this interface with any combination of persistent or temporary LOB instances as the source or destination.

Syntax

```
DBMS_LOB.CONVERTTOBLOB(
  dest_lob       IN OUT     NOCOPY  BLOB,
  src_clob       IN         CLOB CHARACTER SET ANY_CS,
  amount         IN         INTEGER,
  dest_offset    IN OUT     INTEGER,
  src_offset     IN OUT     INTEGER,
  blob_csid      IN         NUMBER,
  lang_context   IN OUT     INTEGER,
  warning        OUT        INTEGER);
```

Parameters

Table 52–9 CONVERTTOBLOB Procedure Parameters

Parameter	Description
dest_lob	LOB locator of the destination LOB instance.
src_blob	LOB locator of the source LOB instance.
amount	Number ofcharacters to convert from the source LOB. If you want to copy the entire LOB, pass the constant DBMS_LOB.LOBMAXSIZE. If you pass any other value, it must be less than or equal to the size of the LOB.
dest_offset	(IN) Offset in bytes in the destination LOB for the start of the write. Specify a value of 1 to start at the beginning of the LOB. (OUT) The new offset in bytes after the end of the write.
src_offset	(IN) Offset in characters in the source LOB for the start of the read. (OUT) Offset in characters in the source LOB right after the end of the read.
blob_csid	Desired character set ID of the converted data.
lang_context	(IN) Language context, such as shift status, for the current conversion. (OUT) The language context at the time when the current conversion is done. This information is returned so you can use it for subsequent conversions without losing or misinterpreting any source data. For the very first conversion, or if do not care, use the default value of zero.

Table 52–9 (Cont.) CONVERTTOBLOB Procedure Parameters

Parameter	Description
warning	(OUT) Warning message. This parameter indicates when something abnormal happened during the conversion. You are responsible for checking the warning message. Currently, the only possible warning is — inconvertible character. This occurs when the character in the source cannot be properly converted to a character in destination. The default replacement character (for example, '?') is used in place of the inconvertible character. The return value of this error message is defined as the constant warn_inconvertible_char in the DBMS_LOB package.

Usage Notes

Preconditions

Before calling the CONVERTTOBLOB procedure, the following preconditions must be met:

- Both the source and destination LOB instances must exist.
- If the destination LOB is a persistent LOB, the row must be locked. To lock the row, select the LOB using the FOR UPDATE clause of the SELECT statement.

Constants and Defaults

All parameters are required. You must pass a variable for each OUT or IN OUT parameter. You must pass either a variable or a value for each IN parameter.

Table 52–10 gives a summary of typical values for each parameter. The first column lists the parameter, the second column lists the typical value, and the last column describes the result of passing the value. Note that constants are used for some values. These constants are defined in the dbmslob.sql package specification file.

Table 52–10 DBMS_LOB.CONVERTTOBLOB Typical Values

Parameter	Value	Description
amount	LOBMAXSIZE (IN)	convert the entire file
dest_offset	1 (IN)	start from the beginning
src_offset	1 (IN)	start from the beginning
csid	DEFAULT_CSID (IN)	default CSID, use same CSID as source LOB
lang_context	DEFAULT_LANG_CTX (IN)	default language context
warning	NO_WARNING (OUT)	no warning message, success
	WARN_INCONVERTIBLE_CHAR (OUT)	character in source cannot be properly converted

General Notes

You must specify the desired character set for the destination LOB in the blob_csid parameter. You can pass a zero value for blob_csid. When you do so, the database assumes that the desired character set is the same as the source LOB character set, and performs a binary copy of the data—no character set conversion is performed.

You must specify the offsets for both the source and destination LOBs, and the number of bytes to copy from the source `LOB`. The `amount` and `src_offset` values are in characters and the `dest_offset` is in bytes. To convert the entire `LOB`, you can specify `LOBMAXSIZE` for the `amount` parameter.

Exceptions

Table 52–11 gives possible exceptions this procedure can throw. The first column lists the exception string and the second column describes the error conditions that can cause the exception.

Table 52–11 CONVERTTOBLOB Procedure Exceptions

Exception	Description
`VALUE_ERROR`	Any of the input parameters are `NULL` or `INVALID`.
`INVALID_ARGVAL`	One or more of the following: - `src_offset` or `dest_offset` < 1. - `src_offset` or `dest_offset` > `LOBMAXSIZE`. - `amount` < 1. - `amount` > `LOBMAXSIZE`.

See Also: *Oracle Database Application Developer's Guide - Large Objects* for more information on using LOBs in application development

CONVERTTOCLOB Procedure

This procedure takes a source BLOB instance, converts the binary data in the source instance to character data using the character set you specify, writes the character data to a destination CLOB or NCLOB instance, and returns the new offsets. You can use this interface with any combination of persistent or temporary LOB instances as the source or destination.

Syntax

```
DBMS_LOB.CONVERTTOCLOB(
   dest_lob       IN OUT NOCOPY  CLOB CHARACTER SET ANY_CS,
   src_blob       IN             BLOB,
   amount         IN             INTEGER,
   dest_offset    IN OUT         INTEGER,
   src_offset     IN OUT         INTEGER,
   blob_csid      IN             NUMBER,
   lang_context   IN OUT         INTEGER,
   warning        OUT            INTEGER);
```

Parameters

Table 52–12 CONVERTTOCLOB Procedure Parameters

Parameter	Description
dest_lob	LOB locator of the destination LOB instance.
src_blob	LOB locator of the source LOB instance.
amount	Number of bytes to convert from the source LOB. If you want to copy the entire BLOB, pass the constant DBMS_LOB.LOBMAXSIZE. If you pass any other value, it must be less than or equal to the size of the BLOB.
dest_offset	(IN) Offset in characters in the destination LOB for the start of the write. Specify a value of 1 to start at the beginning of the LOB. (OUT) The new offset in characters after the end of the write. This offset always points to the beginning of the first complete character after the end of the write.
src_offset	(IN) Offset in bytes in the source LOB for the start of the read. (OUT) Offset in bytes in the source LOB right after the end of the read.
blob_csid	Desired character set ID of the converted data.
lang_context	(IN) Language context, such as shift status, for the current conversion. (OUT) The language context at the time when the current conversion is done. This information is returned so you can use it for subsequent conversions without losing or misinterpreting any source data. For the very first conversion, or if do not care, use the default value of zero.

Table 52–12 (Cont.) CONVERTTOCLOB Procedure Parameters

Parameter	Description
warning	Warning message. This parameter indicates when something abnormal happened during the conversion. You are responsible for checking the warning message. Currently, the only possible warning is — inconvertible character. This occurs when the character in the source cannot be properly converted to a character in destination. The default replacement character (for example, '?') is used in place of the inconvertible character. The return value of this error message is defined as the constant warn_inconvertible_char in the DBMS_LOB package.

Usage Notes

Preconditions

Before calling the CONVERTTOCLOB procedure, the following preconditions must be met:

- Both the source and destination LOB instances must exist.
- If the destination LOB is a persistent LOB, the row must be locked before calling the CONVERTTOCLOB procedure. To lock the row, select the LOB using the FOR UPDATE clause of the SELECT statement.

Constants and Defaults

All parameters are required. You must pass a variable for each OUT or IN OUT parameter. You must pass either a variable or a value for each IN parameter.

Table 52–13 gives a summary of typical values for each parameter. The first column lists the parameter, the second column lists the typical value, and the last column describes the result of passing the value. Note that constants are used for some values. These constants are defined in the dbmslob.sql package specification file.

Table 52–13 DBMS_LOB.CONVERTTOCLOB Typical Values

Parameter	Value	Description
amount	LOBMAXSIZE (IN)	convert the entire file
dest_offset	1 (IN)	start from the beginning
src_offset	1 (IN)	start from the beginning
csid	DEFAULT_CSID (IN)	default CSID, use destination CSID
lang_context	DEFAULT_LANG_CTX (IN)	default language context
warning	NO_WARNING (OUT)	no warning message, success
	WARN_INCONVERTIBLE_CHAR (OUT)	character in source cannot be properly converted

General Notes

You must specify the desired character set for the destination LOB in the blob_csid parameter. You can pass a zero value for blob_csid. When you do so, the database assumes that the BLOB contains character data in the same character set as the destination CLOB, and performs a binary copy of the data to the destination LOB, no character set conversion being performed.

You must specify the offsets for both the source and destination LOBs, and the number of bytes to copy from the source BLOB. The amount and src_offset values are in bytes and the dest_offset is in characters. To convert the entire BLOB, you can specify LOBMAXSIZE for the amount parameter.

Exceptions

Table 52–14 CONVERTTOCLOB Procedure Exceptions

Exception	Description
VALUE_ERROR	Any of the input parameters are NULL or INVALID.
INVALID_ARGVAL	One or more of the following: - src_offset or dest_offset < 1. - src_offset or dest_offset > LOBMAXSIZE. - amount < 1. - amount > LOBMAXSIZE.

See Also: *Oracle Database Application Developer's Guide - Large Objects* for more information on using LOBs in application development

COPY Procedures

This procedure copies all, or a part of, a source internal LOB to a destination internal LOB. You can specify the offsets for both the source and destination LOBs, and the number of bytes or characters to copy.

Syntax

```
DBMS_LOB.COPY (
  dest_lob    IN OUT NOCOPY BLOB,
  src_lob     IN            BLOB,
  amount      IN            INTEGER,
  dest_offset IN            INTEGER := 1,
  src_offset  IN            INTEGER := 1);

DBMS_LOB.COPY (
  dest_lob    IN OUT NOCOPY CLOB  CHARACTER SET ANY_CS,
  src_lob     IN            CLOB  CHARACTER SET dest_lob%CHARSET,
  amount      IN            INTEGER,
  dest_offset IN            INTEGER := 1,
  src_offset  IN            INTEGER := 1);
```

Parameters

Table 52–15 COPY Procedure Parameters

Parameter	Description
dest_lob	LOB locator of the copy target.
src_lob	LOB locator of source for the copy.
amount	Number of bytes (for BLOBs) or characters (for CLOBs) to copy.
dest_offset	Offset in bytes or characters in the destination LOB (origin: 1) for the start of the copy.
src_offset	Offset in bytes or characters in the source LOB (origin: 1) for the start of the copy.

Exceptions

Table 52–16 COPY Procedure Exceptions

Exception	Description
VALUE_ERROR	Any of the input parameters are NULL or invalid.
INVALID_ARGVAL	Either: - src_offset or dest_offset < 1 - src_offset or dest_offset > LOBMAXSIZE - amount < 1 - amount > LOBMAXSIZE

Usage Notes

If the offset you specify in the destination LOB is beyond the end of the data currently in this LOB, then zero-byte fillers or spaces are inserted in the destination BLOB or CLOB respectively. If the offset is less than the current length of the destination LOB, then existing data is overwritten.

It is not an error to specify an amount that exceeds the length of the data in the source LOB. Thus, you can specify a large amount to copy from the source LOB, which copies data from the src_offset to the end of the source LOB.

It is not mandatory that you wrap the LOB operation inside the Open/Close APIs. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit or rollback the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

If you do not wrap the LOB operation inside the Open/Close API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the OPEN or CLOSE statement.

> **See Also:** *Oracle Database Application Developer's Guide - Large Objects* for additional details on usage of this procedure

CREATETEMPORARY Procedures

This procedure creates a temporary BLOB or CLOB and its corresponding index in your default temporary tablespace.

Syntax

```
DBMS_LOB.CREATETEMPORARY (
   lob_loc IN OUT NOCOPY BLOB,
   cache   IN            BOOLEAN,
   dur     IN            PLS_INTEGER := 10);

DBMS_LOB.CREATETEMPORARY (
   lob_loc IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
   cache   IN            BOOLEAN,
   dur     IN            PLS_INTEGER := 10);
```

Parameters

Table 52–17 CREATETEMPORARY Procedure Parameters

Parameter	Description
lob_loc	LOB locator. For more information, see Operational Notes.
cache	Specifies if LOB should be read into buffer cache or not.
dur	1 of 2 predefined duration values (SESSION or CALL) which specifies a hint as to whether the temporary LOB is cleaned up at the end of the session or call. If dur is omitted, then the session duration is used.

See Also:

- *Oracle Database Application Developer's Guide - Large Objects* for additional details on usage of this procedure
- *Oracle Database PL/SQL User's Guide and Reference* for more information about NOCOPY and passing temporary lobs as parameters

ERASE Procedures

This procedure erases an entire internal LOB or part of an internal LOB.

Syntax

```
DBMS_LOB.ERASE (
   lob_loc           IN OUT   NOCOPY   BLOB,
   amount            IN OUT   NOCOPY   INTEGER,
   offset            IN                INTEGER := 1);

DBMS_LOB.ERASE (
   lob_loc           IN OUT   NOCOPY   CLOB CHARACTER SET ANY_CS,
   amount            IN OUT   NOCOPY   INTEGER,
   offset            IN                INTEGER := 1);
```

Parameters

Table 52–18 ERASE Procedure Parameters

Parameter	Description
lob_loc	Locator for the LOB to be erased.For more information, see Operational Notes.
amount	Number of bytes (for BLOBs or BFILES) or characters (for CLOBs or NCLOBs) to be erased.
offset	Absolute offset (origin: 1) from the beginning of the LOB in bytes (for BLOBs) or characters (CLOBs).

Usage Notes

Note: The length of the LOB is not decreased when a section of the LOB is erased. To decrease the length of the LOB value, see the "TRIM Procedures" on page 52-60.

When data is erased from the middle of a LOB, zero-byte fillers or spaces are written for BLOBs or CLOBs respectively.

The actual number of bytes or characters erased can differ from the number you specified in the amount parameter if the end of the LOB value is reached before erasing the specified number. The actual number of characters or bytes erased is returned in the amount parameter.

Exceptions

Table 52–19 ERASE Procedure Exceptions

Exception	Description
VALUE_ERROR	Any input parameter is NULL.
INVALID_ARGVAL	Either: - amount < 1 or amount > LOBMAXSIZE - offset < 1 or offset > LOBMAXSIZE

Usage Notes

It is not mandatory that you wrap the LOB operation inside the Open/Close APIs. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit or rollback the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

If you do not wrap the LOB operation inside the Open/Close API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the OPEN or CLOSE statement.

See Also:

- "TRIM Procedures" on page 52-60
- *Oracle Database Application Developer's Guide - Large Objects* for additional details on usage of this procedure

FILECLOSE Procedure

This procedure closes a BFILE that has already been opened through the input locator.

> **Note:** The database has only read-only access to BFILEs. This means that BFILEs cannot be written through the database.

Syntax

```
DBMS_LOB.FILECLOSE (
    file_loc IN OUT NOCOPY BFILE);
```

Parameters

Table 52–20 FILECLOSE Procedure Parameters

Parameter	Description
file_loc	Locator for the BFILE to be closed.

Exceptions

Table 52–21 FILECLOSE Procedure Exceptions

Exception	Description
VALUE_ERROR	NULL input value for file_loc.
UNOPENED_FILE	File was not opened with the input locator.
NOEXIST_DIRECTORY	Directory does not exist.
NOPRIV_DIRECTORY	You do not have privileges for the directory.
INVALID_DIRECTORY	Directory has been invalidated after the file was opened.
INVALID_OPERATION	File does not exist, or you do not have access privileges on the file.

See Also:

- "FILEOPEN Procedure" on page 52-38
- "FILECLOSEALL Procedure" on page 52-34
- *Oracle Database Application Developer's Guide - Large Objects* for additional details on usage of this procedure

FILECLOSEALL Procedure

This procedure closes all BFILEs opened in the session.

Syntax

```
DBMS_LOB.FILECLOSEALL;
```

Exceptions

Table 52–22 FILECLOSEALL Procedure Exception

Exception	Description
UNOPENED_FILE	No file has been opened in the session.

See Also:

- "FILEOPEN Procedure" on page 52-38
- "FILECLOSE Procedure" on page 52-33
- *Oracle Database Application Developer's Guide - Large Objects* for additional details on usage of this procedure

FILEEXISTS Function

This function finds out if a given BFILE locator points to a file that actually exists on the server's file system.

Syntax

```
DBMS_LOB.FILEEXISTS (
   file_loc     IN    BFILE)
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(FILEEXISTS, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 52–23 FILEEXISTS Function Parameter

Parameter	Description
file_loc	Locator for the BFILE.

Return Values

Table 52–24 FILEEXISTS Function Return Values

Return	Description
0	Physical file does not exist.
1	Physical file exists.

Exceptions

Table 52–25 FILEEXISTS Function Exceptions

Exception	Description
NOEXIST_DIRECTORY	Directory does not exist.
NOPRIV_DIRECTORY	You do not have privileges for the directory.
INVALID_DIRECTORY	Directory has been invalidated after the file was opened.

See Also:

- "FILEISOPEN Function" on page 52-37.
- *Oracle Database Application Developer's Guide - Large Objects* for additional details on usage of this procedure

FILEGETNAME Procedure

This procedure determines the directory object and filename, given a BFILE locator. This function only indicates the directory object name and filename assigned to the locator, not if the physical file or directory actually exists.

The maximum constraint values for the dir_alias buffer is 30, and for the entire path name, it is 2000.

Syntax

```
DBMS_LOB.FILEGETNAME (
   file_loc   IN    BFILE,
   dir_alias  OUT   VARCHAR2,
   filename   OUT   VARCHAR2);
```

Parameters

Table 52–26 FILEGETNAME Procedure Parameters

Parameter	Description
file_loc	Locator for the BFILE
dir_alias	Directory object name
filename	Name of the BFILE

Exceptions

Table 52–27 FILEGETNAME Procedure Exceptions

Exception	Description
VALUE_ERROR	Any of the input parameters are NULL or INVALID.
INVALID_ARGVAL	dir_alias or filename are NULL.

See Also: *Oracle Database Application Developer's Guide - Large Objects* for additional details on usage of this procedure

FILEISOPEN Function

This function finds out whether a BFILE was opened with the given FILE locator.

Syntax

```
DBMS_LOB.FILEISOPEN (
   file_loc   IN    BFILE)
  RETURN INTEGER;
```

Pragmas

```
PRAGMA RESTRICT_REFERENCES(fileisopen, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 52–28 FILEISOPEN Function Parameter

Parameter	Description
file_loc	Locator for the BFILE.

Return Values

INTEGER: 0 = file is not open, 1 = file is open

Usage Notes

If the input FILE locator was never passed to the FILEOPEN procedure, then the file is considered not to be opened by this locator. However, a different locator may have this file open. In other words, openness is associated with a specific locator.

Exceptions

Table 52–29 FILEISOPEN Function Exceptions

Exception	Description
NOEXIST_DIRECTORY	Directory does not exist.
NOPRIV_DIRECTORY	You do not have privileges for the directory.
INVALID_DIRECTORY	Directory has been invalidated after the file was opened.

See Also:

- "FILEEXISTS Function" on page 52-35
- *Oracle Database Application Developer's Guide - Large Objects* for additional details on usage of this procedure

FILEOPEN Procedure

This procedure opens a BFILE for read-only access. BFILE data may not be written through the database.

Syntax

```
DBMS_LOB.FILEOPEN (
   file_loc   IN OUT NOCOPY  BFILE,
   open_mode  IN             BINARY_INTEGER := file_readonly);
```

Parameters

Table 52–30 FILEOPEN Procedure Parameters

Parameter	Description
file_loc	Locator for the BFILE.
open_mode	File access is read-only.

Exceptions

Table 52–31 FILEOPEN Procedure Exceptions

Exception	Description
VALUE_ERROR	file_loc or open_mode is NULL.
INVALID_ARGVAL	open_mode is not equal to FILE_READONLY.
OPEN_TOOMANY	Number of open files in the session exceeds session_max_open_files.
NOEXIST_DIRECTORY	Directory associated with file_loc does not exist.
INVALID_DIRECTORY	Directory has been invalidated after the file was opened.
INVALID_OPERATION	File does not exist, or you do not have access privileges on the file.

See Also:

- "FILECLOSE Procedure" on page 52-33
- "FILECLOSEALL Procedure" on page 52-34
- *Oracle Database Application Developer's Guide - Large Objects* for additional details on usage of this procedure

FREETEMPORARY Procedures

This procedure frees the temporary BLOB or CLOB in your default temporary tablespace.

Syntax

```
DBMS_LOB.FREETEMPORARY (
   lob_loc  IN OUT  NOCOPY BLOB);

DBMS_LOB.FREETEMPORARY (
   lob_loc  IN OUT  NOCOPY CLOB CHARACTER SET ANY_CS);
```

Parameters

Table 52–32 FREETEMPORARY Procedure Parameters

Parameter	Description
lob_loc	LOB locator.For more information, see Operational Notes.

Usage Notes

After the call to FREETEMPORARY, the LOB locator that was freed is marked as invalid.

If an invalid LOB locator is assigned to another LOB locator using OCILobLocatorAssign in OCI or through an assignment operation in PL/SQL, then the target of the assignment is also freed and marked as invalid.

> **See Also:** *Oracle Database Application Developer's Guide - Large Objects* for additional details on usage of this procedure

GET_STORAGE_LIMIT

The DBMS_LOB.GET_STORAGE_LIMIT function returns the LOB storage limit for your database configuration. The DBMS_LOB package supports LOB instances up to this storage limit in size.

Syntax

```
DBMS_LOB.GET_STORAGE_LIMIT
  RETURN INTEGER;
```

Return Value

The value returned from this function is the maximum allowable size for LOB instances for your database configuration. The return value depends on your DB_BLOCK_SIZE initialization parameter setting and is calculated as (4 gigabytes minus 1) times the value of the DB_BLOCK_SIZE initialization parameter.

Usage

Note that BLOB instances are sized in bytes while CLOB and NCLOB instances are sized in characters.

> **See Also:** *Oracle Database Application Developer's Guide - Large Objects* for details on LOB storage limits

GETCHUNKSIZE Functions

When creating the table, you can specify the chunking factor, which can be a multiple of database blocks. This corresponds to the chunk size used by the LOB data layer when accessing or modifying the LOB value. Part of the chunk is used to store system-related information, and the rest stores the LOB value.

This function returns the amount of space used in the LOB chunk to store the LOB value.

Syntax

```
DBMS_LOB.GETCHUNKSIZE (
   lob_loc IN BLOB)
  RETURN INTEGER;

DBMS_LOB.GETCHUNKSIZE (
   lob_loc IN CLOB CHARACTER SET ANY_CS)
  RETURN INTEGER;
```

Pragmas

```
PRAGMA RESTRICT_REFERENCES(getchunksize, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 52–33 GETCHUNKSIZE Function Parameters

Parameter	Description
lob_loc	LOB locator. For more information, see Operational Notes.

Return Values

The value returned for BLOBs is in terms of bytes. The value returned for CLOBs is in terms of characters.

Usage Notes

Performance is improved if you enter read/write requests using a multiple of this chunk size. For writes, there is an added benefit, because LOB chunks are versioned, and if all writes are done on a chunk basis, then no extra or excess versioning is done or duplicated. You could batch up the WRITE until you have enough for a chunk, instead of issuing several WRITE calls for the same chunk.

> **See Also:** *Oracle Database Application Developer's Guide - Large Objects* for additional details on usage of this procedure

GETLENGTH Functions

This function gets the length of the specified LOB. The length in bytes or characters is returned.

The length returned for a BFILE includes the EOF, if it exists. Any 0-byte or space filler in the LOB caused by previous ERASE or WRITE operations is also included in the length count. The length of an empty internal LOB is 0.

Syntax

```
DBMS_LOB.GETLENGTH (
   lob_loc    IN  BLOB)
  RETURN INTEGER;

DBMS_LOB.GETLENGTH (
   lob_loc    IN  CLOB   CHARACTER SET ANY_CS)
  RETURN INTEGER;

DBMS_LOB.GETLENGTH (
   file_loc    IN  BFILE)
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(GETLENGTH, WNDS, WNPS, RNDS, RNPS);
```

Parameters

Table 52–34 GETLENGTH Function Parameter

Parameter	Description
file_loc	The file locator for the LOB whose length is to be returned.

Return Values

The length of the LOB in bytes or characters as an INTEGER. NULL is returned if the input LOB is NULL or if the input lob_loc is NULL. An error is returned in the following cases for BFILEs:

- lob_loc does not have the necessary directory and operating system privileges
- lob_loc cannot be read because of an operating system read error

> **See Also:** *Oracle Database Application Developer's Guide - Large Objects* for additional details on usage of this procedure

INSTR Functions

This function returns the matching position of the *nth* occurrence of the pattern in the LOB, starting from the offset you specify.

Syntax

```
DBMS_LOB.INSTR (
   lob_loc    IN   BLOB,
   pattern    IN   RAW,
   offset     IN   INTEGER := 1,
   nth        IN   INTEGER := 1)
  RETURN INTEGER;

DBMS_LOB.INSTR (
   lob_loc    IN   CLOB      CHARACTER SET ANY_CS,
   pattern    IN   VARCHAR2  CHARACTER SET lob_loc%CHARSET,
   offset     IN   INTEGER := 1,
   nth        IN   INTEGER := 1)
  RETURN INTEGER;

DBMS_LOB.INSTR (
   file_loc   IN   BFILE,
   pattern    IN   RAW,
   offset     IN   INTEGER := 1,
   nth        IN   INTEGER := 1)
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(INSTR, WNDS, WNPS, RNDS, RNPS);
```

Parameters

Table 52–35 INSTR Function Parameters

Parameter	Description
lob_loc	Locator for the LOB to be examined. For more information, see Operational Notes.
file_loc	The file locator for the LOB to be examined.
pattern	Pattern to be tested for. The pattern is a group of RAW bytes for BLOBs, and a character string (VARCHAR2) for CLOBs.The maximum size of the pattern is 16383 bytes.
offset	Absolute offset in bytes (BLOBs) or characters (CLOBs) at which the pattern matching is to start. (origin: 1)
nth	Occurrence number, starting at 1.

Return Values

Table 52–36 INSTR Function Return Values

Return	Description
INTEGER	Offset of the start of the matched pattern, in bytes or characters. It returns 0 if the pattern is not found.

Table 52–36 (Cont.) INSTR Function Return Values

Return	Description
NULL	Either: -any one or more of the IN parameters was NULL or INVALID. -offset < 1 or offset > LOBMAXSIZE. -nth < 1. -nth > LOBMAXSIZE.

Usage Notes

The form of the VARCHAR2 buffer (the pattern parameter) must match the form of the CLOB parameter. In other words, if the input LOB parameter is of type NCLOB, then the buffer must contain NCHAR data. Conversely, if the input LOB parameter is of type CLOB, then the buffer must contain CHAR data.

For BFILEs, the file must be already opened using a successful FILEOPEN operation for this operation to succeed.

Operations that accept RAW or VARCHAR2 parameters for pattern matching, such as INSTR, do not support regular expressions or special matching characters (as in the case of SQL LIKE) in the pattern parameter or substrings.

Exceptions

Table 52–37 INSTR Function Exceptions for BFILES

Exception	Description
UNOPENED_FILE	File was not opened using the input locator.
NOEXIST_ DIRECTORY	Directory does not exist.
NOPRIV_DIRECTORY	You do not have privileges for the directory.
INVALID_ DIRECTORY	Directory has been invalidated after the file was opened.
INVALID_ OPERATION	File does not exist, or you do not have access privileges on the file.

See Also:

- "SUBSTR Functions" on page 52-58
- *Oracle Database Application Developer's Guide - Large Objects* for additional details on usage of this procedure

ISOPEN Functions

This function checks to see if the LOB was already opened using the input locator. This subprogram is for internal and external LOBs.

Syntax

```
DBMS_LOB.ISOPEN (
   lob_loc IN BLOB)
  RETURN INTEGER;

DBMS_LOB.ISOPEN (
   lob_loc IN CLOB CHARACTER SET ANY_CS)
  RETURN INTEGER;

DBMS_LOB.ISOPEN (
   file_loc IN BFILE)
  RETURN INTEGER;
```

Pragmas

```
PRAGMA RESTRICT_REFERENCES(isopen, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 52–38 ISOPEN Function Parameters

Parameter	Description
lob_loc	LOB locator. For more information, see Operational Notes.
file_loc	File locator.

Usage Notes

For BFILES, openness is associated with the locator. If the input locator was never passed to OPEN, the BFILE is not considered to be opened by this locator. However, a different locator may have opened the BFILE. More than one OPEN can be performed on the same BFILE using different locators.

For internal LOBs, openness is associated with the LOB, not with the locator. If locator1 opened the LOB, then locator2 also sees the LOB as open. For internal LOBs, ISOPEN requires a round-trip, because it checks the state on the server to see if the LOB is indeed open.

For external LOBs (BFILEs), ISOPEN also requires a round-trip, because that's where the state is kept.

> **See Also:** *Oracle Database Application Developer's Guide - Large Objects* for additional details on usage of this procedure

ISTEMPORARY Functions

This function determines whether a LOB instance is temporary.

Syntax

```
DBMS_LOB.ISTEMPORARY (
   lob_loc IN BLOB)
  RETURN INTEGER;

DBMS_LOB.ISTEMPORARY (
   lob_loc IN CLOB CHARACTER SET ANY_CS)
  RETURN INTEGER;
```

Pragmas

```
PRAGMA RESTRICT_REFERENCES(istemporary, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 52–39 ISTEMPORARY Procedure Parameters

Parameter	Description
lob_loc	LOB locator. For more information, see Operational Notes.

Return Values

This function returns TRUE in temporary if the locator is pointing to a temporary LOB. It returns FALSE otherwise.

> **See Also:** *Oracle Database Application Developer's Guide - Large Objects* for additional details on usage of this procedure

LOADBLOBFROMFILE Procedure

This procedure loads data from BFILE to internal BLOB. This achieves the same outcome as LOADFROMFILE, and returns the new offsets.

Syntax

```
DBMS_LOB.LOADBLOBFROMFILE (
   dest_lob    IN OUT NOCOPY BLOB,
   src_bfile   IN            BFILE,
   amount      IN            INTEGER,
   dest_offset IN OUT        INTEGER,
   src_offset  IN OUT        INTEGER);
```

Parameters

Table 52–40 LOADBLOBFROMFILE Procedure Parameters

Parameter	Descriptionw
dest_lob	BLOB locator of the target for the load.
src_bfile	BFILE locator of the source for the load.
amount	Number of bytes to load from the BFILE. You can also use DBMS_LOB.LOBMAXSIZE to load until the end of the BFILE.
dest_offset	(IN) Offset in bytes in the destination BLOB (origin: 1) for the start of the write. (OUT) New offset in bytes in the destination BLOB right after the end of this write, which is also where the next write should begin.
src_offset	(IN) Offset in bytes in the source BFILE (origin: 1) for the start of the read. (OUT) Offset in bytes in the source BFILE right after the end of this read, which is also where the next read should begin.

Usage Notes

You can specify the offsets for both the source and destination LOBs, and the number of bytes to copy from the source BFILE. The amount and src_offset, because they refer to the BFILE, are in terms of bytes, and the dest_offset is in bytes for BLOBs.

If the offset you specify in the destination LOB is beyond the end of the data currently in this LOB, then zero-byte fillers or spaces are inserted in the destination BLOB. If the offset is less than the current length of the destination LOB, then existing data is overwritten.

There is an error if the input amount plus offset exceeds the length of the data in the BFILE (unless the amount specified is LOBMAXSIZE which you can specify to continue loading until the end of the BFILE is reached).

It is not mandatory that you wrap the LOB operation inside the OPEN/CLOSE operations. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit or rollback the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

If you do not wrap the LOB operation inside the OPEN/CLOSE, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect

performance. Therefore, it is recommended that you enclose write operations to the LOB within the OPEN or CLOSE statement.

Constants and Defaults

There is no easy way to omit parameters. You must either declare a variable for IN/OUT parameter or provide a default value for the IN parameter. Here is a summary of the constants and the defaults that can be used.

Table 52–41 Suggested Values of the Parameter

Parameter	Default Value	Description
amount	DBMSLOB.LOBMAXSIZE (IN)	Load the entire file
dest_offset	1 (IN)	start from the beginning
src_offset	1 (IN)	start from the beginning

Constants defined in DBMSLOB.SQL

```
lobmaxsize                    CONSTANT INTEGER        := 4294967295;
```

Exceptions

Table 52–42 LOADBLOBFROMFILE Procedure Exceptions

Exception	Description
VALUE_ERROR	Any of the input parameters are NULL or INVALID.
INVALID_ARGVAL	Either: - src_offset or dest_offset < 1. - src_offset or dest_offset > LOBMAXSIZE. - amount < 1. - amount > LOBMAXSIZE.

See Also: *Oracle Database Application Developer's Guide - Large Objects* for additional details on usage of this procedure

LOADCLOBFROMFILE Procedure

This procedure loads data from a BFILE to an internal CLOB/NCLOB with necessary character set conversion and returns the new offsets.

Syntax

```
DBMS_LOB.LOADCLOBFROMFILE (
   dest_lob       IN OUT NOCOPY   BLOB,
   src_bfile      IN              BFILE,
   amount         IN              INTEGER,
   dest_offset    IN OUT          INTEGER,
   src_offset     IN OUT          INTEGER,
   bfile_csid     IN              NUMBER,
   lang_context   IN OUT          INTEGER,
   warning        OUT             INTEGER);
```

Parameters

Table 52–43 LOADCLOBFROMFILE Procedure Parameters

Parameter	Description
dest_lob	CLOB/NCLOB locator of the target for the load.
src_bfile	BFILE locator of the source for the load.
amount	Number of bytes to load from the BFILE. Use DBMS_LOB.LOBMAXSIZE to load until the end of the BFILE.
dest_offset	(IN) Offset in characters in the destination CLOB (origin: 1) for the start of the write. (OUT) The new offset in characters right after the end of this load, which is also where the next load should start. It always points to the beginning of the first complete character after the end of load. If the last character is not complete, offset goes back to the beginning of the partial character.
src_offset	(IN) Offset in bytes in the source BFILE (origin: 1) for the start of the read. (OUT) Offset in bytes in the source BFILE right after the end of this read, which is also where the next read should begin.
bfile_csid	Character set id of the source (BFILE) file.
lang_context	(IN) Language context, such as shift status, for the current load. (OUT) The language context at the time when the current load stopped, and what the next load should be using if continuing loading from the same source. This information is returned to the user so that they can use it for the continuous load without losing or misinterpreting any source data. For the very first load or if do not care, simply use the default 0. The details of this language context is hidden from the user. One does not need to know what it is or what's in it in order to make the call

Table 52–43 (Cont.) LOADCLOBFROMFILE Procedure Parameters

Parameter	Description
warning	(OUT) Warning message. This indicates something abnormal happened during the loading. It may or may not be caused by the user's mistake. The loading is completed as required, and it's up to the user to check the warning message. Currently, the only possible warning is the inconvertible character. This happens when the character in the source cannot be properly converted to a character in destination, and the default replacement character (for example, '?') is used in place. The message is defined as warn_inconvertible_char in DBMSLOB.

Usage Notes

You can specify the offsets for both the source and destination LOBs, and the number of bytes to copy from the source BFILE. The amount and src_offset, because they refer to the BFILE, are in terms of bytes, and the dest_offset is in characters for CLOBs.

If the offset you specify in the destination LOB is beyond the end of the data currently in this LOB, then zero-byte fillers or spaces are inserted in the destination CLOB. If the offset is less than the current length of the destination LOB, then existing data is overwritten.

There is an error if the input amount plus offset exceeds the length of the data in the BFILE (unless the amount specified is LOBMAXSIZE which you can specify to continue loading until the end of the BFILE is reached).

Note the following requirements:

- The destination character set is always the same as the database character set in the case of CLOB and national character set in the case of NCLOB.
- csid=0 indicates the default behavior that uses database csid for CLOB and national csid for NCLOB in the place of source csid. Conversion is still necessary if it is of varying width
- It is not mandatory that you wrap the LOB operation inside the OPEN/CLOSE operations. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit or rollback the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

  If you do not wrap the LOB operation inside the OPEN/CLOSE, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the OPEN or CLOSE statement.

Constants

Here is a summary of the constants and the suggested values that can be used.

Table 52–44 Suggested Values of the Parameter

Parameter	Suggested Value	Description
amount	DBMSLOB.LOBMAXSIZE (IN)	Load the entire file
dest_offset	1 (IN)	start from the beginning
src_offset	1 (IN)	start from the beginning

Table 52–44 (Cont.) Suggested Values of the Parameter

Parameter	Suggested Value	Description
csid	0 (IN)	default csid, use destination csid
lang_context	0 (IN)	default language context
warning	0 (OUT)	no warning message, everything is ok

Constants defined in DBMSLOB.SQL

```
lobmaxsize                  CONSTANT INTEGER       := 18446744073709551615;
warn_inconvertible_char     CONSTANT INTEGER       := 1;
default_csid                CONSTANT INTEGER       := 0;
default_lang_ctx            CONSTANT INTEGER       := 0;
no_warning                  CONSTANT INTEGER       := 0;
```

Exceptions

Table 52–45 LOADCLOBFROMFILE Procedure Exceptions

Exception	Description
VALUE_ERROR	Any of the input parameters are NULL or INVALID.
INVALID_ARGVAL	Either: - src_offset or dest_offset < 1. - src_offset or dest_offset > LOBMAXSIZE. - amount < 1. - amount > LOBMAXSIZE.

See Also: *Oracle Database Application Developer's Guide - Large Objects* for additional details on usage of this procedure

LOADFROMFILE Procedure

This procedure copies all, or a part of, a source external LOB (BFILE) to a destination internal LOB.

Syntax

```
DBMS_LOB.LOADFROMFILE (
   dest_lob    IN OUT NOCOPY BLOB,
   src_file    IN            BFILE,
   amount      IN            INTEGER,
   dest_offset IN            INTEGER  := 1,
   src_offset  IN            INTEGER  := 1);
```

Parameters

Table 52–46 LOADFROMFILE Procedure Parameters

Parameter	Description
dest_lob	LOB locator of the target for the load.
src_file	BFILE locator of the source for the load.
amount	Number of bytes to load from the BFILE.
dest_offset	Offset in bytes or characters in the destination LOB (origin: 1) for the start of the load.
src_offset	Offset in bytes in the source BFILE (origin: 1) for the start of the load.

Usage Notes

You can specify the offsets for both the source and destination LOBs, and the number of bytes to copy from the source BFILE. The amount and src_offset, because they refer to the BFILE, are in terms of bytes, and the dest_offset is either in bytes or characters for BLOBs and CLOBs respectively.

> **Note:** The input BFILE must have been opened prior to using this procedure. No character set conversions are performed implicitly when binary BFILE data is loaded into a CLOB. The BFILE data must already be in the same character set as the CLOB in the database. No error checking is performed to verify this.

If the offset you specify in the destination LOB is beyond the end of the data currently in this LOB, then zero-byte fillers or spaces are inserted in the destination BLOB or CLOB respectively. If the offset is less than the current length of the destination LOB, then existing data is overwritten.

There is an error if the input amount plus offset exceeds the length of the data in the BFILE.

Note: If the character set is varying width, UTF-8 for example, the LOB value is stored in the fixed-width UCS2 format. Therefore, if you are using DBMS_LOB.LOADFROMFILE, the data in the BFILE should be in the UCS2 character set instead of the UTF-8 character set. However, you should use sql*loader instead of LOADFROMFILE to load data into a CLOB or NCLOB because sql*loader will provide the necessary character set conversions.

It is not mandatory that you wrap the LOB operation inside the Open/Close APIs. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit or rollback the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

If you do not wrap the LOB operation inside the Open/Close API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the OPEN or CLOSE statement.

Exceptions

Table 52–47 LOADFROMFILE Procedure Exceptions

Exception	Description
VALUE_ERROR	Any of the input parameters are NULL or INVALID.
INVALID_ARGVAL	Either: - src_offset or dest_offset < 1. - src_offset or dest_offset > LOBMAXSIZE. - amount < 1. - amount > LOBMAXSIZE.

See Also: *Oracle Database Application Developer's Guide - Large Objects* for additional details on usage of this procedure

OPEN Procedures

This procedure opens a LOB, internal or external, in the indicated mode. Valid modes include read-only, and read/write.

Syntax

```
DBMS_LOB.OPEN (
   lob_loc   IN OUT NOCOPY BLOB,
   open_mode IN            BINARY_INTEGER);

DBMS_LOB.OPEN (
   lob_loc   IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
   open_mode IN            BINARY_INTEGER);

DBMS_LOB.OPEN (
   file_loc  IN OUT NOCOPY BFILE,
   open_mode IN            BINARY_INTEGER := file_readonly);
```

Parameters

Table 52–48 OPEN Procedure Parameters

Parameter	Description
lob_loc	LOB locator. For more information, see Operational Notes.
open_mode	Mode in which to open. For BLOB and CLOB types, the mode can be either: LOB_READONLY or LOB_READWRITE. For BFILE types, the mode must be FILE_READONLY.

Usage Notes

Note: If the LOB was opened in read-only mode, and if you try to write to the LOB, then an error is returned. BFILE can only be opened with read-only mode.

OPEN requires a round-trip to the server for both internal and external LOBs. For internal LOBs, OPEN triggers other code that relies on the OPEN call. For external LOBs (BFILEs), OPEN requires a round-trip because the actual operating system file on the server side is being opened.

It is not mandatory that you wrap all LOB operations inside the Open/Close APIs. However, if you open a LOB, you must close it before you commit or rollback the transaction; an error is produced if you do not. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

It is an error to commit the transaction before closing all opened LOBs that were opened by the transaction. When the error is returned, the openness of the open LOBs is discarded, but the transaction is successfully committed. Hence, all the changes made to the LOB and nonLOB data in the transaction are committed, but the domain and function-based indexes are not updated. If this happens, you should rebuild the functional and domain indexes on the LOB column.

See Also: *Oracle Database Application Developer's Guide - Large Objects* for additional details on usage of this procedure

READ Procedures

This procedure reads a piece of a LOB, and returns the specified amount into the buffer parameter, starting from an absolute offset from the beginning of the LOB.

The number of bytes or characters actually read is returned in the amount parameter. If the input offset points past the End of LOB, then amount is set to 0, and a NO_DATA_FOUND exception is raised.

Syntax

```
DBMS_LOB.READ (
   lob_loc   IN             BLOB,
   amount    IN OUT  NOCOPY BINARY_INTEGER,
   offset    IN             INTEGER,
   buffer    OUT            RAW);

DBMS_LOB.READ (
   lob_loc   IN             CLOB CHARACTER SET ANY_CS,
   amount    IN OUT  NOCOPY BINARY_INTEGER,
   offset    IN             INTEGER,
   buffer    OUT            VARCHAR2 CHARACTER SET lob_loc%CHARSET);

DBMS_LOB.READ (
   file_loc   IN              BFILE,
   amount    IN OUT   NOCOPY BINARY_INTEGER,
   offset    IN              INTEGER,
   buffer    OUT             RAW);
```

Parameters

Table 52–49 READ Procedure Parameters

Parameter	Description
lob_loc	Locator for the LOB to be read. For more information, see Operational Notes.
file_loc	The file locator for the LOB to be examined.
amount	Number of bytes (for BLOBs) or characters (for CLOBs) to read, or number that were read.
offset	Offset in bytes (for BLOBs) or characters (for CLOBs) from the start of the LOB (origin: 1).
buffer	Output buffer for the read operation.

Exceptions

Table 52–50 lists exceptions that apply to any LOB instance. Table 52–51 lists exceptions that apply only to BFILEs.

Table 52–50 READ Procedure Exceptions

Exception	Description
VALUE_ERROR	Any of lob_loc, amount, or offset parameters are NULL.

Table 52–50 (Cont.) READ Procedure Exceptions

Exception	Description
INVALID_ARGVAL	Either: - amount < 1 - amount > MAXBUFSIZE - offset < 1 - offset > LOBMAXSIZE - amount is greater, in bytes or characters, than the capacity of buffer.
NO_DATA_FOUND	End of the LOB is reached, and there are no more bytes or characters to read from the LOB: amount has a value of 0.

Table 52–51 READ Procedure Exceptions for BFILEs

Exception	Description
UNOPENED_FILE	File is not opened using the input locator.
NOEXIST_ DIRECTORY	Directory does not exist.
NOPRIV_DIRECTORY	You do not have privileges for the directory.
INVALID_ DIRECTORY	Directory has been invalidated after the file was opened.
INVALID_ OPERATION	File does not exist, or you do not have access privileges on the file.

Usage Notes

The form of the VARCHAR2 buffer must match the form of the CLOB parameter. In other words, if the input LOB parameter is of type NCLOB, then the buffer must contain NCHAR data. Conversely, if the input LOB parameter is of type CLOB, then the buffer must contain CHAR data.

When calling DBMS_LOB.READ from the client (for example, in a BEGIN/END block from within SQL*Plus), the returned buffer contains data in the client's character set. The database converts the LOB value from the server's character set to the client's character set before it returns the buffer to the user.

> **See Also:** *Oracle Database Application Developer's Guide - Large Objects* for additional details on usage of this procedure

SUBSTR Functions

This function returns amount bytes or characters of a LOB, starting from an absolute offset from the beginning of the LOB.

For fixed-width n-byte CLOBs, if the input amount for SUBSTR is greater than (32767/n), then SUBSTR returns a character buffer of length (32767/n), or the length of the CLOB, whichever is lesser. For CLOBs in a varying-width character set, n is the maximum byte-width used for characters in the CLOB.

Syntax

```
DBMS_LOB.SUBSTR (
   lob_loc     IN    BLOB,
   amount      IN    INTEGER := 32767,
   offset      IN    INTEGER := 1)
  RETURN RAW;

DBMS_LOB.SUBSTR (
   lob_loc     IN    CLOB   CHARACTER SET ANY_CS,
   amount      IN    INTEGER := 32767,
   offset      IN    INTEGER := 1)
  RETURN VARCHAR2 CHARACTER SET lob_loc%CHARSET;

DBMS_LOB.SUBSTR (
   file_loc     IN    BFILE,
   amount      IN    INTEGER := 32767,
   offset      IN    INTEGER := 1)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(SUBSTR, WNDS, WNPS, RNDS, RNPS);
```

Parameters

Table 52–52 SUBSTR Function Parameters

Parameter	Description
lob_loc	Locator for the LOB to be read. For more information, see Operational Notes.
file_loc	The file locator for the LOB to be examined.
amount	Number of bytes (for BLOBs) or characters (for CLOBs) to be read.
offset	Offset in bytes (for BLOBs) or characters (for CLOBs) from the start of the LOB (origin: 1).

Return Values

Table 52–53 SUBSTR Function Return Values

Return	Description
RAW	Function overloading that has a BLOB or BFILE in parameter.
VARCHAR2	CLOB version.

Table 52–53 (Cont.) SUBSTR Function Return Values

Return	Description
NULL	Either: - any input parameter is NULL - amount < 1 - amount > 32767 - offset < 1 - offset > LOBMAXSIZE

Exceptions

Table 52–54 SUBSTR Function Exceptions for BFILE operations

Exception	Description
UNOPENED_FILE	File is not opened using the input locator.
NOEXIST_ DIRECTORY	Directory does not exist.
NOPRIV_DIRECTORY	You do not have privileges for the directory.
INVALID_ DIRECTORY	Directory has been invalidated after the file was opened.
INVALID_ OPERATION	File does not exist, or you do not have access privileges on the file.

Usage Notes

The form of the VARCHAR2 buffer must match the form of the CLOB parameter. In other words, if the input LOB parameter is of type NCLOB, then the buffer must contain NCHAR data. Conversely, if the input LOB parameter is of type CLOB, then the buffer must contain CHAR data.

When calling DBMS_LOB.SUBSTR from the client (for example, in a BEGIN/END block from within SQL*Plus), the returned buffer contains data in the client's character set. The database converts the LOB value from the server's character set to the client's character set before it returns the buffer to the user.

See Also:

- "INSTR Functions" on page 52-43
- "READ Procedures" on page 52-56
- *Oracle Database Application Developer's Guide - Large Objects* for additional details on usage of this procedure

TRIM Procedures

This procedure trims the value of the internal LOB to the length you specify in the newlen parameter. Specify the length in bytes for BLOBs, and specify the length in characters for CLOBs.

> **Note:** The TRIM procedure decreases the length of the LOB to the value specified in the newlen parameter.

If you attempt to TRIM an empty LOB, then nothing occurs, and TRIM returns no error. If the new length that you specify in newlen is greater than the size of the LOB, then an exception is raised.

Syntax

```
DBMS_LOB.TRIM (
   lob_loc         IN OUT  NOCOPY BLOB,
   newlen          IN              INTEGER);

DBMS_LOB.TRIM (
   lob_loc         IN OUT  NOCOPY CLOB CHARACTER SET ANY_CS,
   newlen          IN              INTEGER);
```

Parameters

Table 52–55 RIM Procedure Parameters

Parameter	Description
lob_loc	Locator for the internal LOB whose length is to be trimmed. For more information, see Operational Notes.
newlen	New, trimmed length of the LOB value in bytes for BLOBs or characters for CLOBs.

Exceptions

Table 52–56 TRIM Procedure Exceptions

Exception	Description
VALUE_ERROR	lob_loc is NULL.
INVALID_ARGVAL	Either: - new_len < 0 - new_len > LOBMAXSIZE

Usage Notes

It is not mandatory that you wrap the LOB operation inside the Open/Close APIs. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit or rollback the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

If you do not wrap the LOB operation inside the Open/Close API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the OPEN or CLOSE statement.

See Also:

- "ERASE Procedures" on page 52-31
- "WRITEAPPEND Procedures" on page 52-64
- *Oracle Database Application Developer's Guide - Large Objects* for additional details on usage of this procedure

WRITE Procedures

This procedure writes a specified amount of data into an internal LOB, starting from an absolute offset from the beginning of the LOB. The data is written from the buffer parameter.

WRITE replaces (overwrites) any data that already exists in the LOB at the offset, for the length you specify.

Syntax

```
DBMS_LOB.WRITE (
   lob_loc  IN OUT NOCOPY  BLOB,
   amount   IN             BINARY_INTEGER,
   offset   IN             INTEGER,
   buffer   IN             RAW);

DBMS_LOB.WRITE (
   lob_loc  IN OUT  NOCOPY CLOB   CHARACTER SET ANY_CS,
   amount   IN             BINARY_INTEGER,
   offset   IN             INTEGER,
   buffer   IN             VARCHAR2 CHARACTER SET lob_loc%CHARSET);
```

Parameters

Table 52–57 WRITE Procedure Parameters

Parameter	Description
lob_loc	Locator for the internal LOB to be written to. For more information, see Operational Notes.
amount	Number of bytes (for BLOBs) or characters (for CLOBs) to write, or number that were written.
offset	Offset in bytes (for BLOBs) or characters (for CLOBs) from the start of the LOB (origin: 1) for the write operation.
buffer	Input buffer for the write.

Exceptions

Table 52–58 WRITE Procedure Exceptions

Exception	Description
VALUE_ERROR	Any of lob_loc, amount, or offset parameters are NULL, out of range, or INVALID.
INVALID_ARGVAL	Either: - amount < 1 - amount > MAXBUFSIZE - offset < 1 - offset > LOBMAXSIZE

Usage Notes

There is an error if the input amount is more than the data in the buffer. If the input amount is less than the data in the buffer, then only amount bytes or characters from the buffer is written to the LOB. If the offset you specify is beyond the end of the data

currently in the LOB, then zero-byte fillers or spaces are inserted in the BLOB or CLOB respectively.

The form of the VARCHAR2 buffer must match the form of the CLOB parameter. In other words, if the input LOB parameter is of type NCLOB, then the buffer must contain NCHAR data. Conversely, if the input LOB parameter is of type CLOB, then the buffer must contain CHAR data.

When calling DBMS_LOB.WRITE from the client (for example, in a BEGIN/END block from within SQL*Plus), the buffer must contain data in the client's character set. The database converts the client-side buffer to the server's character set before it writes the buffer data to the LOB.

It is not mandatory that you wrap the LOB operation inside the Open/Close APIs. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit or rollback the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

If you do not wrap the LOB operation inside the Open/Close API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the OPEN or CLOSE statement.

See Also:

- "APPEND Procedures" on page 52-18
- "COPY Procedures" on page 52-28
- *Oracle Database Application Developer's Guide - Large Objects* for additional details on usage of this procedure

WRITEAPPEND Procedures

This procedure writes a specified amount of data to the end of an internal LOB. The data is written from the buffer parameter.

Syntax

```
DBMS_LOB.WRITEAPPEND (
   lob_loc IN OUT NOCOPY BLOB,
   amount  IN            BINARY_INTEGER,
   buffer  IN            RAW);

DBMS_LOB.WRITEAPPEND (
   lob_loc IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
   amount  IN            BINARY_INTEGER,
   buffer  IN            VARCHAR2 CHARACTER SET lob_loc%CHARSET);
```

Parameters

Table 52–59 WRITEAPPEND Procedure Parameters

Parameter	Description
lob_loc	Locator for the internal LOB to be written to. For more information, see Operational Notes.
amount	Number of bytes (for BLOBs) or characters (for CLOBs) to write, or number that were written.
buffer	Input buffer for the write.

Usage Notes

There is an error if the input amount is more than the data in the buffer. If the input amount is less than the data in the buffer, then only amount bytes or characters from the buffer are written to the end of the LOB.

Exceptions

Table 52–60 WRITEAPPEND Procedure Exceptions

Exception	Description
VALUE_ERROR	Any of lob_loc, amount, or offset parameters are NULL, out of range, or INVALID.
INVALID_ARGVAL	Either: - amount < 1 - amount > MAXBUFSIZE

Usage Notes

The form of the VARCHAR2 buffer must match the form of the CLOB parameter. In other words, if the input LOB parameter is of type NCLOB, then the buffer must contain NCHAR data. Conversely, if the input LOB parameter is of type CLOB, then the buffer must contain CHAR data.

When calling DBMS_LOB.WRITEAPPEND from the client (for example, in a BEGIN/END block from within SQL*Plus), the buffer must contain data in the client's character set.

The database converts the client-side buffer to the server's character set before it writes the buffer data to the LOB.

It is not mandatory that you wrap the LOB operation inside the Open/Close APIs. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit or rollback the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

If you do not wrap the LOB operation inside the Open/Close API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the OPEN or CLOSE statement.

See Also:

- "APPEND Procedures" on page 52-18
- "COPY Procedures" on page 52-28
- "WRITE Procedures" on page 52-62
- *Oracle Database Application Developer's Guide - Large Objects* for additional details on usage of this procedure

53

DBMS_LOCK

The DBMS_LOCK package provides an interface to Oracle Lock Management services. You can request a lock of a specific mode, give it a unique name recognizable in another procedure in the same or another instance, change the lock mode, and release it.

> **See Also:** For more information, and an example of how to use the DBMS_LOCK package, see "About User Locks" in *Oracle Database Application Developer's Guide - Fundamentals*

This chapter contains the following topics:

Using DBMS_LOCK

Overview

Some uses of user locks:

- Providing exclusive access to a device, such as a terminal
- Providing application-level enforcement of read locks
- Detecting when a lock is released and cleanup after the application
- Synchronizing applications and enforcing sequential processing

Security Model

There might be operating system-specific limits on the maximum number of total locks available. This *must* be considered when using locks or making this package available to other users. Consider granting the `EXECUTE` privilege only to specific users or roles.

A better alternative would be to create a cover package limiting the number of locks used and grant `EXECUTE` privilege to specific users. An example of a cover package is documented in the `DBMS_LOCK.SQL` package specification file. The abbreviations for these locks as they appear in Enterprise Manager monitors are in parentheses.

Constants

The DBMS_LOCK package uses the constants shown in Table 53–1.

Table 53–1 DBMS_LOCK Constants

Name	Alternate Name(s)	Type	Value	OEM Abbreviation	Description
NL_MODE	NuL1	INTEGER	1	-	-
SS_MODE	Sub Shared	INTEGER	2	ULRS	This can be used on an aggregate object to indicate that share locks are being acquired on subparts of the object.
SX_MODE	■ Sub eXclusive ■ Row Exclusive Mode	INTEGER	3	ULRX	This can be used on an aggregate object to indicate that exclusive locks are being acquired on sub-parts of the object.
S_MODE	■ Shared ■ Row Exclusive Mode ■ Intended Exclusive	INTEGER	4	ULRSX	-
SSX_MODE	■ Shared Sub eXclusive ■ Share Row Exclusive Mode	INTEGER	5	-	This indicates that the entire aggregate object has a share lock, but some of the sub-parts may additionally have exclusive locks.
X_MODE	Exclusive	INTEGER	6	ULX	-

These are the various lock modes (nl -> "NuLl", ss -> "Sub Shared", sx -> "Sub eXclusive", s -> "Shared", ssx -> "Shared Sub eXclusive", x -> "eXclusive").

Rules and Limits

When another process holds "held", an attempt to get "get" does the following:

Table 53–2 Lock Compatibility

HELD MODE	GET NL	GET SS	GET SX	GET S	GET SSX	GET X
NL	Success	Success	Success	Success	Success	Success
SS	Success	Success	Success	Success	Success	Fail
SX	Success	Success	Success	Fail	Fail	Fail
S	Success	Success	Fail	Success	Fail	Fail
SSX	Success	Success	Fail	Fail	Fail	Fail
X	Success	Fail	Fail	Fail	Fail	Fail

```
maxwait  constant integer := 32767;
```

The constant `maxwait` waits forever.

Operational Notes

User locks never conflict with Oracle locks because they are identified with the prefix "UL". You can view these locks using the Enterprise Manager lock monitor screen or the appropriate fixed views. User locks are automatically released when a session terminates.The lock identifier is a number in the range of 0 to 1073741823.

Because a reserved user lock is the same as an Oracle lock, it has all the functionality of an Oracle lock, such as deadlock detection. Be certain that any user locks used in distributed transactions are released upon `COMMIT`, or an undetected deadlock may occur.

`DBMS_LOCK` is most efficient with a limit of a few hundred locks for each session. Oracle strongly recommends that you develop a standard convention for using these locks in order to avoid conflicts among procedures trying to use the same locks. For example, include your company name as part of your lock names.

Summary of DBMS_LOCK Subprograms

Table 53–3 DBMS_LOCK Package Subprograms

Subprogram	Description
ALLOCATE_UNIQUE Procedure on page 53-9	Allocates a unique lock ID to a named lock.
CONVERT Function on page 53-11	Converts a lock from one mode to another.
RELEASE Function on page 53-12	Releases a lock.
REQUEST Function on page 53-13	Requests a lock of a specific mode.
SLEEP Procedure on page 53-14	Puts a procedure to sleep for a specific time.

ALLOCATE_UNIQUE Procedure

This procedure allocates a unique lock identifier (in the range of 1073741824 to 1999999999) given a lock name. Lock identifiers are used to enable applications to coordinate their use of locks. This is provided because it may be easier for applications to coordinate their use of locks based on lock names rather than lock numbers.

Syntax

```
DBMS_LOCK.ALLOCATE_UNIQUE (
   lockname         IN  VARCHAR2,
   lockhandle       OUT VARCHAR2,
   expiration_secs  IN  INTEGER   DEFAULT 864000);
```

Parameters

Table 53–4 ALLOCATE_UNIQUE Procedure Parameters

Parameter	Description
lockname	Name of the lock for which you want to generate a unique ID. Do not use lock names beginning with ORA$; these are reserved for products supplied by Oracle.
lockhandle	Returns the handle to the lock ID generated by ALLOCATE_UNIQUE. You can use this handle in subsequent calls to REQUEST, CONVERT, and RELEASE. A handle is returned instead of the actual lock ID to reduce the chance that a programming error accidentally creates an incorrect, but valid, lock ID. This provides better isolation between different applications that are using this package. LOCKHANDLE can be up to VARCHAR2 (128). All sessions using a lock handle returned by ALLOCATE_UNIQUE with the same lock name are referring to the same lock. Therefore, do not pass lock handles from one session to another.
expiration_specs	Number of seconds to wait after the last ALLOCATE_UNIQUE has been performed on a given lock, before permitting that lock to be deleted from the DBMS_LOCK_ALLOCATED table. The default waiting period is 10 days. You should not delete locks from this table. Subsequent calls to ALLOCATE_UNIQUE may delete expired locks to recover space.

Usage Notes

If you choose to identify locks by name, you can use ALLOCATE_UNIQUE to generate a unique lock identification number for these named locks.

The first session to call ALLOCATE_UNIQUE with a new lock name causes a unique lock ID to be generated and stored in the dbms_lock_allocated table. Subsequent calls (usually by other sessions) return the lock ID previously generated.

A lock name is associated with the returned lock ID for at least expiration_secs (defaults to 10 days) past the last call to ALLOCATE_UNIQUE with the given lock name. After this time, the row in the dbms_lock_allocated table for this lock name may be deleted in order to recover space. ALLOCATE_UNIQUE performs a commit.

Note: Named user locks may be less efficient, because Oracle uses SQL to determine the lock associated with a given name.

Exceptions

`ORA-20000,ORU-10003: Unable to find or insert lock <lockname> into catalog dbms_lock_allocated.`

CONVERT Function

This function converts a lock from one mode to another. CONVERT is an overloaded function that accepts either a user-defined lock identifier, or the lock handle returned by the ALLOCATE_UNIQUE procedure.

Syntax

```
DBMS_LOCK.CONVERT(
   id         IN INTEGER ||
   lockhandle IN VARCHAR2,
   lockmode   IN INTEGER,
   timeout    IN NUMBER DEFAULT MAXWAIT)
  RETURN INTEGER;
```

Parameters

Table 53–5 CONVERT Function Parameters

Parameter	Description
id or lockhandle	User assigned lock identifier, from 0 to 1073741823, or the lock handle, returned by ALLOCATE_UNIQUE, of the lock mode you want to change.
lockmode	New mode that you want to assign to the given lock. For the available modes and their associated integer identifiers, see Constants on page 53-5.
timeout	Number of seconds to continue trying to change the lock mode. If the lock cannot be converted within this time period, then the call returns a value of 1 (timeout).

Return Values

Table 53–6 CONVERT Function Return Values

Return Value	Description
0	Success
1	Timeout
2	Deadlock
3	Parameter error
4	Don't own lock specified by id or lockhandle
5	Illegal lock handle

RELEASE Function

This function explicitly releases a lock previously acquired using the REQUEST function. Locks are automatically released at the end of a session. RELEASE is an overloaded function that accepts either a user-defined lock identifier, or the lock handle returned by the ALLOCATE_UNIQUE procedure.

Syntax

```
DBMS_LOCK.RELEASE (
   id         IN INTEGER)
  RETURN INTEGER;

DBMS_LOCK.RELEASE (
   lockhandle IN VARCHAR2)
  RETURN INTEGER;
```

Parameters

Table 53–7 RELEASE Function Parameter

Parameter	Description
id or lockhandle	User assigned lock identifier, from 0 to 1073741823, or the lock handle, returned by ALLOCATE_UNIQUE, of the lock mode you want to change.

Return Values

Table 53–8 RELEASE Function Return Values

Return Value	Description
0	Success
3	Parameter error
4	Do not own lock specified by id or lockhandle
5	Illegal lock handle

REQUEST Function

This function requests a lock with a given mode. REQUEST is an overloaded function that accepts either a user-defined lock identifier, or the lock handle returned by the ALLOCATE_UNIQUE procedure.

Syntax

```
DBMS_LOCK.REQUEST(
   id                 IN  INTEGER ||
   lockhandle         IN  VARCHAR2,
   lockmode           IN  INTEGER DEFAULT X_MODE,
   timeout            IN  INTEGER DEFAULT MAXWAIT,
   release_on_commit  IN  BOOLEAN DEFAULT FALSE)
  RETURN INTEGER;
```

The current default values, such as X_MODE and MAXWAIT, are defined in the DBMS_LOCK package specification.

Parameters

Table 53–9 REQUEST Function Parameters

Parameter	Description
id or lockhandle	User assigned lock identifier, from 0 to 1073741823, or the lock handle, returned by ALLOCATE_UNIQUE, of the lock mode you want to change.
lockmode	Mode that you are requesting for the lock. For the available modes and their associated integer identifiers, see Constants on page 53-5.
timeout	Number of seconds to continue trying to grant the lock. If the lock cannot be granted within this time period, then the call returns a value of 1 (timeout).
release_on_commit	Set this parameter to TRUE to release the lock on commit or roll-back. Otherwise, the lock is held until it is explicitly released or until the end of the session.

Return Values

Table 53–10 REQUEST Function Return Values

Return Value	Description
0	Success
1	Timeout
2	Deadlock
3	Parameter error
4	Already own lock specified by id or lockhandle
5	Illegal lock handle

SLEEP Procedure

This procedure suspends the session for a given period of time.

Syntax

```
DBMS_LOCK.SLEEP (
   seconds  IN NUMBER);
```

Parameters

Table 53–11 SLEEP Procedure Parameters

Parameter	Description
seconds	Amount of time, in seconds, to suspend the session. The smallest increment can be entered in hundredths of a second; for example, 1.95 is a legal time value.

54

DBMS_LOGMNR

The DBMS_LOGMNR package, one of a set of LogMiner packages, contains the subprograms you use to initialize the LogMiner tool and to begin and end a LogMiner session.

> **See Also:** *Oracle Database Utilities* for information regarding LogMiner.

This chapter contains the following topics:

Using DBMS_LOGMNR

This section contains the following topics, which relate to using the `DBMS_LOGMNR` package:

Overview

Oracle LogMiner, which is part of Oracle Database, enables you to query online and archived redo log files through a SQL interface. The `DBMS_LOGMNR` package provides the majority of the tools needed to start and stop LogMiner and specify the redo log files of interest.

All changes made to user data or to the database dictionary are recorded in the Oracle redo log files so that database recovery operations can be performed. You can take advantage of the data recorded in the redo log files to accomplish other tasks, such as:

- Pinpointing when a logical corruption to a database, such as errors made at the application level, may have begun
- Determining what actions you would have to take to perform fine-grained recovery at the transaction level.
- Performance tuning and capacity planning through trend analysis.
- Track any data manipulation language (DML) and data definition language (DDL) statements executed on the database, the order in which they were executed, and who executed them.

> **See Also:** Chapter 55, "DBMS_LOGMNR_D" for information on the package subprograms that extract a LogMiner dictionary and re-create LogMiner tables in alternate tablespaces

Security Model

You must have the EXECUTE_CATALOG_ROLE role to use the DBMS_LOGMNR package.

Constants

The DBMS_LOGMNR package defines several enumerated constants for specifying parameter values. Enumerated constants must be prefixed with the package name, for example, DBMS_LOGMNR.NEW.

Table 54–1 describes the constants for the ADD_LOGFILE options flag in the DBMS_LOGMNR package.

Table 54–1 Constants for ADD_LOGFILE Options Flag

Constant	Description
NEW	Implicitly calls the DBMS_LOGMNR.END_LOGMNR procedure to end the current LogMiner session and then creates a new session. The new session starts a new list of redo log files to be analyzed, beginning with the redo log file you specify.
ADDFILE	Adds the specified redo log file to the list of redo log files to be analyzed. Any attempt to add a duplicate file raises an exception (ORA-01289). This is the default if no options flag is specified.

Table 54–2 describes the constants for the START_LOGMNR options flag in the DBMS_LOGMNR package.

Table 54–2 Constants for START_LOGMNR Options Flag

Constant	Description
COMMITTED_DATA_ONLY	If set, DML statements corresponding to committed transactions are returned. DML statements corresponding to a committed transaction are grouped together. Transactions are returned in their commit order. Transactions that are rolled back or in-progress are filtered out, as are internal redo records (those related to index operations, management, and so on). If this option is not set, all rows for all transactions (committed, rolled back, and in-progress) are returned in the order in which they are found in the redo logs (in order of SCN values).
SKIP_CORRUPTION	Directs a select operation on the V$LOGMNR_CONTENTS view to skip any corruptions in the redo log file being analyzed and continue processing. This option works only when a block in the redo log file (and not the header of the redo log file) is corrupt. You should check the INFO column in the V$LOGMNR_CONTENTS view to determine the corrupt blocks skipped by LogMiner. When a corruption in the redo log file is skipped, the OPERATION column contains the value CORRUPTED_BLOCKS, and the STATUS column contains the value 1343.
DDL_DICT_TRACKING	If the LogMiner dictionary in use is a flat file or in the redo log files, LogMiner updates its internal dictionary if a DDL event occurs. This ensures that correct SQL_REDO and SQL_UNDO information is maintained for objects that are modified after the LogMiner internal dictionary is built. The database to which LogMiner is connected must be open. This option cannot be used in conjunction with the DICT_FROM_ONLINE_CATALOG option and cannot be used when the LogMiner dictionary being used is one that was extracted to a flat file prior to Oracle9*i*.

Table 54–2 (Cont.) Constants for START_LOGMNR Options Flag

Constant	Description
DICT_FROM_ONLINE_CATALOG	Directs LogMiner to use the current online database dictionary rather than a LogMiner dictionary contained in a flat file or in the redo log files being analyzed. This option cannot be used in conjunction with the DDL_DICT_TRACKING option. The database to which LogMiner is connected must be the same one that generated the redo log files. Expect to see a value of 2 in the STATUS column of the V$LOGMNR_CONTENTS view if the table definition in the database does not match the table definition in the redo log file.
DICT_FROM_REDO_LOGS	If set, LogMiner expects to find a LogMiner dictionary in the redo log files that were specified. The redo log files are specified with the DBMS_LOGMNR.ADD_LOGFILE procedure or with the DBMS_LOGMNR.START_LOGMNR procedure with the CONTINUOUS_MINE option.
NO_SQL_DELIMITER	If set, the SQL delimiter (a semicolon) is not placed at the end of reconstructed SQL statements. This is helpful for applications that open a cursor and then execute the reconstructed statements.
NO_ROWID_IN_STMT	If set, the ROWID clause is not included in the reconstructed SQL statements. The redo log file may already contain logically unique identifiers for modified rows if supplemental logging is enabled. When using this option, you must be sure that supplemental logging was enabled in the source database at the appropriate level and that no duplicate rows exist in the tables of interest. LogMiner does not make any guarantee regarding the uniqueness of logical row identifiers.
PRINT_PRETTY_SQL	If set, LogMiner formats the reconstructed SQL statements for ease of reading. These reconstructed SQL statements are not executable.
CONTINUOUS_MINE	Directs LogMiner to automatically add redo log files, as needed, to find the data of interest. You only need to specify the first log to start mining, or just the starting SCN or date to indicate to LogMiner where to begin mining logs. You are not required to specify any redo log files explicitly. LogMiner automatically adds and mines the (archived and online) redo log files for the data of interest. This option requires that LogMiner is connected to the same database instance that is generating the redo log files. It also requires that the database be mounted and that archiving be enabled. Beginning with Oracle Database release 10.1, the CONTINUOUS_MINE options is supported for use in an Oracle Real Application Clusters environment.

Views

The DBMS_LOGMNR package uses the views listed in the section on Accessing LogMiner Operational Information in Views in *Oracle Database Utilities*.

Operational Notes

A **LogMiner session** begins with a call to DBMS_LOGMNR.ADD_LOGFILE or DBMS_LOGMNR.START_LOGMNR (the former if you plan to specify log files explicitly; the latter if you plan to use continuous mining). The session ends with a call to DBMS_LOGMNR.END_LOGMNR. Within a LogMiner session, you can specify the redo log files to be analyzed and the SCN or time range of interest; then you can issue SQL SELECT statements against the V$LOGMNR_CONTENTS view to retrieve the data of interest.

Summary of DBMS_LOGMNR Subprograms

Table 54–3 DBMS_LOGMNR Package Subprograms

Subprogram	Description
ADD_LOGFILE Procedure on page 54-10	Adds a redo log file to the existing or newly created list of redo log files for LogMiner to process, so that if a new list is created, this marks the beginning of a LogMiner session
COLUMN_PRESENT Function on page 54-12	Call this function for any row returned from the `V$LOGMNR_CONTENTS` view to determine if undo or redo column values exist for the column specified by the `column_name` input parameter to this function
END_LOGMNR Procedure on page 54-14	Finishes a LogMiner session
MINE_VALUE Function on page 54-15	Call this function for any row returned from the `V$LOGMNR_CONTENTS` view to retrieve the undo or redo column value of the column specified by the `column_name` input parameter to this function
REMOVE_LOGFILE Procedure on page 54-17	Removes a redo log file from the list of redo log files for LogMiner to process
START_LOGMNR Procedure on page 54-18	Initializes the LogMiner utility and starts LogMiner (unless the session was already started with a call to `DBMS_LOGMNR.ADD_LOGFILE`)

ADD_LOGFILE Procedure

This procedure adds a file to an existing or newly created list of log files for LogMiner to process.

Syntax

```
DBMS_LOGMNR.ADD_LOGFILE (
   LogFileName     IN VARCHAR2,
   options         IN BINARY_INTEGER default ADDFILE );
```

Parameters

Table 54–4 ADD_LOGFILE Procedure Parameters

Parameter	Description
LogFileName	Specifies the name of the redo log file to add to the list of redo log files to be analyzed during this session.
options	Does one of the following: ■ Starts a new LogMiner session and a new list of redo log files for analysis (DBMS_LOGMNR.NEW) ■ Adds a file to an existing list of redo log files for analysis (DBMS_LOGMNR.ADDFILE) See Table 54–1, " Constants for ADD_LOGFILE Options Flag".

Exceptions

Table 54–5 ADD_LOGFILE Procedure Exceptions

Exception	Description
ORA-01284	Specified file cannot be opened.
ORA-01287	Specified file is from a different database incarnation.
ORA-01289	Specified file has already been added to the list. Duplicate redo log files cannot be added.
ORA-01290	Specified file is not in the current list and therefore cannot be removed from the list.
ORA-01324	Specified file cannot be added to the list because there is a DB_ID mismatch.

Usage Notes

- Before querying the V$LOGMNR_CONTENTS view, you must make a successful call to the DBMS_LOGMNR.START_LOGMNR procedure (within the current LogMiner session).
- Unless you specify the CONTINUOUS_MINE option, the LogMiner session must be set up with a list of redo log files to be analyzed. Use the ADD_LOGFILE procedure to specify the list of redo log files to analyze.
- If you are not using the CONTINUOUS_MINE option and you want to analyze more than one redo log file, you must call the ADD_LOGFILE procedure separately for each redo log file. The redo log files do not need to be registered in any particular order.

- Both archived and online redo log files can be mined.
- After you have added the first redo log file to the list, each additional redo log file that you add to the list must be associated with the same database and database RESETLOGS SCN as the first redo log file. (The database RESETLOGS SCN uniquely identifies each execution of an ALTER DATABASE OPEN RESETLOGS statement. When the online redo logs are reset, Oracle creates a new and unique incarnation of the database.)
- To analyze the redo log files from a different database (or a database incarnation with a different database RESETLOGS SCN) than that with which the current list of redo log files is associated, use the END_LOGMNR procedure to end the current LogMiner session, and then build a new list using the ADD_LOGFILE procedure.
- LogMiner matches redo log files by the log sequence number. Thus, two redo log files with different names but with the same log sequence number will return the ORA-01289 exception. For instance, the online counterpart of an archived redo log file has a different name from the archived redo log file, but attempting to register it with LogMiner after registering the archived counterpart will result in the ORA-01289 exception being returned.

COLUMN_PRESENT Function

This function is designed to be used in conjunction with the MINE_VALUE function.

If the MINE_VALUE function returns a NULL value, it can mean either:

- The specified column is not present in the redo or undo portion of the data.
- The specified column is present and has a NULL value.

To distinguish between these two cases, use the COLUMN_PRESENT function, which returns a 1 if the column is present in the redo or undo portion of the data. Otherwise, it returns a 0.

Syntax

```
DBMS_LOGMNR.COLUMN_PRESENT (
     sql_redo_undo      IN  RAW,
     column_name        IN  VARCHAR2 default '') RETURN NUMBER;
```

Parameters

Table 54–6 COLUMN_PRESENT Function Parameters

Parameter	Description
sql_redo_undo	Specifies either the REDO_VALUE or the UNDO_VALUE column in the V$LOGMNR_CONTENTS view from which to extract data values. See the Usage Notes for more information.
column_name	Specifies the fully qualified name (schema.table.column) of the column for which this function will return information.

Return Values

Table 54–7 describes the return values for the COLUMN_PRESENT function. The COLUMN_PRESENT function returns 1 if the self-describing record (the first parameter) contains the column specified in the second parameter. This can be used to determine the meaning of NULL values returned by the DBMS_LOGMNR.MINE_VALUE function.

Table 54–7 Return Values for COLUMN_PRESENT Function

Return	Description
0	Specified column is not present in this row of V$LOGMNR_CONTENTS.
1	Column is present in this row of V$LOGMNR_CONTENTS.

Exceptions

Table 54–8 COLUMN_PRESENT Function Exceptions

Exception	Description
ORA-01323	Currently, a LogMiner dictionary is not associated with the LogMiner session. You must specify a LogMiner dictionary for the LogMiner session.
ORA-00904	Value specified for the column_name parameter is not a fully qualified column name.

Usage Notes

- To use the COLUMN_PRESENT function, you must have successfully started LogMiner.
- The COLUMN_PRESENT function must be invoked in the context of a select operation on the V$LOGMNR_CONTENTS view.
- The COLUMN_PRESENT function does not support LONG, LOB, ADT, or COLLECTION datatypes.
- The value for the sql_redo_undo parameter depends on the operation performed and the data of interest:
  - If an update operation was performed and you want to know what the value was prior to the update operation, specify UNDO_VALUE.
  - If an update operation was performed and you want to know what the value is after the update operation, specify REDO_VALUE.
  - If an insert operation was performed, typically you would specify REDO_VALUE (because the value of a column prior to an insert operation will always be NULL).
  - If a delete operation was performed, typically you would specify UNDO_VALUE (because the value of a column after a delete operation will always be NULL).

END_LOGMNR Procedure

This procedure finishes a LogMiner session. Because this procedure performs cleanup operations that may not otherwise be done, you must use it to properly end a LogMiner session. This procedure is called automatically when you log out of a database session or when you call `DBMS_LOGMNR.ADD_LOGFILE` and specify the `NEW` option.

Syntax

```
DBMS_LOGMNR.END_LOGMNR;
```

Exceptions

Table 54–9 END_LOGMNR Procedure Exception

Exception	Description
`ORA-01307`	No LogMiner session is currently active. The `END_LOGMNR` procedure was called without adding any log files or before the `START_LOGMNR` procedure was called

MINE_VALUE Function

This function facilitates queries based on a column's data value. This function takes two arguments. The first one specifies whether to mine the redo (REDO_VALUE) or undo (UNDO_VALUE) portion of the data. The second argument is a string that specifies the fully qualified name of the column to be mined. The MINE_VALUE function always returns a string that can be converted back to the original datatype.

Syntax

```
DBMS_LOGMNR.MINE_VALUE (
     sql_redo_undo      IN  RAW,
     column_name        IN  VARCHAR2 default '') RETURN VARCHAR2;
```

Parameters

Table 54–10 MINE_VALUE Function Parameters

Parameter	Description
sql_redo_undo	Specifies either the REDO_VALUE or the UNDO_VALUE column in the V$LOGMNR_CONTENTS view from which to extract data values. See the Usage Notes for more information.
column_name	Specifies the fully qualified name (schema.table.column) of the column for which this function will return information.

Return Values

Table 54–11 Return Values for MINE_VALUE Function

Return	Description
NULL	The column is not contained within the self-describing record, or the column value is NULL. To distinguish between the two different null possibilities, use the DBMS_LOGMNR.COLUMN_PRESENT function.
NON-NULL	The column is contained within the self-describing record; the value is returned in string format.

Exceptions

Table 54–12 MINE_VALUE Function Exceptions

Exception	Description
ORA-01323	Invalid state. Currently, a LogMiner dictionary is not associated with the LogMiner session. You must specify a LogMiner dictionary for the LogMiner session.
ORA-00904	Invalid identifier. The value specified for the column_name parameter was not a fully qualified column name.

Usage Notes

- To use the MINE_VALUE function, you must have successfully started LogMiner.
- The MINE_VALUE function must be invoked in the context of a select operation from the V$LOGMNR_CONTENTS view.

- The MINE_VALUE function does not support LONG, LOB, ADT, or COLLECTION datatypes.
- The value for the sql_redo_undo parameter depends on the operation performed and the data of interest:
  - If an update operation was performed and you want to know what the value was prior to the update operation, specify UNDO_VALUE.
  - If an update operation was performed and you want to know what the value is after the update operation, specify REDO_VALUE.
  - If an insert operation was performed, typically you would specify REDO_VALUE (because the value of a column prior to an insert operation will always be null).
  - If a delete operation was performed, typically you would specify UNDO_VALUE (because the value of a column after a delete operation will always be null).

REMOVE_LOGFILE Procedure

This procedure removes a redo log file from an existing list of redo log files for LogMiner to process.

> **Note:** This procedure replaces the REMOVEFILE constant that was an option on the ADD_LOGFILE procedure prior to Oracle Database 10*g*.

Syntax

```
DBMS_LOGMNR.REMOVE_LOGFILE (
   LogFileName     IN VARCHAR2);
```

Parameters

Table 54–13 REMOVE_LOGFILE Procedure Parameters

Parameter	Description
LogFileName	Specifies the name of the redo log file to be removed from the list of redo log files to be analyzed during this session.

Exceptions

Table 54–14 REMOVE_LOGFILE Procedure Exception

Exception	Description
ORA-01290	Cannot remove unlisted log file

Usage Notes

- Before querying the V$LOGMNR_CONTENTS view, you must make a successful call to the DBMS_LOGMNR.START_LOGMNR procedure (within the current LogMiner session).
- You can use this procedure to remove a redo log file from the list of redo log files for LogMiner to process if you know that redo log file does not contain any data of interest.
- Multiple redo log files can be removed by calling this procedure repeatedly.
- The redo log files do not need to be removed in any particular order.
- To start a new list of redo log files for analysis, use the END_LOGMNR procedure to end the current LogMiner session, and then build a new list using the ADD_LOGFILE procedure.
- Even if you remove all redo log files from the list, any subsequent calls you make to the ADD_LOGFILE procedure must match the database ID and RESETLOGS SCN of the removed redo log files. Therefore, to analyze the redo log files from a different database (or a database incarnation with a different database RESETLOGS SCN) than that with which the current list of redo log files is associated, use the END_LOGMNR procedure to end the current LogMiner session, and then build a new list using the ADD_LOGFILE procedure.

START_LOGMNR Procedure

This procedure starts LogMiner by loading the dictionary that LogMiner will use to translate internal schema object identifiers to names.

Syntax

```
DBMS_LOGMNR.START_LOGMNR (
   startScn           IN NUMBER default 0,
   endScn             IN NUMBER default 0,
   startTime          IN DATE default '01-jan-1988',
   endTime            IN DATE default '31-dec-2110',
   DictFileName       IN VARCHAR2 default '',
   Options            IN BINARY_INTEGER default 0 );
```

Parameters

Table 54–15 START_LOGMNR Procedure Parameters

Parameter	Description
`startScn`	Directs LogMiner to return only redo records with an SCN greater than or equal to the `startScn` specified. This fails if there is no redo log file containing the specified `startScn` value. (You can query the `FILENAME`, `LOW_SCN`, and `NEXT_SCN` columns in the `V$LOGMNR_LOGS` view for each redo log file to determine the range of SCN values contained in each redo log file.)
`endScn`	Directs LogMiner to return only redo records with an SCN less than or equal to the `endScn` specified. If you specify an `endScn` value that is beyond the value in any redo log file, then LogMiner will use the greatest endScn value in the redo log file that contains the most recent changes. (You can query the `FILENAME`, `LOW_SCN`, and `NEXT_SCN` columns in the `V$LOGMNR_LOGS` view for each redo log file to determine the range of SCN values contained in each redo log file.)
`startTime`	Directs LogMiner to return only redo records with a timestamp greater than or equal to the `startTime` specified. This fails if there is no redo log file containing the specified `startTime` value. (You can query the `FILENAME`, `LOW_TIME`, and `HIGH_TIME` columns in the `V$LOGMNR_LOGS` view for each redo log file to determine the range of time covered in each redo log file.) This parameter is ignored if `startScn` is specified. See the Usage Notes for additional information.
`endTime`	Directs LogMiner to return only redo records with a timestamp less than or equal to the `endTime` specified. If you specify an `endTime` value that is beyond the value in any redo log file, then LogMiner will use the greatest `endTime` in the redo log file that contains the most recent changes. You can query the `FILENAME`, `LOW_TIME`, and `HIGH_TIME` columns in the V$LOGMNR_LOGS view for each redo log file to determine the range of time covered in each redo log file.) This parameter is ignored if `endScn` is specified. See the Usage Notes for additional information.

Table 54–15 (Cont.) START_LOGMNR Procedure Parameters

Parameter	Description
DictFileName	Specifies the flat file that contains the LogMiner dictionary. It is used to reconstruct SQL_REDO and SQL_UNDO columns in V$LOGMNR_CONTENTS, as well as to fully translate SEG_NAME, SEG_OWNER, SEG_TYPE_NAME, TABLE_NAME, and TABLE_SPACE columns. The fully qualified path name for the LogMiner dictionary file must be specified. (This file must have been created previously through the DBMS_LOGMNR_D.BUILD procedure.) You need to specify this parameter only if neither DICT_FROM_REDO_LOGS nor DICT_FROM_ONLINE_CATALOG is specified.
options	See Table 54–2, " Constants for START_LOGMNR Options Flag".

Exceptions

Table 54–16 START_LOGMNR Procedure Exceptions

Exception	Description
ORA-01280	Internal error encountered.
ORA-01281	startScn or endScn parameter value is not a valid SCN, or endScn is less than startScn.
ORA-01282	value for the startTime parameter was greater than the value specified for the endTime parameter, or there was no redo log file that was compatible with the date range specified with the startTime and endTime parameters.
ORA-01283	Options parameter specified is invalid.
ORA-01284	LogMiner dictionary file specified in the DictFileName parameter has a full path length greater than 256 characters, or the file cannot be opened.
ORA-01285	Error reading specified file.
ORA-01291	Redo log files that are needed to satisfy the user's requested SCN or time range are missing.
ORA-01292	No log file has been specified for the current LogMiner session.
ORA-01293	Mounted database required for specified LogMiner options.
ORA-01294	Error occurred while processing information in the specified dictionary file, possible corruption.
ORA-01295	Specified LogMiner dictionary does not correspond to the database that produced the log files being analyzed.
ORA-01296	Character set mismatch between specified LogMiner dictionary and log files.
ORA-01297	Redo version mismatch between LogMiner dictionary and log files.
ORA-01299	Specified LogMiner dictionary corresponds to a different database incarnation.
ORA-01300	Writable database required for specified LogMiner options.

Usage Notes

- LogMiner can use a dictionary that you previously extracted to the redo log files or to a flat file, or you can specify that LogMiner use the online catalog if LogMiner is mining data from the source system. See *Oracle Database Utilities* and

Chapter 55, "DBMS_LOGMNR_D" in this manual for more information about the LogMiner dictionary.

- After executing the START_LOGMNR procedure, you can query the following views:
  - V$LOGMNR_CONTENTS - contains history of information in redo log files
  - V$LOGMNR_DICTIONARY - contains current information about the LogMiner dictionary file extracted to a flat file
  - V$LOGMNR_PARAMETERS - contains information about the LogMiner session

  (You can query the V$LOGMNR_LOGS view after a redo log file list has been added to the list of files that LogMiner is to mine.)
- Parameters and options are not persistent across calls to DBMS_LOGMNR.START_LOGMNR. You must specify all desired parameters and options (including SCN and time ranges) each time you call DBMS_LOGMNR.START_LOGMNR
- Be aware that specifying redo log files using a timestamp is not precise.
- The CONTINUOUS_MINE option directs LogMiner to automatically add redo log files, as needed, to find the data of interest. You need to specify only the first log to start mining, or just the starting SCN or date to indicate to LogMiner where to begin mining logs. Keep the following in mind when using the CONTINUOUS_MINE option:
  - The database control file will hold information about a limited number of archived redo log files, although the number of entries can be quite large. Query the V$ARCHIVED_LOGS view to determine which redo log file entries will be found by LogMiner.

    Even if an entry is listed in the database control file (and the V$ARCHIVED_LOGS view), the archived redo log file may not be accessible by LogMiner for various reasons. For example, the archived redo log file may have been deleted or moved from its location (maybe because of a backup operation to tape), or the directory where it resides may not be not available.
  - If you specify the CONTINUOUS_MINE option and an ending time or SCN that will occur in the future (or you do not specify an end time or SCN), a query of the V$LOGMNR_CONTENTS view will not finish until the database has generated redo log files beyond the specified time or SCN. In this scenario, LogMiner will automatically add archived redo log files to the LogMiner redo log file list as they are generated. In addition, in this scenario only, LogMiner may automatically remove redo log files from the list to keep it at 50 processed redo files. This is to save PGA memory as LogMiner automatically adds redo log files to the list. If LogMiner did not perform automated removal, memory could eventually be exhausted.
  - LogMiner can mine online redo logs. However, if the CONTINUOUS_MINE option is not specified, it is possible that the database is writing to the online redo log file at the same time that LogMiner is reading the online redo log file. If a log switch occurs while LogMiner is reading an online redo log file, the database will overwrite what LogMiner is attempting to read. The data that LogMiner returns if the file it is trying to read gets overwritten by the database is unpredictable.
- Keep the following in mind regarding starting and ending times or SCN ranges:

- If you specify neither a startTime nor a startScn parameter, LogMiner will set the startScn parameter to use the lowest SCN value from the redo log file that contains the oldest changes.
- If you specify both time and SCN values, LogMiner uses the SCN value or values and ignores the time values.
- If you specify starting and ending time or SCN values and they are found in the LogMiner redo log file list, then LogMiner mines the logs indicated by those values.
- If you specify starting and ending times or SCN values that are not in the LogMiner redo log file list, and you specify DBMS_LOGMNR.START_LOGMNR without the CONTINUOUS_MINE option, and you specify:
  * 0 for the startTime or startScn value, then the lowest SCN in the LogMiner redo log file list will be used as the startScn
  * A nonzero number for the startTime or startScn value, then an error is returned
  * 0 or a nonzero number for the endTime or endScn value, then the highest SCN in the LogMiner redo log file list will be used as the endScn
- If you specify starting and ending times or SCN values and they are not found in the LogMiner redo log file list, and you specify DBMS_LOGMNR.START_LOGMNR with the CONTINUOUS_MINE option, and you specify:
  * 0 for the startTime or startScn value, then an error is returned.
  * A startTime or startScn value that is greater than any value in the database's archived redo log files, then LogMiner starts mining in the online redo log file. LogMiner will continue to process the online redo log file until it finds a change at, or beyond, the requested starting point before it returns rows from the V$LOGMNR_CONTENTS view.
  * An endTime or endScn parameter value that indicates a time or SCN in the future, then LogMiner includes the online redo log files when it mines. When you query the V$LOGMNR_CONTENTS view, rows will be returned from this view as changes are made to the database, and will not stop until LogMiner sees a change beyond the requested ending point.
  * 0 for the endTime or endScn parameter value, then LogMiner includes the online redo log files when it mines. When you query the V$LOGMNR_CONTENTS view, rows will be returned from this view as changes are made to the database, and will not stop until you enter CTL+C or you terminate the PL/SQL cursor.

55

DBMS_LOGMNR_D

The DBMS_LOGMNR_D package, one of a set of LogMiner packages, contains two subprograms:

- The BUILD procedure extracts the LogMiner data dictionary to either the redo log files or to a flat file. This information is saved in preparation for future analysis of redo log files using the LogMiner tool.
- The SET_TABLESPACE procedure re-creates all LogMiner tables in an alternate tablespace.

  The **LogMiner data dictionary** consists of the memory data structures and the database tables that are used to store and retrieve information about objects and their versions. It is referred to as the **LogMiner dictionary** throughout the LogMiner documentation.

  See Also: *Oracle Database Utilities* for information regarding LogMiner.

This chapter contains the following topics:

Using DBMS_LOGMNR_D

This section contains the following topics, which relate to using the `DBMS_LOGMNR_D` package:

Overview

LogMiner requires a dictionary to translate object IDs into object names when it returns redo data to you. LogMiner gives you three options for supplying the dictionary:

- Using the online catalog
- Extracting a LogMiner dictionary to the redo log files
- Extracting a LogMiner dictionary to a flat file

Use the `BUILD` procedure to extract the LogMiner dictionary to the redo log files or a flat file. If you want to specify the online catalog as the dictionary source, you do so when you start LogMiner with the `DBMS_LOGMNR.START_LOGMNR` package.

Use the `SET_TABLESPACE` procedure if you want LogMiner tables to use a tablespace other than the default `SYSAUX` tablespace.

> **See Also:** DBMS_LOGMNR for information on the package subprograms used in running a LogMiner session.

Security Model

You must have the EXECUTE_CATALOG_ROLE role to use the DBMS_LOGMNR_D package.

Summary of DBMS_LOGMNR_D Subprograms

Table 55–1 DBMS_LOGMNR_D Package Subprograms

Subprogram	Description
BUILD Procedure on page 55-6	Extracts the LogMiner dictionary to either a flat file or one or more redo log files
SET_TABLESPACE Procedure on page 55-9	Re-creates all LogMiner tables in an alternate tablespace

BUILD Procedure

This procedure extracts the LogMiner data dictionary to either the redo log files or to a flat file.

Syntax

```
DBMS_LOGMNR_D.BUILD (
     dictionary_filename  IN  VARCHAR2,
     dictionary_location  IN  VARCHAR2,
     options              IN  NUMBER);
```

Parameters

Table 55–2 BUILD Procedure Parameters

Parameter	Description
dictionary_filename	Specifies the name of the LogMiner dictionary file.
dictionary_location	Specifies the path to the LogMiner dictionary file directory.
options	Specifies that the LogMiner dictionary is written to either a flat file (STORE_IN_FLAT_FILE) or the redo log files (STORE_IN_REDO_LOGS).

Exceptions

Table 55–3 BUILD Procedure Exceptions

Exception	Description
ora-01302	Dictionary build options are missing or incorrect. This error is returned under the following conditions: ■ If the value of the OPTIONS parameter is not one of the supported values (STORE_IN_REDO_LOGS, STORE_IN_FLAT_FILE) or is not specified ■ If the STORE_IN_REDO_LOGS option is not specified and neither the dictionary_filename nor the dictionary_location parameter is specified ■ If the STORE_IN_REDO_LOGS option is specified and either the dictionary_filename or the dictionary_location parameter is specified
ora-01308	Initialization parameter UTL_FILE_DIR is not set.
ora-01336	Specified dictionary file cannot be opened. This error is returned under the following conditions: ■ The specified value for the dictionary_location does not exist. ■ The UTL_FILE_DIR initialization parameter is not set to have access to the dictionary_location ■ The dictionary file is read-only.

Usage Notes

- To extract the LogMiner dictionary to a flat file, you must supply a filename and location.

To extract the LogMiner dictionary to the redo log files, specify only the STORE_IN_REDO_LOGS option. The size of the LogMiner dictionary may cause it to be contained in multiple redo log files.

The combinations of parameters used result in the following behavior:

- If you do not specify any parameters, an error is returned.
- If you specify a filename and location, without any options, the LogMiner dictionary is extracted to a flat file with that name.
- If you specify a filename and location, as well as the STORE_IN_FLAT_FILE option, the LogMiner dictionary is extracted to a flat file with the specified name.
- If you do not specify a filename and location, but do specify the STORE_IN_REDO_LOGS option, the LogMiner dictionary is extracted to the redo log files.
- If you specify a filename and location, as well as the STORE_IN_REDO_LOGS option, an error is returned.
- If you do not specify a filename and location, but do specify the STORE_IN_FLAT_FILE option, an error is returned.

- Ideally, the LogMiner dictionary file will be created after all database dictionary changes have been made and prior to the creation of any redo log files that are to be analyzed. As of Oracle9*i* release 1 (9.0.1), you can use LogMiner to dump the LogMiner dictionary to the redo log files or a flat file, perform DDL operations, and dynamically apply the DDL changes to the LogMiner dictionary.
- Do not run the DBMS_LOGMNR_D.BUILD procedure if there are any ongoing DDL operations.
- The database must be open when you run the DBMS_LOGMNR_D.BUILD procedure.
- When extracting a LogMiner dictionary to a flat file, the procedure queries the dictionary tables of the current database and creates a text-based file containing the contents of the tables. To extract a LogMiner dictionary to a flat file, the following conditions must be met:
  - You must specify a directory for use by the PL/SQL procedure. To do so, set the initialization parameter UTL_FILE_DIR in the initialization parameter file. For example:

    ```
    UTL_FILE_DIR = /oracle/dictionary
    ```

    After setting the parameter, you must shut down and restart the database for this parameter to take effect. If you do not set this parameter, the procedure will fail.
  - You must ensure that no DDL operations occur while the LogMiner dictionary build is running. Otherwise, the LogMiner dictionary file may not contain a consistent snapshot of the database dictionary.

  Be aware that the DDL_DICT_TRACKING option to the DBMS_LOGMNR.START_LOGMNR procedure is not supported for flat file dictionaries created prior to Oracle9*i*. If you attempt to use the DDL_DICT_TRACKING option with a LogMiner database extracted to a flat file prior to Oracle9*i*, the ORA-01330 error (problem loading a required build table) is returned.
- To extract a LogMiner dictionary file to the redo log files, the following conditions must be met:

- The DBMS_LOGMNR_D.BUILD procedure must be run on a system that is running Oracle9*i* or later.
- Archivelog mode must be enabled in order to generate usable redo log files.
- The COMPATIBLE parameter in the initialization parameter file must be set to 9.2.0 or higher.
- The database to which LogMiner is attached must be Oracle9*i* or later.

In addition, supplemental logging (at least the minimum level) should be enabled to ensure that you can take advantage of all the features that LogMiner offers. See *Oracle Database Utilities* for information about using supplemental logging with LogMiner.

Examples

Example 1: Extracting the LogMiner Dictionary to a Flat File

The following example extracts the LogMiner dictionary file to a flat file named dictionary.ora in a specified path (/oracle/database).

```
SQL> EXECUTE dbms_logmnr_d.build('dictionary.ora', -
     '/oracle/database/', -
     options => dbms_logmnr_d.store_in_flat_file);
```

Example 2: Extracting the LogMiner Dictionary to the Redo Log Files

The following example extracts the LogMiner dictionary to the redo log files.

```
SQL> EXECUTE dbms_logmnr_d.build( -
     options => dbms_logmnr_d.store_in_redo_logs);
```

SET_TABLESPACE Procedure

By default, all LogMiner tables are created to use the SYSAUX tablespace. However, it may be desirable to have LogMiner tables use an alternate tablespace. Use this procedure to move LogMiner tables to an alternate tablespace.

Syntax

```
DBMS_LOGMNR_D.SET_TABLESPACE (
     new_tablespace        IN VARCHAR2);
```

Parameters

Table 55–4 SET_TABLESPACE Parameter

Parameter	Description
new_tablespace	A string naming a preexisting tablespace. To move all LogMiner tables to employ this tablespace, supply this parameter.

Usage Notes

- Users upgrading from earlier versions of Oracle Database may find LogMiner tables in the SYSTEM tablespace. Oracle encourages such users to consider using the SET_TABLESPACE procedure to move the tables to the SYSAUX tablespace once they are confident that they will not be downgrading to an earlier version of Oracle Database.
- Users of this routine must supply an existing tablespace.

  See Also: *Oracle Database Concepts* and *Oracle Database SQL Reference* for information about tablespaces and how to create them

Example: Using the DBMS_LOGMNR_D.SET_TABLESPACE Procedure

The following example shows the creation of an alternate tablespace and execution of the DBMS_LOGMNR_D.SET_TABLESPACE procedure.

```
SQL> CREATE TABLESPACE  logmnrts$ datafile '/usr/oracle/dbs/logmnrts.f'
     SIZE 25 M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;

SQL> EXECUTE dbms_logmnr_d.set_tablespace('logmnrts$');
```

56

DBMS_LOGSTDBY

The DBMS_LOGSTDBY package provides subprograms for configuring and managing the logical standby database environment.

> **See Also:** *Oracle Data Guard Concepts and Administration* for more information about SQL Apply and logical standby databases

This chapter contains the following topics:

Using DBMS_LOGSTDBY

This section contains topics which relate to using the `DBMS_LOGSTDBY` package.

Overview

The DBMS_LOGSTDBY package helps you manage the SQL Apply (logical standby database) environment. The subprograms in the DBMS_LOGSTDBY package help you to accomplish the following main objectives:

- Manage configuration parameters used by SQL Apply.

  For example, controlling how transactions are applied on the logical standby database, how much shared pool is used, and how many processes are used by SQL Apply to mine and apply the changes.
- Ensure an appropriate level of supplemental logging is enabled, and a LogMiner dictionary is built correctly for logical standby database creation.
- Provide a way to skip the application of changes to selected tables or entire schemas in the logical standby database, and specify ways to handle exceptions encountered by SQL Apply.
- Allow controlled access to tables in the logical standby database that may require maintenance.

Operational Notes

Case Sensitivity

Ensure you use the correct case when supplying schema and table names to the DBMS_LOGSTDBY package. For example, the following statements show incorrect and correct syntax for a SKIP procedure that skips changes to OE.TEST.

Incorrect statement:

```
EXECUTE DBMS_LOGSTDBY.SKIP (stmt => 'DML', schema_name => 'oe', -
   object_name => 'test', proc_name => null);
```

Because the names are specified with lowercase characters, the transactions that update these columns will still be applied to the logical standby database.

Correct statement:

```
EXECUTE DBMS_LOGSTDBY.SKIP (stmt => 'DML', schema_name => 'OE', -
   object_name => 'TEST', proc_name => null);
```

Privileges and Security

A prototype role, LOGSTDBY_ADMINISTRATOR, is created by default with RESOURCE, and EXECUTE on DBMS_LOGSTDBY privileges. If you choose to use this role, consider granting ALTER DATABASE and ALTER SESSION privileges to the role so that the grantee can start and stop SQL Apply and can enable and disable the database guard. Oracle recommends using an account with DBA privileges to perform administration tasks on logical standby databases.

The six procedures associated with skipping transactions (SKIP and UNSKIP, SKIP_ERROR and UNSKIP_ERROR, and SKIP_TRANSACTION and UNSKIP_TRANSACTION) all require DBA privileges to execute because their scope may contain wildcard schemas. Oracle recommends that where SKIP procedures are specified, these be owned by a secure account with appropriate privileges on the schemas they act on (for example, SYS).

Deprecated Subprograms

The `transaction_consistency` parameter of the APPLY_SET Procedure is being deprecated with this release of the Oracle Database. The `transaction_consistency` parameter is being replaced by the `preserve_commit_order` parameter.

Summary of DBMS_LOGSTDBY Subprograms

Table 56–1 DBMS_LOGSTDBY Package Subprograms

Subprogram	Description
APPLY_SET Procedure on page 56-7	Sets the values of various parameters that configure and maintain SQL Apply
APPLY_UNSET Procedure on page 56-9	Restores the default values of various parameters that configure and maintain SQL Apply
BUILD Procedure on page 56-10	Ensures supplemental logging is enabled properly and builds the LogMiner dictionary
INSTANTIATE_TABLE Procedure on page 56-11	Creates and populates a table in the standby database from a corresponding table in the primary database
PREPARE_FOR_NEW_PRIMARY Procedure on page 56-12	Used after a failover, this procedure ensures a local logical standby database that was not involved in the failover has not processed more redo than the new primary database and reports the set of archive redo log files that must be replaced to ensure consistency
PURGE_SESSION Procedure on page 56-14	Identifies the archived redo log files that have been applied to the logical standby database and are no longer needed by SQL Apply
REBUILD Procedure on page 56-15	Records relevant metadata (including the LogMiner Multiversioned Data Dictionary) in the redo stream in case a database that has recently changed its role to a primary database following a failover operation fails to do so during the failover process
SET_TABLESPACE Procedure on page 56-16	Moves metadata tables required by SQL Apply to the user-specified tablespace. By default, the metadata tables are created in the `SYSAUX` tablespace.
SKIP Procedure on page 56-17	Specifies rules that control database operations that should not be applied to the logical standby database
SKIP_ERROR Procedure on page 56-24	Specifies rules regarding what action to take upon encountering errors
SKIP_TRANSACTION Procedure on page 56-28	Specifies transactions that should not be applied on the logical standby database. Be careful in using this procedure, because not applying specific transactions may cause data corruption at the logical standby database.
UNSKIP Procedure on page 56-30	Deletes rules specified by the `SKIP` procedure
UNSKIP_ERROR Procedure on page 56-32	Deletes rules specified by the `SKIP_ERROR` procedure
UNSKIP_TRANSACTION Procedure on page 56-34	Deletes rules specified by the `SKIP_TRANSACTION` procedure

APPLY_SET Procedure

Use this procedure to set values of parameters that configure and manage SQL Apply in a logical standby database environment. SQL Apply cannot be running when you use this procedure.

Syntax

```
DBMS_LOGSTDBY.APPLY_SET (
     parameter          IN VARCHAR,
     value              IN VARCHAR);
```

Parameters

Table 56–2 APPLY_SET Procedure Parameters

Parameter	Description
LOG_AUTO_DELETE	Automatically deletes archived redo log files once they have been applied on the logical standby database. Set to TRUE to enable automatic deletion of archived redo log files, and FALSE to disable automatic deletion.The default value is TRUE.
MAX_SGA	Number of megabytes from shared pool in System Global Area (SGA) that SQL Apply will use. The default value is 30 megabytes or one quarter of the value set for SHARED_POOL_SIZE, whichever is lower.
MAX_SERVERS	Number of parallel query servers that SQL Apply uses to read and apply redo. It defaults to the value of the PARALLEL_MAX_SERVERS initialization parameter or 9, whichever is lower.
MAX_EVENTS_RECORDED	Number of recent events that will be visible through the DBA_LOGSTDBY_EVENTS view. To record all events encountered by SQL Apply, use the DBMS_LOGSTDBY.MAX_EVENTS constant as the number value.
PRESERVE_COMMIT_ORDER	TRUE: Transaction are applied to the logical standby database in the exact order in which they were committed on the primary database. This is the default parameter setting. FALSE: Transactions are applied out of order from how they were committed on the primary database, and no attempt is made to provide read-consistent results. Regardless of the level chosen, modifications done to the same row are always applied in the same order as they happened in the primary database. See the Usage Notes for details and recommendations.
RECORD_SKIP_ERRORS	Controls whether skipped errors (as described by the SKIP_ERROR procedure) are recorded in the DBA_LOGSTDBY_EVENTS table and the alert log. Specify one of the following values: TRUE: Skipped errors are recorded in the DBA_LOGSTDBY_EVENTS table and the alert log. This is the default parameter setting. FALSE: Skipped errors are not recorded in the DBA_LOGSTDBY_EVENTS table and the alert log.

Table 56–2 (Cont.) APPLY_SET Procedure Parameters

Parameter	Description
RECORD_SKIP_DDL	Controls whether skipped DDL statements are recorded in the DBA_LOGSTDBY_EVENTS table and the alert log. Specify one of the following values: TRUE: Skipped DDL statements are recorded in the DBA_LOGSTDBY_EVENTS table and the alert log. This is the default parameter setting. FALSE: Skipped DDL statements are not recorded in the DBA_LOGSTDBY_EVENTS table and the alert log.
RECORD_APPLIED_DDL	Controls whether DDL statements that have been applied to the logical standby database are recorded in the DBA_LOGSTDBY_EVENTS table and the alert log. Specify one of the following values: TRUE: Indicates that DDL statements applied to the logical standby database are recorded in the DBA_LOGSTDBY_EVENTS table and the alert log. FALSE: Indicates that applied DDL statements are not recorded. This is the default parameter setting.
APPLY_SERVERS	Controls the number of APPLIER processes (parallel execution servers) used to apply changes
PREPARE_SERVERS	Controls the number of PREPARER processes (parallel execution servers) used to prepare changes

Exceptions

Table 56–3 APPLY_SET Procedure Exceptions

Exception	Description
ORA-16103	Logical Standby apply must be stopped to allow this operation
ORA-16104	invalid Logical Standby option requested
ORA-16236	Logical Standby metadata operation in progress

Usage Notes

- SQL Apply must be stopped before calling the APPLY_SET procedure.
- Use the APPLY_UNSET procedure to restore the default settings of a parameter.
- See *Oracle Data Guard Concepts and Administration* for help with tuning SQL Apply and for information about setting appropriate values for different parameters.

Examples

To record DDLs in the DBA_LOGSTDBY_EVENTS view and in the alert log, issue the following statement:

```
SQL> EXECUTE DBMS_LOGSTDBY.APPLY_SET('RECORD_APPLIED_DDL', TRUE);
```

APPLY_UNSET Procedure

Use the APPLY_UNSET procedure to restore the default values of the parameters that you changed with the APPLY_SET procedure.

Syntax

```
DBMS_LOGSTDBY.APPLY_UNSET (
     parameter          IN VARCHAR);
```

Parameters

The parameter information for the APPLY_UNSET procedure is the same as that described for the APPLY_SET procedure. See Table 56–2 for complete parameter information.

Exceptions

Table 56–4 APPLY_UNSET Procedure Exceptions

Exception	Description
ORA-16103	Logical Standby apply must be stopped to allow this operation
ORA-16104	invalid Logical Standby option requested
ORA-16236	Logical Standby metadata operation in progress

Usage Notes

- SQL Apply must be stopped before calling the APPLY_UNSET procedure.
- Use the APPLY_SET procedure to specify a nondefault value for a parameter.

Examples

If you previously specified that applied DDLs show up in the DBA_LOGSTDBY_EVENTS view and the alert log, you can restore the default behavior of SQL Apply regarding applied DDL statements with the following statement:

```
SQL> EXECUTE DBMS_LOGSTDBY.APPLY_UNSET('RECORD_APPLIED_DDL');
```

BUILD Procedure

Use this procedure on the primary database to record relevant metadata (LogMiner dictionary) information in the redo log, which will subsequently be used by SQL Apply. This procedure will enable database-wide primary- and unique-key supplemental logging, if necessary.

Syntax

```
DBMS_LOGSTDBY.BUILD;
```

Usage Notes

- Supplemental log information includes extra information in the redo logs that uniquely identifies a modified row in the logical standby database, and also includes information that helps efficient application of changes to the logical standby database.
- LogMiner dictionary information allows SQL Apply to interpret data in the redo logs.
- `DBMS_LOGSTDBY.BUILD` should be run only once for each logical standby database you want to create. You do not need to use DBMS_LOGSTDBY.BUILD for each RAC instance.

Examples

To build the LogMiner dictionary in the redo stream of the primary database and to record additional information so that a logical standby database can be instantiated, issue the following SQL statement at the primary database

```
SQL> EXECUTE DBMS_LOGSTDBY.BUILD;
```

INSTANTIATE_TABLE Procedure

This procedure creates and populates a table in the standby database from a corresponding table in the primary database. The table requires the name of the database link (`dblink`) as an input parameter. If the table already exists in the logical standby database, it will be dropped and re-created based on the table definition at the primary database. This procedure only brings over the data associated with the table, and not the associated indexes and constraints.

Use the `INSTANTIATE_TABLE` procedure to:

- Add a table to a standby database.
- Re-create a table in a standby database.

Syntax

```
DBMS_LOGSTDBY.INSTANTIATE_TABLE (
     schema_name         IN VARCHAR2,
     table_name          IN VARCHAR2,
     dblink              IN VARCHAR2);
```

Parameters

Table 56–5 INSTANTIATE_TABLE Procedure Parameters

Parameter	Description
`schema_name`	Name of the schema
`table_name`	Name of the table to be created or re-created in the standby database
`dblink`	Name of the database link account that has privileges to read and lock the table in the primary database

Exceptions

Table 56–6 INSTANTIATE_TABLE Procedure Exceptions

Exception	Description
`ORA-16103`	Logical Standby apply must be stopped to allow this operation
`ORA-16236`	Logical Standby metadata operation in progress

Usage Notes

- Use this procedure to create and populate a table in a way that keeps the data on the standby database transactionally consistent with the primary database.
- This table will not be synchronized with the rest of the tables being maintained by SQL Apply and SQL Apply will not start to maintain it until the redo log that was current on the primary database at the time of execution is applied to the standby database.

Examples

```
SQL> EXECUTE DBMS_LOGSTDBY.INSTANTIATE_TABLE (-
     SCHEMA_NAME => 'HR', TABLE_NAME => 'EMPLOYEES', -
     DBLINK => 'INSTANTIATE_TBL_LINK');
```

PREPARE_FOR_NEW_PRIMARY Procedure

The PREPARE_FOR_NEW_PRIMARY procedure must be invoked at a logical standby database following a failover if that standby database was not the target of the failover operation. Such a standby database must process the exact same set of redo logs processed at the new primary database. This routine ensures that the local logical standby database has not processed more redo than the new primary database and reports the set of archive logs that must be replaced to ensure consistency. The set of replacement logs will be reported in the alert.log. These logs must be copied to the logical standby and registered using the ALTER DATABASE REGISTER LOGICAL LOGFILE statement.

Syntax

```
DBMS_LOGSTDBY.PREPARE_FOR_NEW_PRIMARY (
           FORMER_STANDBY_TYPE         IN VARCHAR2,
           DBLINK                      IN VARCHAR2);
```

Parameters

Table 56–7 PREPARE_FOR_NEW_PRIMARY Procedure Parameters

Parameter	Description
FORMER_STANDBY_TYPE	The type of standby database that was the target of the failover operation to become the new primary database. Valid values are 'PHYSICAL' if the new primary was formerly a physical standby, and 'LOGICAL' if the new primary database was formerly a logical standby database.
DBLINK	The name of a database link to the new primary database

Exceptions

Table 56–8 PREPARE_FOR_NEW_PRIMARY Procedure Exceptions

Exception	Description
ORA-16104	Invalid Logical Standby option.
ORA-16109	Failed to apply log data from previous primary.

Usage Notes

- This routine is intended only for logical standby systems.
- This routine will fail if the new primary database was formerly a logical standby database and the LogMiner dictionary build has not completed successfully.
- Log files displayed in the alert log will be referred to as *terminal logs*. Users should keep in mind that file paths are relative to the new primary database and may not resolve locally.
- Upon manual registration of the terminal logs, users should complete the process by calling either START LOGICAL STANDBY APPLY if the new primary database was formerly a physical standby database or START LOGICAL STANDBY APPLY NEW PRIMARY if the new primary database was formerly a logical standby database.
- See the alert log for more details regarding the reasons for any exception.

Examples

```
SQL> EXECUTE DBMS_LOGSTDBY.PREPARE_FOR_NEW_PRIMARY (  -
                FORMER_STANDBY_TYPE => 'LOGICAL',    -
                DBLINK => 'dblink_to_newprimary');
```

PURGE_SESSION Procedure

Identifies all archived redo log files that have been applied to the logical standby database and are no longer needed by SQL Apply. Once identified, you can issue operating system commands to delete some or all of the unnecessary archived redo log files.

Syntax

```
DBMS_LOGSTDBY.PURGE_SESSION;
```

Exceptions

Table 56–9 PURGE_SESSION Procedure Exceptions

Exception	Description
ORA-01309	Invalid session

Usage Notes

- This procedure does not delete the archived redo log files. You must issue operating system commands to delete unneeded files.
- This procedure updates the DBA_LOGMNR_PURGED_LOG view that displays the archived redo log files that have been applied to the logical standby database.
- In Oracle Database 10*g* Release 2, metadata related to the archived redo log files (and the actual archived redo log files) are purged automatically based on the default setting of the LOG_AUTO_DELETE parameter described in the DBMS_LOGSTDBY.APPLY_SET procedure described on page 56-7.

Example

To identify and remove unnecessary files:

1. Enter the following statement on the logical standby database:

```
SQL> EXECUTE DBMS_LOGSTDBY.PURGE_SESSION;
```

2. Query the DBA_LOGMNR_PURGED_LOG view to list the archived redo log files that can be removed:

```
SQL> SELECT * FROM DBA_LOGMNR_PURGED_LOG;

FILE_NAME
   -----------------------------------
   /boston/arc_dest/arc_1_40_509538672.log
   /boston/arc_dest/arc_1_41_509538672.log
   /boston/arc_dest/arc_1_42_509538672.log
   /boston/arc_dest/arc_1_43_509538672.log
   /boston/arc_dest/arc_1_44_509538672.log
   /boston/arc_dest/arc_1_45_509538672.log
   /boston/arc_dest/arc_1_46_509538672.log
   /boston/arc_dest/arc_1_47_509538672.log
```

3. Use operating system-specific commands to delete archived redo log files from the file system.

REBUILD Procedure

This procedure is used if a database that has recently changed its role to a primary database following a failover operation fails to record relevant metadata (including the LogMiner Multiversioned Data Dictionary) in the redo stream required for other logical standby databases.

Syntax

```
DBMS_LOGSTDBY.REBUILD;
```

Usage Notes

- LogMiner Multiversioned Data Dictionary information is logged in the redo log files.
- The standby redo log files (if present) are archived.

Examples

```
SQL> EXECUTE DBMS_LOGSTDBY.REBUILD;
```

SET_TABLESPACE Procedure

Moves metadata tables required by SQL Apply to the user-specified tablespace. By default, the metadata tables are created in the SYSAUX tablespace.

Syntax

```
DBMS_LOGSTDBY.SET_TABLESPACE(
           NEW_TABLESPACE IN VARCHAR2)
```

Parameters

Table 56–10 SET_TABLE SPACE Procedure Parameters

Parameter	Description
NEW_TABLESPACE	Name of the new tablespace where metadata tables will reside.

Exceptions

Table 56–11 SET_TABLESPACE Procedure Exceptions

Exception	Description
ORA-16103	Logical Standby apply must be stopped to allow this operation
ORA-16236	Logical Standby metadata operation in progress

Examples

To move metadata tables to a new tablespace named LOGSTDBY_TBS, issue the following statement:

```
SQL> EXECUTE DBMS_LOGSTDBY.SET_TABLESPACE (new_tablespace => 'LOGSTDBY_TBS');
```

SKIP Procedure

The SKIP procedure can be used to define rules that will be used by SQL Apply to skip the application of certain changes to the logical standby database. For example, the SKIP procedure can be used to skip changes to a subset of tables in the logical standby database. It can also be used to specify DDL statements that should not be applied at the logical standby database or should be modified before they are applied in the logical standby database. One reason why a DDL statement may need to be modified is to accommodate a different directory structure on the logical standby database.

Syntax

```
DBMS_LOGSTDBY.SKIP (
     stmt                    IN VARCHAR2,
     schema_name             IN VARCHAR2 DEFAULT NULL,
     object_name             IN VARCHAR2 DEFAULT NULL,
     proc_name               IN VARCHAR2 DEFAULT NULL,
     use_like                IN BOOLEAN DEFAULT TRUE,
     esc                     IN CHAR1 DEFAULT NULL);
```

Parameters

Table 56–12 SKIP Procedure Parameters

Parameter	Description
stmt	Either a keyword that identifies a set of SQL statements or a specific SQL statement. The use of keywords simplifies configuration since keywords, generally defined by the database object, identify all SQL statements that operate on the specified object. Table 56–13 shows a list of keywords and the equivalent SQL statements, either of which is a valid value for this parameter.
schema_name	The name of one or more schemas (wildcards are permitted) associated with the SQL statements identified by the stmt parameter. If not applicable, this value must be set to NULL.
object_name	The name of one or more objects (wildcards are permitted) associated with the SQL statements identified by the stmt. If not applicable, this value must be set to NULL.

Table 56–12 (Cont.) SKIP Procedure Parameters

Parameter	Description
proc_name	Name of a stored procedure to call when SQL Apply determines that a particular statement matches the filter defined by the stmt, schema_name, and object_name parameters. Specify the procedure in the following format: '"schema"."package"."procedure"' This procedure returns a value that directs SQL Apply to perform one of the following: execute the statement, skip the statement, or execute a replacement statement. SQL Apply calls the stored procedure with the following call signature: ■ IN STATEMENT VARCHAR2 -- The SQL statement that matches the filter ■ IN STATEMENT_TYPE VARCHAR2 -- The stmt of the filter ■ IN SCHEMA VARCHAR2 -- The schema_name of the filter, if applicable ■ IN NAME VARCHAR2 -- The object_name of the filter, if applicable ■ IN XIDUSN NUMBER -- Transaction ID part 1 ■ IN XIDSLT NUMBER -- Transaction ID part 2 ■ IN XIDSQN NUMBER -- Transaction ID part 3 ■ OUT SKIP_ACTION NUMBER -- Action to be taken by SQL Apply upon completion of this routine. Valid values are: SKIP_ACTION_APPLY -- Execute the statement SKIP_ACTION_SKIP -- Skip the statement SKIP_ACTION_REPLACE -- Execute the replacement statement supplied in the NEW_STATEMENT output parameter
use_like	Allows pattern matching to isolate the tables that you want to skip on the logical standby database. The use_like parameter matches a portion of one character value to another by searching the first value for the pattern specified by the second, and calculates strings using characters as defined by the input character set. This parameter follows the same rules for pattern matching described in the *Oracle Database SQL Reference*.
esc	Identifies an escape character (such as the character "/") that you can use for pattern matching. If the escape character appears in the pattern before the character "%" or "_" then Oracle interprets this character literally in the pattern, rather than as a special pattern matching character. See*Oracle Database SQL Reference* for more information about pattern matching.

Usage Notes

- This procedure requires DBA privileges to execute.
- You cannot associate a stored procedure to be invoked in the context of a DML statement. For example, the following statement returns the ORA-16104: invalid Logical Standby option requested error:

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP(-
     stmt => 'DML', -
     schema_name => 'HR', -
     object_name => 'EMPLOYEES', -
     proc_name => 'DML_HANDLER');
```

Also, if an event matches multiple rules either because of the use of wildcards while specifying the rule or because of a specification of overlapping rules. For example, if you specify a rule for the SCHEMA_DDL event for the HR.EMPLOYEES table, and a rule for the ALTER TABLE event for the HR.EMPLOYEES table, only one of the matching procedures will be invoked (alphabetically, by procedure). In the following code example, consider the following rules:

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP( -
     stmt => 'SCHEMA DDL', -
     schema_name => 'HR', -
     object_name => 'EMPLOYEES', -
     proc_name => 'SCHEMA_DDL_HANDLER');
SQL> EXECUTE DBMS_LOGSTDBY.SKIP( -
     stmt => 'ALTER TABLE', -
     schema_name => 'HR', -
     object_name => 'EMPLOYEES', -
     proc_name => 'TABLE_ALTER_HANDLER');
```

On encountering an ALTER TABLE statement, the schema_ddl_handler procedure will be invoked because its name will be at the top of an alphabetically sorted list of procedures that are relevant to the statement.

Collisions on a rule set because of a specification containing wildcard entries are resolved in a similar fashion. For example, the rules in the following example will result in the empddl_handler procedure being invoked upon encountering the ALTER TABLE HR.EMPLOYEES ADD COLUMN RATING NUMBER statement:

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP(-
     stmt => 'ALTER TABLE', -
     schema_name => 'HR', -
     object_name => 'EMP%', -
     proc_name => 'EMPDDL_HANDLER');
SQL> EXECUTE DBMS_LOGSTDBY.SKIP( -
     stmt => 'ALTER TABLE', -
     schema_name => 'HR', -
     object_name => 'EMPLOYEES', -
     proc_name => 'EMPLOYEE_DDL_HANDLER');
```

- Use the SKIP procedure with caution, particularly when skipping DDL statements. If a CREATE TABLE statement is skipped, for example, you must also specify other DDL statements that refer to that table in the SKIP procedure. Otherwise, the statements will fail and cause an exception. When this happens, SQL Apply stops running.
- Before calling the SKIP procedure, SQL Apply must be halted. Do this by issuing an ALTER DATABASE STOP LOGICAL STANDBY APPLY statement. Once all desired filters have been specified, issue an ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE statement to start SQL Apply using the new filter settings.
- See the UNSKIP procedure on page 56-30 for information about reversing (undoing) the settings of the SKIP procedure.
- For USER statements, the SCHEMA_NAME parameter will be the user and specify '%' for the OBJECT_NAME parameter.
- If the PROC_NAME parameter is supplied, it must already exist in DBA_PROCEDURES and it must execute with DEFINER rights. If the procedure is declared with INVOKER rights, the ORA-1031: insufficient privileges message will be returned.

- If the procedure returns a REPLACEMENT statement, the REPLACEMENT statement will be executed using the SYSTEM and OBJECT privileges of the owner of the procedure.
- The PL/SQL block of a SKIP procedure cannot contain transaction control statements (for example, COMMIT, ROLLBACK, SAVEPOINT, and SET CONSTRAINT) unless the block is declared to be an autonomous transaction.

Skip Statement Options

Table 56–13 lists the supported values for the stmt parameter of the SKIP procedure. The left column of the table lists the keywords that may be used to identify the set of SQL statements to the right of the keyword. Any of the SQL statements in the right column, however, are also valid values. Note that keywords are generally defined by database object.

Table 56–13 Supported Values for the stmt Parameter

Keyword	Associated SQL Statements
NON_SCHEMA_DDL	*All DDL that does not pertain to a particular schema* **Note:** SCHEMA_NAME and OBJECT_NAME must be null
SCHEMA_DDL	*All DDL statements that create, modify, or drop schema objects (for example: tables, indexes, and columns)* **Note:** SCHEMA_NAME and OBJECT_NAME must *not* be null
DML	Includes DML statements on a table (for example: INSERT, UPDATE, and DELETE)
CLUSTER	AUDIT CLUSTER CREATE CLUSTER DROP CLUSTER TRUNCATE CLUSTER
CONTEXT	CREATE CONTEXT DROP CONTEXT
DATABASE LINK	CREATE DATABASE LINK CREATE PUBLIC DATABASE LINK DROP DATABASE LINK DROP PUBLIC DATABASE LINK
DIMENSION	ALTER DIMENSION CREATE DIMENSION DROP DIMENSION
DIRECTORY	CREATE DIRECTORY DROP DIRECTORY
INDEX	ALTER INDEX CREATE INDEX DROP INDEX

Table 56–13 (Cont.) Supported Values for the stmt Parameter

Keyword	Associated SQL Statements
PROCEDURE[1]	ALTER FUNCTION ALTER PACKAGE ALTER PACKAGE BODY ALTER PROCEDURE CREATE FUNCTION CREATE LIBRARY CREATE PACKAGE CREATE PACKAGE BODY CREATE PROCEDURE DROP FUNCTION DROP LIBRARY DROP PACKAGE DROP PACKAGE BODY DROP PROCEDURE
PROFILE	ALTER PROFILE CREATE PROFILE DROP PROFILE
ROLE	ALTER ROLE CREATE ROLE DROP ROLE SET ROLE
ROLLBACK STATEMENT	ALTER ROLLBACK SEGMENT CREATE ROLLBACK SEGMENT DROP ROLLBACK SEGMENT
SEQUENCE	ALTER SEQUENCE CREATE SEQUENCE DROP SEQUENCE
SYNONYM	CREATE PUBLIC SYNONYM CREATE SYNONYM DROP PUBLIC SYNONYM DROP SYNONYM
TABLE	ALTER TABLE CREATE TABLE DROP TABLE
TABLESPACE	CREATE TABLESPACE DROP TABLESPACE TRUNCATE TABLESPACE
TRIGGER	ALTER TRIGGER CREATE TRIGGER DISABLE ALL TRIGGERS DISABLE TRIGGER DROP TRIGGER ENABLE ALL TRIGGERS ENABLE TRIGGER
TYPE	ALTER TYPE ALTER TYPE BODY CREATE TYPE CREATE TYPE BODY DROP TYPE DROP TYPE BODY

Table 56–13 (Cont.) Supported Values for the stmt Parameter

Keyword	Associated SQL Statements
USER	ALTER USER CREATE USER DROP USER
VIEW	CREATE VIEW DROP VIEW

[1] Java schema objects (sources, classes, and resources) are considered the same as procedure for purposes of skipping (ignoring) SQL statements.

Exceptions

Table 56–14 DBMS_LOGSTDBY.SKIP Procedure Exceptions

Exception	Description
ORA-01031	Insufficient privileges: ■ Procedure used INVOKER rights ■ Procedure needs DBA privileges
ORA-16103	Logical standby apply must be stopped to allow this operation.
ORA-16104	Invalid logical standby option requested.
ORA-16203	"Unable to interpret SKIP procedure return values." Indicates that a SKIP procedure has either generated an exception or has returned ambiguous values. You can identify the offending procedure by examining the DBA_LOGSTDBY_EVENTS view.
ORA-16236	Logical standby metadata operation in progress.

Examples

Example 1 Skipping all DML and DDL changes made to a schema

The following example shows how to specify rules so that SQL Apply will skip both DDL and DML statements made to the HR schema.

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP(STMT => 'SCHEMA DDL', -
     schema_name => 'HR', -
     table_name => '%', -
     proc_name => null);
SQL> EXECUTE DBMS_LOGSTDBY.SKIP(STMT => 'DML', -
     schema_name => 'HR', -
     table_name => '%', -
     proc_name => null);
```

Example 2 Creating a procedure to handle different file system organization

For example, if the file system organization in the logical standby database is different than that in the primary database, you can write a SKIP procedure to handle DDL statements with file specifications transparently.

The following procedure can handle DDL statements as long as you follow a specific naming convention for the file specification string.

1. Create the SKIP procedure to handle tablespace DDL statements:

```
CREATE OR REPLACE PROCEDURE sys.handle_tbs_ddl (
```

```
  old_stmt  IN  VARCHAR2,
  stmt_typ  IN  VARCHAR2,
  schema    IN  VARCHAR2,
  name      IN  VARCHAR2,
  xidusn    IN  NUMBER,
  xidslt    IN  NUMBER,
  xidsqn    IN  NUMBER,
  action    OUT NUMBER,
  new_stmt  OUT VARCHAR2
) AS
BEGIN

-- All primary file specification that contains a directory
-- /usr/orcl/primary/dbs
-- should go to /usr/orcl/stdby directory specification


  new_stmt = replace(old_stmt,
                     '/usr/orcl/primary/dbs',
                     '/usr/orcl/stdby');

  action := DBMS_LOGSTDBY.SKIP_ACTION_REPLACE;

EXCEPTION
  WHEN OTHERS THEN
    action := DBMS_LOGSTDBY.SKIP_ACTION_ERROR;
    new_stmt := NULL;
END handle_tbs_ddl;
```

2. Register the `SKIP` procedure with SQL Apply:

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP (stmt => 'TABLESPACE', -
             proc_name => 'SYS.HANDLE_TBS_DDL');
```

SKIP_ERROR Procedure

Upon encountering an error, the logical standby database uses the criteria contained in this procedure to determine if the error should cause SQL Apply to stop. All errors to be skipped are stored in system tables that describe how exceptions should be handled.

Syntax

```
DBMS_LOGSTDBY.SKIP_ERROR (
     stmt                      IN VARCHAR2,
     schema_name               IN VARCHAR2,
     object_name               IN VARCHAR2,
     proc_name                 IN VARCHAR2,
     use_like                  IN BOOLEAN,
     esc                       IN CHAR1);
```

Parameters

Table 56–15 SKIP_ERROR Procedure Parameters

Parameter	Description
`stmt`	Either a keyword that identifies a set of SQL statements or a specific SQL statement. The use of keywords simplifies configuration because keywords, generally defined by the database object, identify all SQL statements that operate on the specified object. Table 56–13 shows a list of keywords and the equivalent SQL statements, either of which is a valid value for this parameter.
`schema_name`	The name of one or more schemas (wildcards are permitted) associated with the SQL statements identified by the `stmt` parameter. If not applicable, this value must be set to `NULL`.
`object_name`	The name of one or more objects (wildcards are permitted) associated with the SQL statements identified by the `stmt`. If not applicable, this value must be set to `NULL`.

Table 56–15 (Cont.) SKIP_ERROR Procedure Parameters

Parameter	Description
proc_name	Name of a stored procedure to call when SQL Apply determines a particular statement matches the filter defined by the stmt, schema_name, and object_name parameters. Specify the procedure in the following format: '"schema"."package"."procedure"' This procedure returns a value that directs SQL Apply to perform one of the following: execute the statement, skip the statement, or execute a replacement statement. SQL Apply calls the stored procedure with the following call signature: ■ IN STATEMENT VARCHAR(4000) -- The first 4K of the statement ■ IN STATEMENT_TYPE VARCHAR2 -- The stmt of the filter ■ IN SCHEMA VARCHAR2 -- The schema_name of the filter, if applicable ■ IN NAME VARCHAR2 -- The object_name of the filter, if applicable ■ IN XIDUSN NUMBER -- Transaction ID part 1 ■ IN XIDSLT NUMBER -- Transaction ID part 2 ■ IN XIDSQN NUMBER -- Transaction ID part 3 ■ IN ERROR VARCHAR(4000) -- Text of error to be recorded (optional) ■ OUT NEW_ERROR VARCHAR(4000) -- Null or modified error text
use_like	Allows pattern matching to isolate the tables that you want to skip on the logical standby database. The use_like parameter matches a portion of one character value to another by searching the first value for the pattern specified by the second, and calculates strings using characters as defined by the input character set. This parameter follows the same rules for pattern matching described in the *Oracle Database SQL Reference*.
esc	Identifies an escape character (such as the characters "%" or "_") that you can use for pattern matching. If the escape character appears in the pattern before the character "%" or "_" then Oracle interprets this character literally in the pattern, rather than as a special pattern matching character. See*Oracle Database SQL Reference* for more information about pattern matching.

Usage Notes

- A stored procedure provided to the SKIP_ERROR procedure is called when SQL Apply encounters an error that could shut down the application of redo logs to the standby database.
- Running this stored procedure affects the error being written in the STATUS column of the DBA_LOGSTDBY_EVENTS table. The STATUS_CODE column remains unchanged. If the stored procedure is to have no effect, that is, apply will be stopped, then the NEW_ERROR is written to the events table. To truly have no effect, set NEW_ERROR to ERROR in the procedure.
- If the stored procedure requires that a shutdown be avoided, then you must set NEW_ERROR to NULL.
- This procedure requires DBA privileges to execute.

- For USER statements, the SCHEMA_NAME parameter will be the user and you should specify '%' for the OBJECT_NAME parameter.
- If the PROC_NAME parameter is specified, it must already exist in DBA_PROCEDURES and it must execute with DEFINERS rights. If the procedure is declared with INVOKERS rights, the ORA-1031: insufficient privileges message will be returned.
- The PL/SQL block of a SKIP_ERROR procedure cannot contain transaction control statements (for example: COMMIT, ROLLBACK, SAVEPOINT, and SET CONSTRAINT) unless the block is declared to be an autonomous transaction using the following syntax:

```
PRAGMA AUTONOMOUS_TRANSACTION
```

Exceptions

Table 56–16 SKIP_ERROR Procedure Exceptions

Exception	Description
ORA-01031	Insufficient privileges: ■ Procedure used INVOKER rights ■ Procedure needs DBA privileges
ORA-16103	Logical Standby apply must be stopped to allow this operation
ORA-16104	invalid Logical Standby option requested
ORA-16236	Logical Standby metadata operation in progress

Examples

To skip errors on GRANT statements on SYS or HR schemas, define a procedure handle_error_ddl and register it. In the following example, assume that handle_error_ddl is a free-standing procedure in the SYS schema.

1. Create the error-handler procedure:

```
CREATE OR REPLACE PROCEDURE sys.handle_error_ddl (
  old_stmt    IN  VARCHAR2,
  stmt_type   IN  VARCHAR2,
  schema      IN  VARCHAR2,
  name        IN  VARCHAR2,
  xidusn      IN  NUMBER,
  xidslt      IN  NUMBER,
  xidsqn      IN  NUMBER,
  error       IN  VARCHAR2,
  new_stmt    OUT VARCHAR2
) AS

BEGIN
  -- Default to what we already have
  new_stmt := old_stmt;

  -- Ignore any GRANT errors on SYS or HR schemas
  IF INSTR(UPPER(old_stmt),'GRANT') > 0
  THEN
    IF schema IS NULL
    OR (schema IS NOT NULL AND
          (UPPER(schema) = 'SYS'  OR UPPER(schema) = 'HR' )
    THEN
```

```
      new_stmt := NULL;
      -- record the fact that we just skipped an error on 'SYS' or 'HR' schemas
      -- code not shown here
    END IF;
  END IF;

END handle_error_ddl;
/
```

2. Register the error handler with SQL Apply:

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP_ERROR ( -
     statement => 'NON_SCHEMA_DDL', -
     schema_name => NULL, -
     object_name => NULL, -
     proc_name => 'SYS.HANDLE_ERROR_DDL');
```

SKIP_TRANSACTION Procedure

This procedure provides a way to skip (ignore) applying transactions to the logical standby database. You can skip specific transactions by specifying transaction identification information.

Syntax

```
DBMS_LOGSTDBY.SKIP_TRANSACTION (
     XIDUSN                IN NUMBER,
     XIDSLT NUMBER         IN NUMBER,
     XIDSQN NUMBER         IN NUMBER);
```

Parameters

Table 56–17 SKIP_TRANSACTION Procedure Parameters

Parameter	Description
XIDUSN NUMBER	Transaction ID undo segment number of the transaction being skipped
XIDSLT NUMBER	Transaction ID slot number of the transaction being skipped
XIDSQN NUMBER	Transaction ID sequence number of the transaction being skipped

Usage Notes

If SQL Apply stops due to a particular transaction (for example, a DDL transaction), you can specify that transaction ID and then continue to apply. You can call this procedure multiple times for as many transactions as you want SQL Apply to ignore.

> **CAUTION: `SKIP_TRANSACTION` is an inherently dangerous operation. Do not invoke this procedure unless you have examined the transaction in question through the `V$LOGMNR_CONTENTS` view and have taken compensating actions at the logical standby database. `SKIP_TRANSACTION` is not the appropriate procedure to invoke to skip DML changes to a table.**
>
> **To skip a DML failure, use a `SKIP` procedure, such as `SKIP('DML','MySchema','MyFailed Table')`. Using the `SKIP_TRANSACTION` procedure for DML transactions may skip changes for other tables, thus logically corrupting them.**

- This procedure requires DBA privileges to execute.
- Use the DBA_LOGSTDBY_SKIP_TRANSACTION view to list the transactions that are going to be skipped by SQL Apply.
- Also, see the ALTER DATABASE START LOGICAL STANDBY SKIP FAILED TRANSACTION statement in *Oracle Database SQL Reference*.

Exceptions

Table 56–18 SKIP_TRANSACTION Procedure Exceptions

Exception	Description
ORA-01031	Need DBA privileges
ORA-16103	Logical Standby apply must be stopped to allow this operation
ORA-16104	invalid Logical Standby option requested

Examples

To skip a DDL transaction with (XIDUSN, XIDSLT, XIDSQN) of (1.13.1726) you can register a rule as shown in the following example:

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP_TRANSACTION (-
     XIDUSN => 1, XIDSLT => 13, XIDSQN => 1726);
```

UNSKIP Procedure

Use the UNSKIP procedure to delete rules specified earlier with the SKIP procedure. The parameters specified in the UNSKIP procedure must match exactly for it to delete an already-specified rule.

Syntax

```
DBMS_LOGSTDBY.UNSKIP (
     stmt                      IN VARCHAR2,
     schema_name               IN VARCHAR2,
     object_name               IN VARCHAR2);
```

Parameters

The parameter information for the UNSKIP procedure is the same as that described for the SKIP procedure. See Table 56–12 on page 56-17 for complete parameter information.

Exceptions

Table 56–19 UNSKIP Procedure Exceptions

Exception	Description
ORA-01031	Need DBA privileges
ORA-16103	Logical Standby apply must be stopped to allow this operation
ORA-16104	invalid Logical Standby option requested

Usage Notes

CAUTION: If DML changes for a table have been skipped and not compensated for, you must follow the call to the UNSKIP procedure with a call to the INSTANTIATE_TABLE procedure to synchronize this table with those maintained by SQL Apply.

- This procedure requires DBA privileges to execute.
- Wildcards passed in the schema_name or the object_name parameter are not expanded. The wildcard character is matched at the character level. Thus, you can delete only one specified rule by invoking the UNSKIP procedure, and you will need a distinct UNSKIP procedure call to delete each rule that was previously specified.

  For example, assume you have specified the following two rules to skip applying DML statements to the HR.EMPLOYEE and HR.EMPTEMP tables:

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP (STMT => 'DML',-
     SCHEMA_NAME => 'HR', -
     OBJECT_NAME => 'EMPLOYEE', -
     PROC_NAME => null);
SQL> EXECUTE DBMS_LOGSTDBY.SKIP (STMT => 'DML',-
     SCHEMA_NAME => 'HR', -
     OBJECT_NAME => 'EMPTEMP', -
     PROC_NAME => null);
```

In the following example, the wildcard in the TABLE_NAME parameter cannot be used to delete the rules that were specified:

```
SQL> EXECUTE DBMS_LOGSTDBY.UNSKIP (STMT => 'DML',-
     SCHEMA_NAME => 'HR', -
     OBJECT_NAME => 'EMP%');
```

In fact, this UNSKIP procedure matches neither of the rules, because the wildcard character in the TABLE_NAME parameter is not expanded. Instead, the wildcard character will be used in an exact match to find the corresponding SKIP rule.

UNSKIP_ERROR Procedure

Use the UNSKIP_ERROR procedure to delete rules specified earlier with the SKIP_ERROR procedure. The parameters specified in the UNSKIP_ERROR procedure must match exactly for the procedure to delete an already-specified rule.

Syntax

```
DBMS_LOGSTDBY.UNSKIP_ERROR (
     stmt                      IN VARCHAR2,
     schema_name               IN VARCHAR2,
     object_name               IN VARCHAR2);
```

Parameters

The parameter information for the UNSKIP_ERROR procedure is the same as that described for the SKIP_ERROR procedure. See Table 56–15 for complete parameter information.

Exceptions

Table 56–20 UNSKIP_ERROR Procedure Exceptions

Exception	Description
ORA-01031	Need DBA privileges
ORA-16103	Logical Standby apply must be stopped to allow this operation
ORA-16104	invalid Logical Standby option requested

Usage Notes

- This procedure requires DBA privileges to execute.
- Wildcards passed in the schema_name or the object_name parameters are not expanded. Instead, the wildcard character is treated as any other character and an exact match is made. Thus, you can delete only one specified rule by invoking the UNSKIP_ERROR procedure, and you need a distinct UNSKIP_ERROR procedure call to delete each rule that you previously specified.

  For example, assume you have specified the following two rules to handle the HR.EMPLOYEE and HR.EMPTEMP tables:

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP_ERROR (STMT => 'DML',-
     SCHEMA_NAME => 'HR', -
     OBJECT_NAME => 'EMPLOYEE', -
     PROC_NAME => 'hr_employee_handler');
SQL> EXECUTE DBMS_LOGSTDBY.SKIP_ERROR (STMT => 'DML',-
     SCHEMA_NAME => 'HR', -
     OBJECT_NAME => 'EMPTEMP', -
     PROC_NAME => 'hr_tempemp_handler');
```

  In this case, the following UNSKIP procedure cannot be used to delete the rules that you have specified:

```
SQL> EXECUTE DBMS_LOGSTDBY.UNSKIP_ERROR (STMT => 'DML',-
     SCHEMA_NAME => 'HR', -
     OBJECT_NAME => 'EMP%');
```

In fact, the UNSKIP procedure will match neither of the rules, because the wildcard character in the OBJECT_NAME parameter will not be expanded.

Example

To remove a handler that was previously registered with SQL Apply from getting called on encountering an error, you can issue the following statement:

```
DBMS_LOGSTDBY.UNSKIP_ERROR ( -
      statement => 'NON_SCHEMA_DDL', -
      schema_name => NULL, -
      object_name => NULL);
```

UNSKIP_TRANSACTION Procedure

Use the UNSKIP_TRANSACTION procedure to delete rules specified earlier with the SKIP_TRANSACTION procedure. The parameters specified in the UNSKIP_TRANSACTION procedure must match exactly for the procedure to delete an already-specified rule.

Syntax

```
DBMS_LOGSTDBY.UNSKIP_TRANSACTION (
     XIDUSN           NUMBER,
     XIDSLT           NUMBER,
     XIDSQN           NUMBER);
```

Parameters

Table 56–21 UNSKIP_TRANSACTION Procedure Parameters

Parameter	Description
XIDUSN	Transaction ID undo segment number of the transaction being skipped
XIDSLT	Transaction ID slot number of the transaction being skipped
XIDSQN	Transaction ID sequence number of the transaction being skipped

Exceptions

Table 56–22 UNSKIP_TRANSACTION Procedure Exceptions

Exception	Description
ORA-01031	Need DBA privileges
ORA-16103	Logical Standby apply must be stopped to allow this operation
ORA-16104	invalid Logical Standby option requested

Usage Notes

- This procedure requires DBA privileges to execute.
- Query the DBA_LOGSTDBY_SKIP_TRANSACTION view to list the transactions that are going to be skipped by SQL Apply.

Examples

To remove a rule that was originally specified to skip the application of a transaction with (XIDUSN, XIDSLT, XIDSQN) of (1.13.1726) issue the following statement:

```
SQL> DBMS_LOGSTDBY.UNSKIP_TRANSACTION (XIDUSN => 1, XIDSLT => 13, XIDSQN => 1726);
```

57

DBMS_METADATA

The DBMS_METADATA package provides a way for you to retrieve metadata from the database dictionary as XML or creation DDL and to submit the XML to re-create the object.

> **See Also:** *Oracle Database Utilities* for more information and for examples of using the Metadata API

This chapter contains the following topics:

Using DBMS_METADATA

This section contains topics which relate to using the `DBMS_METADATA` package.

Overview

You can use the DBMS_METADATA package to retrieve metadata and also to submit XML.

- Retrieving Metadata
- Submitting XML

Retrieving Metadata

If you are retrieving metadata, you can specify:

- The kind of object to be retrieved. This can be either a particular object type (such as a table, index, or procedure) or a heterogeneous collection of object types that form a logical unit (such as a database export or schema export).
- Optional selection criteria, such as owner or name.
- Parse items (attributes of the returned objects to be parsed and returned separately).
- Optional transformations on the output, implemented by XSLT (Extensible Stylesheet Language Transformation) scripts. By default the output is represented in XML, but you can specify transformations (into SQL DDL, for example), which are implemented by XSLT stylesheets stored in the database or externally.

DBMS_METADATA provides the following retrieval interfaces:

- For programmatic use: OPEN, SET_FILTER, SET_COUNT, GET_QUERY, SET_PARSE_ITEM, ADD_TRANSFORM, SET_TRANSFORM_PARAM, SET_REMAP_PARAM, FETCH_xxx, and CLOSE retrieve multiple objects.
- For use in SQL queries and for browsing: GET_XML and GET_DDL return metadata for a single named object. The GET_DEPENDENT_XML, GET_DEPENDENT_DDL, GET_GRANTED_XML, and GET_GRANTED_DDL interfaces return metadata for one or more dependent or granted objects. These procedures do not support heterogeneous object types.

Submitting XML

If you are submitting XML, you specify:

- The type of object
- Optional transform parameters to modify the object (for example, changing the object's owner)
- Parse items (attributes of the submitted objects to be parsed and submitted separately)
- Whether to execute the operation or simply return the generated DDL

DBMS_METADATA provides a programmatic interface for submission of XML. It is comprised of the following procedures: OPENW, ADD_TRANSFORM, SET_TRANSFORM_PARAM, SET_REMAP_PARAM, SET_PARSE_ITEM, CONVERT, PUT, and CLOSE.

Security Model

The object views of the Oracle metadata model implement security as follows:

- Nonprivileged users can see the metadata of only their own objects.
- `SYS` and users with `SELECT_CATALOG_ROLE` can see all objects.
- Nonprivileged users can also retrieve public synonyms, system privileges granted to them, and object privileges granted to them or by them to others. This also includes privileges granted to `PUBLIC`.
- If callers request objects they are not privileged to retrieve, no exception is raised; the object is simply not retrieved.
- If nonprivileged users are granted some form of access to an object in someone else's schema, they will be able to retrieve the grant specification through the Metadata API, but not the object's actual metadata.
- In stored procedures, functions, and definers-rights packages, roles (such as `SELECT_CATALOG_ROLE`) are disabled. Therefore, such a PL/SQL program can only fetch metadata for objects in its own schema. If you want to write a PL/SQL program that fetches metadata for objects in a different schema (based on the invoker's possession of `SELECT_CATALOG_ROLE`), you must make the program invokers-rights.

Rules and Limits

In an Oracle Shared Server (OSS) environment, the DBMS_METADATA package must disable session migration and connection pooling. This results in any shared server process that is serving a session running the package to effectively become a default, dedicated server for the life of the session. You should ensure that sufficient shared servers are configured when the package is used and that the number of servers is not artificially limited by too small a value for the MAX_SHARED_SERVERS initialization parameter.

Data Structures - Object and Table Types

The DBMS_METADATA package defines, in the SYS schema, the following OBJECT and TABLE types.

```
CREATE TYPE sys.ku$_parsed_item AS OBJECT (
  item            VARCHAR2(30),
  value           VARCHAR2(4000),
  object_row      NUMBER )
/

CREATE PUBLIC SYNONYM ku$_parsed_item FOR sys.ku$_parsed_item;

CREATE TYPE sys.ku$_parsed_items IS TABLE OF sys.ku$_parsed_item
/

CREATE PUBLIC SYNONYM ku$_parsed_items FOR sys.ku$_parsed_items;

CREATE TYPE sys.ku$_ddl AS OBJECT (
   ddlText        CLOB,
parsedItem sys.ku$_parsed_items )
/

CREATE PUBLIC SYNONYM ku$_ddl FOR sys.ku$_ddl;

CREATE TYPE sys.ku$_ddls IS TABLE OF sys.ku$_ddl
/

CREATE PUBLIC SYNONYM ku$_ddls FOR sys.ku$_ddls;

CREATE TYPE sys.ku$_multi_ddl AS OBJECT (
   object_row     NUMBER,
   ddls           sys.ku$_ddls )
/

CREATE OR REPLACE PUBLIC SYNONYM ku$_multi_ddl FOR sys.ku$_multi_ddl;

CREATE TYPE sys.ku$_multi_ddls IS TABLE OF sys.ku$_multi_ddl;
/

CREATE OR REPLACE PUBLIC SYNONYM ku$_multi_ddls FOR
                          sys.ku$_multi_ddls;

CREATE TYPE sys.ku$_ErrorLine IS OBJECT (
   errorNumber    NUMBER,
   errorText      VARCHAR2(2000) )
/

CREATE PUBLIC SYNONYM ku$_ErrorLine FOR sys.ku$_ErrorLine;

CREATE TYPE sys.ku$_ErrorLines IS TABLE OF sys.ku$_ErrorLine
/
CREATE PUBLIC SYNONYM ku$ErrorLines FOR sys.ku$_ErrorLines;

CREATE TYPE sys.ku$_SubmitResult AS OBJECT (
   ddl          sys.ku$_ddl,
   errorLines   sys.ku$_ErrorLines );
/
```

```
CREATE TYPE sys.ku$_SubmitResults IS TABLE OF sys.ku$_SubmitResult
/

CREATE PUBLIC SYNONYM ku$_SubmitResults FOR sys.ku$_SubmitResults;
```

Subprogram Groupings

The DBMS_METADATA subprograms are used to retrieve objects from, and submit XML to, a database. Some subprograms are used for both activities, while others are used only for retrieval or only for submission.

- Table 57–1 provides a summary, in alphabetical order, of DBMS_METADATA subprograms used to retrieve multiple objects from a database.
- Table 57–2 provides a summary, in alphabetical order, of DBMS_METADATA subprograms used to submit XML metadata to a database.

Subprograms for Retrieving Multiple Objects From the Database

Table 57–1 lists the subprograms used for retrieving multiple objects from the database.

Table 57–1 DBMS_METADATA Subprograms for Retrieving Multiple Objects

Subprogram	Description
ADD_TRANSFORM Function on page 57-12	Specifies a transform that `FETCH_xxx` applies to the XML representation of the retrieved objects
CLOSE Procedure on page 57-15	Invalidates the handle returned by `OPEN` and cleans up the associated state
FETCH_xxx Functions and Procedures on page 57-18	Returns metadata for objects meeting the criteria established by `OPEN`, `SET_FILTER`, `SET_COUNT`, `ADD_TRANSFORM`, and so on
GET_QUERY Function on page 57-25	Returns the text of the queries that are used by `FETCH_xxx`
GET_xxx Functions on page 57-21	Fetches the metadata for a specified object as XML or DDL, using only a single call
OPEN Function on page 57-26	Specifies the type of object to be retrieved, the version of its metadata, and the object model
SET_COUNT Procedure on page 57-36	Specifies the maximum number of objects to be retrieved in a single `FETCH_xxx` call
SET_FILTER Procedure on page 57-37	Specifies restrictions on the objects to be retrieved, for example, the object name or schema
SET_PARSE_ITEM Procedure on page 57-47	Enables output parsing by specifying an object attribute to be parsed and returned
SET_TRANSFORM_PARAM and SET_REMAP_PARAM Procedures on page 57-50	Specifies parameters to the XSLT stylesheets identified by `transform_handle`

Subprograms for Submitting XML to the Database

Table 57–2 lists the subprograms used for submitting XML to the database.

Table 57–2 DBMS_METADATA Subprograms for Submitting XML

Subprogram	Description
ADD_TRANSFORM Function on page 57-12	Specifies a transform for the XML documents
CLOSE Procedure on page 57-15	Closes the context opened with `OPENW`
CONVERT Functions and Procedures on page 57-16	Converts an XML document to DDL
OPENW Function on page 57-33	Opens a write context
PUT Function on page 57-34	Submits an XML document to the database
SET_PARSE_ITEM Procedure on page 57-47	Specifies an object attribute to be parsed
SET_TRANSFORM_PARAM and SET_REMAP_PARAM Procedures on page 57-50	`SET_TRANSFORM_PARAM` specifies a parameter to a transform `SET_REMAP_PARAM` specifies a remapping for a transform

Summary of All DBMS_METADATA Subprograms

Table 57–3 DBMS_METADATA Package Subprograms

Subprogram	Description
ADD_TRANSFORM Function on page 57-12	Specifies a transform that FETCH_xxx applies to the XML representation of the retrieved objects
CLOSE Procedure on page 57-15	Invalidates the handle returned by OPEN and cleans up the associated state
CONVERT Functions and Procedures on page 57-16	Converts an XML document to DDL.
FETCH_xxx Functions and Procedures on page 57-18	Returns metadata for objects meeting the criteria established by OPEN, SET_FILTER, SET_COUNT, ADD_TRANSFORM, and so on
GET_xxx Functions on page 57-21	Fetches the metadata for a specified object as XML or DDL, using only a single call
GET_QUERY Function on page 57-25	Returns the text of the queries that are used by FETCH_xxx
OPEN Function on page 57-26	Specifies the type of object to be retrieved, the version of its metadata, and the object model
OPENW Function on page 57-33	Opens a write context
PUT Function on page 57-34	Submits an XML document to the database
SET_COUNT Procedure on page 57-36	Specifies the maximum number of objects to be retrieved in a single FETCH_xxx call
SET_FILTER Procedure on page 57-37	Specifies restrictions on the objects to be retrieved, for example, the object name or schema
SET_PARSE_ITEM Procedure on page 57-47	Enables output parsing by specifying an object attribute to be parsed and returned
SET_TRANSFORM_PARAM and SET_REMAP_PARAM Procedures on page 57-50	Specifies parameters to the XSLT stylesheets identified by transform_handle

ADD_TRANSFORM Function

This function is used for both retrieval and submission:

- When this procedure is used to retrieve objects, it specifies a transform that FETCH_xxx applies to the XML representation of the retrieved objects.
- When used to submit objects, it specifies a transform that CONVERT or PUT applies to the XML representation of the submitted objects. It is possible to add more than one transform.

See Also: For more information about related subprograms:

- Subprograms for Retrieving Multiple Objects From the Database on page 57-9
- Subprograms for Submitting XML to the Database on page 57-10

Syntax

```
DBMS_METADATA.ADD_TRANSFORM (
   handle       IN NUMBER,
   name         IN VARCHAR2,
   encoding     IN VARCHAR2 DEFAULT NULL,
   object_type  IN VARCHAR2 DEFAULT NULL)
 RETURN NUMBER;
```

Parameters

Table 57–4 ADD_TRANSFORM Function Parameters

Parameters	Description
handle	The handle returned from OPEN when this transform is used to retrieve objects. Or the handle returned from OPENW when this transform is used in the submission of XML metadata.
name	The name of the transform. If name contains a period, colon, or forward slash, it is interpreted as the URL of a user-supplied XSLT script. See *Oracle XML DB Developer's Guide.* Otherwise, name designates a transform implemented by this project. The following transforms are defined: ■ DDL - the document is transformed to DDL that creates the object. The output of this transform is *not* an XML document. ■ MODIFY - The document is modified as directed by transform and remap parameters. The output of this transform is an XML document. If no transform or remap parameters are specified, the document is unchanged.
encoding	The name of the Globalization Support character set in which the stylesheet pointed to by name is encoded. This is only valid if name is a URL. If left NULL and the URL is external to the database, UTF-8 encoding is assumed. If left NULL and the URL is internal to the database (that is, it begins with /oradb/), then the encoding is assumed to be the database character set.

Table 57–4 (Cont.) ADD_TRANSFORM Function Parameters

Parameters	Description
object_type	The definition of this parameter depends upon whether you are retrieving objects or submitting XML metadata. 1. When you use ADD_TRANFORM to retrieve objects, the following definition of object_type applies: Designates the object type to which the transform applies. (Note that this is an object type name, not a path name.) By default the transform applies to the object type of the OPEN handle. When the OPEN handle designates a heterogeneous object type, the following behavior can occur: ■ if object_type is omitted, the transform applies to all object types within the heterogeneous collection ■ if object_type is specified, the transform only applies to that specific object type within the collection If you omit this parameter you can add the DDL transform to all objects in a heterogeneous collection with a single call. If you supply this parameter, you can add a transform for a specific object type. 2. When you use ADD_TRANSFORM in the submission of XML metadata, this parameter is the object type to which the transform applies. By default, it is the object type of the OPENW handle. Because the OPENW handle cannot designate a heterogeneous object type, the caller would normally leave this parameter NULL in the ADD_TRANSFORM calls.

Return Values

The opaque handle that is returned is used as input to SET_TRANSFORM_PARAM and SET_REMAP_PARAM. Note that this handle is different from the handle returned by OPEN or OPENW; it refers to the transform, not the set of objects to be retrieved.

Usage Notes

- With no transforms added, objects are returned by default as XML documents. You call ADD_TRANSFORM to specify the XSLT stylesheets to be used to transform the returned XML documents.
- You can call ADD_TRANSFORM more than once to apply multiple transforms to XML documents. Transforms are applied in the order in which they were specified, the output of the first transform being used as input to the second, and so on.
- The output of the DDL transform is *not* an XML document. Therefore, no transform should be added after the DDL transform.

Exceptions

- INVALID_ARGVAL. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- INVALID_OPERATION. ADD_TRANSFORM was called after the first call to FETCH_xxx for the OPEN context. After the first call to FETCH_xxx is made, no further calls to ADD_TRANSFORM for the current OPEN context are permitted.
- INCONSISTENT_ARGS. The arguments are inconsistent. Possible inconsistencies include the following:

- `encoding` is specified even though name is not a URL
- `object_type` is not part of the collection designated by handle

CLOSE Procedure

This procedure is used for both retrieval and submission. This procedure invalidates the handle returned by OPEN (or OPENW) and cleans up the associated state.

See Also: For more information about related subprograms:

- Subprograms for Retrieving Multiple Objects From the Database on page 57-9
- Subprograms for Submitting XML to the Database on page 57-10

Syntax

```
DBMS_METADATA.CLOSE (
   handle  IN NUMBER);
```

Parameters

Table 57–5 CLOSE Procedure Parameters

Parameter	Description
handle	The handle returned from OPEN (or OPENW).

Usage Notes

Note: The following notes apply only to object retrieval

You can prematurely terminate the stream of objects established by OPEN or (OPENW).

- If a call to FETCH_xxx returns NULL, indicating no more objects, a call to CLOSE is made transparently. In this case, you can still call CLOSE on the handle and not get an exception. (The call to CLOSE is not required.)
- If you know that only one specific object will be returned, you should explicitly call CLOSE after the single FETCH_xxx call to free resources held by the handle.

Exceptions

- INVALID_ARGVAL. The value for the handle parameter is NULL or invalid.

CONVERT Functions and Procedures

The CONVERT functions and procedures transform input XML documents. The CONVERT functions return creation DDL. The CONVERT procedures return either XML or DDL, depending on the specified transforms.

> **See Also:** For more information about related subprograms:
>
> - Subprograms for Submitting XML to the Database on page 57-10

Syntax

The CONVERT functions are as follows:

```
DBMS_METADATA.CONVERT (
   handle   IN NUMBER,
   document IN sys.XMLType)
 RETURN sys.ku$_multi_ddls;

DBMS_METADATA.CONVERT (
  handle   IN NUMBER,
  document IN CLOB)
 RETURN sys.ku$_multi_ddls;
```

The CONVERT procedures are as follows:

```
DBMS_METADATA.CONVERT (
  handle   IN NUMBER,
  document IN sys.XMLType,
  result   IN OUT NOCOPY CLOB);

DBMS_METADATA.CONVERT (
  handle   IN NUMBER,
  document IN CLOB,
  result   IN OUT NOCOPY CLOB);
```

Parameters

Table 57–6 CONVERT Subprogram Parameters

Parameter	Description
handle	The handle returned from OPENW.
document	The XML document containing object metadata of the type of the OPENW handle.
result	The converted document.

Return Values

DDL to create the object(s).

Usage Notes

You can think of CONVERT as the second half of FETCH_xxx, either FETCH_DDL (for the function variants) or FETCH_CLOB (for the procedure variants). There are two differences:

- FETCH_xxx gets its XML document from the database, but CONVERT gets its XML document from the caller
- FETCH_DDL returns its results in a sys.ku$_ddls nested table, but CONVERT returns a sys.ku$_multi_ddls nested table

The transforms specified with ADD_TRANSFORM are applied in turn, and the result is returned to the caller. For the function variants, the DDL transform must be specified. If parse items were specified, they are returned in the parsedItems column. Parse items are ignored by the procedure variants.

The encoding of the XML document is embedded in its CLOB or XMLType representation. The version of the metadata is embedded in the XML. The generated DDL is valid for the database version specified in OPENW.

Exceptions

- INVALID_ARGVAL. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- INCONSISTENT_OPERATION. No transform was specified. The DDL transform was not specified (function variants only).
- INCOMPATIBLE_DOCUMENT. The version of the XML document is not compatible with this version of the software.

FETCH_xxx Functions and Procedures

These functions and procedures return metadata for objects meeting the criteria established by OPEN, SET_FILTER, SET_COUNT, ADD_TRANSFORM, and so on. See "Usage Notes" on page 57-19 for the variants.

> **See Also:** For more information about related subprograms:
>
> - Subprograms for Retrieving Multiple Objects From the Database on page 57-9

Syntax

The FETCH functions are as follows:

```
DBMS_METADATA.FETCH_XML (
   handle  IN NUMBER)
RETURN sys.XMLType;
```

> **See Also:** *Oracle XML DB Developer's Guide* for a description of XMLType

```
DBMS_METADATA.FETCH_DDL (
   handle  IN NUMBER)
RETURN sys.ku$_ddls;

DBMS_METADATA.FETCH_CLOB (
   handle       IN NUMBER,
   cache_lob    IN BOOLEAN DEFAULT TRUE,
   lob_duration IN PLS INTEGER DEFAULT DBMS_LOB.SESSION)
RETURN CLOB;
```

The FETCH procedures are as follows:

```
DBMS_METADATA.FETCH_CLOB (
   handle  IN NUMBER,
   doc     IN OUT NOCOPY CLOB);

DBMS_METADATA.FETCH_XML_CLOB (
   handle  IN NUMBER,
   doc     IN OUT NOCOPY CLOB,
   parsed_items OUT sys.ku$_parsed_items,
   object_type_path OUT VARCHAR2);
```

Parameters

Table 57–7 FETCH_xxx Function Parameters

Parameters	Description
handle	The handle returned from OPEN.
cache_lob	TRUE=read LOB into buffer cache
lob_duration	The duration for the temporary LOB created by FETCH_CLOB, either DBMS_LOB.SESSION (the default) or DBMS_LOB.CALL.
doc	The metadata for the objects, or NULL if all objects have been returned.

Table 57–7 (Cont.) FETCH_xxx Function Parameters

Parameters	Description
parsed_items	A nested table containing the items specified by SET_PARSE_ITEM. If SET_PARSE_ITEM was not called, a NULL is returned.
object_type_path	For heterogeneous object types, this is the full path name of the object type for the objects returned by the call to FETCH_XXX. If handle designates a homogeneous object type, a NULL is returned.

Return Values

The metadata for the objects or NULL if all objects have been returned.

Usage Notes

These functions and procedures return metadata for objects meeting the criteria established by the call to OPEN that returned the handle, and subsequent calls to SET_FILTER, SET_COUNT, ADD_TRANSFORM, and so on. Each call to FETCH_xxx returns the number of objects specified by SET_COUNT (or less, if fewer objects remain in the underlying cursor) until all objects have been returned. After the last object is returned, subsequent calls to FETCH_xxx return NULL and cause the stream created by OPEN to be transparently closed.

There are several different FETCH_xxx functions and procedures:

- The FETCH_XML function returns the XML metadata for an object as an XMLType. It assumes that if any transform has been specified, that transform will produce an XML document. In particular, it assumes that the DDL transform has not been specified.
- The FETCH_DDL function returns the DDL (to create the object) in a sys.ku$_ddls nested table. It assumes that the DDL transform has been specified. Each row of the sys.ku$_ddls nested table contains a single DDL statement in the ddlText column; if requested, parsed items for the DDL statement will be returned in the parsedItems column. Multiple DDL statements may be returned under the following circumstances:
  - When you call SET_COUNT to specify a count greater than 1
  - When an object is transformed into multiple DDL statements. For example, A TYPE object that has a DDL transform applied to it can be transformed into both CREATE TYPE and CREATE TYPE BODY statements. A TABLE object can be transformed into a CREATE TABLE, and one or more ALTER TABLE statements
- The FETCH_CLOB function simply returns the object, transformed or not, as a CLOB. By default, the CLOB is read into the buffer cache and has session duration, but these defaults can be overridden with the cache_lob and lob_duration parameters.
- The FETCH_CLOB procedure returns the objects by reference in an IN OUT NOCOPY parameter. This is faster than the function variant, which returns LOBs by value, a practice that involves an expensive LOB copy.
- The FETCH_XML_CLOB procedure returns the XML metadata for the objects as a CLOB in an IN OUT NOCOPY parameter. This helps to avoid LOB copies, which can consume a lot of resources. It also returns a nested table of parse items and the full path name of the object type of the returned objects.

- All LOBs returned by FETCH_xxx are temporary LOBs. You must free the LOB. If the LOB is supplied as an IN OUT NOCOPY parameter, you must also create the LOB.
- If SET_PARSE_ITEM was called, FETCH_DDL and FETCH_XML_CLOB return attributes of the object's metadata (or the DDL statement) in a sys.ku$_parsed_items nested table. For FETCH_XML_CLOB, the nested table is an OUT parameter. For FETCH_DDL, it is a column in the returned sys.ku$_ddls nested table. Each row of the nested table corresponds to an item specified by SET_PARSE_ITEM and contains the following columns:
  - item—the name of the attribute as specified in the name parameter to SET_PARSE_ITEM.
  - value—the attribute value, or NULL if the attribute is not present in the DDL statement.
  - object-row—a positive integer indicating the object to which the parse item applies. If multiple objects are returned by FETCH_xxx, (because SET_COUNT specified a count greater than 1) then object_row=1 for all items for the first object, 2 for the second, and so on.
- The rows of the sys.ku$_parsed_items nested table are ordered by ascending object_row, but otherwise the row order is undetermined. To find a particular parse item within an object row the caller must search the table for a match on item.
- In general there is no guarantee that a requested parse item will be returned. For example, the parse item may not apply to the object type or to the particular line of DDL, or the item's value may be NULL.
- If SET_PARSE_ITEM was not called, NULL is returned as the value of the parsed items nested table.
- It is expected that the same variant of FETCH_xxx will be called for all objects selected by OPEN. That is, programs will not intermix calls to FETCH_XML, FETCH_DDL, FETCH_CLOB, and so on using the same OPEN handle. The effect of calling different variants is undefined; it might do what you expect, but there are no guarantees.
- Every object fetched will be internally consistent with respect to on-going DDL (and the subsequent recursive DML) operations against the dictionary. In some cases, multiple queries may be issued, either because the object type is heterogeneous or for performance reasons (for example, one query for heap tables, one for index-organized tables). Consequently the FETCH_xxx calls may in fact be fetches from different underlying cursors (meaning that read consistency is not guaranteed).

Exceptions

Most exceptions raised during execution of the query are propagated to the caller. Also, the following exceptions may be raised:

- INVALID_ARGVAL. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- INCONSISTENT_OPERATION. Either FETCH_XML was called when the DDL transform had been specified, or FETCH_DDL was called when the DDL transform had *not* been specified.

GET_xxx Functions

The following GET_xxx functions let you fetch metadata for objects with a single call:

- GET_XML
- GET_DDL
- GET_DEPENDENT_XML
- GET_DEPENDENT_DDL
- GET_GRANTED_XML
- GET_GRANTED_DDL

See Also: For more information about related subprograms:

- Subprograms for Retrieving Multiple Objects From the Database on page 57-9

Syntax

```
DBMS_METADATA.GET_XML (
object_type     IN VARCHAR2,
name            IN VARCHAR2,
schema          IN VARCHAR2 DEFAULT NULL,
version         IN VARCHAR2 DEFAULT 'COMPATIBLE',
model           IN VARCHAR2 DEFAULT 'ORACLE',
transform       IN VARCHAR2 DEFAULT NULL)
RETURN CLOB;

DBMS_METADATA.GET_DDL (
object_type     IN VARCHAR2,
name            IN VARCHAR2,
schema          IN VARCHAR2 DEFAULT NULL,
version         IN VARCHAR2 DEFAULT 'COMPATIBLE',
model           IN VARCHAR2 DEFAULT 'ORACLE',
transform       IN VARCHAR2 DEFAULT 'DDL')
RETURN CLOB;

DBMS_METADATA.GET_DEPENDENT_XML (
object_type        IN VARCHAR2,
base_object_name   IN VARCHAR2,
base_object_schema IN VARCHAR2 DEFAULT NULL,
version            IN VARCHAR2 DEFAULT 'COMPATIBLE',
model              IN VARCHAR2 DEFAULT 'ORACLE',
transform          IN VARCHAR2 DEFAULT NULL,
object_count       IN NUMBER   DEFAULT 10000)
RETURN CLOB;

DBMS_METADATA.GET_DEPENDENT_DDL (
object_type         IN VARCHAR2,
base_object_name    IN VARCHAR2,
base_object_schema  IN VARCHAR2 DEFAULT NULL,
version             IN VARCHAR2 DEFAULT 'COMPATIBLE',
model               IN VARCHAR2 DEFAULT 'ORACLE',
transform           IN VARCHAR2 DEFAULT 'DDL',
object_count        IN NUMBER   DEFAULT 10000)
RETURN CLOB;

DBMS_METADATA.GET_GRANTED_XML (
```

```
object_type     IN VARCHAR2,
grantee         IN VARCHAR2 DEFAULT NULL,
version         IN VARCHAR2 DEFAULT 'COMPATIBLE',
model           IN VARCHAR2 DEFAULT 'ORACLE',
transform       IN VARCHAR2 DEFAULT NULL,
object_count    IN NUMBER   DEFAULT 10000)
RETURN CLOB;

DBMS_METADATA.GET_GRANTED_DDL (
object_type     IN VARCHAR2,
grantee         IN VARCHAR2 DEFAULT NULL,
version         IN VARCHAR2 DEFAULT 'COMPATIBLE',
model           IN VARCHAR2 DEFAULT 'ORACLE',
transform       IN VARCHAR2 DEFAULT 'DDL',
object_count    IN NUMBER   DEFAULT 10000)
RETURN CLOB;
```

Parameters

Table 57–8 GET_xxx Function Parameters

Parameter	Description
object_type	The type of object to be retrieved. This parameter takes the same values as the OPEN object_type parameter, except that it cannot be a heterogeneous object type. The attributes of the object type must be appropriate to the function. That is, for GET_xxx it must be a named object.
name	The object name. It is used internally in a NAME filter. (If the name is longer than 30 characters, it will be used in a LONGNAME filter.) If this parameter is NULL, then no NAME or LONGNAME filter is specified See Table 57–17 for a list of filters.
schema	The object schema. It is used internally in a SCHEMA filter. The default is the current user.
version	The version of metadata to be extracted. This parameter takes the same values as the OPEN version parameter.
model	The object model to use. This parameter takes the same values as the OPEN model parameter.
transform	The name of a transformation on the output. This parameter takes the same values as the ADD_TRANSFORM name parameter. For GET_XML this must not be DDL.
base_object_name	The base object name. It is used internally in a BASE_OBJECT_NAME filter.
base_object_schema	The base object schema. It is used internally in a BASE_OBJECT_SCHEMA filter. The default is the current user.
grantee	The grantee. It is used internally in a GRANTEE filter. The default is the current user.
object_count	The maximum number of objects to return. See SET_COUNT Procedure on page 57-36.

Return Values

The metadata for the specified object as XML or DDL.

Usage Notes

- These functions allow you to fetch metadata for objects with a single call. They encapsulate calls to OPEN, SET_FILTER, and so on. The function you use depends on the characteristics of the object type and on whether you want XML or DDL.
  - GET_xxx is used to fetch named objects, especially schema objects (tables, views).
  - GET_DEPENDENT_xxx is used to fetch dependent objects (audits, object grants).
  - GET_GRANTED_xxx is used to fetch granted objects (system grants, role grants).
- For some object types you can use more than one function. For example, you can use GET_xxx to fetch an index by name, or GET_DEPENDENT_xxx to fetch the same index by specifying the table on which it is defined.
- GET_xxx only returns a single named object.
- For GET_DEPENDENT_xxx and GET_GRANTED_xxx, an arbitrary number of dependent or granted objects can match the input criteria. You can specify an object count when fetching these objects. (The default count of 10000 should be adequate in most cases.)
- If the DDL transform is specified, session-level transform parameters are inherited.
- If you invoke these functions from SQL*Plus, you should set the PAGESIZE to 0 and set LONG to some large number to get complete, uninterrupted output.

Exceptions

- INVALID_ARGVAL. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- OBJECT_NOT_FOUND. The specified object was not found in the database.

Examples

Example: Fetch the XML Representation of SCOTT.EMP

To generate complete, uninterrupted output, set the PAGESIZE to 0 and set LONG to some large number, as shown, before executing your query.

```
SET LONG 2000000
SET PAGESIZE 0
SELECT DBMS_METADATA.GET_XML('TABLE','EMP','SCOTT')
FROM DUAL;
```

Example: Fetch the DDL for all Complete Tables in the Current Schema, Filter Out Nested Tables and Overflow Segments

This example fetches the DDL for all "complete" tables in the current schema, filtering out nested tables and overflow segments. The example uses SET_TRANSFORM_PARAM (with the handle value = DBMS_METADATA.SESSION_TRANSFORM meaning "for the current session") to specify that storage clauses are not to be returned in the SQL DDL. Afterwards, the example resets the session-level parameters to their defaults.

To generate complete, uninterrupted output, set the PAGESIZE to 0 and set LONG to some large number, as shown, before executing your query.

```
SET LONG 2000000
```

```
SET PAGESIZE 0
EXECUTE DBMS_METADATA.SET_TRANSFORM_PARAM(DBMS_METADATA.SESSION_
TRANSFORM,'STORAGE',false);
SELECT DBMS_METADATA.GET_DDL('TABLE',u.table_name)
     FROM USER_ALL_TABLES u
     WHERE u.nested='NO'
     AND (u.iot_type is null or u.iot_type='IOT');
EXECUTE DBMS_METADATA.SET_TRANSFORM_PARAM(DBMS_METADATA.SESSION_
TRANSFORM,'DEFAULT');
```

Example: Fetch the DDL For All Object Grants On HR.EMPLOYEES

```
SELECT DBMS_METADATA.GET_DEPENDENT_DDL('OBJECT_GRANT',
    'EMPLOYEES','HR') FROM DUAL;
```

Example: Fetch the DDL For All System Grants Granted To SCOTT

```
SELECT DBMS_METADATA.GET_GRANTED_DDL('SYSTEM_GRANT','SCOTT')
    FROM DUAL;
```

GET_QUERY Function

This function returns the text of the queries that are used by FETCH_xxx. This function assists in debugging.

See Also: For more information about related subprograms:

- Subprograms for Retrieving Multiple Objects From the Database on page 57-9

Syntax

```
DBMS_METADATA.GET_QUERY (
   handle  IN NUMBER)
 RETURN VARCHAR2;
```

Parameters

Table 57–9 GET_QUERY Function Parameters

Parameter	Description
handle	The handle returned from OPEN. It cannot be the handle for a heterogeneous object type.

Return Values

The text of the queries that will be used by FETCH_xxx.

Exceptions

- INVALID_ARGVAL. A NULL or invalid value was supplied for the handle parameter.

OPEN Function

This function specifies the type of object to be retrieved, the version of its metadata, and the object model. The return value is an opaque context handle for the set of objects to be used in subsequent calls.

See Also: For more information about related subprograms:

- Subprograms for Retrieving Multiple Objects From the Database on page 57-9

Syntax

```
DBMS_METADATA.OPEN (
   object_type  IN VARCHAR2,
   version      IN VARCHAR2 DEFAULT 'COMPATIBLE',
   model        IN VARCHAR2 DEFAULT 'ORACLE',
   network_link IN VARCHAR2 DEFAULT NULL)
 RETURN NUMBER;
```

Parameters

Table 57–10 Open Function Parameters

Parameter	Description
`object_type`	The type of object to be retrieved. Table 57–11 lists the valid type names and their meanings. These object types will be supported for the ORACLE model of metadata (see `model` in this table). The Attributes column in Table 57–11 specifies some object type attributes: ■ Schema objects, such as tables, belong to schemas. ■ Named objects have unique names (if they are schema objects, the name is unique to the schema). ■ Dependent objects, such as indexes, are defined with reference to a base schema object. ■ Granted objects are granted or assigned to a user or role and therefore have a named grantee. ■ Heterogeneous object types denote a collection of related objects of different types. See Table 57–12 for a listing of object types returned for the heterogeneous object type. These attributes are relevant when choosing object selection criteria. See "SET_FILTER Procedure" on page 57-37 for more information.
`version`	The version of metadata to be extracted. Database objects or attributes that are incompatible with the version will not be extracted. Legal values for this parameter are as follows: `COMPATIBLE` (default)—the version of the metadata corresponds to the database compatibility level. `LATEST`—the version of the metadata corresponds to the database version. A specific database version, for example, 9.2.0. As of Oracle Database 10*g*, this value cannot be lower than 9.2.0.

Table 57–10 (Cont.) Open Function Parameters

Parameter	Description
model	Specifies which view to use, because the API can support multiple views on the metadata. Only the ORACLE model is supported as of Oracle Database 10*g*.
network_link	Reserved.

Table 57–11 provides the name, meaning, attributes, and notes for the DBMS_METADATA package object types. In the attributes column, S represents a schema object, N represents a named object, D represents a dependent object, G represents a granted object, and H represents a heterogeneous object.

Table 57–11 DBMS_METADATA: Object Types

Type Name	Meaning	Attributes	Notes
AQ_QUEUE	queues	SND	Dependent on table
AQ_QUEUE_TABLE	additional metadata for queue tables	ND	Dependent on table
AQ_TRANSFORM	transforms	SN	None
ASSOCIATION	associate statistics	D	None
AUDIT	audits of SQL statements	DG	Modeled as dependent, granted object. The base object name is the statement audit option name (for example, ALTER SYSTEM). There is no base object schema. The grantee is the user or proxy whose statements are audited.
AUDIT_OBJ	audits of schema objects	D	None
CLUSTER	clusters	SN	None
COMMENT	comments	D	None
CONSTRAINT	constraints	SND	Does not include: ▪ primary key constraint for IOT ▪ column NOT NULL constraints ▪ certain REF SCOPE and WITH ROWID constraints for tables with REF columns
CONTEXT	application contexts	N	None
DATABASE_EXPORT	all metadata objects in a database	H	Corresponds to a full database export
DB_LINK	database links	SN	Modeled as schema objects because they have owners. For public links, the owner is PUBLIC. For private links, the creator is the owner.
DEFAULT_ROLE	default roles	G	Granted to a user by ALTER USER
DIMENSION	dimensions	SN	None
DIRECTORY	directories	N	None
FGA_POLICY	fine-grained audit policies	D	Not modeled as named object because policy names are not unique.
FUNCTION	stored functions	SN	None

Table 57–11 (Cont.) DBMS_METADATA: Object Types

Type Name	Meaning	Attributes	Notes
INDEX_STATISTICS	precomputed statistics on indexes	D	The base object is the index's table.
INDEX	indexes	SND	None
INDEXTYPE	indextypes	SN	None
JAVA_SOURCE	Java sources	SN	None
JOB	jobs	S	None
LIBRARY	external procedure libraries	SN	None
MATERIALIZED_VIEW	materialized views	SN	None
MATERIALIZED_ VIEW_LOG	materialized view logs	D	None
OBJECT_GRANT	object grants	DG	None
OPERATOR	operators	SN	None
OUTLINE	stored outlines	N	This type is being deprecated.
PACKAGE	stored packages	SN	By default, both package specification and package body are retrieved. See "SET_ FILTER Procedure" on page 57-37.
PACKAGE_SPEC	package specifications	SN	None
PACKAGE_BODY	package bodies	SN	None
PROCEDURE	stored procedures	SN	None
PROFILE	profiles	N	None
PROXY	proxy authentications	G	Granted to a user by ALTER USER
REF_CONSTRAINT	referential constraint	SND	None
REFRESH_GROUP	refresh groups	SN	None
RESOURCE_COST	resource cost info		None
RLS_CONTEXT	driving contexts for enforcement of fine-grained access-control policies	D	Corresponds to the DBMS_RLS.ADD_ POLICY_CONTENT procedure
RLS_GROUP	fine-grained access-control policy groups	D	Corresponds to the DBMS_RLS.CREATE_ GROUP procedure
RLS_POLICY	fine-grained access-control policies	D	Corresponds to DBMS_RLS.ADD_ GROUPED_POLICY. Not modeled as named objects because policy names are not unique.
RMGR_CONSUMER_ GROUP	resource consumer groups	SN	Data Pump does not use these object types. Instead, it exports resource manager objects as procedural objects.
RMGR_INTITIAL_ CONSUMER_GROUP	assign initial consumer groups to users	G	None
RMGR_PLAN	resource plans	SN	None
RMGR_PLAN_ DIRECTIVE	resource plan directives	D	Dependent on resource plan
ROLE	roles	N	None

Table 57–11 (Cont.) DBMS_METADATA: Object Types

Type Name	Meaning	Attributes	Notes
ROLE_GRANT	role grants	G	None
ROLLBACK_SEGMENT	rollback segments	N	None
SCHEMA_EXPORT	all metadata objects in a schema	H	Corresponds to user-mode export.
SEQUENCE	sequences	SN	None
SYNONYM	synonyms	See notes	Private synonyms are schema objects. Public synonyms are not, but for the purposes of this API, their schema name is PUBLIC. The name of a synonym is considered to be the synonym itself. For example, in CREATE PUBLIC SYNONYM FOO FOR BAR, the resultant object is considered to have name FOO and schema PUBLIC.
SYSTEM_GRANT	system privilege grants	G	None
TABLE	tables	SN	None
TABLE_DATA	metadata describing row data for a table, nested table, or partition	SND	For partitions, the object name is the partition name. For nested tables, the object name is the storage table name. The base object is the top-level table to which the table data belongs. For nested tables and partitioning, this is the top-level table (*not* the parent table or partition). For nonpartitioned tables and non-nested tables this is the table itself.
TABLE_EXPORT	metadata for a table and its associated objects	H	Corresponds to table-mode export
TABLE_STATISTICS	precomputed statistics on tables	D	None
TABLESPACE	tablespaces	N	None
TABLESPACE_QUOTA	tablespace quotas	G	Granted with ALTER USER
TRANSPORTABLE_EXPORT	metadata for objects in a transportable tablespace set	H	Corresponds to transportable tablespace export
TRIGGER	triggers	SND	None
TRUSTED_DB_LINK	trusted links	N	None
TYPE	user-defined types	SN	By default, both type and type body are retrieved. See "SET_FILTER Procedure" on page 57-37.
TYPE_SPEC	type specifications	SN	None
TYPE_BODY	type bodies	SN	None
USER	users	N	None
VIEW	views	SN	None
XMLSCHEMA	XML schema	SN	The object's name is its URL (which may be longer than 30 characters). Its schema is the user who registered it.

Table 57–12 lists the types of objects returned for the major heterogeneous object types. For SCHEMA_EXPORT, certain object types are only returned if the INCLUDE_USER filter is specified at TRUE. In the table, such object types are marked INCLUDE_USER.

Table 57–12 Object Types Returned for the Heterogeneous Object Type

Object Type	DATABASE_ EXPORT	SCHEMA_ EXPORT	TABLE_ EXPORT	TRANSPORTABLE_ EXPORT
ASSOCIATION	Yes	No	No	No
AUDIT	Yes	No	No	No
AUDIT_OBJ	Yes	Yes	Yes	Yes
CLUSTER	Yes	Yes	No	Yes
COMMENT	Yes	Yes	Yes	Yes
CONSTRAINT	Yes	Yes	Yes	Yes
CONTEXT	Yes	No	No	No
DB_LINK	Yes	Yes	No	No
DEFAULT_ROLE	Yes	INCLUDE_USER	No	No
DIMENSION	Yes	Yes	No	No
DIRECTORY	Yes	No	No	No
FGA_POLICY	Yes	No	No	Yes
FUNCTION	Yes	Yes	No	No
INDEX_STATISTICS	Yes	Yes	Yes	Yes
INDEX	Yes	Yes	Yes	Yes
INDEXTYPE	Yes	Yes	No	No
JAVA_SOURCE	Yes	Yes	No	No
JOB	Yes	Yes	No	No
LIBRARY	Yes	Yes	No	No
MATERIALIED_VIEW	Yes	Yes	No	No
MATERIALIZED_VIEW_LOG	Yes	Yes	No	No
OBJECT_GRANT	Yes	Yes	Yes	Yes
OPERATOR	Yes	Yes	No	No
OUTLINE	If OUTLN user's objects are returned	if user is OUTLN	No	No
PACKAGE	Yes	Yes	No	No
PACKAGE_SPEC	Yes	Yes	No	No
PACKAGE_BODY	Yes	Yes	No	No
PASSWORD_HISTORY	Yes	INCLUDE_USER	No	No
PASSWORD_VERIFY_FUNCTION	Yes	No	No	No
PROCEDURE	Yes	Yes	No	No
PROFILE	Yes	No	No	No
PROXY	Yes	No	No	No

Table 57–12 (Cont.) Object Types Returned for the Heterogeneous Object Type

Object Type	DATABASE_ EXPORT	SCHEMA_ EXPORT	TABLE_ EXPORT	TRANSPORTABLE_ EXPORT
REF_CONSTRAINT	Yes	Yes	Yes	Yes
REFRESH_GROUP	Yes	Yes	No	No
RESOURCE_COST	Yes	No	No	No
RLS_CONTEXT	Yes	No	No	Yes
RLS_GROUP	Yes	No	No	Yes
RLS_POLICY	Yes	Table data is retrieved according to policy	Table data is retrieved according to policy	Yes
ROLE	Yes	No	No	No
ROLE_GRANT	Yes	No	No	No
ROLLBACK_SEGMENT	Yes	No	No	No
SEQUENCE	Yes	Yes	No	No
SYNONYM	Yes	Yes	No	No
SYSTEM_GRANT	Yes	INCLUDE_USER	No	No
TABLE	Yes	Yes	Yes	Yes
TABLE_DATA	Yes	Yes	Yes	Yes
TABLE_STATISTICS	Yes	Yes	Yes	Yes
TABLESPACE	Yes	No	No	No
TABLESPACE_QUOTA	Yes	INCLUDE_USER	No	No
TRIGGER	Yes	Yes	Yes	Yes
TRUSTED_DB_LINK	Yes	No	No	No
TYPE	Yes	Yes	No	Yes, if the types are used by tables in the transportable set
TYPE_SPEC	Yes	Yes	No	Yes, if the types are used by tables in the transportable set
TYPE_BODY	Yes	Yes	No	Yes, if the types are used by tables in the transportable set
USER	Yes	INCLUDE_USER	No	No
VIEW	Yes	Yes	No	No
XMLSCHEMA	Yes	Yes	No	No

Return Values

An opaque handle to the class of objects. This handle is used as input to SET_FILTER, SET_COUNT, ADD_TRANSFORM, GET_QUERY, SET_PARSE_ITEM, FETCH_xxx, and CLOSE.

Exceptions

- `INVALID_ARGVAL`. A `NULL` or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- `INVALID_OBJECT_PARAM`. The `version` or `model` parameter was not valid for the `object_type`.

OPENW Function

This function specifies the type of object to be submitted and the object model. The return value is an opaque context handle.

See Also: For more information about related subprograms:

- Subprograms for Submitting XML to the Database on page 57-10

Syntax

```
DBMS_METADATA.OPENW
  (object_type  IN VARCHAR2,
  version       IN VARCHAR2 DEFAULT 'COMPATIBLE',
  model         IN VARCHAR2 DEFAULT 'ORACLE')
 RETURN NUMBER;
```

Parameters

Table 57–13 OPENW Function Parameters

Parameter	Description
object_type	The type of object to be submitted. Valid types names and their meanings are listed in Table 57–11. The type cannot be a heterogeneous object type.
version	The version of DDL to be generated by the CONVERT function. DDL clauses that are incompatible with the version will not be generated. The legal values for this parameter are as follows: ■ COMPATIBLE - This is the default. The version of the DDL corresponds to the database compatibility level. Database compatibility must be set to 9.2.0 or higher. ■ LATEST - The version of the DDL corresponds to the database version. ■ A specific database version. As of Oracle Database 10*g*, this value cannot be lower than 9.2.0.
model	Specifies which view to use. Only the Oracle proprietary (ORACLE) view is supported by DBMS_METADATA.

Return Values

An opaque handle to write context. This handle is used as input to the ADD_TRANSFORM, CONVERT, PUT, and CLOSE procedures.

Exceptions

- INVALID_ARGVAL. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- INVALID_OBJECT_PARAM. The model parameter was not valid for the object_type.

PUT Function

This function submits an XML document containing object metadata to the database to create the object.

See Also: For more information about related subprograms:

- Subprograms for Submitting XML to the Database on page 57-10

Syntax

```
DBMS_METADATA.PUT (
   handle     IN             NUMBER,
   document   IN             sys.XMLType,
   flags      IN             NUMBER,
   results    IN OUT NOCOPY  sys.ku$_SubmitResults)
  RETURN BOOLEAN;

DBMS_METADATA.PUT (
   handle     IN             NUMBER,
   document   IN             CLOB,
   flags      IN             NUMBER,
   results    IN OUT NOCOPY  sys.ku$_SubmitResults)
 RETURN BOOLEAN;
```

Parameters

Table 57–14 PUT Function Parameters

Parameter	Description
handle	The handle returned from OPENW.
document	The XML document containing object metadata for the type of the OPENW handle.
flags	Reserved for future use
results	Detailed results of the operation.

Return Values

TRUE if all SQL operations succeeded; FALSE if there were any errors.

Usage Notes

The PUT function converts the XML document to DDL just as CONVERT does (applying the specified transforms in turn) and then submits each resultant DDL statement to the database. As with CONVERT, the DDL transform must be specified. The DDL statements and associated parse items are returned in the sys.ku$_SubmitResults nested table. With each DDL statement is a nested table of error lines containing any errors or exceptions raised by the statement.

The encoding of the XML document is embedded in its CLOB or XMLType representation. The version of the metadata is embedded in the XML. The generated DDL is valid for the database version specified in OPENW.

Exceptions

- `INVALID_ARGVAL`. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- `INCONSISTENT_OPERATION`. The DDL transform was not specified.
- `INCOMPATIBLE_DOCUMENT`. The version of the XML document is not compatible with this version of the software.

SET_COUNT Procedure

This procedure specifies the maximum number of objects to be retrieved in a single FETCH_xxx call. By default, each call to FETCH_xxx returns one object. You can use the SET_COUNT procedure to override this default. If FETCH_xxx is called from a client, specifying a count value greater than 1 can result in fewer server round trips and, therefore, improved performance.

For heterogeneous object types, a single FETCH_xxx operation only returns objects of a single object type.

See Also: For more information about related subprograms:

- Subprograms for Retrieving Multiple Objects From the Database on page 57-9

Syntax

```
DBMS_METADATA.SET_COUNT (
   handle           IN NUMBER,
   value            IN NUMBER,
   object_type_path IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 57–15 SET_COUNT Procedure Parameters

Parameter	Description
handle	The handle returned from OPEN.
value	The maximum number of objects to retrieve.
object_type_path	A path name designating the object types to which the count value applies. By default, the count value applies to the object type of the OPEN handle. When the OPEN handle designates a heterogeneous object type, behavior can be either of the following: ■ if object_type_path is omitted, the count applies to all object types within the heterogeneous collection ■ if object_type_path is specified, the count only applies to the specific node (or set of nodes) within the tree of object types forming the heterogeneous collection

Exceptions

- INVALID_ARGVAL. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- INVALID_OPERATION. SET_COUNT was called after the first call to FETCH_xxx for the OPEN context. After the first call to FETCH_xxx is made, no further calls to SET_COUNT for the current OPEN context are permitted.
- INCONSISTENT_ARGS. object_type parameter is not consistent with handle.

SET_FILTER Procedure

This procedure specifies restrictions on the objects to be retrieved, for example, the object name or schema.

See Also: For more information about related subprograms:

- Subprograms for Retrieving Multiple Objects From the Database on page 57-9

Syntax

```
DBMS_METADATA.SET_FILTER (
   handle           IN NUMBER,
   name             IN VARCHAR2,
   value            IN VARCHAR2,
   object_type_path IN VARCHAR2 DEFAULT NULL);

DBMS_METADATA.SET_FILTER (
   handle            IN NUMBER,
   name              IN VARCHAR2,
   value             IN BOOLEAN DEFAULT TRUE,
   object_type_path  IN VARCHAR2 DEFAULT NULL);

DBMS_METADATA.SET_FILTER (
   handle            IN NUMBER,
   name              IN VARCHAR2,
   value             IN NUMBER,
   object_type_path  IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 57–16 SET_FILTER Procedure Parameters

Parameter	Description
handle	The handle returned from OPEN.
name	The name of the filter. For each filter, Table 57–17 lists the object_type it applies to, its name, its datatype (text or Boolean) and its meaning or effect (including its default value, if any). The Datatype column of Table 57–17 also indicates whether a text filter is an expression filter. An expression filter is the right-hand side of a SQL comparison (that is, a SQL comparison operator (=, !=, and so on.)) and the value compared against. The value must contain parentheses and quotation marks where appropriate. Note that in PL/SQL and SQL*Plus, two single quotes (*not* a double quote) are needed to represent an apostrophe. For example, an example of a NAME_EXPR filter in PL/SQL is as follows: 'IN (''DEPT'',''EMP'')' The filter value is combined with a particular object attribute to produce a WHERE condition in the query that fetches the objects. In the preceding example, the filter is combined with the attribute corresponding to an object name; objects named 'DEPT' and 'EMP' are selected.
value	The value of the filter. Text, Boolean, and Numeric filters are supported.

Table 57–16 (Cont.) SET_FILTER Procedure Parameters

Parameter	Description
object_type_path	A path name designating the object types to which the filter applies. By default, the filter applies to the object type of the OPEN handle. When the OPEN handle designates a heterogeneous object type, you can use this parameter to specify a filter for a specific node or set of nodes within the tree of object types that form the heterogeneous collection. See Table 57–18 for a listing of some of the values for this parameter.

Table 57–17 describes the object type, name, datatype, and meaning of the filters available with the SET_FILTER procedure.

Table 57–17 SET_FILTER: Filters

Object Type	Name	Datatype	Meaning
Named objects	NAME	text	Objects with this exact name are selected.
Named objects	NAME_EXPR	text expression	The filter value is combined with the object attribute corresponding to the object name to produce a WHERE condition in the query that fetches the objects. By default, all named objects of object_type are selected.
Named objects	EXCLUDE_NAME_EXPR	text expression	The filter value is combined with the attribute corresponding to the object name to specify objects that are to be excluded from the set of objects fetched. By default, all named objects of the object type are selected.
Schema objects	SCHEMA	text	Objects in this schema are selected. If the object type is SYNONYM, specify PUBLIC to select public synonyms.
Schema objects	SCHEMA_EXPR	text expression	The filter value is combined with the attribute corresponding to the object's schema. The default is determined as follows: - if BASE_OBJECT_SCHEMA is specified, then objects in that schema are selected; - otherwise, objects in the current schema are selected.
PACKAGE, TYPE	SPECIFICATION	Boolean	If TRUE, retrieve the package or type specification. Defaults to TRUE.
PACKAGE, TYPE	BODY	Boolean	If TRUE, retrieve the package or type body. Defaults to TRUE.
TABLE, CLUSTER, INDEX, TABLE_DATA, TABLE_EXPORT, TRANSPORTABLE_EXPORT	TABLESPACE	text	Objects in this tablespace (or having a partition in this tablespace) are selected.

Table 57–17 (Cont.) SET_FILTER: Filters

Object Type	Name	Datatype	Meaning
TABLE, CLUSTER, INDEX,TABLE_DATA, TABLE_EXPORT, TRANSPORTABLE_ EXPORT	TABLESPACE_ EXPR	text expression	The filter value is combined with the attribute corresponding to the object's tablespace (or in the case of a partitioned table or index, the partition's tablespaces). By default, objects in all tablespaces are selected.
TABLE, objects dependent on tables	PRIMARY	Boolean	If TRUE, retrieve primary tables (that is, tables for which the secondary object bit in obj$ is clear. Defaults to TRUE.
TABLE, objects dependent on tables	SECONDARY	Boolean	If TRUE, retrieve secondary tables (that is, tables for which the secondary object bit in obj$ is set). Defaults to TRUE.
Dependent Objects	BASE_OBJECT_ NAME	text	Objects are selected that are defined or granted on objects with this name. Specify SCHEMA for triggers on schemas. Specify DATABASE for database triggers. Column-level comments cannot be selected by column name; the base object name must be the name of the table, view, or materialized view containing the column.
Dependent Objects	BASE_OBJECT_ SCHEMA	text	Objects are selected that are defined or granted on objects in this schema. If BASE_OBJECT_NAME is specified with a value other than SCHEMA or DATABASE, this defaults to the current schema.
Dependent Objects	BASE_OBJECT_ NAME_EXPR	text expression	The filter value is combined with the attribute corresponding to the name of the base object. Not valid for schema and database triggers.
Dependent Objects	EXCLUDE_BASE_ OBJECT_NAME_ EXPR	text expression	The filter value is combined with the attribute corresponding to the name of the base object to specify objects that are to be excluded from the set of objects fetched. Not valid for schema and database triggers.
Dependent Objects	BASE_OBJECT_ SCHEMA_EXPR	text expression	The filter value is combined with the attribute corresponding to the schema of the base object.
Dependent Objects	BASE_OBJECT_ TYPE	text	The object type of the base object.
Dependent Objects	BASE_OBJECT_ TYPE_EXPR	text expression	The filter value is combined with the attribute corresponding to the object type of the base object. By default no filtering is done on object type.
Dependent Objects	BASE_OBJECT_ TABLESPACE	text	The tablespace of the base object.
Dependent Objects	BASE_OBJECT_ TABLESPACE_ EXPR	text expression	The filter value is combined with the attribute corresponding to the tablespaces of the base object. By default, no filtering is done on the tablespace.
INDEX, TRIGGER	SYSTEM_ GENERATED	Boolean	If TRUE, select indexes or triggers even if they are system-generated. If FALSE, omit system-generated indexes or triggers. Defaults to TRUE.
Granted Objects	GRANTEE	text	Objects are selected that are granted to this user or role. Specify PUBLIC for grants to PUBLIC.
Granted Objects	PRIVNAME	text	The name of the privilege or role to be granted. For TABLESPACE_QUOTA, only UNLIMITED can be specified.

Table 57–17 (Cont.) SET_FILTER: Filters

Object Type	Name	Datatype	Meaning
Granted Objects	PRIVNAME_EXPR	text expression	The filter value is combined with the attribute corresponding to the privilege or role name. By default, all privileges/roles are returned.
Granted Objects	GRANTEE_EXPR	text expression	The filter value is combined with the attribute corresponding to the grantee name.
Granted Objects	EXCLUDE_ GRANTEE_EXPR	text expression	The filter value is combined with the attribute corresponding to the grantee name to specify objects that are to be excluded from the set of objects fetched.
OBJECT_GRANT	GRANTOR	text	Object grants are selected that are granted by this user.
SYNONYM, JAVA_ SOURCE, XMLSCHEMA	LONGNAME	text	A name longer than 30 characters. Objects with this exact name are selected. If the object name is 30 characters or less, the NAME filter must be used.
SYNONYM, JAVA_ SOURCE, XMLSCHEMA	LONGNAME_EXPR	text	The filter value is combined with the attribute corresponding to the object's long name. By default, no filtering is done on the long name of an object.
All objects	CUSTOM_FILTER	text	The text of a WHERE condition. The condition is appended to the query that fetches the objects. By default, no custom filter is used. The other filters are intended to meet the needs of the majority of users. Use CUSTOM_FILTER when no defined filters exists for your purpose. Of necessity such a filter depends on the detailed structure of the UDTs and views used in the query. Because filters may change from version to version, upward compatibility is not guaranteed.
SCHEMA_EXPORT	SCHEMA	text	The schema whose objects are selected.
SCHEMA_EXPORT	SCHEMA_EXPR	text expression	The filter value is either: combined with the attribute corresponding to a schema name to produce a WHERE condition in the query that fetches schema objects, combined with the attribute corresponding to a base schema name to produce a WHERE condition in the query that fetches dependent objects. By default the current user's objects are selected.
SCHEMA_EXPORT	INCLUDE_USER	Boolean	If TRUE, retrieve objects containing privileged information about the user. For example, USER, PASSWORD_HISTORY, TABLESPACE_QUOTA. Defaults to FALSE.
TABLE_EXPORT	SCHEMA	text	Objects (tables and their dependent objects) in this schema are selected.
TABLE_EXPORT	SCHEMA_EXPR	text expression	The filter value is either: combined with the attribute corresponding to a schema name to produce a WHERE condition in the query that fetches the tables, combined with the attribute corresponding to a base schema name to produce a WHERE condition in the query that fetches the tables' dependent objects. By default the current user's objects are selected.

Table 57–17 (Cont.) SET_FILTER: Filters

Object Type	Name	Datatype	Meaning
TABLE_EXPORT	NAME	text	The table with this exact name is selected along with its dependent objects.
TABLE_EXPORT	NAME_EXPR	text expression	The filter value is combined with the attribute corresponding to a table name in the queries that fetch tables and their dependent objects. By default all tables in the selected schemas are selected, along with their dependent objects.
Heterogeneous objects	BEGIN_WITH	text	The fully qualified path name of the first object type in the heterogeneous collection to be retrieved. Objects normally fetched prior to this object type will not be retrieved.
Heterogeneous objects	BEGIN_AFTER	text	The fully qualified path name of an object type after which the heterogeneous retrieval should begin. Objects of this type will not be retrieved, nor will objects normally fetched prior to this object type.
Heterogeneous objects	END_BEFORE	text	The fully qualified path name of an object type where the heterogeneous retrieval should end. Objects of this type will not be retrieved, nor will objects normally fetched after this object type.
Heterogeneous objects	END_WITH	text	The fully qualified path name of the last object type in the heterogeneous collection to be retrieved. Objects normally fetched after this object type will not be retrieved.
Heterogeneous objects	INCLUDE_PATH_EXPR, EXCLUDE_PATH_EXPR	text expression	For these two filters, the filter value is combined with the attribute corresponding to an object type path name to produce a WHERE condition in the query that fetches the object types belonging to the heterogeneous collection. Objects of types satisfying this condition are included (INCLUDE_PATH_EXPR) or excluded (EXCLUDE_PATH_EXPR) from the set of object types fetched. Path names in the filter value do not have to be fully qualified. See Table 57–18 for valid path names that can be used with these filters. BEGIN_WITH, BEGIN_AFTER, END_BEFORE, END_WITH, INCLUDE_PATH_EXPR, and EXCLUDE_PATH_EXPR all restrict the set of object types in the heterogeneous collection. By default, objects of all object types in the heterogeneous collection are retrieved.

Usage Notes

- Each call to SET_FILTER causes a WHERE condition to be added to the underlying query that fetches the set of objects. The WHERE conditions are ANDed together, so you can use multiple SET_FILTER calls to refine the set of objects to be returned. For example to specify that you want the object named EMP in schema SCOTT, do the following:

```
SET_FILTER(handle,'SCHEMA','SCOTT');
SET_FILTER(handle,'NAME','EMP');
```

- You can use the same text expression filter multiple times with different values. All the filter conditions will be applied to the query. For example, to get objects with names between Felix and Oscar, do the following:

```
SET_FILTER(handle,'NAME_EXPR','>=''FELIX''');
```

```
SET_FILTER(handle,'NAME_EXPR','<=''OSCAR''');
```

- With SET_FILTER, you can specify the schema of objects to be retrieved, but security considerations may override this specification. If the caller is SYS or has SELECT_CATALOG_ROLE, then any object can be retrieved; otherwise, only the following can be retrieved:
  - Schema objects owned by the current user
  - Public synonyms
  - System privileges granted to the current user or to PUBLIC
  - Grants on objects for which the current user is owner, grantor, or grantee (either explicitly or as PUBLIC).
  - SCHEMA_EXPORT where the name is the current user
  - TABLE_EXPORT where SCHEMA is the current user

  If you request objects that you are not privileged to retrieve, no exception is raised; the object is not retrieved, as if it did not exist.

  In stored procedures, functions, and definers-rights packages, roles (such as SELECT_CATALOG_ROLE) are disabled. Therefore, such a PL/SQL program can only fetch metadata for objects in its own schema. If you want to write a PL/SQL program that fetches metadata for objects in a different schema (based on the invoker's possession of SELECT_CATALOG_ROLE), you must make the program invokers-rights.
- For heterogeneous object types, the BEGIN_WITH and BEGIN_AFTER filters allow restart on an object type boundary. Appropriate filter values are returned by the FETCH_XML_CLOB procedure.

  Filters on heterogeneous objects provide default values for filters on object types within the collection. You can override this default for a particular object type by specifying the appropriate filter for the specific object type path. For example, for SCHEMA_EXPORT the NAME filter specifies the schema to be fetched including all the tables in the schema, but you can further restrict this set of tables by supplying a NAME_EXPR filter explicitly for the TABLE object type path. Table 57–18 lists valid object type path names for the major heterogeneous object types along with an explanation of the scope of each path name. (The same information is available in the following catalog views: DATABASE_EXPORT_OBJECTS, SCHEMA_EXPORT_OBJECTS, and TABLE_EXPORT_OBJECTS.) See Table 57–17 for filters defined for each path name. These path names are valid in the INCLUDE_PATH_EXPR and EXCLUDE_PATH_EXPR filters. Path names marked with an asterisk (*) are *only* valid in those filters; they cannot be used as values of the SET_FILTER object_type_path parameter.

Table 57–18 Object Type Path Names for Heterogeneous Object Types

Heterogeneous Type	Path Name (*=valid only in xxx_PATH_EXPR)	Scope
TABLE_EXPORT	AUDIT_OBJ	Object audits on the selected tables
TABLE_EXPORT	COMMENT	Table and column comments for the selected tables
TABLE_EXPORT	CONSTRAINT	Constraints (including referential constraints) on the selected tables
TABLE_EXPORT	*GRANT	Object grants on the selected tables

Table 57–18 (Cont.) Object Type Path Names for Heterogeneous Object Types

Heterogeneous Type	Path Name (*=valid only in xxx_PATH_EXPR)	Scope
TABLE_EXPORT	INDEX	Indexes (including domain indexes) on the selected tables
TABLE_EXPORT	OBJECT_GRANT	Object grants on the selected tables
TABLE_EXPORT	REF_CONSTRAINT	Referential (foreign key) constraints on the selected tables
TABLE_EXPORT	STATISTICS	Statistics on the selected tables
TABLE_EXPORT	TABLE_DATA	Row data for the selected tables
TABLE_EXPORT	TRIGGER	Triggers on the selected tables
SCHEMA_EXPORT	ASSOCIATION	Statistics type associations for objects in the selected schemas
SCHEMA_EXPORT	AUDIT_OBJ	Audits on all objects in the selected schemas
SCHEMA_EXPORT	CLUSTER	Clusters in the selected schemas and their indexes
SCHEMA_EXPORT	COMMENT	Comments on all objects in the selected schemas
SCHEMA_EXPORT	CONSTRAINT	Constraints (including referential constraints) on all objects in the selected schemas
SCHEMA_EXPORT	DB_LINK	Private database links in the selected schemas
SCHEMA_EXPORT	DEFAULT_ROLE	Default roles granted to users associated with the selected schemas
SCHEMA_EXPORT	DIMENSION	Dimensions in the selected schemas
SCHEMA_EXPORT	FUNCTION	Functions in the selected schemas and their dependent grants and audits
SCHEMA_EXPORT	*GRANT	Grants on objects in the selected schemas
SCHEMA_EXPORT	INDEX	Indexes (including domain indexes) on tables and clusters in the selected schemas
SCHEMA_EXPORT	INDEXTYPE	Indextypes in the selected schemas and their dependent grants and audits
SCHEMA_EXPORT	JAVA_SOURCE	Java sources in the selected schemas and their dependent grants and audits
SCHEMA_EXPORT	JOB	Jobs in the selected schemas
SCHEMA_EXPORT	LIBRARY	External procedure libraries in the selected schemas
SCHEMA_EXPORT	MATERIALIZED_VIEW	Materialized views in the selected schemas
SCHEMA_EXPORT	MATERIALIZED_VIEW_LOG	Materialized view logs on tables in the selected schemas
SCHEMA_EXPORT	OBJECT_GRANT	Grants on objects in the selected schemas
SCHEMA_EXPORT	OPERATOR	Operators in the selected schemas and their dependent grants and audits
SCHEMA_EXPORT	PACKAGE	Packages (both specification and body) in the selected schemas, and their dependent grants and audits
SCHEMA_EXPORT	PACKAGE_BODY	Package bodies in the selected schemas
SCHEMA_EXPORT	PACKAGE_SPEC	Package specifications in the selected schemas
SCHEMA_EXPORT	PASSWORD_HISTORY	The password history for users associated with the selected schemas

Table 57–18 (Cont.) Object Type Path Names for Heterogeneous Object Types

Heterogeneous Type	Path Name (*=valid only in xxx_PATH_EXPR)	Scope
SCHEMA_EXPORT	PROCEDURE	Procedures in the selected schemas and their dependent grants and audits
SCHEMA_EXPORT	REF_CONSTRAINT	Referential (foreign key) constraints on tables in the selected schemas
SCHEMA_EXPORT	REFRESH_GROUP	Refresh groups in the selected schemas
SCHEMA_EXPORT	SEQUENCE	Sequences in the selected schemas and their dependent grants and audits
SCHEMA_EXPORT	STATISTICS	Statistics on tables and indexes in the selected schemas
SCHEMA_EXPORT	SYNONYM	Private synonyms in the selected schemas
SCHEMA_EXPORT	TABLE	Tables in the selected schemas and their dependent objects (indexes, constraints, triggers, grants, audits, comments, table data, and so on)
SCHEMA_EXPORT	TABLE_DATA	Row data for tables in the selected schemas
SCHEMA_EXPORT	TABLESPACE_QUOTA	Tablespace quota granted to users associated with the selected schemas
SCHEMA_EXPORT	TRIGGER	Triggers on tables in the selected schemas
SCHEMA_EXPORT	TYPE	Types (both specification and body) in the selected schemas, and their dependent grants and audits
SCHEMA_EXPORT	TYPE_BODY	Type bodies in the selected schemas
SCHEMA_EXPORT	TYPE_SPEC	Type specifications in the selected schemas
SCHEMA_EXPORT	USER	User definitions for users associated with the selected schemas
SCHEMA_EXPORT	VIEW	Views in the selected schemas and their dependent objects (grants, constraints, comments, audits)
DATABASE_EXPORT	ASSOCIATION	Statistics type associations for objects in the database
DATABASE_EXPORT	AUDIT	Audits of SQL statements
DATABASE_EXPORT	AUDIT_OBJ	Audits on all objects in the database
DATABASE_EXPORT	CLUSTER	Clusters and their indexes
DATABASE_EXPORT	COMMENT	Comments on all objects
DATABASE_EXPORT	CONSTRAINT	Constraints (including referential constraints)
DATABASE_EXPORT	CONTEXT	Application contexts
DATABASE_EXPORT	DB_LINK	Private and public database links
DATABASE_EXPORT	DEFAULT_ROLE	Default roles granted to users in the database
DATABASE_EXPORT	DIMENSION	Dimensions in the database
DATABASE_EXPORT	DIRECTORY	Directory objects in the database
DATABASE_EXPORT	FGA_POLICY	Fine-grained audit policies
DATABASE_EXPORT	FUNCTION	Functions
DATABASE_EXPORT	* GRANT	Object and system grants
DATABASE_EXPORT	INDEX	Indexes (including domain indexes) on tables and clusters
DATABASE_EXPORT	INDEXTYPE	Indextypes and their dependent grants and audits

Table 57–18 (Cont.) Object Type Path Names for Heterogeneous Object Types

Heterogeneous Type	Path Name (*=valid only in xxx_PATH_EXPR)	Scope
DATABASE_EXPORT	JAVA_SOURCE	Java sources and their dependent grants and audits
DATABASE_EXPORT	JOB	Jobs
DATABASE_EXPORT	LIBRARY	External procedure libraries
DATABASE_EXPORT	MATERIALIZED_VIEW	Materialized views
DATABASE_EXPORT	MATERIALIZED_VIEW_LOG	Materialized view logs
DATABASE_EXPORT	OBJECT_GRANT	All object grants in the database
DATABASE_EXPORT	OPERATOR	Operators and their dependent grants and audits
DATABASE_EXPORT	PACKAGE	Packages (both specification and body) and their dependent grants and audits
DATABASE_EXPORT	PACKAGE_BODY	Package bodies
DATABASE_EXPORT	PACKAGE_SPEC	Package specifications
DATABASE_EXPORT	PASSWORD_HISTORY	Password histories for database users
DATABASE_EXPORT	*PASSWORD_VERIFY_FUNCTION	The password complexity verification function
DATABASE_EXPORT	PROCEDURE	Procedures and their dependent grants and objects
DATABASE_EXPORT	PROFILE	Profiles
DATABASE_EXPORT	PROXY	Proxy authentications
DATABASE_EXPORT	REF_CONSTRAINT	Referential (foreign key) constraints on tables in the database
DATABASE_EXPORT	REFRESH_GROUP	Refresh groups
DATABASE_EXPORT	*RESOURCE_ COST	Resource cost information
DATABASE_EXPORT	RLS_CONTEXT	Fine-grained access-control driving contexts
DATABASE_EXPORT	RLS_GROUP	Fine-grained access-control policy groups
DATABASE_EXPORT	RLS_POLICY	Fine-grained access-control policies
DATABASE_EXPORT	ROLE	Roles
DATABASE_EXPORT	ROLE_GRANT	Role grants to users in the database
DATABASE_EXPORT	ROLLBACK_SEGMENT	Rollback segments
DATABASE_EXPORT	*SCHEMA (named object)	Database schemas including for each schema all related and dependent objects: user definitions and their attributes (default roles, role grants, tablespace quotas, and so on), objects in the schema (tables, view, packages, types, and so on), and their dependent objects (grants, audits, indexes, constraints, and so on). The NAME and NAME_EXPR filters can be used with this object type path name to designate the database schemas to be fetched.
DATABASE_EXPORT	SEQUENCE	Sequences
DATABASE_EXPORT	STATISTICS	Statistics on tables and indexes
DATABASE_EXPORT	SYNONYM	Public and private synonyms
DATABASE_EXPORT	SYSTEM_GRANT	System privilege grants

Table 57–18 (Cont.) Object Type Path Names for Heterogeneous Object Types

Heterogeneous Type	Path Name (*=valid only in xxx_PATH_EXPR)	Scope
DATABASE_EXPORT	TABLE	Tables and their dependent objects (indexes, constraints, triggers, grants, audits, comments, table data, and so on)
DATABASE_EXPORT	TABLE_DATA	Row data for all tables
DATABASE_EXPORT	TABLESPACE	Tablespace definitions
DATABASE_EXPORT	TABLESPACE_QUOTA	Tablespace quota granted to users in the database
DATABASE_EXPORT	TRIGGER	Triggers on the database, on schemas, and on schema objects
DATABASE_EXPORT	TRUSTED_DB_LINK	Trusted links
DATABASE_EXPORT	TYPE	Types (both specification and body) and their dependent grants and audits
DATABASE_EXPORT	TYPE_BODY	Type bodies
DATABASE_EXPORT	TYPE_SPEC	Type specifications
DATABASE_EXPORT	USER	User definitions
DATABASE_EXPORT	VIEW	Views

Exceptions

- INVALID_ARGVAL. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- INVALID_OPERATION. SET_FILTER was called after the first call to FETCH_xxx for the OPEN context. After the first call to FETCH_xxx is made, no further calls to SET_FILTER are permitted.
- INCONSISTENT_ARGS. The arguments are inconsistent. Possible inconsistencies include the following:
  - filter name not valid for the object type associated with the OPEN context
  - filter name not valid for the object_type_path
  - object_type_path not part of the collection designated by handle
  - filter value is the wrong datatype

SET_PARSE_ITEM Procedure

This procedure is used for both retrieval and submission. This procedure enables output parsing and specifies an object attribute to be parsed and returned.

> **See Also:** For more information about related subprograms:
>
> - Subprograms for Retrieving Multiple Objects From the Database on page 57-9
> - Subprograms for Submitting XML to the Database on page 57-10

Syntax

The following syntax applies when SET_PARSE_ITEM is used for object retrieval:

```
DBMS_METADATA.SET_PARSE_ITEM (
   handle       IN  NUMBER,
   name         IN  VARCHAR2,
   object_type  IN  VARCHAR2 DEFAULT NULL);
```

The following syntax applies when SET_PARSE_ITEM is used for XML submission:

```
DBMS_METADATA.SET_PARSE_ITEM (
   handle     IN NUMBER,
   name        IN VARCHAR2);
```

Parameters

Table 57–19 SET_PARSE_ITEM Procedure Parameters

Parameter	Description
handle	The handle returned from OPEN (or OPENW).
name	The name of the object attribute to be parsed and returned. See Table 57–20 for the attribute object type, name, and meaning.
object_type	Designates the object type to which the parse item applies (this is an object type name, not a path name). By default, the parse item applies to the object type of the OPEN handle. When the OPEN handle designates a heterogeneous object type, behavior can be either of the following: ▪ if object_type is omitted, the parse item applies to all object types within the heterogeneous collection ▪ if object_type is specified, the parse item only applies to that specific object type within the collection This parameter only applies when SET_PARSE_ITEM is used for object retrieval.

Table 57–20 describes the object type, name, and meaning of the items available in the SET_PARSE_ITEM procedure.

Table 57–20 SET_PARSE_ITEM: Parse Items

Object Type	Name	Meaning
All objects	VERB	If FETCH_XML_CLOB is called, no value is returned. If FETCH_DDL is called, then for every row in the sys.ku$_ddls nested table returned by FETCH_DDL the verb in the corresponding ddlText is returned. If the ddlText is a SQL DDL statement, then the SQL verb (for example, CREATE, GRANT, AUDIT) is returned. If the ddlText is a procedure call (for example, DBMS_AQADM.CREATE_QUEUE_TABLE()) then the package.procedure-name is returned.
All objects	OBJECT_TYPE	If FETCH_XML_CLOB is called, an object type name from Table 57–11 is returned. If FETCH_DDL is called and the ddlText is a SQL DDL statement whose verb is CREATE or ALTER, the object type as used in the DDL statement is returned (for example, TABLE, PACKAGE_BODY, and so on). Otherwise, an object type name from Table 57–11 is returned.
Schema objects	SCHEMA	The object schema is returned. If the object is not a schema object, no value is returned.
Named objects	NAME	The object name is returned. If the object is not a named object, no value is returned.
TABLE, TABLE_DATA, INDEX	TABLESPACE	The name of the object's tablespace or, if the object is a partitioned table, the default tablespace is returned. For a TABLE_DATA object, this is always the tablespace where the rows are stored.
TRIGGER	ENABLE	If the trigger is enabled, ENABLE is returned. If the trigger is disabled, DISABLE is returned.
OBJECT_GRANT, TABLESPACE_QUOTA	GRANTOR	The grantor is returned.
Dependent objects (including domain index secondary tables)	BASE_OBJECT_NAME	The name of the base object is returned. If the object is not a dependent object, no value is returned.
Dependent objects (including domain index secondary tables)	BASE_OBJECT_SCHEMA	The schema of the base object is returned. If the object is not a dependent object, no value is returned.
Dependent objects (including domain index secondary tables)	BASE_OBJECT_TYPE	The object type of the base object is returned. If the object is not a dependent object, no value is returned.
Granted objects	GRANTEE	The grantee is returned. If the object is not a granted object, no value is returned.

Usage Notes

These notes apply when using SET_PARSE_ITEM to retrieve objects.

By default, the FETCH_xxx routines return an object's metadata as XML or creation DDL. By calling SET_PARSE_ITEM you can request that individual attributes of the object be returned as well.

You can call SET_PARSE_ITEM multiple times to ask for multiple items to be parsed and returned. Parsed items are returned in the sys.ku$_parsed_items nested table.

For TABLE_DATA objects, the following parse item return values are of interest:

If Object Is	NAME, SCHEMA	BASE_OBJECT_NAME, BASE_OBJECT_SCHEMA
nonpartitioned table	table name, schema	table name, schema
table partition	partition name, schema	table name, schema
nested table	storage table name, schema	name and schema of top-level table (*not* the parent nested table)

Tables are not usually thought of as dependent objects. However, secondary tables for domain indexes are dependent on the domain indexes. Consequently, the BASE_OBJECT_NAME, BASE_OBJECT_SCHEMA and BASE_OBJECT_TYPE parse items for secondary TABLE objects return the name, schema, and type of the domain index.

See Also:

- "FETCH_xxx Functions and Procedures" on page 57-18
- *Oracle Database Utilities* for information about using the Metadata API

By default, the CONVERT and PUT procedures simply transform an object's XML metadata to DDL. By calling SET_PARSE_ITEM you can request that individual attributes of the object be returned as well.

Exceptions

- INVALID_ARGVAL. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- INVALID_OPERATION. SET_PARSE_ITEM was called after the first call to FETCH_xxx for the OPEN context. After the first call to FETCH_xxx is made, no further calls to SET_PARSE_ITEM are permitted.
- INCONSISTENT_ARGS. The attribute name is not valid for the object type associated with the OPEN context.

SET_TRANSFORM_PARAM and SET_REMAP_PARAM Procedures

These procedures are used for both retrieval and submission. SET_TRANSFORM_PARAM and SET_REMAP_PARAM specify parameters to the XSLT stylesheet identified by transform_handle. Use them to modify or customize the output of the transform.

See Also: For more information about related subprograms:

- Subprograms for Retrieving Multiple Objects From the Database on page 57-9
- Subprograms for Submitting XML to the Database on page 57-10

Syntax

```
DBMS_METADATA.SET_TRANSFORM_PARAM (
   transform_handle   IN NUMBER,
   name               IN VARCHAR2,
   value              IN VARCHAR2,
   object_type        IN VARCHAR2 DEFAULT NULL);

DBMS_METADATA.SET_TRANSFORM_PARAM (
   transform_handle   IN NUMBER,
   name               IN VARCHAR2,
   value              IN BOOLEAN DEFAULT TRUE,
   object_type        IN VARCHAR2 DEFAULT NULL);

DBMS_METADATA.SET_TRANSFORM_PARAM (
   transform_handle   IN NUMBER,
   name               IN VARCHAR2,
   value              IN NUMBER,
   object_type        IN VARCHAR2 DEFAULT NULL);

DBMS_METADATA.SET_REMAP_PARAM (
   transform_handle   IN NUMBER,
   name               IN VARCHAR2,
   old_value          IN VARCHAR2,
   new_value          IN VARCHAR2,
   object_type        IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 57–21 describes the parameters for the SET_TRANSFORM_PARAM and SET_REMAP_PARAM procedures.

Table 57–21 SET_TRANSFORM_PARAM and SET_REMAP_PARAM Parameters

Parameters	Description
transform_handle	Either (1) the handle returned from ADD_TRANSFORM, or (2) the enumerated constant SESSION_TRANSFORM that designates the DDL transform for the whole session. Note that the handle returned by OPEN is not a valid transform handle. For SET_REMAP_PARAM, the transform handle must designate the MODIFY transform.

Table 57–21 (Cont.) SET_TRANSFORM_PARAM and SET_REMAP_PARAM Parameters

Parameters	Description
name	The name of the parameter. Table 57–22 lists the transform parameters defined for the DDL transform, specifying the object_type it applies to, its datatype, and its meaning or effect. This includes its default value, if any, and whether the parameter is additive. Table 57–23 describes the parameters for the MODIFY transform in the SET_TRANSFORM_PARAM procedure. Table 57–24 describes the parameters for the MODIFY transform in the SET_REMAP_PARAM procedure.
value	The value of the transform. This parameter is valid only for SET_TRANSFORM_PARAM.
old_value	The old value for the remapping. This parameter is valid only for SET_REMAP_PARAM.
new_value	The new value for the remapping. This parameter is valid only for SET_REMAP_PARAM.
object_type	Designates the object type to which the transform or remap parameter applies. By default, it applies to the same object type as the transform. In cases where the transform applies to all object types within a heterogeneous collection, the following apply: ▪ If object_type is omitted, the parameter applies to all applicable object types within the heterogeneous collection. ▪ If object_type is specified, the parameter only applies to that object type. This allows a caller who has added a transform to a heterogeneous collection to specify different transform parameters for different object types within the collection.

Table 57–22 describes the object type, name, datatype, and meaning of the parameters for the DDL transform in the SET_TRANSFORM_PARAM procedure.

Table 57–22 SET_TRANSFORM_PARAM: Transform Parameters for the DDL Transform

Object Type	Name	Datatype	Meaning
All objects	PRETTY	BOOLEAN	If TRUE, format the output with indentation and line feeds. Defaults to TRUE.
All objects	SQLTERMINATOR	BOOLEAN	If TRUE, append a SQL terminator (; or /) to each DDL statement. Defaults to FALSE.
TABLE	SEGMENT_ATTRIBUTES	BOOLEAN	If TRUE, emit segment attributes (physical attributes, storage attributes, tablespace, logging). Defaults to TRUE.
TABLE	STORAGE	BOOLEAN	If TRUE, emit storage clause. (Ignored if SEGMENT_ATTRIBUTES is FALSE.) Defaults to TRUE.
TABLE	TABLESPACE	BOOLEAN	If TRUE, emit tablespace. (Ignored if SEGMENT_ATTRIBUTES is FALSE.) Defaults to TRUE.
TABLE	CONSTRAINTS	BOOLEAN	If TRUE, emit all non-referential table constraints. Defaults to TRUE.
TABLE	REF_CONSTRAINTS	BOOLEAN	If TRUE, emit all referential constraints (foreign keys). Defaults to TRUE.

Table 57–22 (Cont.) SET_TRANSFORM_PARAM: Transform Parameters for the DDL Transform

Object Type	Name	Datatype	Meaning
TABLE	CONSTRAINTS_AS_ALTER	BOOLEAN	If TRUE, emit table constraints as separate ALTER TABLE (and, if necessary, CREATE INDEX) statements. If FALSE, specify table constraints as part of the CREATE TABLE statement. Defaults to FALSE. Requires that CONSTRAINTS be TRUE.
TABLE	OID	BOOLEAN	If TRUE, emit the OID clause for object tables. Defaults to FALSE.
TABLE	SIZE_BYTE_KEYWORD	BOOLEAN	If TRUE, emit the BYTE keyword as part of the size specification of CHAR and VARCHAR2 columns that use byte semantics. If FALSE, omit the keyword. Defaults to FALSE.
INDEX, CONSTRAINT, ROLLBACK_ SEGMENT, CLUSTER, TABLESPACE	SEGMENT_ATTRIBUTES	BOOLEAN	If TRUE, emit segment attributes (physical attributes, storage attributes, tablespace, logging). Defaults to TRUE.
INDEX, CONSTRAINT, ROLLBACK_ SEGMENT, CLUSTER	STORAGE	BOOLEAN	If TRUE, emit storage clause. (Ignored if SEGMENT_ATTRIBUTES is FALSE.) Defaults to TRUE.
INDEX, CONSTRAINT, ROLLBACK_ SEGMENT, CLUSTER	TABLESPACE	BOOLEAN	If TRUE, emit tablespace. (Ignored if SEGMENT_ ATTRIBUTES is FALSE.) Defaults to TRUE.
TYPE	SPECIFICATION	BOOLEAN	If TRUE, emit the type specification. Defaults to TRUE.
TYPE	BODY	BOOLEAN	If TRUE, emit the type body. Defaults to TRUE.
TYPE	OID	BOOLEAN	If TRUE, emit the OID clause. Defaults to FALSE.
PACKAGE	SPECIFICATION	BOOLEAN	If TRUE, emit the package specification. Defaults to TRUE.
PACKAGE	BODY	BOOLEAN	If TRUE, emit the package body. Defaults to TRUE.
VIEW	FORCE	BOOLEAN	If TRUE, use the FORCE keyword in the CREATE VIEW statement. Defaults to TRUE.
OUTLINE	INSERT	BOOLEAN	If TRUE, emit the INSERT statements into the OL$ dictionary tables that will create the outline and its hints. If FALSE, emit a CREATE OUTLINE statement. Defaults to FALSE. Note: This object type is being deprecated.
All objects	DEFAULT	BOOLEAN	Calling SET_TRANSFORM_PARAM with this parameter set to TRUE has the effect of resetting all parameters for the transform to their default values. Setting this FALSE has no effect. There is no default.

Table 57–22 (Cont.) SET_TRANSFORM_PARAM: Transform Parameters for the DDL Transform

Object Type	Name	Datatype	Meaning
All objects	INHERIT	BOOLEAN	If TRUE, inherits session-level parameters. Defaults to FALSE. If an application calls ADD_TRANSFORM to add the DDL transform, then by default the only transform parameters that apply are those explicitly set for that transform handle. This has no effect if the transform handle is the session transform handle.
ROLE	REVOKE_FROM	Text	The name of a user from whom the role must be revoked. If this is a non-null string and if the CREATE ROLE statement grants you the role, a REVOKE statement is emitted after the CREATE ROLE. Note: When you issue a CREATE ROLE statement, Oracle may grant you the role. You can use this transform parameter to undo the grant. Defaults to null string.
TABLESPACE	REUSE	BOOLEAN	If TRUE, include the REUSE parameter for datafiles in a tablespace to indicate that existing files can be reused. Defaults to FALSE.
CLUSTER, INDEX, ROLLBACK_SEGMENT, TABLE, TABLESPACE	PCTSPACE	NUMBER	A number representing the percentage by which space allocation for the object type is to be modified. The value is the number of one-hundreths of the current allocation. For example, 100 means 100%. If the object type is TABLESPACE, the following size values are affected: - in file specifications, the value of SIZE - MINIMUM EXTENT - EXTENT MANAGEMENT LOCAL UNIFORM SIZE For other object types, INITIAL and NEXT are affected.

Table 57–23 describes the object type, name, datatype, and meaning of the parameters for the MODIFY transform in the SET_TRANSFORM_PARAM procedure.

Table 57–23 SET_TRANSFORM_PARAM: Transform Parameters for the MODIFY Transform

Object Type	Name	Datatype	Meaning
All objects	OBJECT_ROW	NUMBER	A number designating the object row for an object. The object in the document that corresponds to this number will be copied to the output document. This parameter is additive. By default, all objects are copied to the output document.

Table 57–24 describes the object type, name, datatype, and meaning of the parameters for the MODIFY transform in the SET_REMAP_PARAM procedure.

Table 57–24 SET_REMAP_PARAM: Transform Parameters for the MODIFY Transform

Object Type	Name	Datatype	Meaning
LIBRARY, TABLESPACE, DIRECTORY	REMAP_DATAFILE	Text	Objects in the document will have their filespecs renamed as follows: any filespec matching old_value will be changed to new_value. Filespecs should *not* be enclosed in quotes. This parameter is additive. By default, filespecs are not renamed.
Schema Objects, Dependent Objects, Granted Objects, USER	REMAP_SCHEMA	Text	Any schema object in the document whose name matches old_value will have its schema name changed to new_value. Any dependent object whose base object schema name matches old_value will have its base object schema name changed to new_value. Any granted object whose grantee name matches old_value will have its grantee name changed to new_value. Any user whose name matches old_value will have its name changed to new_value. This parameter is additive. By default, schemas are not remapped.
TABLE, CLUSTER, CONSTRAINT, INDEX, ROLLBACK_SEGMENT, MATERIALIZED_VIEW, MATERIALIZED_VIEW_LOG, TABLESPACE_QUOTA	REMAP_TABLESPACE	Text	Objects in the document will have their tablespaces renamed as follows: any tablespace name matching old_value will be changed to new_value. This parameter is additive. By default, tablespaces are not remapped.

Exceptions

- INVALID_ARGVAL. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- INVALID_OPERATION. Either SET_TRANSFORM_PARAM or SET_REMAP_PARAM was called after the first call to FETCH_xxx for the OPEN context. After the first call to FETCH_xxx is made, no further calls to SET_TRANSFORM_PARAM or SET_REMAP_PARAM are permitted.
- INCONSISTENT_ARGS. The arguments are inconsistent. This can mean the following:
  - The transform parameter name is not valid for the object type associated with the OPEN context or for the transform associated with the transform handle.
  - The transform applies to all object types in a heterogeneous collection, but object_type is not part of the collection.

Usage Notes

XSLT allows parameters to be passed to stylesheets. You call SET_TRANSFORM_PARAM or SET_REMAP_PARAM to specify the value of a parameter to be passed to the stylesheet identified by transform_handle.

Normally, if you call SET_TRANSFORM_PARAMETER multiple times for the same parameter name, each call overrides the prior call. For example, the following sequence simply sets the STORAGE transform parameter to TRUE.

```
SET_TRANSFORM_PARAM(tr_handle,'STORAGE',false);
SET_TRANSFORM_PARAM(tr_handle,'STORAGE',true);
```

However, some transform parameters are additive which means that all specified parameter values are applied to the document, not just the last one. For example, the OBJECT_ROW parameter to the MODIFY transform is additive. If you specify the following, then both specified rows are copied to the output document.

```
SET_TRANSFORM_PARAM(tr_handle,'OBJECT_ROW',5);
SET_TRANSFORM_PARAM(tr_handle,'OBJECT_ROW',8);
```

The REMAP_TABLESPACE parameter is also additive. If you specify the following, then tablespaces TBS1 and TBS3 are changed to TBS2 and TBS4, respectively.

```
SET_REMAP_PARAM(tr_handle,'REMAP_TABLESPACE','TBS1','TBS2');
SET_REMAP_PARAM(tr_handle,'REMAP_TABLESPACE','TBS3','TBS4');
```

The order in which the transformations are performed is undefined. For example, if you specify the following, the result is undefined.

```
SET_REMAP_PARAM(tr_handle,'REMAP_TABLESPACE','TBS1','TBS2');
SET_REMAP_PARAM(tr_handle,'REMAP_TABLESPACE','TBS2','TBS3');
```

> **Note:** The number of remap parameters that can be specified for a MODIFY transform is limited to ten. That is, you can specify up to ten REMAP_DATAFILE parameters, up to ten REMAP_SCHEMA parameters and so on. Additional instances are ignored. To work around this, you can perform another DBMS_METADATA.ADD_TRANSFORM and specify additional remap parameters.

The GET_DDL, GET_DEPENDENT_DDL, and GET_GRANTED_DDL functions allow the casual browser to extract the creation DDL for an object. So that you can specify transform parameters, this package defines an enumerated constant SESSION_TRANSFORM as the handle of the DDL transform at the session level. You can call SET_TRANSFORM_PARAM using DBMS_METADATA.SESSION_TRANSFORM as the transform handle to set transform parameters for the whole session. GET_DDL, GET_DEPENDENT_DDL, and GET GRANTED_DDL inherit these parameters when they invoke the DDL transform.

> **Note:** The enumerated constant must be prefixed with the package name DBMS_METADATA.SESSION_TRANSFORM.

58

DBMS_MGWADM

DBMS_MGWADM defines the Messaging Gateway administrative interface. The package and object types are owned by SYS.

Note: You must run the catmgw.sql script to load the Messaging Gateway packages and types into the database.

See Also: *Oracle Streams Advanced Queuing User's Guide and Reference* contains information on loading database objects and using DBMS_MGWADM

This chapter contains the following topics:

Using DBMS_MGWADM

Constants

Table 58–1 DBMS_MGWADM Constants—Cleanup Actions

Name	Type	Description
CLEAN_STARTUP_STATE	CONSTANT BINARY_INTEGER	Sets the Messaging Gateway agent to a known state so that it can be started
CLEAN_LOG_QUEUES	CONSTANT BINARY_INTEGER	Messaging Gateway agent will clean log queues for all configured messaging system links
RESET_SUB_MISSING_LOG_REC	CONSTANT BINARY_INTEGER	Messaging Gateway agent recovers a Messaging Gateway subscriber that has failed due to a missing log record
RESET_SUB_MISSING_MESSAGE	CONSTANT BINARY_INTEGER	Messaging Gateway agent recovers a Messaging Gateway subscriber that has failed due to a missing persistent source message

Table 58–2 DBMS_MGWADM Constants—Force Values

Name	Type	Description
FORCE	CONSTANT BINARY_INTEGER	Represents a forced action
NO_FORCE	CONSTANT BINARY_INTEGER	Represents a normal, nonforced action

Table 58–3 DBMS_MGWADM Constants—Logging Levels

Name	Type	Description
BASIC_LOGGING	CONSTANT BINARY_INTEGER	The standard (the least) information written to the log file
TRACE_DEBUG_LOGGING	CONSTANT BINARY_INTEGER	The greatest information written to the log file
TRACE_HIGH_LOGGING	CONSTANT BINARY_INTEGER	The third level of detail of logging information written to the log file
TRACE_LITE_LOGGING	CONSTANT BINARY_INTEGER	The second level detail of logging information written to the log file

Table 58–4 DBMS_MGWADM Constants—Named Property Constants

Name	Type	Description
MGWPROP_PREFIX	CONSTANT VARCHAR2	A constant (MGWPROP$_) for the reserved property name prefix
MGWPROP_REMOVE	CONSTANT VARCHAR2	A constant (MGWPROP$_REMOVE) for the reserved property name used to remove an existing property
MGWPROP_REMOVE_ALL	CONSTANT VARCHAR2	A constant (MGWPROP$_REMOVE_ALL) for the reserved property name used to remove all properties

Table 58–5 DBMS_MGWADM Constants—Other Constants

Name	Type	Description
JMS_CONNECTION	CONSTANT BINARY_INTEGER	Used to indicate that JMS connections will be used to access JMS destinations in a domain-independent manner that supports a unified messaging model
JMS_QUEUE_CONNECTION	CONSTANT BINARY_INTEGER	Used to indicate that JMS queue connections will be used to access JMS destinations
JMS_TOPIC_CONNECTION	CONSTANT BINARY_INTEGER	Used to indicate that JMS topic connections will be used to access JMS destinations
NO_CHANGE	CONSTANT VARCHAR2	Indicates that an existing value should be preserved (not changed). This is used for certain APIs where the desire is to change one or more parameters but leave others unchanged.

Table 58–6 DBMS_MGWADM Constants—Propagation Types

Name	Type	Description
INBOUND_PROPAGATION	CONSTANT BINARY_INTEGER	Represents the propagation type for non-Oracle to Oracle Streams AQ propagation. The propagation source is a queue in a foreign (non-Oracle) messaging system and the destination is a local Oracle Streams AQ queue.
OUTBOUND_PROPAGATION	CONSTANT BINARY_INTEGER	Represents the propagation type for Oracle Streams AQ to non-Oracle propagation. The propagation source is a local Oracle Streams AQ queue and the destination is a queue in a foreign (non-Oracle) messaging system.

Table 58–7 DBMS_MGWADM Constants—Queue Domain Types

Name	Type	Description
DOMAIN_QUEUE	CONSTANT BINARY_INTEGER	Represents a queue destination. A JMS queue (point-to-point model) is classified as a queue.
DOMAIN_TOPIC	CONSTANT BINARY_INTEGER	Represents a topic destination. A JMS topic (publish-subscribe model) is classified as a topic.

Table 58–8 DBMS_MGWADM Constants—Shutdown Modes

Name	Type	Description
SHUTDOWN_IMMEDIATE	CONSTANT BINARY_INTEGER	Represents the immediate shutdown mode
SHUTDOWN_NORMAL	CONSTANT BINARY_INTEGER	Represents the normal shutdown mode

Table 58–9 DBMS_MGWADM Constants—WebSphere MQ Interface Types

Name	Type	Description
MQSERIES_BASE_JAVA_INTERFACE	CONSTANT BINARY_INTEGER	Represents the Base Java interface for the WebSphere MQ messaging system

Views

The views listed in Table 58–10 provide Messaging Gateway configuration, status, and statistical information. Unless otherwise indicated, the SELECT privilege is granted to MGW_ADMINISTRATOR_ROLE so that only Messaging Gateway administrators have access to the views. All views are owned by SYS.

Table 58–10 Database Views

Name	Description
MGW_GATEWAY View	Configuration and status information for Messaging Gateway
MGW_LINKS View	Names and types of messaging system links currently created
MGW_MQSERIES_LINKS View	Messaging system properties for WebSphere MQ links
MGW_TIBRV_LINKS View	Messaging system properties for TIB/Rendezvous links
MGW_FOREIGN_QUEUES View	Queue properties of registered queues
MGW_SUBSCRIBERS View	Subscriber properties, status, and statistical information
MGW_SCHEDULES View	Schedule properties and status

MGW_GATEWAY View

This view lists configuration and status information for Messaging Gateway, as shown in Table 58–11.

Table 58–11 MGW_GATEWAY View Properties

Name	Type	Description
AGENT_DATABASE	VARCHAR2	The database connect string used by the Messaging Gateway agent. NULL indicates that a local connection is used.
AGENT_INSTANCE	NUMBER	The database instance on which the Messaging Gateway agent is currently running. This should be NULL if the agent is not running.
AGENT_JOB	NUMBER	Job number of the queued job used to start the Messaging Gateway agent process. The job number is set when Messaging Gateway is started and cleared when it shuts down.
AGENT_PING	VARCHAR2	Gateway agent ping status. Values: ■ NULL means no ping attempt was made. ■ REACHABLE means ping attempt was successful. ■ UNREACHABLE means ping attempt failed. AGENT_PING attempts to contact the Messaging Gateway agent. There is a short delay (up to 5 seconds) if the ping attempt fails. No ping is attempted if the AGENT_STATUS is NOT_STARTED or START_SCHEDULED.
AGENT_START_TIME	TIMESTAMP	The time when the Messaging Gateway agent job currently running was started. This should be NULL if the agent is not running.

Table 58–11 (Cont.) MGW_GATEWAY View Properties

Name	Type	Description
AGENT_STATUS	VARCHAR2	Status of the Messaging Gateway agent. Values: ■ NOT_STARTED means the Messaging Gateway agent has not been started ■ START_SCHEDULED means Messaging Gateway agent has been scheduled to start. That is, Messaging Gateway has been started using DBMS_MGWADM.STARTUP, but the queued job used to start the Messaging Gateway agent has not yet run. ■ STARTING means Messaging Gateway agent is starting. That is, Messaging Gateway has been started using DBMS_MGWADM.STARTUP, the queued job has run, and the Messaging Gateway agent is starting up. ■ INITIALIZING means the Messaging Gateway agent has started and is initializing ■ RUNNING means the Messaging Gateway agent is running ■ SHUTTING_DOWN means the Messaging Gateway agent is shutting down ■ BROKEN means an unexpected condition has been encountered that prevents the Messaging Gateway agent from starting. DBMS_MGWADM.CLEANUP_GATEWAY must be called before the agent can be started.
AGENT_USER	VARCHAR2	Database username used by the Messaging Gateway agent to connect to the database
LAST_ERROR_DATE	DATE	Date of last Messaging Gateway agent error. The last error information is cleared when Messaging Gateway is started. It is set if the Messaging Gateway agent fails to start or terminates due to an abnormal condition.
LAST_ERROR_MSG	VARCHAR2	Message for last Messaging Gateway agent error
LAST_ERROR_TIME	VARCHAR2	Time of last Messaging Gateway agent error
MAX_CONNECTIONS	NUMBER	Maximum number of messaging connections to Oracle Database
MAX_MEMORY	NUMBER	Maximum heap size used by the Messaging Gateway agent (in MB)
MAX_THREADS	NUMBER	Maximum number of messaging threads created by the Messaging Gateway agent

MGW_LINKS View

This view lists the names and types of messaging system links currently defined. Table 58–12 lists the MGW_LINKS view properties.

Table 58–12 MGW_LINKS View Properties

Name	Type	Description
LINK_COMMENT	VARCHAR2	User comment for the link
LINK_NAME	VARCHAR2	Name of the messaging system link
LINK_TYPE	VARCHAR2	Type of messaging system link. Values ■ MQSERIES is for WebSphere MQ links. ■ TIBRV is for TIB/Rendezvous links.

MGW_MQSERIES_LINKS View

This view lists information for the WebSphere MQ messaging system links. The view includes most of the messaging system properties specified when the link is created. Table 58–13 lists the MGW_MQSERIES_LINKS view properties.

Table 58–13 MGW_MQSERIES_LINKS View Properties

Name	Type	Description
CHANNEL	VARCHAR2	Connection channel
HOSTNAME	VARCHAR2	Name of the WebSphere MQ host
INBOUND_LOG_QUEUE	VARCHAR2	Inbound propagation log queue
INTERFACE_TYPE	VARCHAR2	Messaging interface type. Values: ■ BASE_JAVA is for WebSphere MQ Base Java interface ■ JMS_CONNECTION is for WebSphere MQ JMS unified, domain-independent connections ■ JMS_QUEUE_CONNECTION is for WebSphere MQ JMS queue connections ■ JMS_TOPIC_CONNECTION is for WebSphere MQ JMS topic connections
LINK_COMMENT	VARCHAR2	User comment for the link
LINK_NAME	VARCHAR2	Name of the messaging system link
MAX_CONNECTIONS	NUMBER	Maximum number of messaging connections
OPTIONS	SYS.MGW_ PROPERTIES	Link options
OUTBOUND_LOG_QUEUE	VARCHAR2	Outbound propagation log queue
PORT	NUMBER	Port number
QUEUE_MANAGER	VARCHAR2	Name of the WebSphere MQ queue manager

MGW_TIBRV_LINKS View

This view lists information for TIB/Rendezvous messaging system links. The view includes most of the messaging system properties specified when the link was created. Table 58–14 lists the MGW_TIBRV_LINKS view properties.

Table 58–14 MGW_TIBRV_LINKS View Properties

Property Name	Type	Description
CM_LEDGER	VARCHAR2	TIB/Rendezvous CM ledger file name
CM_NAME	VARCHAR2	TIB/Rendezvous CM correspondent name
DAEMON	VARCHAR2	TIB/Rendezvous daemon parameter for rvd transport
LINK_COMMENT	VARCHAR2	User comment for the link
LINK_NAME	VARCHAR2	Name of the messaging system link
NETWORK	VARCHAR2	TIB/Rendezvous network parameter for rvd transport
OPTIONS	SYS.MGW_ PROPERTIES	Link options
SERVICE	VARCHAR2	TIB/Rendezvous service parameter for rvd transport

MGW_FOREIGN_QUEUES View

This view lists information for foreign queues. The view includes most of the queue properties specified when the queue is registered. Table 58–15 lists the MGW_FOREIGN_QUEUES view properties.

Table 58–15 MGW_FOREIGN_QUEUES View Properties

Name	Type	Description
DOMAIN	VARCHAR2	Queue domain type. Values: ■ NULL means the queue domain type is automatically determined by the messaging system ■ QUEUE is for a queue (point-to-point) model ■ TOPIC is for a topic (publish-subscribe) model
LINK_NAME	VARCHAR2	Name of the messaging system link
NAME	VARCHAR2	Name of the registered queue
OPTIONS	SYS.MGW_PROPERTIES	Optional queue properties
PROVIDER_QUEUE	VARCHAR2	Message provider (native) queue name
QUEUE_COMMENT	VARCHAR2	User comment for the foreign queue

MGW_SUBSCRIBERS View

This view lists configuration and status information for Messaging Gateway subscribers. The view includes most of the subscriber properties specified when the subscriber is added, as well as other status and statistical information. Table 58–16 lists the MGW_SUBSCRIBERS view properties.

Table 58–16 MGW_SUBSCRIBERS View Properties

Name	Type	Description
DESTINATION	VARCHAR2	Destination queue to which messages are propagated
EXCEPTIONQ_MSGS	NUMBER	Number of messages moved to the propagation exception queue since the last time the agent was started
EXCEPTION_QUEUE	VARCHAR2	Exception queue used for logging purposes
FAILURES	NUMBER	Number of propagation failures
LAST_ERROR_DATE	DATE	Date of last propagation error
LAST_ERROR_MSG	VARCHAR2	Message for last propagation error
LAST_ERROR_TIME	VARCHAR2	Time of last propagation error
OPTIONS	SYS.MGW_PROPERTIES	Subscriber options
PROP_STYLE	VARCHAR2	Message propagation style. Values: ■ NATIVE is for native message propagation ■ JMS is for JMS message propagation
PROPAGATED_MSGS	NUMBER	Number of messages propagated to the destination queue since the last time the agent was started

Table 58–16 (Cont.) MGW_SUBSCRIBERS View Properties

Name	Type	Description
PROPAGATION_TYPE	VARCHAR2	Propagation type. Values: ■ OUTBOUND is for Oracle Streams AQ to non-Oracle propagation ■ INBOUND is for non-Oracle to Oracle Streams AQ propagation
QUEUE_NAME	VARCHAR2	Subscriber source queue
RULE	VARCHAR2	Subscription rule
STATUS	VARCHAR2	Subscriber status. Values: ■ ENABLED means the subscriber is enabled ■ DELETE_PENDING means subscriber removal is pending, usually because DBMS_MGWADM.REMOVE_SUBSCRIBER has been called but certain cleanup tasks pertaining to this subscriber are still outstanding
SUBSCRIBER_ID	VARCHAR2	Propagation subscriber identifier
TRANSFORMATION	VARCHAR2	Transformation used for message conversion

MGW_SCHEDULES View

This view lists configuration and status information for Messaging Gateway schedules. The view includes most of the schedule properties specified when the schedule is created, as well as other status information. Table 58–17 lists the MGW_SCHEDULES view properties.

Table 58–17 MGW_SCHEDULES View Properties

Name	Type	Description
DESTINATION	VARCHAR2	Propagation destination
LATENCY	NUMBER	Propagation window latency (in seconds)
NEXT_TIME	VARCHAR2	Reserved for future use
PROPAGATION_TYPE	VARCHAR2	Propagation type. Values: ■ OUTBOUND is for Oracle Streams AQ to non-Oracle propagation ■ INBOUND is for non-Oracle to Oracle Streams AQ propagation
PROPAGATION_WINDOW	NUMBER	Reserved for future use
SCHEDULE_DISABLED	VARCHAR2	Indicates whether the schedule is disabled. Y means the schedule is disabled. N means the schedule is enabled.
SCHEDULE_ID	VARCHAR2	Propagation schedule identifier
SOURCE	VARCHAR2	Propagation source
START_DATE	DATE	Reserved for future use
START_TIME	VARCHAR2	Reserved for future use

Data Structures

The DBMS_MGWADM package defines the following OBJECT types.

Object Types

- SYS.MGW_MQSERIES_PROPERTIES Object Type
- SYS.MGW_PROPERTIES Object Type
- SYS.MGW_PROPERTY Object Type
- SYS.MGW_TIBRV_PROPERTIES Object Type

SYS.MGW_MQSERIES_PROPERTIES Object Type

This type specifies basic properties for a WebSphere MQ messaging system link.

Syntax

```
TYPE SYS.MGW_MQSERIES_PROPERTIES IS OBJECT (
   queue_manager       VARCHAR2(64),
   hostname            VARCHAR2(64),
   port                INTEGER,
   channel             VARCHAR2(64),
   interface_type      INTEGER,
   max_connections     INTEGER,
   username            VARCHAR2(64),
   password            VARCHAR2(64),
   inbound_log_queue   VARCHAR2(64),
   outbound_log_queue  VARCHAR2(64),

-- Methods
STATIC FUNCTION construct
RETURN SYS.MGW_MQSERIES_PROPERTIES,

STATIC FUNCTION alter_construct
RETURN SYS.MGW_MQSERIES_PROPERTIES );
```

Attributes

Table 58–18 SYS.MGW_MQSERIES_PROPERTIES Attributes

Attribute	Description
queue_manager	The name of the WebSphere MQ queue manager
hostname	The host on which the WebSphere MQ messaging system resides. If hostname is NULL, then a WebSphere MQ bindings connection is used. If not NULL, then a client connection is used and requires that a port and channel be specified.
port	The port number. This is used only for client connections; that is, when hostname is not NULL.
channel	The channel used when establishing a connection to the queue manager. This is used only for client connections; that is, when hostname is not NULL.
interface_type	The type of messaging interface to use. Values: ■ DBMS_MGWADM.MQSERIES_BASE_JAVA_INTERFACE if the WebSphere MQ Base Java interface should be used. ■ DBMS_MGWADM.JMS_CONNECTION if the link is to be used to access JMS destinations in a unified, domain-independent manner. ■ DBMS_MGWADM.JMS_QUEUE_CONNECTION if the link is to be used for accessing JMS queues ■ DBMS_MGWADM.JMS_TOPIC_CONNECTION if the link is to be used for accessing JMS topics.
max_connections	The maximum number of messaging connections to the WebSphere MQ messaging system
username	The username used for authentication to the WebSphere MQ messaging system

Table 58–18 (Cont.) SYS.MGW_MQSERIES_PROPERTIES Attributes

Attribute	Description
password	The password used for authentication to the WebSphere MQ messaging system
inbound_log_queue	The name of the WebSphere MQ queue used for propagation recovery purposes when this messaging link is used for inbound propagation; that is, when queues associated with this link serve as a propagation source: ■ For MQSERIES_BASE_JAVA_INTERFACE, this is the name of a physical WebSphere MQ queue created using WebSphere MQ administration tools. ■ For the JMS_CONNECTION interface and the JMS_QUEUE_CONNECTION interface, this is the name of a physical WebSphere MQ queue created using WebSphere MQ administration tools. ■ For JMS_TOPIC_CONNECTION interface, this specifies the name of a WebSphere MQ JMS topic. The physical WebSphere MQ queue used by subscribers of that topic must be created using WebSphere MQ administration tools. By default, the physical queue used is SYSTEM.JMS.D.SUBSCRIBER.QUEUE.
outbound_log_queue	The name of the WebSphere MQ queue used for propagation recovery purposes when this messaging link is used for outbound propagation; that is, when queues associated with this link serve as a propagation destination: ■ For MQSERIES_BASE_JAVA_INTERFACE, this is the name of a physical WebSphere MQ queue created using WebSphere MQ administration tools. ■ For the JMS_CONNECTION interface and the JMS_QUEUE_CONNECTION interface, this is the name of a physical WebSphere MQ queue created using WebSphere MQ administration tools. ■ For JMS_TOPIC_CONNECTION interface, this specifies the name of a WebSphere MQ JMS topic. The physical WebSphere MQ queue used by subscribers of that topic must be created using WebSphere MQ administration tools. By default, the physical queue used is SYSTEM.JMS.D.SUBSCRIBER.QUEUE.

Methods

Table 58–19 SYS.MGW_MQSERIES_PROPERTIES Methods

Method	Description
construct	Constructs a new SYS.MGW_MQSERIES_PROPERTIES instance. All attributes are assigned a value of NULL
alter_construct	Constructs a new SYS.MGW_MQSERIES_PROPERTIES instance for altering the properties of an existing messaging link. All attributes having a VARCHAR2 data type are assigned a value of DBMS_MGWADM.NO_CHANGE. Attributes of other data types are assigned a value of NULL.

SYS.MGW_PROPERTIES Object Type

This type specifies an array of properties.

Syntax

```
TYPE SYS.MGW_PROPERTIES AS VARRAY (2000) OF SYS.MGW_PROPERTY;
```

Attributes

Table 58–20 SYS.MGW_PROPERTIES Attributes

Attribute	Description
`name`	Property name
`value`	Property value

Usage Notes

Unless noted otherwise, Messaging Gateway uses named properties as follows:

- Names with the `MGWPROP$_` prefix are reserved. They are used for special purposes and are invalid when used as a normal property name.
- A property name can exist only once in a property list; that is, a list can contain only one value for a given name. The name is case-insensitive.
- In general, a property list is order-independent, and the property names may appear in any order. An alter property list is an exception.
- You can use a new property list to alter an existing property list. Each new property modifies the original list in one of the following ways: adds a new property, modifies a property, removes a property, or removes all properties.

The alter list is processed in order, from the first element to the last element. Thus the order in which the elements appear in the alter list is meaningful, especially when the alter list is used to remove properties from an existing list.

The property name and value are used to determine how that element affects the original list. The following rules apply:

- Add or modify property

  ```
  MGW_PROPERTY.NAME = property_name
  MGW_PROPERTY.VALUE = property_value
  ```

  If a property of the given name already exists, then the current value is replaced with the new value; otherwise the new property is added to the end of the list.

- Remove property

  ```
  MGW_PROPERTY.NAME = 'MGWPROP$_REMOVE'
  MGW_PROPERTY.VALUE = name_of_property_to_remove
  ```

  No action is taken if the property name does not exist in the original list.

- Remove all properties

  ```
  MGW_PROPERTY.NAME = 'MGWPROP$_REMOVE_ALL'
  MGW_PROPERTY.VALUE = not used
  ```

See Also: "The DBMS_MGWADM package defines constants to represent the reserved property names on Table 58–4, " DBMS_MGWADM Constants—Named Property Constants"

SYS.MGW_PROPERTY Object Type

This type specifies a named property which is used to specify optional properties for messaging links, foreign queues, and subscribers.

Syntax

```
TYPE SYS.MGW_PROPERTY IS OBJECT(
   name  VARCHAR2(500),
   value VARCHAR2(4000),

-- Methods
STATIC FUNCTION construct    --- (1)
RETURN SYS.MGW_PROPERTY,

STATIC FUNCTION construct(   --- (2)
   p_name   IN VARCHAR2,
   p_value  IN VARCHAR2)
RETURN SYS.MGW_PROPERTY );
```

Attributes

Table 58–21 SYS.MGW_PROPERTY Attributes

Attribute	Description
`name`	Property name
`value`	Property value

Methods

Table 58–22 SYS.MGW_PROPERTY Methods

Method	Description
`construct --- (1)`	Constructs a new `MGW_PROPERTY` instance. All attributes are assigned a value of `NULL`
`construct --- (2)`	Constructs a new `MGW_PROPERTY` instance initialized using the given parameters

SYS.MGW_TIBRV_PROPERTIES Object Type

A type that specifies basic properties for a TIB/Rendezvous messaging system link. The Messaging Gateway agent creates a TIB/Rendezvous transport of type `TibrvRvdTransport` for each Messaging Gateway link.

Syntax

```
TYPE SYS.MGW_TIBRV_PROPERTIES IS OBJECT(
   service   VARCHAR2(128),
   daemon    VARCHAR2(128),
   network   VARCHAR2(256),
   cm_name   VARCHAR2(256),
   cm_ledger VARCHAR2(256),

-- Methods
STATIC FUNCTION construct
RETURN SYS.MGW_TIBRV_PROPERTIES,

STATIC FUNCTION alter_construct
RETURN SYS.MGW_TIBRV_PROPERTIES );
```

Attributes

Table 58–23 SYS.MGW_TIBRV_PROPERTIES Attributes

Attribute	Description
`service`	The service parameter for the rvd transport
`daemon`	The daemon parameter for the rvd transport
`network`	The network parameter for the rvd transport
`cm_name`	The CM correspondent name. Reserved for future use.
`cm_ledger`	The CM ledger file name. Reserved for future use.

Methods

Table 58–24 SYS.MGW_TIBRV_PROPERTIES Methods

Method	Description
`construct`	Constructs a new `SYS.MGW_TIBRV_PROPERTIES` instance. All attributes will be assigned a value of `NULL`.
`alter_construct`	Constructs a new `SYS.MGW_TIBRV_PROPERTIES` instance. This function is useful for altering the properties of an existing messaging link. All attributes having a `VARCHAR2` data type will be assigned a value of `DBMS_MGWADM.NO_CHANGE`. Attributes of other data types will be assigned a value of `NULL`.

Summary of DBMS_MGWADM Subprograms

Table 58–25 DBMS_MGWADM Package Subprograms

Subprogram	Description
ADD_SUBSCRIBER Procedure on page 58-20	Adds a subscriber used to consume messages from a source queue for propagation to a destination
ALTER_AGENT Procedure on page 58-23	Alters Messaging Gateway agent parameters
ALTER_MSGSYSTEM_LINK Procedure for TIB/Rendezvous on page 58-24	Alters the properties of a TIB/Rendezvous messaging system link
ALTER_MSGSYSTEM_LINK Procedure for WebSphere MQ on page 58-25	Alters the properties of a WebSphere MQ messaging system link
ALTER_PROPAGATION_ SCHEDULE Procedure on page 58-26	Alters a propagation schedule
ALTER_SUBSCRIBER Procedure on page 58-27	Alters the parameters of a subscriber used to consume messages from a source queue for propagation to a destination
CLEANUP_GATEWAY Procedure on page 58-29	Cleans up Messaging Gateway
CREATE_MSGSYSTEM_LINK Procedure for TIB/Rendezvous on page 58-32	Creates a messaging system link to a TIB/Rendezvous messaging system
CREATE_MSGSYSTEM_LINK Procedure for WebSphere MQ on page 58-33	Creates a messaging system link to a WebSphere MQ messaging system
DB_CONNECT_INFO Procedure on page 58-34	Configures connection information used by the Messaging Gateway agent for connections to Oracle Database
DISABLE_PROPAGATION_ SCHEDULE Procedure on page 58-35	Disables a propagation schedule
ENABLE_PROPAGATION_ SCHEDULE Procedure on page 58-36	Enables a propagation schedule
REGISTER_FOREIGN_QUEUE Procedure on page 58-37	Registers a non-Oracle queue entity in Messaging Gateway
REMOVE_MSGSYSTEM_LINK Procedure on page 58-38	Removes a messaging system link for a non-Oracle messaging system
REMOVE_SUBSCRIBER Procedure on page 58-39	Removes a subscriber used to consume messages from a source queue for propagation to a destination
RESET_SUBSCRIBER Procedure on page 58-40	Resets the propagation error state for a subscriber
SCHEDULE_PROPAGATION Procedure on page 58-41	Schedules message propagation from a source to a destination
SET_LOG_LEVEL Procedure on page 58-43	Dynamically alters the Messaging Gateway agent logging level

Table 58–25 (Cont.) DBMS_MGWADM Package Subprograms

Subprogram	Description
SHUTDOWN Procedure on page 58-44	Shuts down the Messaging Gateway agent
STARTUP Procedure on page 58-45	Starts the Messaging Gateway agent
UNREGISTER_FOREIGN_QUEUE Procedure on page 58-46	Removes a non-Oracle queue entity in Messaging Gateway
UNSCHEDULE_PROPAGATION Procedure on page 58-47	Removes a propagation schedule

ADD_SUBSCRIBER Procedure

This procedure adds a subscriber used to consume messages from a source queue for propagation to a destination.

Syntax

```
DBMS_MGWADM.ADD_SUBSCRIBER(
   subscriber_id     IN VARCHAR2,
   propagation_type  IN BINARY_INTEGER,
   queue_name        IN VARCHAR2,
   destination       IN VARCHAR2,
   rule              IN VARCHAR2 DEFAULT NULL,
   transformation    IN VARCHAR2 DEFAULT NULL,
   exception_queue   IN VARCHAR2 DEFAULT NULL
   options           IN SYS.MGW_PROPERTIES DEFAULT NULL);
```

Parameters

Table 58–26 ADD_SUBSCRIBER Procedure Parameters

Parameter	Description
subscriber_id	Specifies a user-defined name that identifies this subscriber
propagation_type	Specifies the type of message propagation. DBMS_MGWADM.OUTBOUND_PROPAGATION is for Oracle Streams AQ to non-Oracle propagation. DBMS_MGWADM.INBOUND_PROPAGATION is for non-Oracle to Oracle Streams AQ propagation
queue_name	Specifies the source queue to which this subscriber is being added. The syntax and interpretation of this parameter depend on the value specified for propagation_type.
destination	Specifies the destination queue to which messages consumed by this subscriber are propagated. The syntax and interpretation of this parameter depend on the value specified for propagation_type.
rule	Specifies an optional subscription rule used by the subscriber to dequeue messages from the source queue. This is NULL if no rule is needed. The syntax and interpretation of this parameter depend on the value specified for propagation_type.
transformation	Specifies the transformation needed to convert between the Oracle Streams AQ payload and an ADT defined by Messaging Gateway. The type of transformation needed depends on the value specified for propagation_type. If NULL, then the Oracle Streams AQ payload type must be supported by Messaging Gateway.
exception_queue	Specifies a queue used for exception message logging purposes. This queue must be on the same messaging system as the propagation source. If NULL, then an exception queue is not used and propagation stops if a problem occurs. The syntax and interpretation of this parameter depend on the value specified for propagation_type. The source queue and exception queue cannot be the same queue.
options	Optional subscriber properties. NULL if there are none. Typically these are lesser used configuration properties supported by the messaging system.

Usage Notes

> **See Also:** "Handling Arbitrary Payload Types Using Message Transformations", in *Oracle Streams Advanced Queuing User's Guide and Reference* for more information regarding message conversion and transformation

If the non-Oracle messaging link being accessed for the subscriber uses a JMS interface, then the Messaging Gateway agent will use the Oracle JMS interface to access the Oracle Streams AQ queues. Otherwise the native Oracle Streams AQ interface will be used. Parameters are interpreted differently when the Messaging Gateway agent uses Oracle JMS for JMS connections.

Transformations are not currently supported if the Oracle JMS interface is used for propagation. The transformation parameter must be `NULL`.

> **See Also:** For additional information regarding subscriber options
>
> - "WebSphere MQ System Properties" in *Oracle Streams Advanced Queuing User's Guide and Reference*
> - "TIB/Rendezvous System Properties" in *Oracle Streams Advanced Queuing User's Guide and Reference*

OUTBOUND_PROPAGATION Subscribers

The parameters for a subscriber used for outbound propagation are interpreted as follows:

- `queue_name` specifies the local Oracle Streams AQ queue that is the propagation source. This must have a syntax of `schema.queue`.
- `destination` specifies the foreign queue to which messages are propagated. This must have a syntax of `registered_queue@message_link`.
- `rule` specifies an optional Oracle Streams AQ subscriber rule if the native Oracle Streams AQ interface is used, or a JMS selector if the Oracle JMS interface is used. If `NULL`, then no rule or selector is used.
- `transformation` specifies the transformation used to convert the Oracle Streams AQ payload to an ADT defined by Messaging Gateway.

  Messaging Gateway propagation dequeues messages from the Oracle Streams AQ queue using the transformation to convert the Oracle Streams AQ payload to a known ADT defined by Messaging Gateway. The message is then enqueued in the foreign messaging system based on the Messaging Gateway ADT.
- `exception_queue` specifies the name of a local Oracle Streams AQ queue to which messages are moved if an exception occurs. This must have a syntax of `schema.queue`.

If the native Oracle Streams AQ interface is used, then a subscriber will be added to the Oracle Streams AQ queue when this procedure is called, whether or not Messaging Gateway is running. The local subscriber will be of the form `sys.aq$_agent('MGW_`*`subscriber_id`*`',NULL, NULL)`.

If the Oracle JMS interface is used, then the Messaging Gateway agent will create a JMS durable subscriber with the name of `MGW_`*`subscriber_id`*. If the agent is not running when this procedure is called, then the durable subscriber will be created the next time the agent starts.

The exception queue has the following caveats:

- The user is responsible for creating the Oracle Streams AQ queue to be used as the exception queue.
- The payload type of the source and exception queue must match.
- The exception queue must be created as a queue type of `DBMS_AQADM.NORMAL_QUEUE` rather than `DBMS_AQADM.EXCEPTION_QUEUE`. Enqueue restrictions prevent Messaging Gateway propagation from using an Oracle Streams AQ queue of type `EXCEPTION_QUEUE` as a Messaging Gateway exception queue.

INBOUND_PROPAGATION Subscribers

The parameters for a subscriber used for inbound propagation are interpreted as follows:

- `queue_name` specifies the foreign queue that is the propagation source. This must have a syntax of `registered_queue@message_link`.
- `destination` specifies the local Oracle Streams AQ queue to which messages are propagated. This must have a syntax of `schema.queue`.
- `rule` specifies an optional subscriber rule that is valid for the foreign messaging system. This is `NULL` if no rule is needed.
- `transformation` specifies the transformation used to convert an ADT defined by Messaging Gateway to the Oracle Streams AQ payload type.

  Messaging Gateway propagation dequeues messages from the foreign messaging system and converts the message body to a known ADT defined by Messaging Gateway. The transformation is used to convert the Messaging Gateway ADT to an Oracle Streams AQ payload type when the message is enqueued to the Oracle Streams AQ queue.
- `exception_queue` specifies the name of a foreign queue to which messages are moved if an exception occurs. This must have a syntax of `registered_queue@message_link`.

Whether or not a subscriber is needed depends on the requirements of the non-Oracle messaging system. If a durable subscriber is necessary, then it will be created by the Messaging Gateway agent. If the agent is not running at the time this procedure is called, then the creation of the subscriber on the non-Oracle messaging system will occur when the agent next starts.

The exception queue has the following caveats:

- The exception queue must be a registered non-Oracle queue.
- The source and exception queues must use the same messaging system link.

ALTER_AGENT Procedure

This procedure configures Messaging Gateway agent parameters.

Syntax

```
DBMS_MGWADM.ALTER_AGENT (
   max_connections  IN BINARY_INTEGER DEFAULT NULL,
   max_memory       IN BINARY_INTEGER DEFAULT NULL,
   max_threads      IN BINARY_INTEGER DEFAULT NULL);
```

Parameters

Table 58–27 ALTER_AGENT Procedure Parameters

Parameter	Description
max_connections	The maximum number of messaging connections to Oracle Database used by the Messaging Gateway agent. If it is NULL, then the current value is unchanged.
max_memory	The maximum heap size, in MB, used by the Messaging Gateway agent. If it is NULL, then the current value is unchanged.
max_threads	The number of messaging threads that the Messaging Gateway agent creates. If it is NULL, then the current value is unchanged.

Usage Notes

Default values for these configuration parameters are set when the Messaging Gateway agent is installed.

Changes to the max_memory and max_threads parameters take effect the next time the Messaging Gateway agent is active. If the Messaging Gateway agent is currently active, then it must be shut down and restarted for the changes to take effect.

ALTER_MSGSYSTEM_LINK Procedure for TIB/Rendezvous

Alters the properties of a TIB/Rendezvous messaging system link.

Syntax

```
DBMS_MGWADM.ALTER_MSGSYSTEM_LINK (
   linkname      IN  VARCHAR2,
   properties    IN  SYS.MGW_TIBRV_PROPERTIES,
   options       IN  SYS.MGW_PROPERTIES DEFAULT NULL,
   comment       IN  VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE );
```

Parameters

Table 58–28 ALTER_MSGSYSTEM_LINK Procedure Parameters for TIB/Rendezvous

Parameters	Description
`linkname`	The messaging system link name
`properties`	Basic properties for a TIB/Rendezvous messaging system link. If `NULL`, then no link properties will be changed.
`options`	Optional link properties. If `NULL`, then no options will be changed. If not `NULL`, then the properties specified in this list are combined with the current options properties to form a new set of link options.
`comment`	A user-specified description, or `NULL` if one is not desired. If `DBMS_MGWADM.NO_CHANGE`, then the current value will not be changed.

Usage Notes

To retain an existing value for a messaging link property with a `VARCHAR2` data type, specify `DBMS_MGWADM.NO_CHANGE` for that particular property. To preserve an existing value for a property of another data type, specify `NULL` for that property.

The `options` parameter specifies a set of properties used to alter the current optional properties. Each property affects the current property list in a particular manner: add a new property, replace an existing property, remove an existing property, or remove all properties.

See Also: SYS.MGW_PROPERTIES Object Type on page 58-14

Some properties cannot be modified, and this procedure will fail if an attempt is made to alter such a property. For properties and options that can be changed, a few are dynamic, and Messaging Gateway uses the new values immediately. Others require the Messaging Gateway agent to be shut down and restarted before they take effect.

See Also: "TIB/Rendezvous System Properties" in *Oracle Streams Advanced Queuing User's Guide and Reference* for more information about the messaging system properties and options

ALTER_MSGSYSTEM_LINK Procedure for WebSphere MQ

This procedure alters the properties of a WebSphere MQ messaging system link.

Syntax

```
DBMS_MGWADM.ALTER_MSGSYSTEM_LINK (
   linkname   IN  VARCHAR2,
   properties IN  SYS.MGW_MQSERIES_PROPERTIES,
   options    IN  SYS.MGW_PROPERTIES DEFAULT NULL,
   comment    IN  VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE);
```

Parameters

Table 58–29 ALTER_MSGSYSTEM_LINK Procedure Parameters for WebSphere MQ

Parameters	Description
`linkname`	The messaging system link name
`properties`	Basic properties for a WebSphere MQ messaging system link. If it is `NULL`, then no link properties are changed.
`options`	Optional link properties. `NULL` if no options are changed. If not `NULL`, then the properties specified in this list are combined with the current options properties to form a new set of link options.
`comment`	An optional description or `NULL` if not desired. If `DBMS_MGWADM.NO_CHANGE` is specified, then the current value is not changed.

Usage Notes

To retain an existing value for a messaging link property with a `VARCHAR2` data type, specify `DBMS_MGWADM.NO_CHANGE` for that particular property. To preserve an existing value for a property of another data type, specify `NULL` for that property.

The `options` parameter specifies a set of properties used to alter the current optional properties. Each property affects the current property list in a particular manner: add a new property, replace an existing property, remove an existing property, or remove all properties.

See Also: SYS.MGW_PROPERTIES Object Type on page 58-14

Some properties cannot be modified, and this procedure will fail if an attempt is made to alter such a property. For properties and options that can be changed, a few are dynamic, and Messaging Gateway uses the new values immediately. Others require the Messaging Gateway agent to be shut down and restarted before they take effect.

See Also: "WebSphere MQ System Properties" in *Oracle Streams Advanced Queuing User's Guide and Reference* for more information about the messaging system properties and options

ALTER_PROPAGATION_SCHEDULE Procedure

This procedure alters a propagation schedule.

Syntax

```
DBMS_MGWADM.ALTER_PROPAGATION_SCHEDULE (
   schedule_id  IN VARCHAR2,
   duration     IN NUMBER DEFAULT NULL,
   next_time    IN VARCHAR2 DEFAULT NULL,
   latency      IN NUMBER DEFAULT NULL);
```

Parameters

Table 58–30 ALTER_PROPAGATION_SCHEDULE Procedure Parameters

Parameter	Description
`schedule_id`	Identifies the propagation schedule to be altered
`duration`	Reserved for future use
`next_time`	Reserved for future use
`latency`	Specifies the polling interval, in seconds, used by the Messaging Gateway agent when checking for messages in the source queue. If no messages are available in the source queue, then the agent will not poll again until the polling interval has passed. Once the agent detects a message it will continue propagating messages as long as any are available. Values: `NULL` or value > 0. If `latency` is `NULL`, then the Messaging Gateway agent default polling interval will be used. The default polling interval is 5 seconds, but it can be overridden by the Messaging Gateway initialization file.

Usage Notes

This procedure always overwrites the existing value for each parameter. If a given parameter is not specified, then the existing values are overwritten with the default value.

ALTER_SUBSCRIBER Procedure

This procedure alters the parameters of a subscriber used to consume messages from a source queue for propagation to a destination.

Syntax

```
DBMS_MGWADM.ALTER_SUBSCRIBER (
   subscriber_id    IN VARCHAR2,
   rule             IN VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE,
   transformation   IN VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE,
   exception_queue  IN VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE,
   options          IN SYS.MGW_PROPERTIES DEFAULT NULL );
```

Parameters

Table 58–31 ALTER_SUBSCRIBER Procedure Parameters

Parameter	Description
subscriber_id	Identifies the subscriber to be altered
rule	Specifies an optional subscription rule used by the subscriber to dequeue messages from the source queue. The syntax and interpretation of this parameter depend on the subscriber propagation type. A NULL value indicates that no subscription rule is needed. If DBMS_MGWADM.NO_CHANGE, then the current value is unchanged.
transformation	Specifies the transformation needed to convert between the Oracle Streams AQ payload and an ADT defined by Messaging Gateway. The type of transformation needed depends on the subscriber propagation type. A NULL value indicates that no transformation is needed. If DBMS_MGWADM.NO_CHANGE, then the current value is unchanged.
exception_queue	Specifies a queue used for exception message logging. This queue must be on the same messaging system as the propagation source. If no exception queue is associated with the subscriber, then propagation stops if a problem occurs. The syntax and interpretation of this parameter depend on the subscriber propagation type. A NULL value indicates that no exception queue is used. If DBMS_MGWADM.NO_CHANGE, then the current value is unchanged. The source queue and exception queue cannot be the same queue.
options	Optional subscriber properties. If NULL, then no options will be changed. If not NULL, then the properties specified in this list are combined with the current optional properties to form a new set of subscriber options.

Usage Notes

If the non-Oracle messaging link being accessed for the subscriber uses a JMS interface, then the Messaging Gateway agent will use the Oracle JMS interface to access the Oracle Streams AQ queues. Otherwise the native Oracle Streams AQ

interface will be used. Parameters are interpreted differently when the Messaging Gateway agent uses Oracle JMS for JMS connections.

When propagating from a JMS source, the subscriber rule cannot be altered. Instead, the subscriber must be removed and added with the new rule. For JMS, changing the message selector on a durable subscription is equivalent to deleting and re-creating the subscription.

Transformations are not currently supported if the Oracle JMS interface is used for propagation. The transformation parameter must be `DBMS_MGWADM.NO_CHANGE` (the default value).

The `options` parameter specifies a set of properties used to alter the current optional properties. Each property affects the current property list in a particular manner: add a new property, replace an existing property, remove an existing property, or remove all properties.

See Also:

- SYS.MGW_PROPERTIES Object Type on page 58-14 for more information on the options parameter
- "WebSphere MQ System Properties" in *Oracle Streams Advanced Queuing User's Guide and Reference* for more information about WebSphere MQ subscriber options
- "TIB/Rendezvous System Properties" in *Oracle Streams Advanced Queuing User's Guide and Reference* for more information about TIB/Rendezvous subscriber options
- "OUTBOUND_PROPAGATION Subscribers on page 58-21 for outbound propagation parameter interpretation
- "INBOUND_PROPAGATION Subscribers on page 58-22 for inbound propagation parameter interpretation

CLEANUP_GATEWAY Procedure

This procedure cleans up Messaging Gateway. The procedure performs cleanup or recovery actions that may be needed when Messaging Gateway is left in some abnormal or unexpected condition. The MGW_GATEWAY view lists Messaging Gateway status and configuration information that pertains to the cleanup actions.

Syntax

```
DBMS_MGWADM.CLEANUP_GATEWAY(
   action  IN BINARY_INTEGER,
   sarg    IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 58–32 CLEANUP_GATEWAY Procedure Parameters

Parameter	Description
action	The cleanup action to be performed. Values: ■ DBMS_MGWADM.CLEAN_STARTUP_STATE for Messaging Gateway start up state recovery. ■ DBMS_MGWADM.CLEAN_LOG_QUEUES for log queue cleanup. ■ DBMS_MGWADM.RESET_SUB_MISSING_LOG_REC for subscriber recovery due to missing log record. ■ DBMS_MGWADM.RESET_SUB_MISSING_MESSAGE for subscriber recovery due to missing message.
sarg	Optional argument whose meaning depends on the value specified for action. This should be NULL if it is not used for the specified action.

Usage Notes

CLEAN_STARTUP_STATE

sarg is not used and must be NULL.

The CLEAN_STARTUP_STATE action recovers Messaging Gateway to a known state when the Messaging Gateway agent has crashed or some other abnormal event occurs, and Messaging Gateway cannot be restarted. This should be done only when the Messaging Gateway agent has been started but appears to have crashed or has been nonresponsive for an extended period of time.

The CLEAN_STARTUP_STATE action may be needed when the MGW_GATEWAY view shows that the AGENT_STATUS value is something other than NOT_STARTED or START_SCHEDULED, and the AGENT_PING value is UNREACHABLE for an extended period of time.

If the AGENT_STATUS value is BROKEN, then the Messaging Gateway agent cannot be started until the problem has been resolved and the CLEAN_STARTUP_STATE action used to reset the agent status. A BROKEN status can indicate that the Messaging Gateway start job detected a Messaging Gateway agent already running. This condition that should never occur under normal use.

Cleanup tasks include:

- Removing the queued job used to start the external Messaging Gateway agent process.
- Setting certain configuration information to a known state. For example, setting the agent status to NOT_STARTED.

Execution of this command fails if:

- The agent status is NOT_STARTED or START_SCHEDULED.
- No shutdown attempt has been made prior to calling this procedure, except if the agent status is STARTING.
- The Messaging Gateway agent is successfully contacted.

  The assumption is that the agent is active, and this procedure fails. If the agent does not respond after several attempts have been made, then the cleanup tasks are performed. This procedure takes at least several seconds and possibly up to one minute. This is expected behavior under conditions where this particular cleanup action is appropriate and necessary.

> **Note:** Terminate any Messaging Gateway agent process that may still be running after a CLEAN_STARTUP_STATE action has been successfully performed. This should be done before calling DBMS_MGWADM.STARTUP to start Messaging Gateway. The process is usually named extprocmgwextproc.

CLEAN_LOG_QUEUES

sarg is not used and must be NULL.

The Messaging Gateway agent will clean log queues for all configured messaging system links. The agent will temporarily stop all propagation activity and then remove all obsolete and bad log records from the log queues for all links. The procedure will fail if the Messaging Gateway agent is not running.

This cleanup action is automatically performed each time the Messaging Gateway agent is started.

> **Note:** For Oracle Database 10*g*, the CLEAN_LOG_QUEUES action is performed only on agent startup. If this procedure is called when the agent is running, then the Messaging Gateway agent ignores it.

RESET_SUB_MISSING_LOG_REC

sarg specifies a Messaging Gateway subscriber ID to be reset. It must be not NULL.

The Messaging Gateway agent recovers a Messaging Gateway subscriber that has failed due to a missing log record. The agent will reset the source and destination log records. The procedure will fail if the Messaging Gateway agent is not running.

> **Caution:** If the messages in the source queue had already been propagated to the destination queue, then this action may result in duplicate messages.

RESET_SUB_MISSING_MESSAGE

sarg specifies a Messaging Gateway subscriber ID to be reset. It must be not NULL.

The Messaging Gateway agent recovers a Messaging Gateway subscriber that has failed due to a missing persistent source message. The agent will treat the message as a non-persistent message and continue processing that subscriber. The procedure will fail if the Messaging Gateway agent is not running.

CREATE_MSGSYSTEM_LINK Procedure for TIB/Rendezvous

Creates a link to a TIB/Rendezvous messaging system.

Syntax

```
DBMS_MGWADM.CREATE_MSGSYSTEM_LINK (
   linkname     IN  VARCHAR2,
   properties   IN  SYS.MGW_TIBRV_PROPERTIES,
   options      IN  SYS.MGW_PROPERTIES DEFAULT NULL,
   comment      IN  VARCHAR2 DEFAULT NULL );
```

Parameters

Table 58–33 CREATE_MSGSYSTEM_LINK Procedure Parameters for TIB/Rendezvous

Parameter	Description
`linkname`	A user-defined name to identify this messaging system link
`properties`	Basic properties of a TIB/Rendezvous messaging system link.
`options`	Optional link properties. `NULL` if there are none. These are less frequently used configuration properties supported by the messaging system
`comment`	A user-specified description. `NULL` if one is not desired.

Usage Notes

See Also: "TIB/Rendezvous System Properties" in *Oracle Streams Advanced Queuing User's Guide and Reference* for more information about the messaging system properties and options

CREATE_MSGSYSTEM_LINK Procedure for WebSphere MQ

This procedure creates a messaging system link to a WebSphere MQ messaging system.

Syntax

```
DBMS_MGWADM.CREATE_MSGSYSTEM_LINK(
   linkname    IN VARCHAR2,
   properties  IN SYS.MGW_MQSERIES_PROPERTIES,
   options     IN SYS.MGW_PROPERTIES DEFAULT NULL,
   comment     IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 58–34 CREATE_MSGSYSTEM_LINK Procedure Parameters for WebSphere MQ

Parameter	Description
`linkname`	A user-defined name to identify the messaging system link
`properties`	Basic properties of a WebSphere MQ messaging system link
`options`	Optional link properties. `NULL` if there are none. These are less frequently used configuration properties supported by the messaging system.
`comment`	A user-specified description. `NULL` if one is not desired

Usage Notes

See Also: "WebSphere MQ System Properties" in *Oracle Streams Advanced Queuing User's Guide and Reference* for more information about the messaging system properties and options

DB_CONNECT_INFO Procedure

This procedure configures connection information used by the Messaging Gateway agent for connections to Oracle Database.

Syntax

```
DBMS_MGWADM.DB_CONNECT_INFO (
   username      IN VARCHAR2,
   password      IN VARCHAR2,
   database      IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 58–35 DB_CONNECT_INFO Procedure Parameters

Parameter	Description
username	The username used for connections to Oracle Database. NULL is not allowed
password	The password used for connections to Oracle Database. NULL is not allowed
database	The database connect string used by the Messaging Gateway agent. NULL indicates that a local connection should be used. Oracle strongly recommends that a not NULL value be specified. Usually it will be a net service name from tnsnames.ora.

Usage Notes

The Messaging Gateway agent connects to Oracle Database as the user configured by this procedure. An Oracle administrator should create the user, grant it the role MGW_AGENT_ROLE, and then call this procedure to configure Messaging Gateway. Role MGW_AGENT_ROLE is used to grant this user special privileges needed to access Messaging Gateway configuration information stored in the database, enqueue or dequeue messages to and from Oracle Streams AQ queues, and perform certain Oracle Streams AQ administration tasks.

DISABLE_PROPAGATION_SCHEDULE Procedure

This procedure disables a propagation schedule.

Syntax

```
DBMS_MGWADM.DISABLE_PROPAGATION_SCHEDULE (
   schedule_id  IN VARCHAR2 );
```

Parameters

Table 58–36 DISABLE_PROPAGATION_SCHEDULE Procedure Parameters

Parameter	Description
schedule_id	Identifies the propagation schedule to be disabled

ENABLE_PROPAGATION_SCHEDULE Procedure

This procedure enables a propagation schedule.

Syntax

```
DBMS_MGWADM.ENABLE_PROPAGATION_SCHEDULE (
   schedule_id  IN VARCHAR2 );
```

Parameters

Table 58–37 ENABLE_PROPAGATION_SCHEDULE Procedure Parameters

Parameter	Description
schedule_id	Identifies the propagation schedule to be enabled

REGISTER_FOREIGN_QUEUE Procedure

This procedure registers a non-Oracle queue entity in Messaging Gateway.

Syntax

```
DBMS_MGWADM.REGISTER_FOREIGN_QUEUE(
   name            IN VARCHAR2,
   linkname        IN VARCHAR2,
   provider_queue  IN VARCHAR2 DEFAULT NULL,
   domain          IN INTEGER DEFAULT NULL,
   options         IN SYS.MGW_PROPERTIES DEFAULT NULL,
   comment         IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 58–38 REGISTER_FOREIGN_QUEUE Procedure Parameters

Parameters	Description
`name`	The registered queue name. This name identifies the foreign queue within Messaging Gateway and need not match the name of the queue in the foreign messaging system.
`linkname`	The link name for the messaging system on which this queue exists
`provider_queue`	The message provider (native) queue name. If `NULL`, then the value provided for the `name` parameter is used as the provider queue name.
`domain`	The domain type of the queue. `NULL` means the domain type is automatically determined based on the messaging system of the queue. `DBMS_MGWADM.DOMAIN_QUEUE` is for a queue (point-to-point model). `DBMS_MGWADM.DOMAIN_TOPIC` is for a topic (publish-subscribe model)
`options`	Optional queue properties
`comment`	A user-specified description. Can be `NULL`.

Usage Notes

This procedure does not create the physical queue in the non-Oracle messaging system. The non-Oracle queue must be created using the administration tools for that messaging system.

> **See Also:** For more information when registering queues for the WebSphere MQ messaging system or the TIB/Rendezvous messaging system, specifically "Optional Foreign Queue Configuration Properties" in *Oracle Streams Advanced Queuing User's Guide and Reference*.

REMOVE_MSGSYSTEM_LINK Procedure

This procedure removes a messaging system link for a non-Oracle messaging system.

Syntax

```
DBMS_MGWADM.REMOVE_MSGSYSTEM_LINK(
   linkname  IN VARCHAR2);
```

Parameters

Table 58–39 REMOVE_MSGSYSTEM_LINK Procedure Parameters

Parameters	Description
`linkname`	The messaging system link name

Usage Notes

All registered queues associated with this link must be removed before the messaging system link can be removed. This procedure fails if there is a registered foreign (non-Oracle) queue that references this link.

REMOVE_SUBSCRIBER Procedure

This procedure removes a subscriber used to consume messages from a source queue for propagation to a destination.

Syntax

```
DBMS_MGWADM.REMOVE_SUBSCRIBER (
   subscriber_id  IN VARCHAR2,
   force          IN BINARY_INTEGER DEFAULT DBMS_MGWADM.NO_FORCE );
```

Parameters

Table 58–40 REMOVE_SUBSCRIBER Procedure Parameters

Parameter	Description
subscriber_id	Identifies the subscriber to be removed
force	Specifies whether this procedure should succeed even if Messaging Gateway is not able to perform all cleanup actions pertaining to this subscriber. DBMS_MGWADM.NO_FORCE (0) means the subscriber is not removed if Messaging Gateway is unable to clean up successfully. DBMS_MGWADM.FORCE (1) means the subscriber is removed, even though all cleanup actions may not be done. The Messaging Gateway agent uses various resources of Oracle Database and the non-Oracle messaging system for its propagation work. These resources are typically associated with each subscriber and need to be released when the subscriber is no longer needed. Therefore, this procedure should only be called when the Messaging Gateway agent is running and able to access the non-Oracle messaging system associated with this subscriber.

Usage Notes

For outbound propagation, a local subscriber is removed from the Oracle Streams AQ queue.

RESET_SUBSCRIBER Procedure

This procedure resets the propagation error state for a subscriber.

Syntax

```
DBMS_MGWADM.RESET_SUBSCRIBER (
   subscriber_id  IN VARCHAR2 );
```

Parameters

Table 58–41 RESET_SUBSCRIBER Procedure Parameters

Parameter	Description
`subscriber_id`	Identifies the subscriber

SCHEDULE_PROPAGATION Procedure

This procedure schedules message propagation from a source to a destination. The schedule must be enabled and Messaging Gateway started in order for messages to be propagated.

Syntax

```
DBMS_MGWADM.SCHEDULE_PROPAGATION (
   schedule_id       IN VARCHAR2,
   propagation_type  IN BINARY_INTEGER,
   source            IN VARCHAR2,
   destination       IN VARCHAR2,
   start_time        IN DATE DEFAULT SYSDATE,
   duration          IN NUMBER DEFAULT NULL,
   next_time         IN VARCHAR2 DEFAULT NULL,
   latency           IN NUMBER DEFAULT NULL);
```

Parameters

Table 58–42 SCHEDULE_PROPAGATION Procedure Parameters

Parameter	Description
schedule_id	Specifies a user-defined name that identifies the schedule
propagation_type	Specifies the type of message propagation. DBMS_MGWADM.OUTBOUND_PROPAGATION is for Oracle Streams AQ to non-Oracle propagation. DBMS_MGWADM.INBOUND_PROPAGATION is for non-Oracle to Oracle Streams AQ propagation.
source	Specifies the source queue whose messages are to be propagated. The syntax and interpretation of this parameter depend on the value specified for propagation_type.
destination	Specifies the destination queue to which messages are propagated. The syntax and interpretation of this parameter depend on the value specified for propagation_type.
start_time	Reserved for future use
duration	Reserved for future use
next_time	Reserved for future use
latency	Specifies the polling interval, in seconds, used by the Messaging Gateway agent when checking for messages in the source queue. If no messages are available in the source queue, then the agent will not poll again until the polling interval has passed. Once the agent detects a message it will continue propagating messages as long as any are available. Values: NULL or value > 0. If latency is NULL, then the Messaging Gateway agent default polling interval will be used. The default polling interval is 5 seconds but it can be overridden by the Messaging Gateway initialization file.

Usage Notes

For outbound propagation, parameters are interpreted as follows:

- source specifies the local Oracle Streams AQ queue from which messages are propagated. This must have a syntax of schema.queue.

- `destination` specifies the foreign queue to which messages are propagated. This must have a syntax of `registered_queue@message_link`.

For inbound propagation, parameters are interpreted as follows:

- `source` specifies the foreign queue from which messages are propagated. This must have a syntax of `registered_queue@message_link`.
- `destination` specifies the local Oracle Streams AQ queue to which messages are propagated. This must have a syntax of `schema.queue`.

The schedule is set to an enabled state when it is created.

SET_LOG_LEVEL Procedure

This procedure dynamically alters the Messaging Gateway agent logging level. The Messaging Gateway agent must be running.

Syntax

```
DBMS_MGWADM.SET_LOG_LEVEL (
   log_level  IN BINARY_INTEGER);
```

Parameters

Table 58–43 SET_LOG_LEVEL Procedure Parameters

Parameter	Description
`log_level`	Level at which the Messaging Gateway agent logs information. `DBMS_MGWADM.BASIC_LOGGING` generates the least information while `DBMS_MGWADM.TRACE_DEBUG_LOGGING` generates the most information.

See Also: Table 58–3, " DBMS_MGWADM Constants—Logging Levels" on page 58-3 for details on the `log_level` parameter

SHUTDOWN Procedure

This procedure shuts down the Messaging Gateway agent. No propagation activity occurs until Messaging Gateway is restarted.

Syntax

```
DBMS_MGWADM.SHUTDOWN (
   sdmode  IN BINARY_INTEGER DEFAULT DBMS_MGWADM.SHUTDOWN_NORMAL);
```

Parameters

Table 58–44 SHUTDOWN Procedure Parameters

Parameter	Description
sdmode	The shutdown mode. The only value currently supported is DBMS_MGWADM.SHUTDOWN_NORMAL for normal shutdown. The Messaging Gateway agent may attempt to complete any propagation work currently in progress.

STARTUP Procedure

This procedure starts the Messaging Gateway agent. It must be called before any propagation activity can take place.

Syntax

```
DBMS_MGWADM.STARTUP(
   instance  IN  BINARY_INTEGER DEFAULT 0,
   force     IN  BINARY_INTEGER DEFAULT DBMS_MGWADM.NO_FORCE);
```

Parameters

Table 58–45 STARTUP Procedure Parameters

Parameter	Description
`instance`	Specifies which instance can run the job queue job used to start the Messaging Gateway agent. If this is zero, then the job can be run by any instance.
`force`	If this is `DBMS_MGWADM.FORCE`, then any positive integer is acceptable as the job instance. If this is `DBMS_MGWADM.NO_FORCE` (the default), then the specified instance must be running; otherwise the routine raises an exception.

Usage Notes

The Messaging Gateway agent cannot be started until an agent user has been configured using `DBMS_MGWADM.DB_CONNECT_INFO`.

This procedure submits a job queue job, which starts the Messaging Gateway agent when it runs. The `instance` and `force` parameters are used for job queue affinity, which you use to indicate whether a particular instance or any instance can run a submitted job.

UNREGISTER_FOREIGN_QUEUE Procedure

This procedure removes a non-Oracle queue entity in Messaging Gateway.

Syntax

```
DBMS_MGWADM.UNREGISTER_FOREIGN_QUEUE(
   name         IN VARCHAR2,
   linkname     IN VARCHAR2);
```

Parameters

Table 58–46 UNREGISTER_FOREIGN_QUEUE Procedure Parameters

Parameter	Description
`name`	The queue name
`linkname`	The link name for the messaging system on which the queue exists

Usage Notes

This procedure does not remove the physical queue in the non-Oracle messaging system.

All subscribers and schedules referencing this queue must be removed before it can be unregistered. This procedure fails if a subscriber or propagation schedule references the non-Oracle queue.

UNSCHEDULE_PROPAGATION Procedure

This procedure removes a propagation schedule.

Syntax

```
DBMS_MGWADM.UNSCHEDULE_PROPAGATION (
   schedule_id   IN VARCHAR2 );
```

Parameters

Table 58–47 UNSCHEDULE_PROPAGATION Procedure Parameters

Parameter	Description
schedule_id	Identifies the propagation schedule to be removed

59

DBMS_MGWMSG

DBMS_MGWMSG provides:

- Object types used by the canonical message types to convert message bodies.
- Methods, constants, and subprograms for working with Messaging Gateway message types.

> **See Also:** Chapter 58, "DBMS_MGWADM" which describes the Messaging Gateway administrative interface, DBMS_MGWADM

This chapter contains the following topics:

Using DBMS_MGWMSG

Security Model

The EXECUTE privilege is granted to PUBLIC on all types defined in the DBMS_MGWMSG package as well as the canonical types. The DBMS_MGWMSG packages and object types are owned by SYS.

Note: You must run the catmgw.sql script to load the Messaging Gateway packages and object types into the database. Refer to the *Oracle Streams Advanced Queuing User's Guide and Reference* for information on loading database objects and using DBMS_MGWMSG.

Constants

Table 59–1 DBMS_MGWMSG Constants: Value Types and Constants Representing the Type of Value for a SYS.MGW_NAME_VALUE_T Object

Value	Constant
TEXT_VALUE	CONSTANT BINARY_INTEGER := 1
RAW_VALUE	CONSTANT BINARY_INTEGER := 2
BOOLEAN_VALUE	CONSTANT BINARY_INTEGER := 3
BYTE_VALUE	CONSTANT BINARY_INTEGER := 4
SHORT_VALUE	CONSTANT BINARY_INTEGER := 5
INTEGER_VALUE	CONSTANT BINARY_INTEGER := 6
LONG_VALUE	CONSTANT BINARY_INTEGER := 7
FLOAT_VALUE	CONSTANT BINARY_INTEGER := 8
DOUBLE_VALUE	CONSTANT BINARY_INTEGER := 9
DATE_VALUE	CONSTANT BINARY_INTEGER := 10

Table 59–2 DBMS_MGWMSG Constants: Boolean Values—Constants Representing a Boolean as a Numeric Value

Value	Constant
BOOLEAN_FALSE	CONSTANT BINARY_INTEGER := 0
BOOLEAN_TRUE	CONSTANT BINARY_INTEGER := 1

Table 59–3 DBMS_MGWMSG Constants: Case Comparisons

Value	Constant
CASE_SENSITIVE	CONSTANT BINARY_INTEGER := 0
CASE_INSENSITIVE	CONSTANT BINARY_INTEGER := 1

Table 59–4 Constants for the TIB/Rendezvous field type

Value	Constant
TIBRVMSG_BOOL	CONSTANT INTEGER := 1
TIBRVMSG_F32	CONSTANT INTEGER := 2
TIBRVMSG_F64	CONSTANT INTEGER := 3
TIBRVMSG_I8	CONSTANT INTEGER := 4
TIBRVMSG_I16	CONSTANT INTEGER := 5
TIBRVMSG_I32	CONSTANT INTEGER := 6
TIBRVMSG_I64	CONSTANT INTEGER := 7
TIBRVMSG_IPADDR32	CONSTANT INTEGER := 8
TIBRVMSG_IPPORT16	CONSTANT INTEGER := 9
TIBRVMSG_DATETIME	CONSTANT INTEGER := 10
TIBRVMSG_F32ARRAY	CONSTANT INTEGER := 11

Table 59–4 (Cont.) Constants for the TIB/Rendezvous field type

Value	Constant
TIBRVMSG_F64ARRAY	CONSTANT INTEGER := 12
TIBRVMSG_I8ARRAY	CONSTANT INTEGER := 13
TIBRVMSG_I16ARRAY	CONSTANT INTEGER := 14
TIBRVMSG_I32ARRAY	CONSTANT INTEGER := 15
TIBRVMSG_I64ARRAY	CONSTANT INTEGER := 16
TIBRVMSG_OPAQUE	CONSTANT INTEGER := 17
TIBRVMSG_STRING	CONSTANT INTEGER := 18
TIBRVMSG_XML	CONSTANT INTEGER := 19

Types

SYS.MGW_NAME_VALUE_T Type

This type specifies a named value. The `name` attribute, `type` attribute, and one of the `<>_value` attributes are typically not `NULL`.

Syntax

```
TYPE SYS.MGW_NAME_VALUE_T IS OBJECT(
   name               VARCHAR2(250),
   type               INTEGER,
   integer_value      INTEGER,
   number_value       NUMBER,
   text_value         VARCHAR2(4000),
   raw_value          RAW(2000),
   date_value         DATE,

-- Methods
STATIC FUNCTION CONSTRUCT
RETURN SYS.MGW_NAME_VALUE_T,

STATIC FUNCTION CONSTRUCT_BOOLEAN (
   name   IN VARCHAR2,
   value  IN INTEGER )
RETURN SYS.MGW_NAME_VALUE_T,

STATIC FUNCTION CONSTRUCT_BYTE (
   name   IN VARCHAR2,
   value  IN INTEGER )
RETURN SYS.MGW_NAME_VALUE_T,

STATIC FUNCTION CONSTRUCT_SHORT (
   name   IN VARCHAR2,
   value  IN INTEGER )
RETURN SYS.MGW_NAME_VALUE_T,

STATIC FUNCTION CONSTRUCT_INTEGER (
   name   IN VARCHAR2,
   value  IN INTEGER )
RETURN SYS.MGW_NAME_VALUE_T,

STATIC FUNCTION CONSTRUCT_LONG (
   name   IN VARCHAR2,
```

```
   value  IN NUMBER )
RETURN SYS.MGW_NAME_VALUE_T,

STATIC FUNCTION CONSTRUCT_FLOAT (
   name   IN VARCHAR2,
   value  IN NUMBER )
RETURN SYS.MGW_NAME_VALUE_T,

STATIC FUNCTION CONSTRUCT_DOUBLE (
   name   IN VARCHAR2,
   value  IN NUMBER )
RETURN SYS.MGW_NAME_VALUE_T,

STATIC FUNCTION CONSTRUCT_TEXT (
   name   IN VARCHAR2,
   value  IN VARCHAR2 )
RETURN SYS.MGW_NAME_VALUE_T,

STATIC FUNCTION CONSTRUCT_RAW (
   name   IN VARCHAR2,
   value  IN RAW )
RETURN SYS.MGW_NAME_VALUE_T,

STATIC FUNCTION CONSTRUCT_DATE (
   name   IN VARCHAR2,
   value  IN DATE )
RETURN SYS.MGW_NAME_VALUE_T );
```

Attributes

Table 59–5 SYS.MGW_NAME_VALUE_T Attributes

Attribute	Description
name	Name associated with the value
type	Value type. Refer to the DBMS_MGWMSG.<>_VALUE constants in Table 59–1. This indicates which Java datatype and class are associated with the value. It also indicates which attribute stores the value.
integer_value	Stores a numeric integer value
number_value	Stores a numeric float or large integer value
text_value	Stores a text value
raw_value	Stores a RAW (bytes) value
date_value	Stores a date value

SYS.MGW_NAME_VALUE_T Type-Attribute Mapping

Table 59–6 shows the mapping between the value type and the attribute used to store the value.

Table 59–6 SYS.MGW_NAME_VALUE_T Type Attribute Mapping

Type	Value Stored in Attribute
DBMS_MGWMSG.TEXT_VALUE	text_value
DBMS_MGWMSG.RAW_VALUE	raw_value
DBMS_MGWMSG.BOOLEAN_VALUE	integer_value

Table 59–6 (Cont.) SYS.MGW_NAME_VALUE_T Type Attribute Mapping

Type	Value Stored in Attribute
DBMS_MGWMSG.BYTE_VALUE	integer_value
DBMS_MGWMSG.SHORT_VALUE	integer_value
DBMS_MGWMSG.INTEGER_VALUE	integer_value
DBMS_MGWMSG.LONG_VALUE	number_value
DBMS_MGWMSG.FLOAT_VALUE	number_value
DBMS_MGWMSG.DOUBLE_VALUE	number_value
DBMS_MGWMSG.DATE_VALUE	date_value

CONSTRUCT Method

This method constructs a new SYS.MGW_NAME_VALUE_T instance. All attributes are assigned a value of NULL.

Syntax

```
STATIC FUNCTION CONSTRUCT
RETURN SYS.MGW_NAME_VALUE_T;
```

CONSTRUCT_*TYPE* Methods

These methods construct a new SYS.MGW_NAME_VALUE_T instance initialized with the value of a specific type. Each method sets the name and type attributes and one of the <>_value attributes, as shown in the mappings in Table 59–6.

Syntax

```
STATIC FUNCTION CONSTRUCT_<> (
   name   IN VARCHAR2,
   value  IN datatype )
RETURN SYS.MGW_NAME_VALUE_T;
```

Usage Notes

The construct_boolean method sets the value to either DBMS_MGWMSG.BOOLEAN_TRUE or DBMS_MGWMSG.BOOLEAN_FALSE.

SYS.MGW_NAME_TYPE_ARRAY_T Type

This type specifies an array of name-value pairs. An object of SYS.MGW_NAME_VALUE_ARRAY_T type can have up to 1024 elements.

Syntax

```
TYPE SYS.MGW_NAME_VALUE_ARRAY_T
  AS VARRAY (1024) OF SYS.MGW_NAME_VALUE_T;
```

SYS.MGW_TEXT_VALUE_T Type

This type specifies a TEXT value. It can store a large value as a CLOB or a smaller value (size <= 4000) as VARCHAR2. Only one of the < >_ value attributes should be set.

Syntax

```
TYPE SYS.MGW_TEXT_VALUE_T IS OBJECT(
   small_value VARCHAR2(4000),
```

```
   large_value CLOB,

-- Methods
STATIC FUNCTION CONSTRUCT
RETURN SYS.MGW_TEXT_VALUE_T);
```

Attributes

Table 59–7 SYS.MGW_TEXT_VALUE_T Attributes

Attribute	Description
small_value	Small TEXT value. Used for values <= 4000.
large_value	Large TEXT value. Used when the value is too large for the small_value attribute.

CONSTRUCT Method

This method constructs a new SYS.MGW_TEXT_VALUE_T instance. All attributes are assigned a value of NULL.

Syntax

```
STATIC FUNCTION CONSTRUCT
RETURN SYS.MGW_TEXT_VALUE_T;
```

SYS.MGW_RAW_VALUE_T Type

This type specifies a RAW value. This type can store a large value as a BLOB or a smaller value (size <= 2000) as RAW. You must set no more than one of the < >_value attributes.

Syntax

```
TYPE SYS.MGW_RAW_VALUE_T IS OBJECT(
   small_value RAW(2000),
   large_value BLOB,

--Methods
STATIC FUNCTION CONSTRUCT
RETURN SYS.MGW_RAW_VALUE_T);
```

Attributes

Table 59–8 SYS.MGW_RAW_VALUE_T Attributes

Attribute	Description
small_value	Small RAW (bytes) value <= 2000
large_value	Large RAW value. Used when the value is too large for the small_value attribute.

CONSTRUCT Method

This method constructs a new SYS.MGW_RAW_VALUE_T instance. All attributes are assigned a value of NULL.

Syntax

```
STATIC FUNCTION CONSTRUCT
```

```
RETURN SYS.MGW_RAW_VALUE_T;
```

SYS.MGW_BASIC_MSG_T Type

This is a canonical type for a basic TEXT or RAW message. Only a single TEXT or RAW value is typically set. An object of this type must not have both TEXT and RAW set to a not NULL value at the same time.

Syntax

```
TYPE SYS.MGW_BASIC_MSG_T IS OBJECT(
   header     SYS.MGW_NAME_VALUE_ARRAY_T,
   text_body  SYS.MGW_TEXT_VALUE_T,
   raw_body   SYS.MGW_RAW_VALUE_T,

--Methods
STATIC FUNCTION CONSTRUCT
RETURN SYS.MGW_BASIC_MSG_T);
```

Attributes

Table 59–9 SYS.MGW_BASIC_MSG_T Attributes

Attribute	Description
header	Message header information as an array of name-value pairs
text_body	Message body for a TEXT message
raw_body	Message body for a RAW (bytes) message

CONSTRUCT Method

This method constructs a new SYS.MGW_BASIC_MSG_T instance. All attributes are assigned a value of NULL.

Syntax

```
STATIC FUNCTION CONSTRUCT
RETURN SYS.MGW_BASIC_MSG_T;
```

SYS.MGW_NUMBER_ARRAY_T Type

A type that specifies an array of numbers.

Syntax

```
TYPE SYS.MGW_NUMBER_ARRAY_T AS VARRAY(1024) OF NUMBER;
```

SYS.MGW_TIBRV_FIELD_T Type

A type representing a TIB/Rendezvous message field, typically used in a read-only fashion to retrieve field information from a SYS.MGW_TIBRV_MSG_T instance.

Syntax

```
TYPE SYS.MGW_TIBRV_FIELD_T IS OBJECT(
   field_name       VARCHAR2(256),
   field_id         INTEGER,
   field_type       INTEGER,
   number_value     NUMBER,
   number_array_value  SYS.MGW_NUMBER_ARRAY_T,
```

```
  text_value       VARCHAR2(4000),
  raw_value        RAW(2000),
  date_value       DATE,
  clob_value       CLOB,
  blob_value       BLOB);
```

Attributes

Table 59–10 SYS.MGW_TIBRV_FIELD_T Attributes

Attribute	Description
field_name	Field name. This will be NULL if the field has no name.
field_id	Field identifier. If the field identifier is zero (0), then that field is considered not to have a field identifier. Otherwise the field identifier is a nonzero value that is unique for all fields of that message.
field_type	Field wire format datatype. The DBMS_MGWMSG.TIBRVMSG_<> constants represent valid values for this attribute. The value of this field discriminates which value attribute is used to store the field data.
number_value	Used to store a numeric value
number_array_value	Used to store a numeric array value
text_value	Used to store a small text value
raw_value	Used to store a small raw value
date_value	Used to store a date value
clob_value	Used to store a large text value. This is used when the text data will not fit in text_value, that is, when size is larger than 4000.
blob_value	Used to store a large raw value. This is used when the raw data will not fit in raw_value; that is, when size is larger than 2000.

SYS.MGW_TIBRV_FIELD_T Type and Attribute Mapping

Table 59–11 describes the mapping in type SYS.MGW_TIBRV_FIELD_T between the field type and attribute used to store the value.

Table 59–11 SYS.MGW_TIBRV_FIELD_T Type and Attribute Mapping

Field Type (DBMS_MGWMSG constant)	Value Stored in Attribute
TIBRVMSG_BOOL	number_value
TIBRVMSG_F32	number_value
TIBRVMSG_F64	number_value
TIBRVMSG_I8	number_value
TIBRVMSG_I16	number_value
TIBRVMSG_I32	number_value
TIBRVMSG_I64	number_value
TIBRVMSG_IPADDR32	text_value
TIBRVMSG_IPPORT16	number_value
TIBRVMSG_DATETIME	date_value

Table 59–11 (Cont.) SYS.MGW_TIBRV_FIELD_T Type and Attribute Mapping

Field Type (DBMS_MGWMSG constant)	Value Stored in Attribute
TIBRVMSG_F32ARRAY	number_array_value
TIBRVMSG_F64ARRAY	number_array_value
TIBRVMSG_I8ARRAY	number_array_value
TIBRVMSG_I16ARRAY	number_array_value
TIBRVMSG_I32ARRAY	number_array_value
TIBRVMSG_I64ARRAY	number_array_value
TIBRVMSG_OPAQUE	raw_value or blob_value
TIBRVMSG_STRING	text_value or clob_value
TIBRVMSG_XML	raw_value or blob_value

SYS.MGW_TIBRV_MSG_T Type

A type representing a TIB/Rendezvous message. You must never directly reference the attributes of this type. Instead use the type methods.

Syntax

```
TYPE SYS.MGW_TIBRV_MSG_T IS OBJECT(
   send_subject    VARCHAR2(256),
   reply_subject   VARCHAR2(256),
   cm_time_limit   NUMBER,
   cm_sender_name  VARCHAR2(256),
   cm_sequence_num NUMBER,
   fields          SYS.MGW_TIBRV_IFIELDS_T,
   clob_data1      CLOB,
   clob_data2      CLOB,
   clob_data3      CLOB,
   blob_data1      BLOB,
   blob_data2      BLOB,
   blob_data3      BLOB,

STATIC FUNCTION construct
RETURN SYS.MGW_TIBRV_MSG_T,

MEMBER PROCEDURE add_bool (
    name  IN   VARCHAR2,
    id    IN   INTEGER,
    value IN   INTEGER ),

MEMBER PROCEDURE add_f32 (
    name  IN   VARCHAR2,
    id    IN   INTEGER,
    value IN   FLOAT ),

MEMBER PROCEDURE add_f64 (
    name  IN   VARCHAR2,
    id    IN   INTEGER,
    value IN   DOUBLE ),

MEMBER PROCEDURE add_i8 (
    name  IN  VARCHAR2,
    id    IN  INTEGER,
```

```
    value IN  INTEGER ),

MEMBER PROCEDURE add_i16 (
     name  IN  VARCHAR2,
     id    IN  INTEGER,
     value IN  INTEGER ),

MEMBER PROCEDURE add_i32 (
     name  IN  VARCHAR2,
     id    IN  INTEGER,
     value IN  INTEGER ),

MEMBER PROCEDURE add_i64 (
     name  IN  VARCHAR2,
     id    IN  INTEGER,
     value IN  NUMBER ),

MEMBER PROCEDURE add_ipaddr32 (
     name  IN  VARCHAR2,
     id    IN  INTEGER,
     value IN  VARCHAR2 ),

MEMBER PROCEDURE add_ipport16 (
     name  IN  VARCHAR2,
     id    IN  INTEGER,
     value IN  INTEGER ),

MEMBER PROCEDURE add_datetime (
     name  IN  VARCHAR2,
     id    IN  INTEGER,
     value IN  DATE ),

MEMBER PROCEDURE add_f32array (
     name  IN  VARCHAR2,
     id    IN  INTEGER,
     value IN  SYS.MGW_NUMBER_ARRAY_T ),

MEMBER PROCEDURE add_f64array (
     name  IN  VARCHAR2,
     id    IN  INTEGER,
     value IN  SYS.MGW_NUMBER_ARRAY_T ),

MEMBER PROCEDURE add_i8array (
     name  IN  VARCHAR2,
     id    IN  INTEGER,
     value IN  SYS.MGW_NUMBER_ARRAY_T ),

MEMBER PROCEDURE add_i16array (
     name  IN  VARCHAR2,
     id    IN  INTEGER,
     value IN  SYS.MGW_NUMBER_ARRAY_T ),

MEMBER PROCEDURE add_i32array (
     name  IN  VARCHAR2,
     id    IN  INTEGER,
     value IN  SYS.MGW_NUMBER_ARRAY_T ),

MEMBER PROCEDURE add_i64array (
     name  IN  VARCHAR2,
     id    IN  INTEGER,
```

```
     value IN  SYS.MGW_NUMBER_ARRAY_T ),

MEMBER PROCEDURE add_string (
     name  IN  VARCHAR2,
     id    IN  INTEGER,
     value IN  VARCHAR2 ),

MEMBER PROCEDURE add_string (
     name  IN  VARCHAR2,
     id    IN  INTEGER,
     value IN  CLOB ),

MEMBER PROCEDURE add_opaque (
     name  IN  VARCHAR2,
     id    IN  INTEGER,
     value IN  RAW ),

MEMBER PROCEDURE add_opaque (
     name  IN  VARCHAR2,
     id    IN  INTEGER,
     value IN  BLOB ),

MEMBER PROCEDURE add_xml (
     name  IN  VARCHAR2,
     id    IN  INTEGER,
     value IN  RAW ),

MEMBER PROCEDURE add_xml (
     name  IN  VARCHAR2,
     id    IN  INTEGER,
     value IN  BLOB ),

MEMBER PROCEDURE set_send_subject (
     value  IN  VARCHAR2 ),

MEMBER PROCEDURE set_reply_subject (
     value  IN  VARCHAR2 ),

MEMBER PROCEDURE set_cm_time_limit (
     value  IN  NUMBER ),

MEMBER PROCEDURE set_cm_sender_name (
     value IN  VARCHAR2 ),

MEMBER PROCEDURE set_cm_sequence_num (
     value IN NUMBER ),

MEMBER FUNCTION get_send_subject
RETURN VARCHAR2,

MEMBER FUNCTION get_reply_subject
RETURN VARCHAR2,

MEMBER FUNCTION get_cm_time_limit
RETURN NUMBER,

MEMBER FUNCTION get_cm_sender_name
RETURN VARCHAR2,

MEMBER FUNCTION get_cm_sequence_num
```

```
RETURN NUMBER,

MEMBER FUNCTION get_field_count
RETURN INTEGER,

MEMBER FUNCTION get_field (
     idx    IN INTEGER )
RETURN SYS.MGW_TIBRV_FIELD_T,

MEMBER FUNCTION get_field_by_name (
     name   IN  VARCHAR2 )
RETURN SYS.MGW_TIBRV_FIELD_T,

MEMBER FUNCTION get_field_by_id (
     id     IN  INTEGER )
RETURN SYS.MGW_TIBRV_FIELD_T,

MEMBER FUNCTION find_field_name (
     name   IN  VARCHAR2,
     start_idx IN INTEGER )
RETURN INTEGER,

MEMBER FUNCTION find_field_id (
     id     IN INTEGER,
     start_idx IN INTEGER )
RETURN INTEGER
);
```

Attributes

Table 59–12 SYS.MGW_TIBRV_MSG_T Type Attributes

Attribute	Description
send_subject	Send subject name
reply_subject	Reply subject name
cm_time_limit	Time limit for a certified message
cm_sender_name	Sender name of a certified message
cm_sequence_num	Sequence number of a certified message
fields	Collection of message fields
clob_data1	Used to store a large text value
clob_data2	Used to store a large text value
clob_data3	Used to store a large text value
blob_data1	Used to store a large raw value
blob_data2	Used to store a large raw value
blob_data3	Used to store a large raw value

Construct Method

Constructs a new SYS.MGW_TIBRV_MSG_T instance. All attributes are set to NULL.

Syntax

```
STATIC FUNCTION construct
RETURN SYS.MGW_TIBRV_MSG_T;
```

ADD_<> Procedures

Adds a new field to the message.

Syntax

```
MEMBER PROCEDURE ADD_<> (
   name  IN  VARCHAR2,
   id    IN  INTEGER,
   value IN  datatype );
```

Parameters

Table 59–13 SYS.MGW_TIBRV_MSG_T ADD_<> Method Parameters

Parameter	Description
name	Field name
id	Field identifier
value	Field data

Table 59–14 shows, for each add method, the field type that will be assigned and valid values for the field data.

Table 59–14 MGW_TIBRV_MSG_T Add Method Field Types

Method Name	Field Type Assigned	Comment
add_bool	TIBRVMSG_BOOL	Valid values: 0 (false), 1 (true)
add_f32	TIBRVMSG_F32	n/a
add_f64	TIBRVMSG_F64	n/a
add_i8	TIBRVMSG_I8	Valid range: -128...127
add_i16	TIBRVMSG_I16	Valid range: -32768...32767
add_i32	TIBRVMSG_I32	Valid range: -2147483648... 2147483647
add_i64	TIBRVMSG_I64	n/a
add_ipaddr32	TIBRVMSG_IPADDR32	n/a
add_ipport16	TIBRVMSG_IPPORT16	n/a
add_datetime	TIBRVMSG_DATETIME	n/a
add_f32array	TIBRVMSG_F32ARRAY	n/a
add_f64array	TIBRVMSG_F64ARRAY	n/a
add_i8array	TIBRVMSG_I8ARRAY	Valid range: -128...127
add_i16array	TIBRVMSG_I16ARRAY	Valid range: -32768...32767
add_i32array	TIBRVMSG_I32ARRAY	Valid range: -2147483648... 2147483647
add_i64array	TIBRVMSG_I64ARRAY	n/a
add_opaque	TIBRVMSG_OPAQUE	Value stored as RAW if size < 2000; otherwise value stored in BLOB
add_string	TIBRVMSG_STRING	Value stored as VARCHAR2 if size < 4000; otherwise value stored in CLOB

Table 59–14 (Cont.) MGW_TIBRV_MSG_T Add Method Field Types

Method Name	Field Type Assigned	Comment
add_xml	TIBRVMSG_XML	Value stored as RAW if size < 2000; otherwise value stored in BLOB

SET_<> Methods

Accessor methods to set an instance attribute to a specific value.

Syntax

```
MEMBER PROCEDURE SET_<> (
  value IN datatype );
```

Parameters

Table 59–15 SYS.MGW_TIBRV_MSG_T SET_<> Method Parameters

Parameter	Description
value	Value to be assigned

GET_<> Methods

Accessor methods to retrieve the value for an instance attribute.

Syntax

```
MEMBER PROCEDURE GET_<>
RETURN datatype;
```

Parameters

None

Return Values

Returns the attribute value.

GET_FIELD_COUNT Procedure

Gets the number of message fields.

Syntax

```
MEMBER PROCEDURE get_field_count
RETURN INTEGER;
```

Parameters

None

Return Values

Returns the number of fields, or zero (0) if there are none.

GET_FIELD Procedure

Retrieves field information for the field having a given field collection index. This method should only be called if the GET_FIELD_COUNT Procedure returns a

nonzero value and idx must specify a valid collection index; that is, 1<=idx<=get_field_count().

Syntax

```
MEMBER PROCEDURE get_field (
    idx    IN    INTEGER )
RETURN SYS.MGW_TIBRV_FIELD_T;
```

Parameters

Table 59–16 SYS.MGW_TIBRV_MSG_T GET_FIELD Procedure Parameters

Parameter	Description
idx	Specifies the 1-based field collection index of the field to retrieve

Note: A 1-based index begins at one (1) instead of zero (0).

Return Values

Returns the field information.

GET_FIELD_BY_NAME Procedure

Retrieves field information for the first field that has a given field name. The name comparison is case-sensitive.

Syntax

```
MEMBER PROCEDURE get_field_by_name (
   name  IN   VARCHAR2 )
RETURN SYS.MGW_TIBRV_FIELD_T;
```

Parameters

Table 59–17 SYS.MGW_TIBRV_MSG_T GET_FIELD_BY_NAME Procedure Parameters

Parameter	Description
name	Specifies the field name to search for. This can be NULL to find the first field that does not have a field name.

Return Values

Returns the field information, or NULL if no match was found.

GET_FIELD_BY_ID Procedure

Retrieves field information for the first field that has a given field identifier.

A field can have either a unique identifier or no identifier. If the field identifier value is zero (0) or NULL, then the field is considered to have no identifier. Otherwise, the identifier is a nonzero value that is unique for all the fields of this message.

Syntax

```
MEMBER PROCEDURE get_field_by_id (
   id   IN  INTEGER )
```

```
RETURN SYS.MGW_TIBRV_FIELD_T;
```

Parameters

Table 59–18 SYS.MGW_TIBRV_MSG_T GET_FIELD_BY_ID Procedure Parameters

Parameter	Description
id	Specifies the field identifier to search for. This can be zero (0) or NULL to find the first field that does not have an identifier.

Return Values

Returns the field information, or NULL if no match was found.

FIND_FIELD_NAME Procedure

Searches for a field with a given field name, starting from a given index of the field collection. It returns the index of that field. The name comparison is case-sensitive. This function is useful for finding all the fields that have the same name.

Syntax

```
MEMBER PROCEDURE find_field_name (
   name      IN  VARCHAR2,
   start_idx IN  INTEGER )
RETURN INTEGER;
```

Parameters

Table 59–19 SYS.MGW_TIBRV_MSG_T FIND_FIELD_NAME Procedure Parameters

Parameter	Description
name	Specifies the field name to search for. This can be NULL to search for a field that does not have a field name.
start_idx	Specifies the 1-based field collection index from which the search should start.

Return Values

Returns the field index (> 0) if a match was found, or zero (0) if no match was found.

FIND_FIELD_ID Procedure

Searches for a field with a given field identifier, starting from a given index of the field collection. It returns the index of that field.

Syntax

```
MEMBER PROCEDURE find_field_id (
     id        IN  INTEGER,
     start_idx IN INTEGER )
RETURN    INTEGER;
```

Parameters

Table 59–20 SYS.MGW_TIBRV_MSG_T FIND_FIELD_ID Procedure Parameters

Parameter	Description
`id`	Specifies the field identifier to search for. This can be zero (0) or `NULL` to find a field that does not have an identifier.
`start_idx`	Specifies the 1-based field collection index from which the search should start.

Return Values

Returns the field index (> 0) if a match was found, or zero (0) if no match was found.

Summary of DBMS_MGWMSG Subprograms

Table 59–21 DBMS_MGWMSG Package Subprograms

Subprogram	Description
LCR_TO_XML Function on page 59-22	Converts a SYS.ANYDATA object encapsulating a row LCR (LCR$_ROW_RECORD) or a DDL LCR (LCR$_DDL_RECORD) to a SYS.XMLTYPE object
NVARRAY_ADD Procedure on page 59-23	Appends a name-value element to the end of a name-value array
NVARRAY_FIND_NAME Function on page 59-24	Searches a name-value array for the element with the name you specify in p_name
NVARRAY_FIND_NAME_TYPE Function on page 59-25	Searches a name-value array for an element with the name and value type you specify
NVARRAY_GET Function on page 59-26	Gets the name-value element of the name you specify in p_name from a name-value array
NVARRAY_GET_BOOLEAN Function on page 59-27	Gets the value of the name-value array element that you specify in p_name and with the BOOLEAN_VALUE value type
NVARRAY_GET_BYTE Function on page 59-28	Gets the value of the name-value array element that you specify in p_name and with the BYTE_VALUE value type
NVARRAY_GET_DATE Function on page 59-29	Gets the value of the name-value array element that you specify in p_name and with the DATE_VALUE value type
NVARRAY_GET_DOUBLE Function on page 59-30	Gets the value of the name-value array element that you specify in p_name and with the DOUBLE_VALUE value type
NVARRAY_GET_FLOAT Function on page 59-31	Gets the value of the name-value array element that you specify in p_name and with the FLOAT_VALUE value type
NVARRAY_GET_INTEGER Function on page 59-32	Gets the value of the name-value array element that you specify in p_name and with the INTEGER_VALUE value type
NVARRAY_GET_LONG Function on page 59-33	Gets the value of the name-value array element that you specify in p_name and with the LONG_VALUE value type
NVARRAY_GET_RAW Function on page 59-34	Gets the value of the name-value array element that you specify in p_name and with the RAW_VALUE value type
NVARRAY_GET_SHORT Function on page 59-35	Gets the value of the name-value array element that you specify in p_name and with the SHORT_VALUE value type
NVARRAY_GET_TEXT Function on page 59-36	Gets the value of the name-value array element that you specify in p_name and with the TEXT_VALUE value type
XML_TO_LCR Function on page 59-37	Converts a SYS.XMLTYPE object to a SYS.ANYDATA object encapsulating a row LCR (LCR$_ROW_RECORD) or a DDL LCR (LCR$_DDL_RECORD)

LCR_TO_XML Function

This function converts a SYS.ANYDATA object encapsulating a row LCR (Logical Change Record, in this case a LCR$_ROW_RECORD) or a DDL LCR (LCR$_DDL_RECORD) to a SYS.XMLTYPE object.

See Also: XML_TO_LCR Function on page 59-37

Syntax

```
DBMS_MGWMSG.LCR_TO_XML (
   p_anydata IN SYS.ANYDATA )
 RETURN SYS.XMLTYPE;
```

Parameters

Table 59–22 LCR_TO_XML Function Parameters

Parameter	Description
p_anydata	An ANYDATA object to be converted

Return Values

Returns a SYS.XMLTYPE object.

Usage Notes

An exception is raised if the encapsulated type p_anydata is not an LCR.

NVARRAY_ADD Procedure

This procedure appends a name-value element to the end of a name-value array.

Syntax

```
DBMS_MGWMSG.NVARRAY_ADD (
   p_array  IN OUT SYS.MGW_NAME_VALUE_ARRAY_T,
   p_value  IN     SYS.MGW_NAME_VALUE_T );
```

Parameters

Table 59–23 NVARRAY_ADD Procedure Parameters

Parameter	Description
p_array	On input, the name-value array instance to modify. If NULL, then a new array is created. On output, the modified name-value array instance.
p_value	The value to add. If NULL, then p_array is not changed.

NVARRAY_FIND_NAME Function

This function searches a name-value array for the element with the name you specify in `p_name`.

Syntax

```
DBMS_MGWMSG.NVARRAY_FIND_NAME (
   p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
   p_name     IN VARCHAR2,
   p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN BINARY_INTEGER;
```

Parameters

Table 59–24 NVARRAY_FIND_NAME Function Parameters

Parameters	Description
`p_array`	The name-value array to search
`p_name`	The name to find
`p_compare`	Name comparison method. Values are `CASE_SENSITIVE` and `CASE_INSENSITIVE`.

Return Values

Returns a positive integer that is the array index of the matching element or zero (`0`) if the specified name is not found.

NVARRAY_FIND_NAME_TYPE Function

This function searches a name-value array for an element with the name and value type you specify.

Syntax

```
DBMS_MGWMSG.NVARRAY_FIND_NAME_TYPE (
   p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
   p_name     IN VARCHAR2,
   p_type     IN BINARY_INTEGER
   p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN BINARY_INTEGER;
```

Parameters

Table 59–25 NVARRAY_FIND_NAME_TYPE Function Parameters

Parameter	Description
p_array	The name-value array to search
p_name	The name to find
p_type	The value type. Refer to the value type constants in Table 59–1 on page 59-4.
p_compare	Name comparison method. Values are CASE_SENSITIVE and CASE_INSENSITIVE.

Return Values

Returns a positive integer that is the array index of the matching element, zero (0) if the specified name is not found, or negative one (-1) if the specified name is found but a type mismatch exists.

NVARRAY_GET Function

This function gets the name-value element of the name you specify in `p_name` from a name-value array.

Syntax

```
DBMS_MGWMSG.NVARRAY_GET (
   p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
   p_name     IN VARCHAR2,
   p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN SYS.MGW_NAME_VALUE_T;
```

Parameters

Table 59–26 NVARRAY_GET Function Parameters

Parameter	Description
`p_array`	The name-value array
`p_name`	The value name
`p_compare`	Name comparison method. Values are `CASE_SENSITIVE` and `CASE_INSENSITIVE`.

Return Values

Returns the matching element, or `NULL` if the specified name is not found.

NVARRAY_GET_BOOLEAN Function

This function gets the value of the name-value array element that you specify in p_name and with the BOOLEAN_VALUE value type .

Syntax

```
DBMS_MGWMSG.NVARRAY_GET_BOOLEAN (
   p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
   p_name     IN VARCHAR2,
   p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN INTEGER;
```

Parameters

Table 59–27 NVARRAY_GET_BOOLEAN Function Parameters

Parameter	Description
p_array	The name-value array
p_name	The value name
p_compare	Name comparison method. Values are CASE_SENSITIVE and CASE_INSENSITIVE.

Return Values

Returns the value, or NULL if either the specified name is not found or a type mismatch exists.

NVARRAY_GET_BYTE Function

This function gets the value of the name-value array element that you specify in p_name and with the BYTE_VALUE value type.

Syntax

```
DBMS_MGWMSG.NVARRAY_GET_BYTE (
   p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
   p_name     IN VARCHAR2,
   p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN INTEGER;
```

Parameters

Table 59–28 NVARRAY_GET_BYTE Function

Parameter	Description
p_array	The name-value array
p_name	The value name
p_compare	Name comparison method. Values are CASE_SENSITIVE and CASE_INSENSITIVE.

Return Values

Returns the value, or NULL if either the specified name is not found or a type mismatch exists.

NVARRAY_GET_DATE Function

This function gets the value of the name-value array element that you specify in p_name and with the DATE_VALUE value type .

Syntax

```
DBMS_MGWMSG.NVARRAY_GET_DATE (
   p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
   p_name     IN VARCHAR2,
   p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN DATE;
```

Parameters

Table 59–29 NVARRAY_GET_DATE Function Parameters

Parameters	Description
p_array	The name-value array
p_name	The value name
p_compare	Name comparison method. Values are CASE_SENSITIVE and CASE_INSENSITIVE.

Return Values

Returns the value, or NULL if either the specified name is not found or a type mismatch exists.

NVARRAY_GET_DOUBLE Function

This function gets the value of the name-value array element that you specify in p_name and with the DOUBLE_VALUE value type.

Syntax

```
DBMS_MGWMSG.NVARRAY_GET_DOUBLE (
   p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
   p_name     IN VARCHAR2,
   p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN NUMBER;
```

Parameters

Table 59–30 NVARRAY_GET_DOUBLE Function Parameters

Parameter	Description
p_array	The name-value array
p_name	The value name
p_compare	Name comparison method. Values are CASE_SENSITIVE and CASE_INSENSITIVE.

Return Values

Returns the value, or NULL if either the specified name is not found or a type mismatch exists.

NVARRAY_GET_FLOAT Function

This function gets the value of the name-value array element that you specify in p_name and with the FLOAT_VALUE value type.

Syntax

```
DBMS_MGWMSG.NVARRAY_GET_FLOAT (
   p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
   p_name     IN VARCHAR2,
   p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN NUMBER;
```

Parameters

Table 59–31 NVARRAY_GET_FLOAT Function Parameters

Parameter	Description
p_array	The name-value array
p_name	The value name
p_compare	Name comparison method. Values are CASE_SENSITIVE and CASE_INSENSITIVE.

Return Values

Returns the value, or NULL if either the specified name is not found or a type mismatch exists.

NVARRAY_GET_INTEGER Function

This function gets the value of the name-value array element that you specify in p_name and with the INTEGER_VALUE value type.

Syntax

```
DBMS_MGWMSG.NVARRAY_GET_INTEGER (
   p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
   p_name     IN VARCHAR2,
   p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN INTEGER;
```

Parameters

Table 59–32 NVARRAY_GET_INTEGER Function Parameters

Parameter	Description
p_array	The name-value array
p_name	The value name
p_compare	Name comparison method. Values are CASE_SENSITIVE and CASE_INSENSITIVE.

Return Values

Returns the value, or NULL if either the specified name is not found or a type mismatch exists.

NVARRAY_GET_LONG Function

This function gets the value of the name-value array element that you specify in p_name and with the LONG_VALUE value type .

Syntax

```
DBMS_MGWMSG.NVARRAY_GET_LONG (
   p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
   p_name     IN VARCHAR2,
   p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN NUMBER;
```

Parameters

Table 59–33 NVARRAY_GET_LONG Function Parameters

Parameter	Description
p_array	The name-value array
p_name	The value name
p_compare	Name comparison method. Values are CASE_SENSITIVE and CASE_INSENSITIVE.

Return Values

Returns the value, or NULL if either the specified name is not found or a type mismatch exists.

NVARRAY_GET_RAW Function

This function gets the value of the name-value array element that you specify in p_name and with the RAW_VALUE value type .

Syntax

```
DBMS_MGWMSG.NVARRAY_GET_RAW (
   p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
   p_name     IN VARCHAR2,
   p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN RAW;
```

Parameters

Table 59–34 NVARRAY_GET_RAW Function Parameters

Parameter	Description
p_array	The name-value array
p_name	The value name
p_compare	Name comparison method. Values are CASE_SENSITIVE and CASE_INSENSITIVE.

Return Values

Returns the value, or NULL if either the specified name is not found or a type mismatch exists.

NVARRAY_GET_SHORT Function

This function gets the value of the name-value array element that you specify in p_name and with the SHORT_VALUE value type.

Syntax

```
DBMS_MGWMSG.NVARRAY_GET_SHORT (
   p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
   p_name     IN VARCHAR2,
   p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN INTEGER;
```

Parameters

Table 59–35 NVARRAY_GET_SHORT Function Parameters

Parameter	Description
p_array	The name-value array
p_name	The value name
p_compare	Name comparison method. Values are CASE_SENSITIVE and CASE_INSENSITIVE.

Return Values

Returns the value, or NULL if either the specified name is not found or a type mismatch exists.

NVARRAY_GET_TEXT Function

This function gets the value of the name-value array element that you specify in p_name and with the TEXT_VALUE value type.

Syntax

```
DBMS_MGWMSG.NVARRAY_GET_TEXT (
   p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
   p_name     IN VARCHAR2,
   p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN VARCHAR2;
```

Parameters

Table 59–36 NVARRAY_GET_TEXT Function Parameters

Parameter	Description
p_array	The name-value array
p_name	The value name
p_compare	Name comparison method. Values are CASE_SENSITIVE and CASE_INSENSITIVE.

Return Values

Returns the value, or NULL if either the specified name is not found or a type mismatch exists.

XML_TO_LCR Function

This function converts a SYS.XMLTYPE object to a SYS.ANYDATA object encapsulating a row LCR (LCR$_ROW_RECORD) or a DDL LCR (LCR$_DDL_RECORD).

See Also: LCR_TO_XML Function on page 59-22

Syntax

```
DBMS_MGWMSG.XML_TO_LCR (
   p_xmldata IN SYS.XMLTYPE )
 RETURN SYS.ANYDATA;
```

Parameters

Table 59–37 XML_TO_LCR Function Parameters

Parameter	Description
p_xmldata	An XMLTYPE object representing an LCR

Return Values

Returns a SYS.ANYDATA object.

Usage Notes

An exception is raised if p_xmldata cannot be converted to an LCR.

60

DBMS_MONITOR

The DBMS_MONITOR package let you use PL/SQL for controlling additional tracing and statistics gathering.

The chapter contains the following topics:

Summary of DBMS_MONITOR Subprograms

Table 60–1 DBMS_MONITOR Package Subprograms

Subprogram	Description
CLIENT_ID_STAT_DISABLE Procedure on page 60-3	Disables statistic gathering previously enabled for a given Client Identifier
CLIENT_ID_STAT_ENABLE Procedure on page 60-4	Enables statistic gathering for a given Client Identifier
CLIENT_ID_TRACE_DISABLE Procedure on page 60-5	Disables the trace previously enabled for a given Client Identifier globally for the database
CLIENT_ID_TRACE_ENABLE Procedure on page 60-6	Enables the trace for a given Client Identifier globally for the database
DATABASE_TRACE_DISABLE Procedure on page 60-7	Disables SQL trace for the whole database or a specific instance
DATABASE_TRACE_ENABLE Procedure on page 60-8	Enables SQL trace for the whole database or a specific instance
SERV_MOD_ACT_STAT_ DISABLE Procedure on page 60-9	Disables statistic gathering enabled for a given combination of Service Name, `MODULE` and `ACTION`
SERV_MOD_ACT_STAT_ ENABLE Procedure on page 60-10	Enables statistic gathering for a given combination of Service Name, `MODULE` and `ACTION`
SERV_MOD_ACT_TRACE_ DISABLE Procedure on page 60-12	Disables the trace for ALL enabled instances for a or a given combination of Service Name, `MODULE` and `ACTION` name globally
SERV_MOD_ACT_TRACE_ ENABLE Procedure on page 60-13	Enables SQL tracing for a given combination of Service Name, `MODULE` and `ACTION` globally unless an `instance_name` is specified
SESSION_TRACE_DISABLE Procedure on page 60-15	Disables the previously enabled trace for a given database session identifier (SID) on the local instance
SESSION_TRACE_ENABLE Procedure on page 60-16	Enables the trace for a given database session identifier (SID) on the local instance

CLIENT_ID_STAT_DISABLE Procedure

This procedure will disable statistics accumulation for all instances and remove the accumulated results from V$CLIENT_STATS view enabled by the CLIENT_ID_STAT_ENABLE Procedure.

Syntax

```
DBMS_MONITOR.CLIENT_ID_STAT_DISABLE(
   client_id            IN   VARCHAR2);
```

Parameters

Table 60–2 CLIENT_ID_STAT_DISABLE Procedure Parameters

Parameter	Description
client_id	The Client Identifier for which statistic aggregation is disabled.

Examples

To disable accumulation:

```
EXECUTE DBMS_MONITOR.CLIENT_ID_STAT_DISABLE('janedoe');
```

CLIENT_ID_STAT_ENABLE Procedure

This procedure enables statistic gathering for a given Client Identifier. Statistics gathering is global for the database and persistent across instance starts and restarts. That is, statistics are enabled for all instances of the same database, including restarts. Statistics are viewable through V$CLIENT_STATS views.

Syntax

```
DBMS_MONITOR.CLIENT_ID_STAT_ENABLE(
   client_id            IN   VARCHAR2);
```

Parameters

Table 60–3 CLIENT_ID_STAT_ENABLE Procedure Parameters

Parameter	Description
client_id	The Client Identifier for which statistic aggregation is enabled.

Examples

To enable statistic accumulation for a client with a given client ID:

```
EXECUTE DBMS_MONITOR.CLIENT_ID_STAT_ENABLE('janedoe');
```

CLIENT_ID_TRACE_DISABLE Procedure

This procedure will disable tracing enabled by the CLIENT_ID_TRACE_ENABLE Procedure.

Syntax

```
DBMS_MONITOR.CLIENT_ID_TRACE_DISABLE(
 client_id            IN   VARCHAR2);
```

Parameters

Table 60–4 CLIENT_ID_TRACE_DISABLE Procedure Parameters

Parameter	Description
client_id	The Client Identifier for which SQL tracing is disabled.

Examples

```
EXECUTE DBMS_MONITOR.CLIENT_ID_TRACE_DISABLE ('janedoe');
```

CLIENT_ID_TRACE_ENABLE Procedure

This procedure will enable the trace for a given client identifier globally for the database.

Syntax

```
DBMS_MONITOR.CLIENT_ID_TRACE_ENABLE(
   client_id    IN  VARCHAR2,
   waits        IN  BOOLEAN DEFAULT TRUE,
   binds        IN  BOOLEAN DEFAULT FALSE);
```

Parameters

Table 60–5 CLIENT_ID_TRACE_ENABLE Procedure Parameters

Parameter	Description
client_id	Database Session Identifier for which SQL tracing is enabled.
waits	If TRUE, wait information is present in the trace.
binds	If TRUE, bind information is present in the trace.

Usage Notes

- The trace will be written to multiple trace files because more than one Oracle shadow process can work on behalf of a given client identifier.
- The tracing is enabled for all instances and persistent across restarts.

Examples

```
EXECUTE DBMS_MONITOR.CLIENT_ID_TRACE_ENABLE('janedoe', TRUE,
FALSE);
```

DATABASE_TRACE_DISABLE Procedure

This procedure disables SQL trace for the whole database or a specific instance.

Syntax

```
DBMS_MONITOR.DATABASE_TRACE_DISABLE(
   instance_name  IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 60–6 DATABASE_TRACE_DISABLE Procedure Parameters

Parameter	Description
instance_name	Disables tracing for the named instance

DATABASE_TRACE_ENABLE Procedure

This procedure enables SQL trace for the whole database or a specific instance.

Syntax

```
DBMS_MONITOR.DATABASE_TRACE_ENABLE(
   waits          IN BOOLEAN DEFAULT TRUE,
   binds          IN BOOLEAN DEFAULT FALSE,
   instance_name  IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 60–7 DATABASE_TRACE_ENABLE Procedure Parameters

Parameter	Description
`waits`	If `TRUE`, wait information will be present in the trace
`binds`	If `TRUE`, bind information will be present in the trace
`instance_name`	If set, restricts tracing to the named instance

SERV_MOD_ACT_STAT_DISABLE Procedure

This procedure will disable statistics accumulation and remove the accumulated results from V$SERV_MOD_ACT_STATS view. Statistics disabling is persistent for the database. That is, service statistics are disabled for instances of the same database (plus dblinks that have been activated as a result of the enable).

Syntax

```
DBMS_MONITOR.SERV_MOD_ACT_STAT_DISABLE(
   service_name    IN VARCHAR2,
   module_name     IN VARCHAR2,
   action_name     IN VARCHAR2 DEFAULT ALL_ACTIONS);
```

Parameters

Table 60–8 SERV_MOD_ACT_STAT_DISABLE Procedure Parameters

Parameter	Description
service_name	Name of the service for which statistic aggregation is disabled.
module_name	Name of the MODULE. An additional qualifier for the service. It is a required parameter.
action_name	Name of the ACTION. An additional qualifier for the Service and MODULE name. Omitting the parameter (or supplying ALL_ACTIONS constant) means enabling aggregation for all Actions for a given Server/Module combination. In this case, statistics are aggregated on the module level.

SERV_MOD_ACT_STAT_ENABLE Procedure

This procedure enables statistic gathering for a given combination of Service Name, MODULE and ACTION. Calling this procedure enables statistic gathering for a hierarchical combination of Service name, MODULE name, and ACTION name on all instances for the same database. Statistics are accessible by means of the V$SERV_MOD_ACT_STATS view.

Syntax

```
DBMS_MONITOR.SERV_MOD_ACT_STAT_ENABLE(
   service_name    IN VARCHAR2,
   module_name     IN VARCHAR2,
   action_name     IN VARCHAR2 DEFAULT ALL_ACTIONS);
```

Parameters

Table 60–9 SERV_MOD_ACT_STAT_ENABLE Procedure Parameters

Parameter	Description
service_name	Name of the service for which statistic aggregation is enabled.
module_name	Name of the MODULE. An additional qualifier for the service. It is a required parameter.
action_name	Name of the ACTION. An additional qualifier for the Service and MODULE name. Omitting the parameter (or supplying ALL_ACTIONS constant) means enabling aggregation for all Actions for a given Server/Module combination. In this case, statistics are aggregated on the module level.

Usage Notes

Enabling statistic aggregation for the given combination of Service/Module/Action names is slightly complicated by the fact that the Module/Action values can be empty strings which are indistinguishable from NULLs. For this reason, we adopt the following conventions:

A special constant (unlikely to be a real action names) is defined:

```
ALL_ACTIONS constant VARCHAR2 := '###ALL_ACTIONS';
```

Using ALL_ACTIONS for a module specification means that aggregation is enabled for all actions with a given module name, while using NULL (or empty string) means that aggregation is enabled for an action whose name is an empty string.

Examples

To enable statistic accumulation for a given combination of Service name and MODULE:

```
EXECUTE DBMS_MONITOR.SERV_MOD_ACT_STAT_ENABLE( 'APPS1','PAYROLL');
```

To enable statistic accumulation for a given combination of Service name, MODULE and ACTION:

```
EXECUTE
DBMS_MONITOR.SERV_MOD_ACT_STAT_ENABLE('APPS1','GLEDGER','DEBIT_ENTRY');
```

If both of the preceding commands are issued, statistics are accumulated as follows:

- For the APPS1 service, because accumulation for each Service Name is the default.

- For all actions in the PAYROLL Module.
- For the DEBIT_ENTRY Action within the GLEDGER Module.

SERV_MOD_ACT_TRACE_DISABLE Procedure

This procedure will disable the trace at ALL enabled instances for a given combination of Service Name, MODULE, and ACTION name globally.

Syntax

```
DBMS_MONITOR.SERV_MOD_ACT_TRACE_DISABLE(
   service_name    IN VARCHAR2,
   module_name     IN VARCHAR2,
   action_name     IN VARCHAR2 DEFAULT ALL_ACTIONS,
   instance_name   IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 60–10 SERV_MOD_ACT_TRACE_DISABLE Procedure Parameters

Parameter	Description
service_name	Name of the service for which tracing is disabled.
module_name	Name of the MODULE. An additional qualifier for the service.
action_name	Name of the ACTION. An additional qualifier for the Service and MODULE name.
instance_name	If set, this restricts tracing to the named instance_name.

Usage Notes

Specifying NULL for the module_name parameter means that statistics will no longer be accumulated for the sessions which do not set the MODULE attribute.

Examples

To enable tracing for a Service named APPS1:

```
EXECUTE DBMS_MONITOR.SERV_MOD_ACT_TRACE_ENABLE('APPS1',
            DBMS_MONITOR.ALL_MODULES,  DBMS_MONITOR.ALL_ACTIONS,TRUE,
FALSE,NULL);
```

To disable tracing specified in the previous step:

```
EXECUTE DBMS_MONITOR.SERV_MOD_ACT_TRACE_DISABLE('APPS1');
```

To enable tracing for a given combination of Service and MODULE (all ACTIONs):

```
EXECUTE DBMS_MONITOR.SERV_MOD_ACT_TRACE_ENABLE('APPS1','PAYROLL',
            DBMS_MONITOR.ALL_ACTIONS,TRUE,FALSE,NULL);
```

To disable tracing specified in the previous step:

```
EXECUTE DBMS_MONITOR.SERV_MOD_ACT_TRACE_DISABLE('APPS1','PAYROLL');
```

SERV_MOD_ACT_TRACE_ENABLE Procedure

This procedure will enable SQL tracing for a given combination of Service Name, MODULE and ACTION globally unless an instance_name is specified.

Syntax

```
DBMS_MONITOR.SERV_MOD_ACT_TRACE_ENABLE(
   service_name    IN VARCHAR2,
   module_name     IN VARCHAR2 DEFAULT ANY_MODULE,
   action_name     IN VARCHAR2 DEFAULT ANY_ACTION,
   waits           IN BOOLEAN DEFAULT TRUE,
   binds           IN BOOLEAN DEFAULT FALSE,
   instance_name   IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 60–11 SERV_MOD_ACT_TRACE_ENABLE Procedure Parameters

Parameter	Description
service_name	Name of the service for which tracing is enabled.
module_name	Name of the MODULE. An optional additional qualifier for the service.
action_name	Name of the ACTION. An optional additional qualifier for the Service and MODULE name.
waits	If TRUE, wait information is present in the trace.
binds	If TRUE, bind information is present in the trace.
instance_name	If set, this restricts tracing to the named instance_name.

Usage Notes

- The procedure enables a trace for a given combination of Service, MODULE and ACTION name. The specification is strictly hierarchical: Service Name or Service Name/MODULE, or Service Name, MODULE, and ACTION name must be specified. Omitting a qualifier behaves like a wild-card, so that not specifying an ACTION means all ACTIONs. Using the ALL_ACTIONS constant achieves the same purpose.
- This tracing is useful when an application MODULE and optionally known ACTION is experiencing poor service levels.
- By default, tracing is enabled globally for the database. The instance_name parameter is provided to restrict tracing to named instances that are known, for example, to exhibit poor service levels.
- Tracing information is present in multiple trace files and you must use the trcsess tool to collect it into a single file.
- Specifying NULL for the module_name parameter means that statistics will be accumulated for the sessions which do not set the MODULE attribute.

Examples

To enable tracing for a Service named APPS1:

```
EXECUTE DBMS_MONITOR.SERV_MOD_ACT_TRACE_ENABLE('APPS1',
            DBMS_MONITOR.ALL_MODULES,  DBMS_MONITOR.ALL_ACTIONS,TRUE,
```

```
FALSE,NULL);
```

To enable tracing for a given combination of Service and MODULE (all ACTIONs):

```
EXECUTE DBMS_MONITOR.SERV_MOD_ACT_TRACE_ENABLE('APPS1','PAYROLL',
            DBMS_MONITOR.ALL_ACTIONS,TRUE,FALSE,NULL);
```

SESSION_TRACE_DISABLE Procedure

This procedure will disable the trace for a given database session at the local instance.

Syntax

```
DBMS_MONITOR.SESSION_TRACE_DISABLE(
   session_id      IN     BINARY_INTEGER DEFAULT NULL,
   serial_num      IN     BINARY_INTEGER DEFAULT NULL);
```

Parameters

Table 60–12 SESSION_TRACE_DISABLE Procedure Parameters

Parameter	Description
session_id	Name of the service for which SQL trace is disabled.
serial_num	Serial number for this session.

Usage Notes

If serial_num is NULL but session_id is specified, a session with a given session_id is no longer traced irrespective of its serial number. If both session_id and serial_num are NULL, the current user session is no longer traced. It is illegal to specify NULL session_id and non-NULL serial_num. In addition, the NULL values are default and can be omitted.

Examples

To enable tracing for a client with a given client session ID:

```
EXECUTE DBMS_MONITOR.SESSION_TRACE_ENABLE(7,4634, TRUE, FALSE);
```

To disable tracing specified in the previous step:

```
EXECUTE DBMS_MONITOR.SESSION_TRACE_DISABLE(7,4634);;
```

SESSION_TRACE_ENABLE Procedure

This procedure enables a SQL trace for the given Session ID on the local instance

Syntax

```
DBMS_MONITOR.SESSION_TRACE_ENABLE(
    session_id   IN  BINARY_INTEGER DEFAULT NULL,
    serial_num   IN  BINARY_INTEGER DEFAULT NULL,
    waits        IN  BOOLEAN DEFAULT TRUE,
    binds        IN  BOOLEAN DEFAULT FALSE)
```

Parameters

Table 60–13 SESSION_TRACE_ENABLE Procedure Parameters

Parameter	Description
session_id	Database Session Identifier for which SQL tracing is enabled. Specifying NULL means that my current session should be traced.
serial_num	Serial number for this session. Specifying NULL means that any session which matches session_id (irrespective of serial number) should be traced.
waits	If TRUE, wait information is present in the trace.
binds	If TRUE, bind information is present in the trace.

Usage Notes

The procedure enables a trace for a given database session, and is still useful for client/server applications. The trace is enabled only on the instance to which the caller is connected, since database sessions do not span instances. This tracing is strictly local to an instance.

If serial_num is NULL but session_id is specified, a session with a given session_id is traced irrespective of its serial number. If both session_id and serial_num are NULL, the current user session is traced. It is illegal to specify NULL session_id and non-NULL serial_num. In addition, the NULL values are default and can be omitted.

Examples

To enable tracing for a client with a given client session ID:

```
EXECUTE DBMS_MONITOR.SESSION_TRACE_ENABLE(7,4634, TRUE, FALSE);
```

To disable tracing specified in the previous step:

```
EXECUTE DBMS_MONITOR.SESSION_TRACE_DISABLE(7,4634);
```

Either

```
EXECUTE DBMS_MONITOR.SESSION_TRACE_ENABLE(5);
```

or

```
EXECUTE DBMS_MONITOR.SESSION_TRACE_ENABLE(5, NULL);
```

traces the session with session ID of 5, while either

```
EXECUTE DBMS_MONITOR.SESSION_TRACE_ENABLE();
```

or

```
EXECUTE DBMS_MONITOR.SESSION_TRACE_ENABLE(NULL, NULL);
```

traces the current user session. Also,

```
EXECUTE DBMS_MONITOR.SESSION_TRACE_ENABLE(NULL, NULL, TRUE, TRUE);
```

traces the current user session including waits and binds. The same can be also expressed using keyword syntax:

```
EXECUTE DBMS_MONITOR.SESSION_TRACE_ENABLE(binds=>TRUE);
```

61

DBMS_MVIEW

DBMS_MVIEW enables you to understand capabilities for materialized views and potential materialized views, including their rewrite availability. It also enables you to refresh materialized views that are not part of the same refresh group and purge logs.

Note: DBMS_SNAPSHOT is a synonym for DBMS_MVIEW.

See Also:

- *Oracle Database Advanced Replication* for more information about using materialized views in a replication environment
- *Oracle Database Data Warehousing Guide* for more information about using materialized views in a data warehousing environment

This chapter contains the following topics:

Using DBMS_MVIEW

This section contains topics which relate to using the `DBMS_MVIEW` package.

Operational Notes

If a query is less than 256 characters long, you can invoke `EXPLAIN_REWRITE` using the `EXECUTE` command from SQL*Plus. Otherwise, the recommended method is to use a PL/SQL `BEGIN..END` block, as shown in the examples in `/rdbms/demo/smxrw.sql`.

Rules and Limits

The EXPLAIN_REWRITE procedure cannot accept queries longer than 32627 characters. These restrictions also apply when passing the defining query of a materialized view to the EXPLAIN_MVIEW procedure.

Summary of DBMS_MVIEW Subprograms

Table 61–1 DBMS_MVIEW Package Subprograms

Subprogram	Description
BEGIN_TABLE_ REORGANIZATION Procedure on page 61-6	Performs a process to preserve materialized view data needed for refresh
END_TABLE_ REORGANIZATION Procedure on page 61-7	Ensures that the materialized view data for the master table is valid and that the master table is in the proper state
ESTIMATE_MVIEW_SIZE Procedure on page 61-8	Estimates the size of a materialized view that you might create, in bytes and rows
EXPLAIN_MVIEW Procedure on page 61-9	Explains what is possible with a materialized view or potential materialized view
EXPLAIN_REWRITE Procedure on page 61-10	Explains why a query failed to rewrite or why the optimizer chose to rewrite a query with a particular materialized view or materialized views
I_AM_A_REFRESH Function on page 61-12	Returns the value of the `I_AM_REFRESH` package state
PMARKER Function on page 61-13	Returns a partition marker from a rowid, and is used for Partition Change Tracking (PCT)
PURGE_DIRECT_LOAD_LOG Procedure on page 61-14	Purges rows from the direct loader log after they are no longer needed by any materialized views (used with data warehousing)
PURGE_LOG Procedure on page 61-15	Purges rows from the materialized view log
PURGE_MVIEW_FROM_LOG Procedure on page 61-16	Purges rows from the materialized view log
REFRESH Procedures on page 61-17	Refreshes one or more materialized views that are not members of the same refresh group
REFRESH_ALL_MVIEWS Procedure on page 61-19	Refreshes all materialized views that do not reflect changes to their master table or master materialized view
REFRESH_DEPENDENT Procedures on page 61-20	Refreshes all table-based materialized views that depend on a specified master table or master materialized view, or list of master tables or master materialized views
REGISTER_MVIEW Procedure on page 61-22	Enables the administration of individual materialized views
UNREGISTER_MVIEW Procedure on page 61-24	Enables the administration of individual materialized views once invoked at a master site or master materialized view site to unregister a materialized view

BEGIN_TABLE_REORGANIZATION Procedure

This procedure performs a process to preserve materialized view data needed for refresh. It must be called before a master table is reorganized.

Syntax

```
DBMS_MVIEW.BEGIN_TABLE_REORGANIZATION (
   tabowner    IN   VARCHAR2,
   tabname     IN   VARCHAR2);
```

Parameters

Table 61–2 BEGIN_TABLE_REORGANIZATION Procedure Parameters

Parameter	Description
`tabowner`	Owner of the table being reorganized
`tabname`	Name of the table being reorganized

END_TABLE_REORGANIZATION Procedure

This procedure ensures that the materialized view data for the master table is valid and that the master table is in the proper state. It must be called after a master table is reorganized.

Syntax

```
DBMS_MVIEW.END_TABLE_REORGANIZATION (
   tabowner    IN   VARCHAR2,
   tabname     IN   VARCHAR2);
```

Parameters

Table 61–3 END_TABLE_REORGANIZATION Procedure Parameters

Parameter	Description
tabowner	Owner of the table being reorganized
tabname	Name of the table being reorganized

ESTIMATE_MVIEW_SIZE Procedure

This procedure estimates the size of a materialized view that you might create, in bytes and number of rows.

Syntax

```
DBMS_MVIEW.ESTIMATE_MVIEW_SIZE (
   stmt_id       IN  VARCHAR2,
   select_clause IN  VARCHAR2,
   num_rows      OUT NUMBER,
   num_bytes     OUT NUMBER);
```

Parameters

Table 61–4 ESTIMATE_MVIEW_SIZE Procedure Parameters

Parameter	Description
`stmt_id`	Arbitrary string used to identify the statement in an `EXPLAIN PLAN`
`select_clause`	The `SELECT` statement to be analyzed
`num_rows`	Estimated cardinality
`num_bytes`	Estimated number of bytes

EXPLAIN_MVIEW Procedure

This procedure enables you to learn what is possible with a materialized view or potential materialized view. For example, you can determine if a materialized view is fast refreshable and what types of query rewrite you can perform with a particular materialized view.

Using this procedure is straightforward. You simply call DBMS_MVIEW.EXPLAIN_MVIEW, passing in as parameters the schema and materialized view name for an existing materialized view. Alternatively, you can specify the SELECT string or CREATE MATERIALIZED VIEW statement for a potential materialized view. The materialized view or potential materialized view is then analyzed and the results are written into either a table called MV_CAPABILITIES_TABLE, which is the default, or to an array called MSG_ARRAY.

The procedure is overloaded:

- The first version is for explaining an existing or potential materialized view with output to MV_CAPABILITIES_TABLE.
- The second version is for explaining an existing or potential materialized view with output to a VARRAY.

Syntax

```
DBMS_MVIEW.EXPLAIN_MVIEW (
   mv            IN VARCHAR2,
   statement_id  IN VARCHAR2:= NULL);

DBMS_MVIEW.EXPLAIN_MVIEW (
   mv          IN VARCHAR2,
   msg_array   OUT SYS.ExplainMVArrayType);
```

Parameters

Table 61–5 EXPLAIN_MVIEW Procedure Parameters

Parameter	Description
mv	The name of an existing materialized view (optionally qualified with the owner name separated by a ".") or a SELECT statement or a CREATE MATERIALIZED VIEW statement for a potential materialized view.
statement_id	A client-supplied unique identifier to associate output rows with specific invocations of EXPLAIN_MVIEW
msg_array	The PL/SQL VARRAY that receives the output. Use this parameter to direct EXPLAIN_MVIEW's output to a PL/SQL VARRAY rather than MV_CAPABILITIES_TABLE.

Usage Notes

You must run the utlxmv.sql script to create MV_CAPABILITIES_TABLE in the current schema prior to calling EXPLAIN_MVIEW except when you direct output to a VARRAY. The script is found in the ADMIN directory.

EXPLAIN_REWRITE Procedure

This procedure enables you to learn why a query failed to rewrite, or, if it rewrites, which materialized views will be used. Using the results from the procedure, you can take the appropriate action needed to make a query rewrite if at all possible. The query specified in the EXPLAIN_REWRITE statement is never actually executed.

A demo file, xrwutl.sql, is available to help format the output from EXPLAIN_REWRITE.

Syntax

You can obtain the output from DBMS_MVIEW.EXPLAIN_REWRITE in two ways. The first is to use a table, while the second is to create a VARRAY. The following shows the basic syntax for using an output table:

```
DBMS_MVIEW.EXPLAIN_REWRITE (
    query           VARCHAR2,
    mv              VARCHAR2(30),
    statement_id    VARCHAR2(30));
```

You can create an output table called REWRITE_TABLE by executing the utlxrw.sql script.

The query parameter is a text string representing the SQL query. The parameter, mv, is a fully qualified materialized view name in the form of schema.mv. This is an optional parameter. When it is not specified, EXPLAIN_REWRITE returns any relevant messages regarding all the materialized views considered for rewriting the given query. When schema is omitted and only mv is specified, EXPLAIN_REWRITE looks for the materialized view in the current schema.

If you want to direct the output of EXPLAIN_REWRITE to a VARRAY instead of a table, you should call the procedure as follows:

```
DBMS_MVIEW.EXPLAIN_REWRITE (
    query           [VARCHAR2 | CLOB],
    mv               VARCHAR2(30),
    output_array     SYS.RewriteArrayType);
```

Note that if the query is less than 256 characters long, EXPLAIN_REWRITE can be easily invoked with the EXECUTE command from SQL*Plus. Otherwise, the recommended method is to use a PL/SQL BEGIN... END block, as shown in the examples in /rdbms/demo/smxrw*.

You can also use EXPLAIN_REWRITE with multiple materialized views, in which case the syntax will be the same as with a single materialized view, except that the materialized views are specified by a comma-delimited string. For example, to find out whether a given set of materialized views mv1, mv2, and mv3 could be used to rewrite the query, query_txt, and, if not, why not, use EXPLAIN_REWRITE as follows:

```
DBMS_MVIEW.EXPLAIN_REWRITE(query_txt, 'mv1, mv2, mv3')
```

See *Oracle Database Data Warehousing Guide* for more information on using the EXPLAIN_REWRITE procedure.

Parameters

Table 61–6 EXPLAIN_REWRITE Procedure Parameters

Parameter	Description
query	SQL SELECT statement to be explained
mv	The fully qualified name of an existing materialized view in the form of SCHEMA.MV. For multiple materialized views, you can provide a comma-delimited list of names.
statement_id	A client-supplied unique identifier to distinguish output messages
msg_array	The PL/SQL VARRAY that receives the output. Use this parameter to direct EXPLAIN_REWRITE's output to a PL/SQL VARRAY.

Usage Notes

To obtain the output into a table, you must run the utlxrw.sql script before calling EXPLAIN_REWRITE. This script creates a table named REWRITE_TABLE in the current schema.

I_AM_A_REFRESH Function

This function returns the value of the `I_AM_REFRESH` package state.

Syntax

```
DBMS_MVIEW.I_AM_A_REFRESH
   RETURN BOOLEAN;
```

Return Values

A return value of `true` indicates that all local replication triggers for materialized views are effectively disabled in this session because each replication trigger first checks this state. A return value of `false` indicates that these triggers are enabled.

PMARKER Function

This function returns a partition marker from a rowid. It is used for Partition Change Tracking (PCT).

Syntax

```
DBMS_MVIEW.PMARKER(
   rid IN ROWID)
 RETURN NUMBER;
```

Parameters

Table 61–7 PMARKER Procedure Parameters

Parameter	Description
rid	The rowid of a row entry in a master table

PURGE_DIRECT_LOAD_LOG Procedure

This procedure removes entries from the direct loader log after they are no longer needed for any known materialized view. This procedure usually is used in environments using Oracle's data warehousing technology.

> **See Also:** *Oracle Database Data Warehousing Guide* for more information

Syntax

```
DBMS_MVIEW.PURGE_DIRECT_LOAD_LOG();
```

PURGE_LOG Procedure

This procedure purges rows from the materialized view log.

Syntax

```
DBMS_MVIEW.PURGE_LOG (
   master         IN   VARCHAR2,
   num            IN   BINARY_INTEGER := 1,
   flag           IN   VARCHAR2       := 'NOP');
```

Parameters

Table 61–8 PURGE_LOG Procedure Parameters

Parameter	Description
`master`	Name of the master table or master materialized view.

`num`	Number of least recently refreshed materialized views whose rows you want to remove from materialized view log. For example, the following statement deletes rows needed to refresh the two least recently refreshed materialized views:
	`DBMS_MVIEW.PURGE_LOG('master_table', 2);`
	To delete all rows in the materialized view log, indicate a high number of materialized views to disregard, as in this example:
	`DBMS_MVIEW.PURGE_LOG('master_table',9999);`
	This statement completely purges the materialized view log that corresponds to `master_table` if fewer than 9999 materialized views are based on `master_table`. A simple materialized view whose rows have been purged from the materialized view log must be completely refreshed the next time it is refreshed.

`flag`	Specify `delete` to guarantee that rows are deleted from the materialized view log for at least one materialized view. This parameter can override the setting for the parameter `num`. For example, the following statement deletes rows from the materialized view log that has dependency rows in the least recently refreshed materialized view: `DBMS_MVIEW.PURGE_LOG('master_table',1,'delete');`

PURGE_MVIEW_FROM_LOG Procedure

This procedure is called on the master site or master materialized view site to delete the rows in materialized view refresh related data dictionary tables maintained at the master for the specified materialized view identified by its `mview_id` or the combination of the `mviewowner`, `mviewname`, and the `mviewsite`. If the materialized view specified is the oldest materialized view to have refreshed from any of the master tables or master materialized views, then the materialized view log is also purged. This procedure does not unregister the materialized view.

Syntax

```
DBMS_MVIEW.PURGE_MVIEW_FROM_LOG (
   mview_id       IN   BINARY_INTEGER  |
   mviewowner     IN   VARCHAR2,
   mviewname      IN   VARCHAR2,
   mviewsite      IN   VARCHAR2);
```

> **Note:** This procedure is overloaded. The `mview_id` parameter is mutually exclusive with the three remaining parameters: `mviewowner`, `mviewname`, and `mviewsite`.

Parameters

Table 61–9 PURGE_MVIEW_FROM_LOG Procedure Parameters

Parameter	Description
`mview_id`	If you want to execute this procedure based on the identification of the target materialized view, specify the materialized view identification using the `mview_id` parameter. Query the `DBA_BASE_TABLE_MVIEWS` view at the materialized view log site for a listing of materialized view IDs. Executing this procedure based on the materialized view identification is useful if the target materialized view is not listed in the list of registered materialized views (`DBA_REGISTERED_MVIEWS`).
`mviewowner`	If you do not specify an `mview_id`, enter the owner of the target materialized view using the `mviewowner` parameter. Query the `DBA_REGISTERED_MVIEWS` view at the materialized view log site to view the materialized view owners.
`mviewname`	If you do not specify an `mview_id`, enter the name of the target materialized view using the `mviewname` parameter. Query the `DBA_REGISTERED_MVIEWS` view at the materialized view log site to view the materialized view names.
`mviewsite`	If you do not specify an `mview_id`, enter the site of the target materialized view using the `mviewsite` parameter. Query the `DBA_REGISTERED_MVIEWS` view at the materialized view log site to view the materialized view sites.

Usage Notes

If there is an error while purging one of the materialized view logs, the successful purge operations of the previous materialized view logs are not rolled back. This is to minimize the size of the materialized view logs. In case of an error, this procedure can be invoked again until all the materialized view logs are purged.

REFRESH Procedures

This procedure refreshes a list of materialized views.

Syntax

```
DBMS_MVIEW.REFRESH (
   { list                 IN     VARCHAR2,
   | tab                  IN     DBMS_UTILITY.UNCL_ARRAY,}
   method                 IN     VARCHAR2       := NULL,
   rollback_seg           IN     VARCHAR2       := NULL,
   push_deferred_rpc      IN     BOOLEAN        := true,
   refresh_after_errors   IN     BOOLEAN        := false,
   purge_option           IN     BINARY_INTEGER := 1,
   parallelism            IN     BINARY_INTEGER := 0,
   heap_size              IN     BINARY_INTEGER := 0,
   atomic_refresh         IN     BOOLEAN        := true,
   nested                 IN     BOOLEAN        := false);
```

Note: This procedure is overloaded. The `list` and `tab` parameters are mutually exclusive.

Parameters

Table 61–10 REFRESH Procedure Parameters

Parameter	Description
`list | tab`	Comma-delimited list of materialized views that you want to refresh. (Synonyms are not supported.) These materialized views can be located in different schemas and have different master tables or master materialized views. However, all of the listed materialized views must be in your local database. Alternatively, you may pass in a PL/SQL index-by table of type `DBMS_UTILITY.UNCL_ARRAY`, where each element is the name of a materialized view.

`method`	A string of refresh methods indicating how to refresh the listed materialized views. An `f` indicates fast refresh, `?` indicates force refresh, `C` or `c` indicates complete refresh, and `A` or `a` indicates always refresh. `A` and `C` are equivalent. `P` or `p` refreshes by recomputing the rows in the materialized view affected by changed partitions in the detail tables.
	
	If a materialized view does not have a corresponding refresh method (that is, if more materialized views are specified than refresh methods), then that materialized view is refreshed according to its default refresh method. For example, consider the following `EXECUTE` statement within SQL*Plus:
	
	`DBMS_MVIEW.REFRESH`
	`('countries_mv,regions_mv,hr.employees_mv','cf');`
	
	This statement performs a complete refresh of the `countries_mv` materialized view, a fast refresh of the `regions_mv` materialized view, and a default refresh of the `hr.employees` materialized view.

`rollback_seg`	Name of the materialized view site rollback segment to use while refreshing materialized views

Table 61–10 (Cont.) REFRESH Procedure Parameters

Parameter	Description
`push_deferred_rpc`	Used by updatable materialized views only. Set this parameter to `true` if you want to push changes from the materialized view to its associated master tables or master materialized views before refreshing the materialized view. Otherwise, these changes may appear to be temporarily lost.
`refresh_after_errors`	If this parameter is `true`, an updatable materialized view continues to refresh even if there are outstanding conflicts logged in the `DEFERROR` view for the materialized view's master table or master materialized view. If this parameter is `true` and `atomic_refresh` is `false`, this procedure continues to refresh other materialized views if it fails while refreshing a materialized view.
`purge_option`	If you are using the parallel propagation mechanism (in other words, parallelism is set to 1 or greater), 0 means do not purge, 1 means lazy purge, and 2 means aggressive purge. In most cases, lazy purge is the optimal setting. Set purge to aggressive to trim the queue if multiple master replication groups are pushed to different target sites, and updates to one or more replication groups are infrequent and infrequently pushed. If all replication groups are infrequently updated and pushed, then set this parameter to `0` and occasionally execute `PUSH` with this parameter set to `2` to reduce the queue.
`parallelism`	0 specifies serial propagation. *n* > 1 specifies parallel propagation with *n* parallel processes. 1 specifies parallel propagation using only one parallel process.
`heap_size`	Maximum number of transactions to be examined simultaneously for parallel propagation scheduling. Oracle automatically calculates the default setting for optimal performance. Note**:** Do not set this parameter unless directed to do so by Oracle Support Services.
`atomic_refresh`	If this parameter is set to `true`, then the list of materialized views is refreshed in a single transaction. All of the refreshed materialized views are updated to a single point in time. If the refresh fails for any of the materialized views, none of the materialized views are updated. If this parameter is set to `false`, then each of the materialized views is refreshed in a separate transaction.
`nested`	If `true`, then perform nested refresh operations for the specified set of materialized views. Nested refresh operations refresh all the depending materialized views and the specified set of materialized views based on a dependency order to ensure the nested materialized views are truly fresh with respect to the underlying base tables.

REFRESH_ALL_MVIEWS Procedure

This procedure refreshes all materialized views that have the following properties:

- The materialized view has not been refreshed since the most recent change to a master table or master materialized view on which it depends.
- The materialized view and all of the master tables or master materialized views on which it depends are local.
- The materialized view is in the view DBA_MVIEWS.

This procedure is intended for use with data warehouses.

Syntax

```
DBMS_MVIEW.REFRESH_ALL_MVIEWS (
   number_of_failures     OUT   BINARY_INTEGER,
   method                 IN    VARCHAR2         := NULL,
   rollback_seg           IN    VARCHAR2         := NULL,
   refresh_after_errors   IN    BOOLEAN          := false,
   atomic_refresh         IN    BOOLEAN          := true);
```

Parameters

Table 61–11 REFRESH_ALL_MVIEWS Procedure Parameters

Parameter	Description
number_of_ failures	Returns the number of failures that occurred during processing
method	A single refresh method indicating the type of refresh to perform for each materialized view that is refreshed. F or f indicates fast refresh, ? indicates force refresh, C or c indicates complete refresh, and A or a indicates always refresh. A and C are equivalent. If no method is specified, a materialized view is refreshed according to its default refresh method. P or p refreshes by recomputing the rows in the materialized view affected by changed partitions in the detail tables.
rollback_seg	Name of the materialized view site rollback segment to use while refreshing materialized views
refresh_after_ errors	If this parameter is true, an updatable materialized view continues to refresh even if there are outstanding conflicts logged in the DEFERROR view for the materialized view's master table or master materialized view. If this parameter is true and atomic_refresh is false, this procedure continues to refresh other materialized views if it fails while refreshing a materialized view.
atomic_refresh	If this parameter is set to true, then the refreshed materialized views are refreshed in a single transaction. All of the refreshed materialized views are updated to a single point in time. If the refresh fails for any of the materialized views, none of the materialized views are updated. If this parameter is set to false, then each of the refreshed materialized views is refreshed in a separate transaction.

REFRESH_DEPENDENT Procedures

This procedure refreshes all materialized views that have the following properties:

- The materialized view depends on a master table or master materialized view in the list of specified masters.
- The materialized view has not been refreshed since the most recent change to a master table or master materialized view on which it depends.
- The materialized view and all of the master tables or master materialized views on which it depends are local.
- The materialized view is in the view DBA_MVIEWS.

This procedure is intended for use with data warehouses.

Syntax

```
DBMS_MVIEW.REFRESH_DEPENDENT (
   number_of_failures     OUT    BINARY_INTEGER,
   { list                 IN     VARCHAR2,
   | tab                  IN     DBMS_UTILITY.UNCL_ARRAY,}
   method                 IN     VARCHAR2    := NULL,
   rollback_seg           IN     VARCHAR2    := NULL,
   refresh_after_errors   IN     BOOLEAN     := false,
   atomic_refresh         IN     BOOLEAN     := true,
   nested                 IN     BOOLEAN     := false);
```

Note: This procedure is overloaded. The list and tab parameters are mutually exclusive.

Parameters

Table 61–12 REFRESH_DEPENDENT Procedure Parameters

Parameter	Description
number_of_ failures	Returns the number of failures that occurred during processing
list | tab	Comma-delimited list of master tables or master materialized views on which materialized views can depend. (Synonyms are not supported.) These tables and the materialized views that depend on them can be located in different schemas. However, all of the tables and materialized views must be in your local database. Alternatively, you may pass in a PL/SQL index-by table of type DBMS_ UTILITY.UNCL_ARRAY, where each element is the name of a table.

Table 61–12 (Cont.) REFRESH_DEPENDENT Procedure Parameters

Parameter	Description
method	A string of refresh methods indicating how to refresh the dependent materialized views. All of the materialized views that depend on a particular table are refreshed according to the refresh method associated with that table. F or f indicates fast refresh, ? indicates force refresh, C or c indicates complete refresh, and A or a indicates always refresh. A and C are equivalent. P or p refreshes by recomputing the rows in the materialized view affected by changed partitions in the detail tables. If a table does not have a corresponding refresh method (that is, if more tables are specified than refresh methods), then any materialized view that depends on that table is refreshed according to its default refresh method. For example, the following EXECUTE statement within SQL*Plus: `DBMS_MVIEW.REFRESH_DEPENDENT ('employees,deptartments,hr.regions','cf');` performs a complete refresh of the materialized views that depend on the employees table, a fast refresh of the materialized views that depend on the departments table, and a default refresh of the materialized views that depend on the hr.regions table.
rollback_seg	Name of the materialized view site rollback segment to use while refreshing materialized views
refresh_after_errors	If this parameter is true, an updatable materialized view continues to refresh even if there are outstanding conflicts logged in the DEFERROR view for the materialized view's master table or master materialized view. If this parameter is true and atomic_refresh is false, this procedure continues to refresh other materialized views if it fails while refreshing a materialized view.
atomic_refresh	If this parameter is set to true, then the refreshed materialized views are refreshed in a single transaction. All of the refreshed materialized views are updated to a single point in time. If the refresh fails for any of the materialized views, none of the materialized views are updated. If this parameter is set to false, then each of the refreshed materialized views is refreshed in a separate transaction.
nested	If true, then perform nested refresh operations for the specified set of tables. Nested refresh operations refresh all the depending materialized views of the specified set of tables based on a dependency order to ensure the nested materialized views are truly fresh with respect to the underlying base tables.

REGISTER_MVIEW Procedure

This procedure enables the administration of individual materialized views. It is invoked at a master site or master materialized view site to register a materialized view.

Note that, typically, a materialized view is registered automatically during materialized view creation. You should only run this procedure to manually register a materialized view if the automatic registration failed or if the registration information was deleted.

Syntax

```
DBMS_MVIEW.REGISTER_MVIEW (
   mviewowner  IN   VARCHAR2,
   mviewname   IN   VARCHAR2,
   mviewsite   IN   VARCHAR2,
   mview_id    IN   DATE | BINARY_INTEGER,
   flag        IN   BINARY_INTEGER,
   qry_txt     IN   VARCHAR2,
   rep_type    IN   BINARY_INTEGER := DBMS_MVIEW.REG_UNKNOWN);
```

Parameters

Table 61–13 REGISTER_MVIEW Procedure Parameters

Parameter	Description
mviewowner	Owner of the materialized view.
mviewname	Name of the materialized view.
mviewsite	Name of the materialized view site for a materialized view registering at an Oracle database version 8.x and higher master site or master materialized view site. This name should not contain any double quotes.
mview_id	The identification number of the materialized view. Specify an Oracle database version 8.x and higher materialized view as a BINARY_INTEGER. Specify an Oracle database version 7 materialized view registering at an Oracle database version 8.x and higher master sites or master materialized view sites as a DATE.
flag	A constant that describes the properties of the materialized view being registered. Valid constants that can be assigned include the following: DBMS_MVIEW.REG_ROWID_MVIEW for a rowid materialized view DBMS_MVIEW.REG_PRIMARY_KEY_MVIEW for a primary key materialized view DBMS_MVIEW.REG_OBJECT_ID_MVIEW for an object id materialized view DBMS_MVIEW.REG_FAST_REFRESHABLE_MVIEW for a materialized view that can be fast refreshed DBMS_MVIEW.REG_UPDATABLE_MVIEW for a materialized view that is updatable A materialized view can have more than one of these properties. In this case, use the plus sign (+) to specify more than one property. For example, if a primary key materialized view can be fast refreshed, you can enter the following for this parameter: DBMS_MVIEW.REG_PRIMARY_KEY_MVIEW + DBMS_MVIEW.REG_FAST_REFRESHABLE_MVIEW You can determine the properties of a materialized view by querying the ALL_MVIEWS data dictionary view.
qry_txt	The first 32,000 bytes of the materialized view definition query.

Table 61–13 (Cont.) REGISTER_MVIEW Procedure Parameters

Parameter	Description
rep_type	Version of the materialized view. Valid constants that can be assigned include the following: DBMS_MVIEW.REG_V7_SNAPSHOT if the materialized view is at an Oracle database version 7 site ■ DBMS_MVIEW.REG_V8_SNAPSHOT reg_repapi_snapshot if the materialized view is at an Oracle database version 8.x or higher site DBMS_MVIEW.REG_UNKNOWN (the default) if you do not know whether the materialized view is at an Oracle database version 7 site or an Oracle database version 8.x (or higher) site

Usage Notes

This procedure is invoked at the master site or master materialized view site by a remote materialized view site using a remote procedure call. If REGISTER_MVIEW is called multiple times with the same mviewowner, mviewname, and mviewsite, then the most recent values for mview_id, flag, and qry_txt are stored. If a query exceeds the maximum VARCHAR2 size, then qry_txt contains the first 32000 characters of the query and the remainder is truncated. When invoked manually, the value of mview_id must be looked up in the materialized view data dictionary views by the person who calls the procedure.

UNREGISTER_MVIEW Procedure

This procedure enables the administration of individual materialized views. It is invoked at a master site or master materialized view site to unregister a materialized view.

Syntax

```
DBMS_MVIEW.UNREGISTER_MVIEW (
   mviewowner      IN   VARCHAR2,
   mviewname       IN   VARCHAR2,
   mviewsite       IN   VARCHAR2);
```

Parameters

Table 61–14 UNREGISTER_MVIEW Procedure Parameters

Parameters	Description
mviewowner	Owner of the materialized view
mviewname	Name of the materialized view
mviewsite	Name of the materialized view site

62

DBMS_OBFUSCATION_TOOLKIT

DBMS_OBFUSCATION_TOOLKIT enables an application to encrypt data using either the Data Encryption Standard (DES) or the Triple DES algorithms.

This chapter contains the following topics:

Using DBMS_OBFUSCATION_TOOLKIT

Overview

The Data Encryption Standard (DES), also known as the Data Encryption Algorithm (DEA) by the American National Standards Institute (ANSI) and DEA-1 by the International Standards Organization (ISO), has been a worldwide encryption standard for over 20 years. The banking industry has also adopted DES-based standards for transactions between private financial institutions, and between financial institutions and private individuals. DES will eventually be replaced by a new Advanced Encryption Standard (AES).

DES is a symmetric key cipher; that is, the same key is used to encrypt data as well as decrypt data. DES encrypts data in 64-bit blocks using a 56-bit key. The DES algorithm ignores 8 bits of the 64-bit key that is supplied; however, you must supply a 64-bit key to the algorithm.

Triple DES (3DES) is a far stronger cipher than DES; the resulting ciphertext (encrypted data) is much harder to break using an exhaustive search: 2**112 or 2**168 attempts instead of 2**56 attempts. Triple DES is also not as vulnerable to certain types of cryptanalysis as is DES.

Security Model

Oracle installs this package in the SYS schema. You can then grant package access to existing users and roles as needed. The package also grants access to the PUBLIC role so no explicit grant needs to be done.

Operational Notes

Key Management

Key management, including both generation and secure storage of cryptographic keys, is one of the most important aspects of encryption. If keys are poorly chosen or stored improperly, then it is far easier for a malefactor to break the encryption. Rather than using an exhaustive key search attack (that is, cycling through all the possible keys in hopes of finding the correct decryption key), cryptanalysts typically seek weaknesses in the choice of keys, or the way in which keys are stored.

Key generation is an important aspect of encryption. Typically, keys are generated automatically through a random-number generator. Provided that the random number generation is cryptographically secure, this can be an acceptable form of key generation. However, if random numbers are not cryptographically secure, but have elements of predictability, the security of the encryption may be easily compromised.

The `DBMS_OBFUSCATION_TOOLKIT` package includes tools for generating random material that can be used for encryption keys, but it does not provide a mechanism for maintaining them. Care must be taken by the application developer to ensure the secure generation and storage of encryption keys used with this package. Furthermore, the encryption and decryption done by the `DBMS_OBFUSCATION_TOOLKIT` takes place on the server, not the client. If the key is passed over the connection between the client and the server, the connection must be protected by using network encryption. Otherwise, the key is vulnerable to capture over the wire. See *Oracle Database Advanced Security Administrator's Guide* for information about configuring and using network encryption for Oracle Net.

Key storage is one of the most important, yet difficult aspects of encryption and one of the hardest to manage properly. To recover data encrypted with a symmetric key, the key must be accessible to the application or user seeking to decrypt data. The key needs to be easy enough to retrieve that users can access encrypted data when they need to without significant performance degradation. The key also needs to be secure enough that it is not easily recoverable by unauthorized users trying to access encrypted data that they are not supposed to see.

The three options available are:

- Store the key in the database
- Store the key in the operating system
- Have the user manage the key

Storing the Key in the Database

Storing the keys in the database cannot always provide bullet-proof security if you are trying to protect data against the DBA accessing encrypted data (since an all-privileged DBA can access tables containing encryption keys), but it can provide security against the casual snooper, or against someone compromising the database files on the operating system. Furthermore, the security you can obtain by storing keys in the database does not have to be bullet-proof in order to be extremely useful.

For example, suppose you want to encrypt an employee's social security number, one of the columns in table EMP. You could encrypt each employee's SSN using a key which is stored in a separate column in EMP. However, anyone with SELECT access on the EMP table could retrieve the encryption key and decrypt the matching social security number. Alternatively, you could store the encryption keys in another table, and use a package to retrieve the correct key for the encrypted data item, based on a primary key-foreign key relationship between the tables.

You can envelope both the `DBMS_OBFUSCATION_TOOLKIT` package and the procedure to retrieve the encryption keys supplied to the package. Furthermore, the encryption key itself could be transformed in some way (for example, XORed with the foreign key to the EMP table) so that the key itself is not stored in easily recoverable form.

Oracle recommends using the wrap utility of PL/SQL to obfuscate the code within a PL/SQL package itself that does the encryption. That prevents people from breaking the encryption by looking at the PL/SQL code that handles keys, calls encrypting routines, and so on. In other words, use the wrap utility to obfuscate the PL/SQL packages themselves. This scheme is secure enough to prevent users with `SELECT` access to `EMP` from reading unencrypted sensitive data, and a DBA from easily retrieving encryption keys and using them to decrypt data in the `EMP` table. It can be made more secure by changing encryption keys regularly, or having a better key storage algorithm (so the keys themselves are encrypted, for example).

Storing the Key in the Operating System

Storing keys in a flat file in the operating system is another option. You can make callouts from PL/SQL, which you can use to retrieve encryption keys. If you store keys in a file and make callouts to retrieve the keys, the security of your encrypted data is only as secure as the protection of the key file on the operating system. Of course, a user retrieving keys from the operating system would have to be able to either access the Oracle database files (to decrypt encrypted data), or be able to gain access to the table in which the encrypted data is stored as a legitimate user.

User-Supplied Keys

If you ask a user to supply the key, it is crucial that you use network encryption, such as that provided by Oracle Advanced Security, so the key is not passed from client to server in the clear. The user must remember the key, or your data is not recoverable.

Summary of DBMS_OBFUSCATION Subprograms

Table 62–1 DBMS_OBFUSCATION Package Subprograms

Subprogram	Description
DES3DECRYPT Procedures and Functions on page 62-8	Generates the decrypted form of the input data
DES3ENCRYPT Procedures and Functions on page 62-10	Generates the encrypted form of the input data by passing it through the Triple DES encryption algorithm
DES3GETKEY Procedures and Functions on page 62-8	Takes a random value and uses it to generate an encryption key, using Triple DES
DESDECRYPT Procedures and Functions on page 62-13	Generates the decrypted form of the input data
DESENCRYPT Procedures and Functions on page 62-15	Generates the encrypted form of the input data
DESGETKEY Procedures and Functions on page 62-17	Takes a random value and uses it to generate an encryption key
MD5 Procedures and Functions on page 62-18	Generates MD5 hashes of data

DES3DECRYPT Procedures and Functions

These subprograms generate the decrypted form of the input data.

For a discussion of the initialization vector that you can use with this procedure, see the section, "DES3ENCRYPT Procedures and Functions" on page 62-10.

Syntax

```
DBMS_OBFUSCATION_TOOLKIT.DES3DECRYPT(
   input             IN   RAW,
   key               IN   RAW,
   decrypted_data    OUT  RAW,
   which             IN   PLS_INTEGER  DEFAULT TwoKeyMode
   iv                IN   RAW          DEFAULT NULL);

DBMS_OBFUSCATION_TOOLKIT.DES3DECRYPT(
   input_string      IN   VARCHAR2,
   key_string        IN   VARCHAR2,
   decrypted_string  OUT  VARCHAR2,
   which             IN   PLS_INTEGER  DEFAULT TwoKeyMode
   iv_string         IN   VARCHAR2     DEFAUTL NULL);

DBMS_OBFUSCATION_TOOLKIT.DES3DECRYPT(
   input         IN  RAW,
   key           IN  RAW,
   which         IN  PLS_INTEGER DEFAULT TwoKeyMode
   iv            IN  RAW         DEFAULT NULL)
  RETURN RAW;

DBMS_OBFUSCATION_TOOLKIT.DES3DECRYPT(
   input_string  IN  VARCHAR2,
   key_string    IN  VARCHAR2,
   which         IN  PLS_INTEGER DEFAULT TwoKeyMode
   iv_string     IN  VARCHAR2    DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 62–2 DES3DECRYPT Parameters for Raw Data

Parameter	Description
input	Data to be decrypted
key	Decryption key
decrypted_data	Decrypted data
which	If = 0, (default), then TwoKeyMode is used. If = 1, then ThreeKeyMode is used.
iv	Initialization vector
input_string	String to be decrypted
key_string	Decryption key string
decrypted_ string	Decrypted string
iv_string	Initialization vector

Usage Notes

If the input data or key given to the DES3DECRYPT procedure is empty, then the procedure raises the error ORA-28231 "Invalid input to Obfuscation toolkit."

If the input data given to the DES3DECRYPT procedure is not a multiple of 8 bytes, the procedure raises the error ORA-28232 "Invalid input size for Obfuscation toolkit." ORA-28233 is NOT applicable for the DES3DECRYPT function.

If the key length is missing or is less than 8 bytes, then the procedure raises the error ORA-28234 "Key length too short." Note that if larger keys are used, extra bytes are ignored. So a 9-byte key will not generate an exception.

If an incorrect value is specified for the WHICH parameter, ORA-28236 "Invalid Triple DES mode" is generated. Only the values 0 (TwoKeyMode) and 1 (ThreeKeyMode) are valid.

Restrictions

You must supply a single key of either 128 bits for a 2-key implementation (of which only 112 are used), or a single key of 192 bits for a 3-key implementation (of which 168 bits are used). Oracle automatically truncates the supplied key into 56-bit lengths for decryption. This key length is fixed and cannot be altered.

Note: Both the key length limitation and the prevention of multiple encryption passes are requirements of U.S. regulations governing the export of cryptographic products.

DES3ENCRYPT Procedures and Functions

These subprograms generate the encrypted form of the input data by passing it through the Triple DES (3DES) encryption algorithm.

Oracle's implementation of 3DES supports either a 2-key or 3-key implementation, in outer cipher-block-chaining (CBC) mode.

Syntax

```
DBMS_OBFUSCATION_TOOLKIT.DES3Encrypt(
   input           IN     RAW,
   key             IN     RAW,
   encrypted_data  OUT    RAW,
   which           IN     PLS_INTEGER  DEFAULT TwoKeyMode
   iv              IN     RAW          DEFAULT NULL);

DBMS_OBFUSCATION_TOOLKIT.DES3Encrypt(
   input_string      IN     VARCHAR2,
   key_string        IN     VARCHAR2,
   encrypted_string  OUT    VARCHAR2,
   which             IN     PLS_INTEGER  DEFAULT TwoKeyMode
   iv_string         IN     VARCHAR2     DEFAULT NULL);

DBMS_OBFUSCATION_TOOLKIT.DES3Encrypt(
   input        IN RAW,
   key          IN RAW,
   which        IN PLS_INTEGER DEFAULT TwoKeyMode
   iv           IN RAW         DEFAULT NULL)
  RETURN RAW;

DBMS_OBFUSCATION_TOOLKIT.DES3Encrypt(
   input_string  IN VARCHAR2,
   key_string    IN VARCHAR2,
   which         IN PLS_INTEGER DEFAULT TwoKeyMode
   iv_string     IN VARCHAR2    DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 62–3 DES3ENCRYPT Parameters Procedure and Function

Parameter	Description
`input`	Data to be encrypted.
`key`	Encryption key.
`encrypted_data`	Encrypted data.
`which`	If = 0, (default), then `TwoKeyMode` is used. If = 1, then `ThreeKeyMode` is used.
iv	Initialization vector.
`input_string`	String to be encrypted.
`key_string`	Encryption key string.
`encrypted_ string`	Encrypted string.
iv_string	Initialization vector.

Usage Notes

If you are using Oracle's 3DES interface with a 2-key implementation, you must supply a single key of 128 bits as an argument to the DES3ENCRYPT procedure. With a 3-key implementation, you must supply a single key of 192 bits. Oracle then breaks the supplied key into two 64-bit keys. As with DES, the 3DES algorithm throws away 8 bits of each derived key. However, you must supply a single 128-bit key for the 2-key 3DES implementation or a single 192-bit key for the 3-key 3DES implementation; otherwise the package will raise an error. The DES3ENCRYPT procedure uses the 2-key implementation by default.

You also have the option of providing an *initialization vector* (IV) with the DES3ENCRYPT procedure. An IV is a block of random data prepended to the data you intend to encrypt. The IV has no meaning. It is there to make each message unique. Prepending an IV to your input data avoids starting encrypted blocks of data with common header information, which may give cryptanalysts information they can use to decrypt your data.

If the input data or key given to the PL/SQL DES3ENCRYPT procedure is empty, then the procedure raises the error `ORA-28231 "Invalid input to Obfuscation toolkit."`

If the input data given to the DES3ENCRYPT procedure is not a multiple of 8 bytes, the procedure raises the error `ORA-28232 "Invalid input size for Obfuscation toolkit."`

If you try to double encrypt data using the DES3ENCRYPT procedure, then the procedure raises the error `ORA-28233 "Double encryption not supported."`

If the key length is missing or is less than 8 bytes, then the procedure raises the error `ORA-28234 "Key length too short."` Note that if larger keys are used, extra bytes are ignored. So a 9-byte key will not generate an exception.

If an incorrect value is specified for the `which` parameter, `ORA-28236 "Invalid Triple DES mode"` is generated. Only the values 0 (`TwoKeyMode`) and 1 (`ThreeKeyMode`) are valid.

Restrictions

The DES3ENCRYPT procedure has two restrictions. The first is that the DES key length for encryption is fixed at 128 bits (for 2-key DES) or 192 bits (for 3-key DES); you cannot alter these key lengths.

The second is that you cannot execute multiple passes of encryption using 3DES. (Note: the 3DES algorithm itself encrypts data multiple times; however, you cannot call the DES3ENCRYPT function itself more than once to encrypt the same data using 3DES.)

> **Note:** Both the key length limitation and the prevention of multiple encryption passes are requirements of U.S. regulations governing the export of cryptographic products.

DES3GETKEY Procedures and Functions

These subprograms take a random value and uses it to generate an encryption key. For Triple DES, you specify the mode so that the returned key has the proper length.

Syntax

```
DBMS_OBFUSCATION_TOOLKIT.DES3GetKey(
   which        IN   PLS_INTEGER DEFAULT TwoKeyMode,
   seed         IN   RAW,
   key          OUT  RAW);

DBMS_OBFUSCATION_TOOLKIT.DES3GetKey(
   which        IN   PLS_INTEGER DEFAULT TwoKeyMode,
   seed_string  IN   VARCHAR2,
   key          OUT  VARCHAR2);

DBMS_OBFUSCATION_TOOLKIT.DES3GetKey(
   which  IN  PLS_INTEGER DEFAULT TwoKeyMode,
   seed   IN  RAW)
 RETURN RAW;

DBMS_OBFUSCATION_TOOLKIT.DES3GetKey(
   which        IN  PLS_INTEGER DEFAULT TwoKeyMode,
   seed_string  IN  VARCHAR2)
 RETURN VARCHAR2;
```

Parameters

Table 62–4 DES3GETKEY Procedure and Function Parameters

Parameter	Description
`which`	If = 0, (default), then `TwoKeyMode` is used. If = 1, then `ThreeKeyMode` is used.
`seed`	A value at least 80 characters long.
`key`	Encryption key.
`seed_string`	A value at least 80 characters long.
`key`	Encryption key.

DESDECRYPT Procedures and Functions

These subprograms generate the decrypted form of the input data.

Syntax

```
DBMS_OBFUSCATION_TOOLKIT.DESDecrypt(
   input             IN   RAW,
   key               IN   RAW,
   decrypted_data    OUT  RAW);

DBMS_OBFUSCATION_TOOLKIT.DESDecrypt(
   input_string      IN   VARCHAR2,
   key_string        IN   VARCHAR2,
   decrypted_string  OUT  VARCHAR2);

DBMS_OBFUSCATION_TOOLKIT.DESDecrypt(
   input            IN  RAW,
   key              IN  RAW)
  RETURN RAW;

DBMS_OBFUSCATION_TOOLKIT.DESDecrypt(
   input_string     IN  VARCHAR2,
   key_string       IN  VARCHAR2)
  RETURN VARCHAR2;
```

Parameters

Table 62–5 DESDECRYPT Procedure and Function Parameters

Parameter	Description
`input`	Data to be decrypted.
`key`	Decryption key.
`decrypted_data`	Decrypted data.
`input_string`	String to be decrypted.
`key_string`	Decryption key string.
`decrypted_string`	Decrypted string.

Usage Notes

If the input data or key given to the PL/SQL `DESDECRYPT` function is empty, then Oracle raises `ORA error 28231 "Invalid input to Obfuscation toolkit."`

If the input data given to the `DESDECRYPT` function is not a multiple of 8 bytes, Oracle raises `ORA error 28232 "Invalid input size for Obfuscation toolkit."`

If the key length is missing or is less than 8 bytes, then the procedure raises the error `ORA-28234 "Key length too short."` Note that if larger keys are used, extra bytes are ignored. So a 9-byte key will not generate an exception.

> **Note:** `ORA-28233` is not applicable to the `DESDECRYPT` function.

Restrictions

The DES key length for encryption is fixed at 64 bits (of which 56 bits are used); you cannot alter this key length.

Note: The key length limitation is a requirement of U.S. regulations governing the export of cryptographic products.

DESENCRYPT Procedures and Functions

These subprograms generate the encrypted form of the input data.

Syntax

```
DBMS_OBFUSCATION_TOOLKIT.DESEncrypt(
   input            IN    RAW,
   key              IN    RAW,
   encrypted_data   OUT   RAW);

DBMS_OBFUSCATION_TOOLKIT.DESEncrypt(
   input_string     IN    VARCHAR2,
   key_string       IN    VARCHAR2,
   encrypted_string OUT   VARCHAR2);

DBMS_OBFUSCATION_TOOLKIT.DESEncrypt(
   input         IN  RAW,
   key           IN  RAW)
  RETURN RAW;

DBMS_OBFUSCATION_TOOLKIT.DESEncrypt(
   input_string  IN  VARCHAR2,
   key_string    IN  VARCHAR2)
  RETURN VARCHAR2;
```

Parameters

Table 62–6 DESENCRYPT Procedure and Function Parameters

Parameter	Description
`input`	Data to be encrypted.
`key`	Encryption key.
`encrypted_data`	Encrypted data.
`input_string`	String to be encrypted.
`key_string`	Encryption key string.
`encrypted_string`	Encrypted string.

Usage Notes

The DES algorithm encrypts data in 64-bit blocks using a 56-bit key. The DES algorithm throws away 8 bits of the supplied key (the particular bits which are thrown away is beyond the scope of this documentation). However, when using the algorithm, you must supply a 64-bit key or the package will raise an error.

If the input data or key given to the PL/SQL DESEncrypt procedure is empty, then the procedure raises the error ORA-28231 "Invalid input to Obfuscation toolkit".

If the input data given to the `DESENCRYPT` procedure is not a multiple of 8 bytes, the procedure raises the error `ORA-28232 "Invalid input size for Obfuscation toolkit."`

If you try to double-encrypt data using the `DESENCRYPT` procedure, then the procedure raises the error `ORA-28233 "Double encryption not supported."`

If the key length is missing or is less than 8 bytes, then the procedure raises the error `ORA-28234 "Key length too short."` Note that if larger keys are used, extra bytes are ignored. So a 9-byte key will not generate an exception.

Restrictions

The `DESENCRYPT` procedure has the following restrictions:

- The DES key length for encryption is fixed at 56 bits; you cannot alter this key length.
- You cannot execute multiple passes of encryption. That is, you cannot re-encrypt previously encrypted data by calling the function twice.

> **Note:** Both the key length limitation and the prevention of multiple encryption passes are requirements of U.S. regulations governing the export of cryptographic products.

DESGETKEY Procedures and Functions

These subprograms take a random value and use it to generate an encryption key.

Syntax

```
DBMS_OBFUSCATION_TOOLKIT.DESGetKey(
   seed         IN   RAW,
   key          OUT  RAW);

DBMS_OBFUSCATION_TOOLKIT.DESGetKey(
   seed_string  IN   VARCHAR2,
   key          OUT  VARCHAR2);

DBMS_OBFUSCATION_TOOLKIT.DESGetKey(
   seed IN RAW)
 RETURN RAW;

DBMS_OBFUSCATION_TOOLKIT.DESGetKey(
   seed_string IN VARCHAR2)
 RETURN VARCHAR2;
```

Parameters

Table 62–7 DESGETKEY Procedure and Function Parameters

Parameter	Description
`seed`	A value at least 80 characters long.
`key`	Encryption key.
`seed_string`	A value at least 80 characters long.
`key`	Encryption key.

MD5 Procedures and Functions

These subprograms generate MD5 hashes of data. The MD5 algorithm ensures data integrity by generating a 128-bit cryptographic message digest value from given data.

Syntax

```
DBMS_OBFUSCATION_TOOLKIT.MD5(
   input            IN   RAW,
   checksum         OUT  raw_checksum);

DBMS_OBFUSCATION_TOOLKIT.MD5(
   input_string     IN   VARCHAR2,
   checksum_string  OUT  varchar2_checksum);

DBMS_OBFUSCATION_TOOLKIT.MD5(
   input         IN  RAW)
  RETURN raw_checksum;

DBMS_OBFUSCATION_TOOLKIT.MD5(
   input_string  IN  VARCHAR2)
  RETURN varchar2_checksum;
```

Parameters

Table 62–8 MD5 Procedure and Function Parameters

Parameter Name	Description
`input`	Data to be hashed
`checksum`	128-bit cryptographic message digest
`input_string`	Data to be hashed
`checksum_ string`	128-bit cryptographic message digest

63

DBMS_ODCI

`DBMS_ODCI` package contains a single user function related to the use of Data Cartridges.

See Also:

- *Oracle Database Data Cartridge Developer's Guide*

This chapter contains the following topic:

Summary of DBMS_ODCI Subprograms

Table 63–1 DBMS_ODCI Package Subprograms

Subprogram	Description
ESTIMATE_CPU_UNITS Function on page 63-3	Returns the approximate number of CPU instructions (in thousands) corresponding to a specified time interval (in seconds)

ESTIMATE_CPU_UNITS Function

This function returns the approximate number of CPU instructions (in thousands) corresponding to a specified time interval (in seconds). This information can be used to associate the CPU cost with a user-defined function for the extensible optimizer.

The function takes as input the elapsed time of the user function, measures CPU units by multiplying the elapsed time by the processor speed of the machine, and returns the approximate number of CPU instructions that should be associated with the user function. For a multiprocessor machine, ESTIMATE_CPU_UNITS considers the speed of a single processor.

Syntax

```
DBMS_ODCI.ESTIMATE_CPU_UNITS(
   elapsed_time     NUMBER)
  RETURN NUMBER;
```

Parameters

Parameter	Description
elapsed_time	The elapsed time in seconds that it takes to execute a function.

Usage Notes

When associating CPU cost with a user-defined function, use the full number of CPU units rather than the number of *thousands* of CPU units returned by ESTIMATE_CPU_UNITS; multiply the number returned by ESTIMATE_CPU_UNITS by 1,000.

64

DBMS_OFFLINE_OG

The `DBMS_OFFLINE_OG` package contains the public interface for offline instantiation of master groups.

This chapter contains the following topics:

Documentation of DBMS_OFFLINE_OG

For a complete description of this package within the context of Replication, see DBMS_OFFLINE_OG in the *Oracle Database Advanced Replication Management API Reference*.

65

DBMS_OLAP

Note: With Oracle Database 10*g*, the DBMS_OLAP package has been replaced with improved technology. While Oracle recommends you not begin development using DBMS_OLAP, Oracle continues to support DBMS_OLAP, and your existing applications using DBMS_OLAP will continue to work.

- If you are developing new or substantially modified applications and had previously used the Summary Advisor in DBMS_OLAP, you should now use the SQL Access Advisor described in Chapter 12, "DBMS_ADVISOR".
- If you had previously used DBMS_OLAP.VALIDATE_DIMENSION, you should now use DBMS_DIMENSION.VALIDATE_DIMENSION described in Chapter 35, "DBMS_DIMENSION".
- If you had previously used DBMS_OLAP.ESTIMATE_MVIEW_SIZE, you should now use DBMS_MVIEW.ESTIMATE_MVIEW_SIZE described in Chapter 61, "DBMS_MVIEW".

The DBMS_OLAP package, presented here for reasons of backward compatibility, provides a collection of materialized view analysis and advisory functions that are callable from any PL/SQL program. Some of the functions generate output tables.

> **See Also:** *Oracle Database Data Warehousing Guide* for more information.

This chapter contains the following topics:

Using DBMS_OLAP

This section contains topics which relate to using the `DBMS_OLAP` package.

Overview

DBMS_OLAP performs seven major functions, which include materialized view strategy recommendation, materialized view strategy evaluation, reporting and script generation, repository management, workload management, filter management, and dimension validation.

To perform materialized view strategy recommendation and evaluation functions, the workload information can either be provided by the user or synthesized by the Advisor engine. In the former case, cardinality information of all tables and materialized views referenced in the workload are required. In the latter case, dimension objects must be present and cardinality information for all dimension tables, fact tables, and materialized views are required. Cardinality information should be gathered with the DBMS_STATS.GATHER_TABLE_STATS procedure. Once these functions are completed, the analysis results can be presented with the reporting and script generation function.

The workload management function handles three types of workload, which are user-specified workload, SQL cache workload, and Oracle Trace workload. To process the user-specified workload, a user-defined workload table must be present in the user's schema. To process Oracle Trace workload, the Oracle Trace formatter must be run to preprocess collected workload statistics into default V-tables in the user's schema.

Views

Several views are created when using DBMS_OLAP. All are in the SYSTEM schema. To access these views, you must have a DBA role.

SYSTEM.MVIEW_EVALUATIONS

Table 65–1 SYSTEM.MVIEW_EVALUATIONS

Column	NULL?	Datatype	Description
RUNID	NOT NULL	NUMBER	Run ID identifying a unique Advisor call.
MVIEW_OWNER	-	VARCHAR2(30)	Owner of materialized view.
MVIEW_NAME	-	VARCHAR2(30)	Name of an exiting materialized view in this database.
RANK	NOT NULL	NUMBER	Rank of this materialized view in descending order of BENEFIT_TO_CSOT_RATIO.
STORAGE_IN_BYTES	-	NUMBER	Size of the materialized view in bytes.
FREQUENCY	-	NUMBER	Number of times this materialized view appears in the workload.
CUMULATIVE_BENEFIT	-	NUMBER	The cumulative benefit of the materialized view.
BENEFIT_TO_COST_RATIO	NOT NULL	NUMBER	The ratio of CUMULATIVE_BENEFIT to STORAGE_IN_BYTES.

SYSTEM.MVIEW_EXCEPTIONS

Table 65–2 SYSTEM.MVIEW_EXCEPTIONS

Column	NULL?	Datatype	Description
RUNID	-	NUMBER	Run ID identifying a unique Advisor call.
OWNER	-	VARCHAR2(30)	Owner name.
TABLE_NAME	-	VARCHAR2(30)	Table name.
DIMENSION_NAME	-	VARCHAR2(30)	Dimension name.
RELATIONSHIP	-	VARCHAR2(11)	Violated relation name.
BAD_ROWID	-	ROWID	Location of offending entry.

SYSTEM.MVIEW_FILTER

Table 65–3 SYSTEM.MVIEW_FILTER

Column	NULL?	Datatype	Description
FILTERID	NOT NULL	NUMBER	Unique number used to identify the operation that used this filter.
SUBFILTERNUM	NOT NULL	NUMBER	A unique ID number that groups all filter items together. A corresponding filter header record can be found in the MVIEW_LOG table.
SUBFILTERTYPE	-	VARCHAR2(12)	Filter item number.

Table 65–3 (Cont.) SYSTEM.MVIEW_FILTER

Column	NULL?	Datatype	Description
STR_VALUE	–	VARCHAR2(1028)	String attribute for items that require strings.
NUM_VALUE1	–	NUMBER	Numeric low for items that require numbers.
NUM_VALUE2	–	NUMBER	Numeric high for items that require numbers.
DATE_VALUE1	–	DATE	Date low for items that require dates.
DATE_VALUE2	–	DATE	Date high for items that require dates.

SYSTEM.MVIEW_FILTERINSTANCE

Table 65–4 SYSTEM.MVIEW_FILTERINSTANCE

Column	NULL?	Datatype	Description
RUNID	NOT NULL	NUMBER	Unique number used to identify the operation that used this filter.
FILTERID	–	NUMBER	A unique ID number that groups all filter items together. A corresponding filter header record can be found in the MVIEW_LOG table.
SUBFILTERNUM	–	NUMBER	Filter item number.
SUBFILTERTYPE	–	VARCHAR2(12)	Filter item type.
STR_VALUE	–	VARCHAR2(1028)	String attribute for items that require strings.
NUM_VALUE1	–	NUMBER	Numeric low for items that require numbers.
NUM_VALUE2	–	NUMBER	Numeric high for items that require numbers.
DATE_VALUE1	–	DATE	Date low for items that require dates.
DATE_VALUE2	–	DATE	Date high for items that require dates.

SYSTEM.MVIEW_LOG

Table 65–5 SYSTEM.MVIEW_LOG

Column	NULL?	Datatype	Description
ID	NOT NULL	NUMBER	Unique number used to identify the table entry. The number must be created using the CREATE_ID routine.
FILTERID	–	NUMBER	Optional filter ID. Zero indicates no user-supplied filter has been applied to the operation.
RUN_BEGIN	–	DATE	Date at which the operation began.
RUN_END	–	DATE	Date at which the operation ended.
TYPE	–	VARCHAR2(11)	A name that identifies the type of operation.
STATUS	–	VARCHAR2(11)	The current operational status.
MESSAGE	–	VARCHAR2(2000)	Informational message indicating current operation or condition.
COMPLETED	–	NUMBER	Number of steps completed by operation.
TOTAL	–	NUMBER	Total number steps to be performed.
ERROR_CODE	–	VARCHAR2(20)	Oracle error code in the event of an error.

SYSTEM.MVIEW_RECOMMENDATIONS

Table 65–6 SYSTEM.MVIEW_RECOMMENDATIONS

Column	NULL?	Datatype	Description
RUNID	-	NUMBER	Run ID identifying a unique Advisor call.
ALL_TABLES	-	VARCHAR2(2000)	A comma-delimited list of fully qualified table names for structured recommendations.
FACT_TABLES	-	VARCHAR2(1000)	A comma-delimited list of grouping levels, if any, for structured recommendation.
GROUPING_LEVELS	-	VARCHAR2(2000)	-
QUERY_TEXT	-	LONG	Query text of materialized view if RECOMMENDED_ACTION is CREATE; null otherwise.
RECOMMENDATION_NUMBER	NOT NULL	NUMBER	Unique identifier for this recommendation.
RECOMMENDED_ACTION	-	VARCHAR2(6)	CREATE, RETAIN, or DROP.
MVIEW_OWNER	-	VARCHAR2(30)	Owner of the materialized view if RECOMMENDED_ACTION is RETAIN or DROP; null otherwise.
MVIEW_NAME	-	VARCHAR2(30)	Name of the materialized view if RECOMMENDED_ACTION is RETAIN or DROP; null otherwise.
STORAGE_IN_BYTES	-	NUMBER	Actual or estimated storage in bytes.
PCT_PERFORMANCE_GAIN	-	NUMBER	The expected incremental improvement in performance obtained by accepting this recommendation relative to the initial condition, assuming that all previous recommendations have been accepted, or NULL if unknown.
BENEFIT_TO_COST_RATIO	NOT NULL	NUMBER	Ratio of the incremental improvement in performance to the size of the materialized view in bytes, or NULL if unknown.

SYSTEM.MVIEW_WORKLOAD

Table 65–7 SYSTEM.MVIEW_WORKLOAD

Column	NULL?	Datatype	Description
APPLICATION	-	VARCHAR2(30)	Optional application name for the query.
CARDINALITY	-	NUMBER	Total cardinality of all of tables in query.
WORKLOADID	-	NUMBER	Workload ID identifying a unique sampling.
FREQUENCY	-	NUMBER	Number of times query executed.
IMPORT_TIME	-	DATE	Date at which item was collected.
LASTUSE	-	DATE	Last date of execution.
OWNER	-	VARCHAR2(30)	User who last executed query.
PRIORITY	-	NUMBER	User-supplied ranking of query.
QUERY	-	LONG	Query text.

Table 65–7 (Cont.) SYSTEM.MVIEW_WORKLOAD

Column	NULL?	Datatype	Description
QUERYID	-	NUMBER	Id number identifying a unique query.
RESPONSETIME	-	NUMBER	Execution time in seconds.
RESULTSIZE	-	NUMBER	Total bytes selected by the query.

Deprecated Subprograms

The DBMS_OLAP subprograms have been replaced with improved technology: see Chapter 12, "DBMS_ADVISOR", Chapter 35, "DBMS_DIMENSION" and Chapter 61, "DBMS_MVIEW". All DBMS_OLAP subprograms are obsolete with Oracle Database 10g, and while Oracle will continue to support them, they are documented only for reasons of backward compatibility.

- ADD_FILTER_ITEM Procedure
- CREATE_ID Procedure
- ESTIMATE_MVIEW_SIZE Procedure
- EVALUATE_MVIEW_STRATEGY Procedure
- GENERATE_MVIEW_REPORT Procedure
- GENERATE_MVIEW_SCRIPT Procedure
- LOAD_WORKLOAD_CACHE Procedure
- LOAD_WORKLOAD_TRACE Procedure
- PURGE_FILTER Procedure
- PURGE_RESULTS Procedure
- PURGE_WORKLOAD Procedure
- RECOMMEND_MVIEW_STRATEGY Procedure
- SET_CANCELLED Procedure
- VALIDATE_DIMENSION Procedure
- VALIDATE_WORKLOAD_CACHE Procedure
- VALIDATE_WORKLOAD_TRACE Procedure
- VALIDATE_WORKLOAD_USER Procedure

Summary of DBMS_OLAP Subprograms

Note: The DBMS_OLAP subprograms have been replaced with improved technology:

- If you are developing new or substantially modified applications and had previously used the Summary Advisor in DBMS_OLAP, you should now use the SQL Access Advisor described in Chapter 12, "DBMS_ADVISOR".
- If you had previously used DBMS_OLAP.VALIDATE_DIMENSION, you should now use DBMS_DIMENSION.VALIDATE_DIMENSION described in Chapter 35, "DBMS_DIMENSION".
- If you had previously used DBMS_OLAP.ESTIMATE_MVIEW_SIZE, you should now use DBMS_MVIEW.ESTIMATE_MVIEW_SIZE described in Chapter 61, "DBMS_MVIEW"

Table 65–8 DBMS_OLAP Package Subprograms

Subprogram	Description
ADD_FILTER_ITEM Procedure on page 65-11	Filters the contents being used during the recommendation process [see Deprecated Subprograms on page 65-8]
CREATE_ID Procedure on page 65-13	Generates an internal ID used by a new workload collection, a new filter, or a new Advisor run [see Deprecated Subprograms on page 65-8]
ESTIMATE_MVIEW_SIZE Procedure on page 65-14	Estimates the size of a materialized view that you might create, in bytes and rows [see Deprecated Subprograms on page 65-8]
EVALUATE_MVIEW_STRATEGY Procedure on page 65-15	Measures the utilization of each existing materialized view [see Deprecated Subprograms on page 65-8]
GENERATE_MVIEW_REPORT Procedure on page 65-16	Generates an HTML-based report on the given Advisor run [see Deprecated Subprograms on page 65-8]
GENERATE_MVIEW_SCRIPT Procedure on page 65-17	Generates a simple script containing the SQL commands to implement Summary Advisor recommendations [see Deprecated Subprograms on page 65-8]
LOAD_WORKLOAD_CACHE Procedure on page 65-18	Obtains a SQL cache workload [see Deprecated Subprograms on page 65-8]
LOAD_WORKLOAD_TRACE Procedure on page 65-19	Loads a workload collected by Oracle Trace [see Deprecated Subprograms on page 65-8]
LOAD_WORKLOAD_USER Procedure on page 65-20	Loads a user-defined workload [see Deprecated Subprograms on page 65-8]
PURGE_FILTER Procedure on page 65-21	Deletes a specific filter or all filters [see Deprecated Subprograms on page 65-8]
PURGE_RESULTS Procedure on page 65-22	Removes all results or those for a specific run [see Deprecated Subprograms on page 65-8]
PURGE_WORKLOAD Procedure on page 65-23	Deletes all workloads or a specific collection [see Deprecated Subprograms on page 65-8]

Table 65–8 (Cont.) DBMS_OLAP Package Subprograms

Subprogram	Description
RECOMMEND_MVIEW_STRATEGY Procedure on page 65-24	Generates a set of recommendations about which materialized views should be created, retained, or dropped [see Deprecated Subprograms on page 65-8]
SET_CANCELLED Procedure on page 65-26	Stops the Advisor if it takes too long returning results [see Deprecated Subprograms on page 65-8]
VALIDATE_DIMENSION Procedure on page 65-27	Verifies that the relationships specified in a `dimension` are correct [see Deprecated Subprograms on page 65-8]
VALIDATE_WORKLOAD_CACHE Procedure on page 65-28	Validates the SQL Cache workload before performing load operations [see Deprecated Subprograms on page 65-8]
VALIDATE_WORKLOAD_TRACE Procedure on page 65-29	Validates the Oracle Trace workload before performing load operations [see Deprecated Subprograms on page 65-8]
VALIDATE_WORKLOAD_USER Procedure on page 65-30	Validates the user-supplied workload before performing load operations [see Deprecated Subprograms on page 65-8]

ADD_FILTER_ITEM Procedure

> **Note:** See Deprecated Subprograms on page 65-8.

This procedure adds a new filter item to an existing filter to make it more restrictive. It also creates a filter to restrict what is analyzed for the workload.

Syntax

```
ADD_FILTER_ITEM (
   filter_id    IN NUMBER,
   filter_name  IN VARCHAR2,
   string_list  IN VARCHAR2,
   number_min   IN NUMBER,
   number_max   IN NUMBER,
   date_min     IN VARCHAR2,
   date_max     IN VARCHAR2);
```

Parameters

Table 65–9 ADD_FILTER_ITEM Procedure Parameters

Parameter	Description
filter_id	An ID that uniquely describes the filter. It is generated by the DBMS_OLAP.CREATE_ID procedure
filter_name	■ APPLICATION: String-workloads application column. An example of how to load a SQL Cache workload follows. ■ BASETABLE: String-based tables referenced by workload queries. Name must be fully qualified including owner and table name (for example, SH.SALES). ■ CARDINALITY: Numerical-sum of cardinality of the referenced base tables. ■ FREQUENCY: Numerical-workloads frequency column. ■ LASTUSE: Date-workloads lastuse column. Not used by SQL Cache workload. ■ OWNER: String-workloads owner column. Expected in uppercase unless owner defined explicitly to be not all in uppercase. ■ PRIORITY: Numerical-workloads priority column. Not used by SQL Cache workload. ■ RESPONSETIME: Numerical-workloads response time column. Not used by SQL Cache workload. ■ SCHEMA: String-based schema referenced by workload filter. ■ TRACENAME: String-list of oracle trace collection names. Only used by a Trace Workload.
string_list	A comma-delimited list of strings. This parameter is only used by the filter items of the string type.
number_min	The lower bound of a numerical range. NULL represents the lowest possible value. This parameter is only used by the parameters of the numerical type.
number_max	The upper bound of a numerical range, NULL for no upper bound. NULL represents the highest possible value. This parameter is only used by the parameters of the numerical type.

Table 65–9 (Cont.) ADD_FILTER_ITEM Procedure Parameters

Parameter	Description
`date_min`	The lower bound of a date range. `NULL` represents the lowest possible date value. This parameter is only used by the parameters of the date type.
`date_max`	The upper bound of a date range. `NULL` represents the highest possible date value. This parameter is only used by the parameters of the date type.

CREATE_ID Procedure

> **Note:** See Deprecated Subprograms on page 65-8.

This procedure creates a unique identifier, which is used to identify a filter, a workload or results of an Advisor or dimension validation run.

Syntax

```
CALL DBMS_OLAP.CREATE_ID (
  id        OUT NUMBER);
```

Parameters

Table 65–10 CREATE_ID Procedure Parameters

Parameter	Description
id	The unique identifier that can be used to identify a filter, a workload, or an Advisor run

ESTIMATE_MVIEW_SIZE Procedure

Note: See Deprecated Subprograms on page 65-8.

This procedure estimates the size of a materialized view that you might create, in bytes and number of rows.

Syntax

```
DBMS_OLAP.ESTIMATE_MVIEW_SIZE (
   stmt_id       IN  VARCHAR2,
   select_clause IN  VARCHAR2,
   num_rows      OUT NUMBER,
   num_bytes     OUT NUMBER);
```

Parameters

Table 65–11 ESTIMATE_MVIEW_SIZE Procedure Parameters

Parameter	Description
stmt_id	Arbitrary string used to identify the statement in an EXPLAIN PLAN
select_clause	The SELECT statement to be analyzed
num_rows	Estimated cardinality
num_bytes	Estimated number of bytes

EVALUATE_MVIEW_STRATEGY Procedure

> **Note:** See Deprecated Subprograms on page 65-8.

This procedure measures the utilization of each existing materialized view based on the materialized view usage statistics collected from the workload. The `workload_id` is optional. If not provided, `EVALUATE_MVIEW_STRATEGY` uses a hypothetical workload.

Syntax

```
DBMS_OLAP.EVALUATE_MVIEW_STRATEGY (
run_id        IN NUMBER,
workload_id   IN NUMBER,
filter_id     IN NUMBER);
```

Parameters

Table 65–12 EVALUATE_MVIEW_STRATEGY Procedure Parameters

Parameter	Description
`run_id`	An ID generated by the `DBMS_OLAP.CREATE_ID` procedure to identify results of a run
`workload_id`	An optional workload ID that maps to a workload in the current repository. Use the parameter `DBMS_OLAP.WORKLOAD_ALL` to choose all workloads.
`filter_id`	Specify filter for the workload to be used. The value `DBMS_OLAP.FILTER_NONE` indicates no filtering.

Usage Notes

Periodically, the unused results can be purged from the system by calling the `DBMS_OLAP.PURGE_RESULTS` procedure.

GENERATE_MVIEW_REPORT Procedure

Note: See Deprecated Subprograms on page 65-8.

This procedure generates an HTML-based report on the given Advisor run.

Syntax

```
DBMS_OLAP.GENERATE_MVIEW_REPORT (
   filename     IN VARCHAR2,
   id           IN NUMBER,
   flags        IN NUMBER);
```

Parameters

Table 65–13 GENERATE_MVIEW_REPORT Procedure Parameters

Parameter	Description
filename	Fully qualified output file name to receive HTML data. Note that the Oracle server restricts file access within Oracle stored procedures. See the "Security and Performance" section of the Java Developer's Guide for more information on file permissions.
id	An ID that identifies an Advisor run. Or use the parameter DBMS_OLAP.RUNID_ALL to indicate all Advisor runs should be reported.
flags	Bit masked flags indicating what sections should be reported ■ DBMS_OLAP.RPT_ACTIVITY -- Overall activities ■ DBMS_OLAP.RPT_JOURNAL -- Runtime journals ■ DBMS_OLAP.RPT_WORKLOAD_FILTER -- Filters ■ DBMS_OLAP.RPT_WORKLOAD_DETAIL -- Workload information ■ DBMS_OLAP.RPT_WORKLOAD_QUERY -- Workload query information ■ DBMS_OLAP.RPT_RECOMMENDATION -- Recommendations ■ DBMS_OLAP.RPT_USAGE -- Materialized view usage ■ DBMS_OLAP.RPT_ALL -- All sections

GENERATE_MVIEW_SCRIPT Procedure

> **Note:** See Deprecated Subprograms on page 65-8.

This procedure generates a simple script containing the SQL commands to implement Summary Advisor recommendations.

Syntax

```
DBMS_OLAP.GENERATE_MVIEW_SCRIPT(
   filename      IN VARCHAR2,
   id            IN NUMBER,
   tspace        IN VARCHAR2);
```

Parameters

Table 65–14 GENERATE_MVIEW_SCRIPT Procedure Parameters

Parameter	Description
`filename`	Fully qualified output file name to receive HTML data. Note that the Oracle server restricts file access within Oracle stored procedures. See the "Security and Performance" section of the Java Developer's Guide for more information on file permissions.
`id`	An ID that identifies an Advisor run. The parameter `DBMS_OLAP.RUNID_ALL` indicates all Advisor runs should be reported.
`tspace`	Optional tablespace name to use when creating materialized views.

LOAD_WORKLOAD_CACHE Procedure

> **Note:** See Deprecated Subprograms on page 65-8.

This procedure loads a SQL cache workload.

Syntax

```
DBMS_OLAP.LOAD_WORKLOAD_CACHE (
   workload_id  IN NUMBER,
   flags        IN NUMBER,
   filter_id    IN NUMBER,
   application  IN VARCHAR2,
   priority     IN NUMBER);
```

Parameters

Table 65–15 LOAD_WORKLOAD_CACHE Procedure Parameters

Parameter	Description
workload_id	Fully qualified output file name to receive HTML data. Note that the Oracle server restricts file access within Oracle stored procedures. See the "Security and Performance" section of the Java Developer's Guide for more information on file permission.
flags	■ DBMS_OLAP.WORKLOAD_OVERWRITE: The load routine will explicitly remove any existing queries from the workload that are owned by the specified collection ID. ■ DBMS_OLAP.WORKLOAD_APPEND: The load routine preserves any existing queries in the workload. Any queries collected by the load operation will be appended to the end of the specified workload. ■ DBMS_OLAP.WORKLOAD_NEW: The load routine assumes there are no existing queries in the workload. If it finds an existing workload element, the call will fail with an error. Note: the flags have the same behavior irrespective of the LOAD_WORKLOAD operation.
filter_id	Specify filter for the workload to be loaded
application	The default business application name. This value will be used for a query if one is not found in the target workload.
priority	The default business priority to be assigned to every query in the target workload

LOAD_WORKLOAD_TRACE Procedure

> **Note:** See Deprecated Subprograms on page 65-8.

This procedure loads an Oracle Trace workload.

Syntax

```
DBMS_OLAP.LOAD_WORKLOAD_TRACE (
   workload_id  IN NUMBER,
   flags        IN NUMBER,
   filter_id    IN NUMBER,
   application  IN VARCHAR2,
   priority     IN NUMBER,
   owner_name   IN VARCHAR2);
```

Parameters

Table 65–16 LOAD_WORKLOAD_TRACE Procedure Parameters

Parameter	Description
collectionid	Fully qualified output file name to receive HTML data. Note that the Oracle server restricts file access within Oracle stored procedures. See the "Security and Performance" section of the Java Developer's Guide for more information on file permission.
flags	■ DBMS_OLAP.WORKLOAD_OVERWRITE: The load routine will explicitly remove any existing queries from the workload that are owned by the specified collection ID. ■ DBMS_OLAP.WORKLOAD_APPEND: The load routine preserves any existing queries in the workload. Any queries collected by the load operation will be appended to the end of the specified workload. ■ DBMS_OLAP.WORKLOAD_NEW: The load routine assumes there are no existing queries in the workload. If it finds an existing workload element, the call will fail with an error. Note: the flags have the same behavior irrespective of the LOAD_WORKLOAD operation.
filter_id	Specify filter for the workload to be loaded
application	The default business application name. This value will be used for a query if one is not found in the target workload.
priority	The default business priority to be assigned to every query in the target workload
owner_name	The schema that contains the Oracle Trace data. If omitted, the current user will be used

LOAD_WORKLOAD_USER Procedure

Note: See Deprecated Subprograms on page 65-8.

This procedure loads a user-defined workload.

Syntax

```
DBMS_OLAP.LOAD_WORKLOAD_USER (
   workload_id   IN    NUMBER,
   flags         IN    NUMBER,
   filter_id     IN    NUMBER,
   owner_name    IN    VARCHAR2,
   table_name    IN    VARCHAR2);
```

Parameters

Table 65–17 LOAD_WORKLOAD_USER Procedure Parameters

Parameter	Description
workload_id	The required id that was returned by the DBMS_OLAP.CREATE_ID call
flags	■ DBMS_OLAP.WORKLOAD_OVERWRITE: The load routine will explicitly remove any existing queries from the workload that are owned by the specified collection ID ■ DBMS_OLAP.WORKLOAD_APPEND: The load routine preserves any existing queries in the workload. Any queries collected by the load operation will be appended to the end of the specified workload ■ DBMS_OLAP.WORKLOAD_NEW: The load routine assumes there are no existing queries in the workload. If it finds an existing workload element, the call will fail with an error Note: the flags have the same behavior irrespective of the LOAD_WORKLOAD operation.
filter_id	Specify filter for the workload to be loaded
owner_name	The schema that contains the user supplied table or view
table_name	The table or view name containing valid workload data

PURGE_FILTER Procedure

> **Note:** See Deprecated Subprograms on page 65-8.

This procedure removes a filter at any time. You can delete a specific filter or all filters.

Syntax

```
DBMS_OLAP.PURGE_FILTER (
   filter_id    IN    NUMBER);
```

Parameters

Table 65–18 PURGE_FILTER Procedure Parameters

Parameter	Description
`filter_id`	The parameter `DBMS_OLAP.FILTER_ALL` indicates all filters should be removed.

PURGE_RESULTS Procedure

Note: See Deprecated Subprograms on page 65-8.

Many procedures in the DBMS_OLAP package generate output in system tables, such as recommendation results for RECOMMEND_MVIEW_STRATEGY and evaluation results for EVALUATE_MVIEW_STRATEGY, and dimension validation results for VALIDATE_DIMENSION. When these outputs are no longer required, they should be removed using the procedure PURGE_RESULTS. You can remove all results or those for a specific run.

Syntax

```
DBMS_OLAP.PURGE_RESULTS (
   run_id    IN   NUMBER);
```

Parameters

Table 65–19 PURGE_RESULTS Procedure Parameters

Parameter	Description
run_id	An ID generated with the DBMS_OLAP.CREATE_ID procedure. The ID should be associated with a RECOMMEND_MVIEW_STRATEGY or a EVALUATE_MVIEW_STRATEGY or a VALIDATE_DIMENSION run. Use the value DBMS_OLAP.RUNID_ALL to specify all such runs.

PURGE_WORKLOAD Procedure

> **Note:** See Deprecated Subprograms on page 65-8.

This procedure removes workloads when they are no longer needed. You can delete all workloads or a specific collection.

Syntax

```
DBMS_OLAP.PURGE_WORKLOAD (
   workload_id   IN   NUMBER);
```

Parameters

Table 65–20 PURGE_WORKLOAD Procedure Parameters

Parameter	Description
workload_id	An ID number originally assigned by the create_id call. If the value of workload_id is set to DBMS_OLAP.WORKLOAD_ALL, then all workloads for the current user will be deleted.

RECOMMEND_MVIEW_STRATEGY Procedure

Note: See Deprecated Subprograms on page 65-8.

This procedure generates a set of recommendations about which materialized views should be created, retained, or dropped, based on information in the workload (gathered by Oracle Trace, the user workload, or the SQL cache), and an analysis of table and column cardinality statistics gathered by the DBMS_STATS.GATHER_TABLE_STATS procedure.

RECOMMEND_MVIEW_STRATEGY requires that you have run the GATHER_TABLE_STATS procedure to gather table and column cardinality statistics and have collected and formatted the workload statistics.

The workload is aggregated to determine the count of each request in the workload, and this count is used as a weighting factor during the optimization process. If the workload_id is not provided, then RECOMMEND_MVIEW_STRATEGY uses a hypothetical workload based on dimension definitions and other embedded statistics.

The space of all dimensional materialized views that include the specified fact tables identifies the set of materialized views that optimize performance across the workload. The recommendation results are stored in system tables, which can be accessed through the view SYSTEM.MVIEW_RECOMMENDATIONS.

Syntax

```
DBMS_OLAP.RECOMMEND_MVIEW_STRATEGY (
   run_id             IN  NUMBER,
   workload_id        IN  NUMBER,
   filter_id          IN  NUMBER,
   storage_in_bytes   IN  NUMBER,
   retention_pct      IN  NUMBER,
   retention_list     IN  VARCHAR2,
   fact_table_filter  IN  VARCHAR2);
```

Parameters

Table 65–21 RECOMMEND_MVIEW_STRATEGY Procedure Parameters

Parameter	Description
run_id	An ID generated by the DBMS_OLAP.CREATE_ID procedure to uniquely identify results of a run
workload_id	An optional workload ID that maps to a workload in the current repository. Use the parameter DBMS_OLAP.WORKLOAD_ALL to choose all workloads. If the workload_id is set to NULL, the call will use a hypothetical workload.
filter_id	An optional filter ID that maps to a set of user-supplied filter items. Use the parameter DBMS_OLAP.FILTER_NONE to avoid filtering.
storage_in_bytes	Maximum storage, in bytes, that can be used for storing materialized views. This number must be nonnegative.

Table 65–21 (Cont.) RECOMMEND_MVIEW_STRATEGY Procedure Parameters

Parameter	Description
retention_pct	Number between 0 and 100 that specifies the percent of existing materialized view storage that must be retained, based on utilization on the actual or hypothetical workload. A materialized view is retained if the cumulative space, ranked by utilization, is within the retention threshold specified (or if it is explicitly listed in retention_list). Materialized views that have a NULL utilization (for example, nondimensional materialized views) are always retained.
retention_list	A comma-delimited list of materialized view table names. A drop recommendation is not made for any materialized view that appears in this list.
fact_table_filter	Optional list of fact tables used to filter real or ideal workload

Usage Notes

Periodically, the unused results can be purged from the system by calling the PURGE_RESULTS procedure.

SET_CANCELLED Procedure

Note: See Deprecated Subprograms on page 65-8.

If the Summary Advisor takes too long to make its recommendations using the procedures `RECOMMEND_MVIEW_STRATEGY`, you can stop it by calling the procedure `SET_CANCELLED` and passing in the `run_id` for this recommendation process.

Syntax

```
DBMS_OLAP.SET_CANCELLED (
   run_id     IN  NUMBER);
```

Parameters

Table 65–22 SET_CANCELLED Procedure Parameters

Parameter	Description
`run_id`	ID that uniquely identifies an Advisor analysis operation. This call can be used to cancel a long running workload collection as well as an Advisor analysis session

VALIDATE_DIMENSION Procedure

> **Note:** See Deprecated Subprograms on page 65-8.

This procedure verifies that the hierarchical and attribute relationships, and join relationships, specified in an existing dimension object are correct. This provides a fast way to ensure that referential integrity is maintained.

The validation results are stored in system tables, which can be accessed through the view SYSTEM.MVIEW_EXCEPTIONS.

Syntax

```
DBMS_OLAP.VALIDATE_DIMENSION (
   dimension_name    IN VARCHAR2,
   dimension_owner   IN VARCHAR2,
   incremental       IN BOOLEAN,
   check_nulls       IN BOOLEAN,
   run_id            IN NUMBER);
```

Parameters

Table 65–23 VALIDATE_DIMENSION Procedure Parameters

Parameter	Description
dimension_name	Name of the dimension to analyze
dimension_owner	Name of the dimension owner
incremental	If TRUE, then tests are performed only for the rows specified in the sumdelta$ table for tables of this dimension; otherwise, check all rows.
check_nulls	■ If TRUE, then all level columns are verified to be non-NULL; otherwise, this check is omitted. ■ Specify FALSE when non-NULLness is guaranteed by other means, such as NOT NULL constraints.
run_id	An ID generated by the DBMS_OLAP.CREATE_ID procedure to identify a run

Usage Notes

Periodically, the unused results can be purged from the system by calling the PURGE_RESULTS procedure.

VALIDATE_WORKLOAD_CACHE Procedure

Note: See Deprecated Subprograms on page 65-8.

This procedure validates the SQL Cache workload before performing load operations.

Syntax

```
DBMS_OLAP.VALIDATE_WORKLOAD_CACHE (
   valid            OUT NUMBER,
   error            OUT VARCHAR2);
```

Parameters

Table 65–24 VALIDATE_WORKLOAD_USER Procedure Parameters

Parameter	Description
valid	Return DBMS_OLAP.VALID or DBMS_OLAP.INVALID. Indicates whether a workload is valid
error	VARCHAR2, return error set

VALIDATE_WORKLOAD_TRACE Procedure

> **Note:** See Deprecated Subprograms on page 65-8.

This procedure validates the Oracle Trace workload before performing load operations.

Syntax

```
DBMS_OLAP.VALIDATE_WORKLOAD_TRACE (
   owner_name       IN  VARCHAR2,
   valid            OUT NUMBER,
   error            OUT VARCHAR2);
```

Parameters

Table 65–25 VALIDATE_WORKLOAD_TRACE Procedure Parameters

Parameter	Description
owner_name	Owner of the trace workload table
valid	Return DBMS_OLAP.VALID or DBMS_OLAP.INVALID. Indicates whether a workload is valid.
error	VARCHAR2, return error text

VALIDATE_WORKLOAD_USER Procedure

Note: See Deprecated Subprograms on page 65-8.

This procedure validates the user-supplied workload before performing load operations.

Syntax

```
DBMS_OLAP.VALIDATE_WORKLOAD_USER (
   owner_name       IN  VARCHAR2,
   table_name       IN  VARCHAR2,
   valid            OUT NUMBER,
   error            OUT VARCHAR2);
```

Parameters

Table 65–26 VALIDATE_WORKLOAD_USER Procedure Parameters

Parameter	Description
owner_name	Owner of the user workload table
table_name	User workload table name
valid	Return DBMS_OLAP.VALID or DBMS_OLAP.INVALID Indicate whether a workload is valid.
error	VARCHAR2, return error set

66

DBMS_OUTLN

The DBMS_OUTLN package, synonymous with OUTLN_PKG, contains the functional interface for subprograms associated with the management of stored outlines.

> **See Also:** For more information about using the DBMS_OUTLN package, see "Using Plan Stability" in *Oracle Database Performance Tuning Guide*.

This chapter contains the following topics:

Using DBMS_OUTLN

Overview

A stored outline is the stored data that pertains to an execution plan for a given SQL statement. It enables the optimizer to repeatedly re-create execution plans that are equivalent to the plan originally generated along with the outline.The data stored in an outline consists, in part, of a set of hints that are used to achieve plan stability.

Security Model

DBMS_OUTLN contains management procedures that should be available to appropriate users only. EXECUTE privilege is not extended to the general user community unless the DBA explicitly does so.

PL/SQL functions that are available for outline management purposes can be executed only by users with EXECUTE privilege on the procedure (or package).

Summary of DBMS_OUTLN Subprograms

Table 66–1 DBMS_OUTLN Package Subprograms

Subprogram	Description
CLEAR_USED Procedure on page 66-6	Clears the outline 'used' flag
CREATE_OUTLINE Procedure on page 66-7	Generates outlines from the shared cursor identified by hash value and child number
DROP_BY_CAT Procedure on page 66-8	Drops outlines that belong to a specified category
DROP_UNUSED Procedure on page 66-9	Drops outlines that have never been applied in the compilation of a SQL statement
EXACT_TEXT_SIGNATURES Procedure on page 66-10	Updates outline signatures to those that compute based on exact text matching
UPDATE_BY_CAT Procedure on page 66-11	Changes the category of outlines in one category to a new category
UPDATE_SIGNATURES Procedure on page 66-12	Updates outline signatures to the current version's signature

CLEAR_USED Procedure

This procedure clears the outline 'used' flag.

Syntax

```
DBMS_OUTLN.CLEAR_USED (
   name     IN     VARCHAR2);
```

Parameters

Table 66–2 CLEAR_USED Procedure Parameters

Parameter	Description
name	Name of the outline.

CREATE_OUTLINE Procedure

This procedure generates an outline from the shared cursor identified by hash value and child number.

Syntax

```
DBMS_OUTLN.CREATE_OUTLINE (
   hash_value    IN NUMBER,
   child_number  IN NUMBER,
   category      IN VARCHAR2 DEFAULT 'DEFAULT');
```

Parameters

Table 66–3 CREATE_OUTLINE Procedure Parameters

Parameter	Description
hash_value	Hash value identifying the target shared cursor.
child_number	Child number of the target shared cursor.
category	Category in which to create outline (optional).

DROP_BY_CAT Procedure

This procedure drops outlines that belong to a particular category. While outlines are put into the DEFAULT category unless otherwise specified, users have the option of grouping their outlines into groups called categories.

Syntax

```
DBMS_OUTLN.DROP_BY_CAT (
   cat VARCHAR2);
```

Parameters

Table 66–4 DROP_BY_CAT Procedure Parameters

Parameter	Description
cat	Category of outlines to drop.

Usage Notes

This procedure purges a category of outlines in a single call.

Examples

This example drops all outlines in the DEFAULT category:

```
DBMS_OUTLN.DROP_BY_CAT('DEFAULT');
```

DROP_UNUSED Procedure

This procedure drops outlines that have never been applied in the compilation of a SQL statement.

Syntax

```
DBMS_OUTLN.DROP_UNUSED;
```

Usage Notes

You can use `DROP_UNUSED` for outlines generated by an application for one-time use SQL statements created as a result of dynamic SQL. These outlines are never used and take up valuable disk space.

EXACT_TEXT_SIGNATURES Procedure

This procedure updates outline signatures to those that compute based on exact text matching.

Syntax

```
DBMS.OUTLN.EXACT_TEXT_SIGNATURES;
```

Usage Notes

This procedure is relevant only for downgrading an outline to 8.1.6 or earlier.

UPDATE_BY_CAT Procedure

This procedure changes the category of all outlines in one category to a new category.

Syntax

```
DBMS.OUTLN.UPDATE_BY_CAT (
   oldcat    VARCHAR2 default 'DEFAULT',
   newcat    VARCHAR2 default 'DEFAULT');
```

Parameters

Table 66–5 UPDATE_BY_CAT Procedure Parameters

Parameter	Description
`oldcat`	The current category of outlines.
`newcat`	The new category of outlines.

UPDATE_SIGNATURES Procedure

This procedure updates outline signatures to the current version's signature.

Syntax

```
DBMS.OUTLN.UPDATE_SIGNATURES;
```

Usage Notes

You should execute this procedure if you have imported outlines generated in an earlier release to ensure that the signatures are compatible with the current release's computation algorithm.

67

DBMS_OUTLN_EDIT

The `DBMS_OUTLN_EDIT` package is an invoker's rights package.

> **See Also:** For more information about using the `DBMS_OUTLN_EDIT` package, see "Using Plan Stability" in *Oracle Database Performance Tuning Guide*.

This chapter contains the following topic:

- Summary of DBMS_OUTLN_EDIT Subprograms

Summary of DBMS_OUTLN_EDIT Subprograms

Table 67–1 DBMS_OUTLN_EDIT Package Subprograms

Subprogram	Description
CHANGE_JOIN_POS Procedure on page 67-3	Changes the join position for the hint identified by outline name and hint number to the position specified by `newpos`
CREATE_EDIT_TABLES Procedure on page 67-4	Creates outline editing tables in calling a user's schema
DROP_EDIT_TABLES Procedure on page 67-5	Drops outline editing tables in calling the user's schema
GENERATE_SIGNATURE Procedure on page 67-6	Generates a signature for the specified SQL text
REFRESH_PRIVATE_OUTLINE Procedure on page 67-7	Refreshes the in-memory copy of the outline, synchronizing its data with the edits made to the outline hints

CHANGE_JOIN_POS Procedure

This function changes the join position for the hint identified by outline name and hint number to the position specified by `newpos`.

Syntax

```
DBMS_OUTLN_EDIT.CHANGE_JOIN_POS (
   name     VARCHAR2
   hintno   NUMBER
   newpos   NUMBER);
```

Parameters

Table 67–2 CHANGE_JOIN_POS Procedure Parameters

Parameter	Description
`name`	Name of the private outline to be modified.
`hintno`	Hint number to be modified.
`newpos`	New join position for the target hint.

CREATE_EDIT_TABLES Procedure

This procedure creates outline editing tables in calling a user's schema.

Syntax

```
DBMS_OUTLN_EDIT.CREATE_EDIT_TABLES;
```

Usage Notes

Beginning with Oracle Database 10g Release 1 (10.1) you will not need to use this statement because the outline editing tables will already exist as temporary tables in the `SYSTEM` schema.

DROP_EDIT_TABLES Procedure

This procedure drops outline editing tables in calling the user's schema.

Syntax

```
DBMS_OUTLN_EDIT.DROP_EDIT_TABLES;
```

GENERATE_SIGNATURE Procedure

This procedure generates a signature for the specified SQL text.

Syntax

```
DBMS_OUTLN.GENERATE_SIGNATURE (
   sqltxt      IN   VARCHAR2,
   signature   OUT  RAW);
```

Parameters

Table 67–3 GENERATE_SIGNATURE Procedure Parameters

Parameter	Description
sqltxt	The specified SQL.
signature	The signature to be generated.

REFRESH_PRIVATE_OUTLINE Procedure

This procedure refreshes the in-memory copy of the outline, synchronizing its data with the edits made to the outline hints.

Syntax

```
DBMS_OUTLN_EDIT.REFRESH_PRIVATE_OUTLINE (
   name  IN  VARCHAR2);
```

Parameters

Table 67–4 REFRESH_PRIVATE_OUTLINE Procedure Parameters

Parameter	Description
name	Name of the private outline to be refreshed.

68

DBMS_OUTPUT

The DBMS_OUTPUT package enables you to send messages from stored procedures, packages, and triggers. The package is especially useful for displaying PL/SQL debugging information.

This chapter contains the following topics:

Using DBMS_OUTPUT

This section contains topics which relate to using the `DBMS_OUTPUT` package.

Overview

The package is typically used for debugging, or for displaying messages and reports to SQL*DBA or SQL*Plus (such as are produced by applying the SQL command `DESCRIBE` to procedures).

The PUT Procedure and PUT_LINE Procedure in this package enable you to place information in a buffer that can be read by another trigger, procedure, or package. In a separate PL/SQL procedure or anonymous block, you can display the buffered information by calling the GET_LINE Procedure and GET_LINES Procedure.

If the package is disabled, all calls to subprograms are ignored. In this way, you can design your application so that subprograms are available only when a client is able to process the information.

Security Model

The `dbmsotpt.sql` script must be run as user `SYS`. This creates the public synonym `DBMS_OUTPUT`, and `EXECUTE` permission on this package is granted to `public`.

Operational Notes

- If you do not call `GET_LINE`, or if you do not display the messages on your screen in SQL*Plus, the buffered messages are ignored.
- SQL*Plus calls `GET_LINES` after issuing a SQL statement or anonymous PL/SQL calls.
- Typing `SET SERVEROUTPUT ON` in SQL*Plus has the effect of invoking

  ```
  DBMS_OUTPUT.ENABLE (buffer_size => NULL);
  ```

  with no limit on the output.
- You should generally avoid having application code invoke either the DISABLE Procedure or ENABLE Procedure because this could subvert the attempt of an external tool like SQL*Plus to control whether or not to display output.

Note: Messages sent using `DBMS_OUTPUT` are not actually sent until the sending subprogram or trigger completes. There is no mechanism to flush output during the execution of a procedure.

Exceptions

DBMS_OUTPUT subprograms raise the application error ORA-20000, and the output procedures can return the following errors:

Table 68–1 DBMS_OUTPUT Errors

Error	Description
ORU-10027:	Buffer overflow
ORU-10028:	Line length overflow

Rules and Limits

- The maximum line size is 32767 bytes.
- The default buffer size is 20000 bytes. The minimum size is 2000 bytes and the maximum is unlimited.

Examples

Example 1: Using a Trigger to Produce Output

You can use a trigger to print out some output from the debugging process. For example, you could code the trigger to invoke:

```
DBMS_OUTPUT.PUT_LINE('I got here:'||:new.col||' is the new value');
```

If you have enabled the DBMS_OUTPUT package, then the text produced by this PUT_LINE would be buffered, and you could, after executing the statement (presumably some INSERT, DELETE, or UPDATE that caused the trigger to fire), retrieve the line of information. For example:

```
BEGIN
   DBMS_OUTPUT.GET_LINE(:buffer, :status);
END;
```

You could then optionally display the buffer on the screen. You repeat calls to GET_LINE until status comes back as nonzero. For better performance, you should use calls to GET_LINES Procedure which can return an array of lines.

Example 2: Debugging Stored Procedures and Triggers

The DBMS_OUTPUT package is commonly used to debug stored procedures and triggers. This package can also be used to enable you to retrieve information about an object and format this output, as shown in "Example 3: Retrieving Information About an Object" on page 68-9.

This function queries the employee table and returns the total salary for a specified department. The function includes several calls to the PUT_LINE procedure:

```
CREATE FUNCTION dept_salary (dnum NUMBER) RETURN NUMBER IS
   CURSOR emp_cursor IS
      SELECT sal, comm FROM emp WHERE deptno = dnum;
   total_wages    NUMBER(11, 2) := 0;
   counter        NUMBER(10) := 1;
BEGIN

   FOR emp_record IN emp_cursor LOOP
      emp_record.comm := NVL(emp_record.comm, 0);
      total_wages := total_wages + emp_record.sal
         + emp_record.comm;
      DBMS_OUTPUT.PUT_LINE('Loop number = ' || counter ||
         '; Wages = '|| TO_CHAR(total_wages));    /* Debug line */
      counter := counter + 1; /* Increment debug counter */
   END LOOP;
   /* Debug line */
   DBMS_OUTPUT.PUT_LINE('Total wages = ' ||
     TO_CHAR(total_wages));
   RETURN total_wages;

END dept_salary;
```

Assume the EMP table contains the following rows:

```
EMPNO          SAL     COMM     DEPT
-----        ------- -------- -------
1002           1500      500       20
1203           1000                30
1289           1000                10
```

```
1347          1000      250      20
```

Assume the user executes the following statements in SQL*Plus:

```
SET SERVEROUTPUT ON
VARIABLE salary NUMBER;
EXECUTE :salary := dept_salary(20);
```

The user would then see the following information displayed in the output pane:

```
Loop number = 1; Wages = 2000
Loop number = 2; Wages = 3250
Total wages = 3250

PL/SQL procedure successfully executed.
```

Example 3: Retrieving Information About an Object

In this example, the user has used the EXPLAIN PLAN command to retrieve information about the execution plan for a statement and has stored it in PLAN_TABLE. The user has also assigned a statement ID to this statement. The example EXPLAIN_OUT procedure retrieves the information from this table and formats the output in a nested manner that more closely depicts the order of steps undergone in processing the SQL statement.

```
 /*****************************************************************/
/* Create EXPLAIN_OUT procedure. User must pass STATEMENT_ID to */
/* to procedure, to uniquely identify statement.                 */
/*****************************************************************/
CREATE OR REPLACE PROCEDURE explain_out
   (statement_id IN VARCHAR2) AS

   -- Retrieve information from PLAN_TABLE into cursor EXPLAIN_ROWS.

   CURSOR explain_rows IS
      SELECT level, id, position, operation, options,
         object_name
      FROM plan_table
      WHERE statement_id = explain_out.statement_id
      CONNECT BY PRIOR id = parent_id
         AND statement_id = explain_out.statement_id
      START WITH id = 0
       ORDER BY id;

BEGIN

   -- Loop through information retrieved from PLAN_TABLE:

   FOR line IN explain_rows LOOP

      -- At start of output, include heading with estimated cost.

      IF line.id = 0 THEN
         DBMS_OUTPUT.PUT_LINE ('Plan for statement '
            || statement_id
            || ', estimated cost = ' || line.position);
      END IF;

      -- Output formatted information. LEVEL determines indention level.

      DBMS_OUTPUT.PUT_LINE (lpad(' ',2*(line.level-1)) ||
```

```
            line.operation || ' ' || line.options || ' ' ||
            line.object_name);
      END LOOP;

END;
```

See Also: Chapter 167, "UTL_FILE"

Data Structures

The DBMS_OUTPUT package declares 2 collection types for use with the GET_LINES Procedure.

TABLE Types

CHARARR Table Type

OBJECT Types

DBMSOUTPUT_LINESARRAY Object Type

CHARARR Table Type

This package type is to be used with the GET_LINES Procedure to obtain text submitted through the PUT Procedure and PUT_LINE Procedure.

Syntax

```
TYPE CHARARR IS TABLE OF VARCHAR2(32767) INDEX BY BINARY_INTEGER;
```

DBMSOUTPUT_LINESARRAY Object Type

This package type is to be used with the GET_LINES Procedure to obtain text submitted through the PUT Procedure and PUT_LINE Procedure.

Syntax

```
TYPE DBMSOUTPUT_LINESARRAY IS
     VARRAY(2147483647) OF VARCHAR2(32767);
```

Summary of DBMS_OUTPUT Subprograms

Table 68–2 DBMS_OUTPUT Package Subprograms

Subprogram	Description
DISABLE Procedure on page 68-15	Disables message output
ENABLE Procedure on page 68-16	Enables message output
GET_LINE Procedure on page 68-17	Retrieves one line from buffer
GET_LINES Procedure on page 68-18	Retrieves an array of lines from buffer
NEW_LINE Procedure on page 68-19	Terminates a line created with `PUT`
PUT Procedure on page 68-20	Places a line in the buffer
PUT_LINE Procedure on page 68-21	Places partial line in buffer

Note: The PUT Procedure that take a number are obsolete and, while currently supported, are included in this release for legacy reasons only.

DISABLE Procedure

This procedure disables calls to `PUT`, `PUT_LINE`, `NEW_LINE`, `GET_LINE`, and `GET_LINES`, and purges the buffer of any remaining information.

As with the ENABLE Procedure, you do not need to call this procedure if you are using the `SERVEROUTPUT` option of SQL*Plus.

Syntax

```
DBMS_OUTPUT.DISABLE;
```

Pragmas

```
pragma restrict_references(disable,WNDS,RNDS);
```

ENABLE Procedure

This procedure enables calls to PUT, PUT_LINE, NEW_LINE, GET_LINE, and GET_LINES. Calls to these procedures are ignored if the DBMS_OUTPUT package is not activated.

Syntax

```
DBMS_OUTPUT.ENABLE (
   buffer_size IN INTEGER DEFAULT 20000);
```

Pragmas

```
pragma restrict_references(enable,WNDS,RNDS);
```

Parameters

Table 68–3 ENABLE Procedure Parameters

Parameter	Description
buffer_size	Upper limit, in bytes, the amount of buffered information. Setting buffer_size to NULL specifies that there should be no limit.

Usage Notes

- It is not necessary to call this procedure when you use the SET SERVEROUTPUT option of SQL*Plus.
- If there are multiple calls to ENABLE, then buffer_size is the last of the values specified. The maximum size is 1,000,000, and the minimum is 2,000 when the user specifies buffer_size (NOT NULL).
- NULL is expected to be the usual choice. The default is 20,000 for backwards compatibility with earlier database versions that did not support unlimited buffering.

GET_LINE Procedure

This procedure retrieves a single line of buffered information.

Syntax

```
DBMS_OUTPUT.GET_LINE (
   line    OUT VARCHAR2,
   status  OUT INTEGER);
```

Parameters

Table 68–4 GET_LINE Procedure Parameters

Parameter	Description
`line`	Returns a single line of buffered information, excluding a final newline character. You should declare the actual for this parameter as `VARCHAR2 (32767)` to avoid the risk of "`ORA-06502`: PL/SQL: numeric or value error: character string buffer too small".
`status`	If the call completes successfully, then the status returns as 0. If there are no more lines in the buffer, then the status is 1.

Usage Notes

- You can choose to retrieve from the buffer a single line or an array of lines. Call the `GET_LINE` procedure to retrieve a single line of buffered information. To reduce the number of calls to the server, call the `GET_LINES` procedure to retrieve an array of lines from the buffer.
- You can choose to automatically display this information if you are using SQL*Plus by using the special `SET SERVEROUTPUT ON` command.
- After calling `GET_LINE` or `GET_LINES`, any lines not retrieved before the next call to `PUT`, `PUT_LINE`, or `NEW_LINE` are discarded to avoid confusing them with the next message.

GET_LINES Procedure

This procedure retrieves an array of lines from the buffer.

Syntax

```
DBMS_OUTPUT.GET_LINES (
   lines       OUT     CHARARR,
   numlines    IN OUT  INTEGER);

DBMS_OUTPUT.GET_LINES (
   lines       OUT     DBMSOUTPUT_LINESARRAY,
   numlines    IN OUT INTEGER);
```

Parameters

Table 68–5 GET_LINES Procedure Parameters

Parameter	Description
`lines`	Returns an array of lines of buffered information. The maximum length of each line in the array is 32767 bytes. It is recommended that you use the `VARRAY` overload version in a 3GL host program to execute the procedure from a PL/SQL anonymous block.
`numlines`	Number of lines you want to retrieve from the buffer. After retrieving the specified number of lines, the procedure returns the number of lines actually retrieved. If this number is less than the number of lines requested, then there are no more lines in the buffer.

Usage Notes

- You can choose to retrieve from the buffer a single line or an array of lines. Call the `GET_LINE` procedure to retrieve a single line of buffered information. To reduce the number of calls to the server, call the `GET_LINES` procedure to retrieve an array of lines from the buffer.
- You can choose to automatically display this information if you are using SQL*Plus by using the special `SET SERVEROUTPUT ON` command.
- After calling `GET_LINE` or `GET_LINES`, any lines not retrieved before the next call to `PUT`, `PUT_LINE`, or `NEW_LINE` are discarded to avoid confusing them with the next message.

NEW_LINE Procedure

This procedure puts an end-of-line marker. The GET_LINE Procedure and the GET_LINES Procedure return "lines" as delimited by "newlines". Every call to the PUT_LINE Procedure or NEW_LINE Procedure generates a line that is returned by `GET_LINE(S)`.

Syntax

```
DBMS_OUTPUT.NEW_LINE;
```

PUT Procedure

This procedure places a partial line in the buffer.

> **Note:** The PUT procedure that takes a NUMBER is obsolete and, while currently supported, is included in this release for legacy reasons only.

Syntax

```
DBMS_OUTPUT.PUT (
    item IN VARCHAR2);
```

Parameters

Table 68–6 PUT and PUT_LINE Procedure Parameters

Parameter	Description
item	Item to buffer.

Exceptions

Table 68–7 PUT and PUT_LINE Procedure Exceptions

Error	Description
ORA-20000, ORU-10027:	Buffer overflow, limit of <buf_limit> bytes.
ORA-20000, ORU-10028:	Line length overflow, limit of 32767 bytes for each line.

Usage Notes

- You can build a line of information piece by piece by making multiple calls to PUT, or place an entire line of information into the buffer by calling PUT_LINE.
- When you call PUT_LINE the item you specify is automatically followed by an end-of-line marker. If you make calls to PUT to build a line, then you must add your own end-of-line marker by calling NEW_LINE. GET_LINE and GET_LINES do not return lines that have not been terminated with a newline character.
- If your lines exceed the line limit, you receive an error message.
- Output that you create using PUT or PUT_LINE is buffered. The output cannot be retrieved until the PL/SQL program unit from which it was buffered returns to its caller.

  For example, SQL*Plus does not display DBMS_OUTPUT messages until the PL/SQL program completes. There is no mechanism for flushing the DBMS_OUTPUT buffers within the PL/SQL program.

```
SQL> SET SERVEROUTPUT ON
SQL> BEGIN
  2  DBMS_OUTPUT.PUT_LINE ('hello');
  3  DBMS_LOCK.SLEEP (10);
  4  END;
```

PUT_LINE Procedure

This procedure places a line in the buffer.

> **Note:** The PUT_LINE procedure that takes a NUMBER is obsolete and, while currently supported, is included in this release for legacy reasons only.

Syntax

```
DBMS_OUTPUT.PUT_LINE (
   item IN VARCHAR2);
```

Parameters

Table 68–8 PUT and PUT_LINE Procedure Parameters

Parameter	Description
item	Item to buffer.

Exceptions

Table 68–9 PUT and PUT_LINE Procedure Exceptions

Error	Description
ORA-20000, ORU-10027:	Buffer overflow, limit of <buf_limit> bytes.
ORA-20000, ORU-10028:	Line length overflow, limit of 32767 bytes for each line.

Usage Notes

- You can build a line of information piece by piece by making multiple calls to PUT, or place an entire line of information into the buffer by calling PUT_LINE.
- When you call PUT_LINE the item you specify is automatically followed by an end-of-line marker. If you make calls to PUT to build a line, then you must add your own end-of-line marker by calling NEW_LINE. GET_LINE and GET_LINES do not return lines that have not been terminated with a newline character.
- If your lines exceeds the line limit, you receive an error message.
- Output that you create using PUT or PUT_LINE is buffered. The output cannot be retrieved until the PL/SQL program unit from which it was buffered returns to its caller.

  For example, SQL*Plus does not display DBMS_OUTPUT messages until the PL/SQL program completes. There is no mechanism for flushing the DBMS_OUTPUT buffers within the PL/SQL program. For example:

```
SQL> SET SERVEROUTPUT ON
SQL> BEGIN
   2 DBMS_OUTPUT.PUT_LINE ('hello');
   3 DBMS_LOCK.SLEEP (10);
   4 END;
```

69

DBMS_PCLXUTIL

The DBMS_PCLXUTIL package provides intra-partition parallelism for creating partition-wise local indexes. DBMS_PCLXUTIL circumvents the limitation that, for local index creation, the degree of parallelism is restricted to the number of partitions as only one slave process for each partition is used.

> **See Also:** There are several rules concerning partitions and indexes. For more information, see *Oracle Database Concepts* and *Oracle Database Administrator's Guide*.

This chapter contains the following topics:

Using DBMS_PCLXUTIL

Overview

DBMS_PCLXUTIL uses the DBMS_JOB package to provide a greater degree of parallelism for creating a local index for a partitioned table. This is achieved by asynchronous inter-partition parallelism using the background processes (with DBMS_JOB), in combination with intra-partition parallelism using the parallel query slave processes.

DBMS_PCLXUTIL works with both range and range-hash composite partitioning.

The DBMS_PCLXUTIL package can be used during the following DBA tasks:

1. Local index creation

   The procedure BUILD_PART_INDEX assumes that the dictionary information for the local index already exists. This can be done by issuing the create index SQL command with the UNUSABLE option.

   ```
   CREATE INDEX <idx_name> on <tab_name>(...) local(...) unusable;
   ```

This causes the dictionary entries to be created without "building" the index itself, the time consuming part of creating an index. Now, invoking the procedure BUILD_PART_INDEX causes a concurrent build of local indexes with the specified degree of parallelism.

```
EXECUTE dbms_pclxutil.build_part_index(4,4,<tab_name>,<idx_name>,FALSE);
```

For composite partitions, the procedure automatically builds local indexes for all subpartitions of the composite table.

1. Local index maintenance

   By marking desired partitions usable or unusable, the BUILD_PART_INDEX procedure also enables selective rebuilding of local indexes. The force_opt parameter provides a way to override this and build local indexes for all partitions.

   ```
   ALTER INDEX <idx_name> local(...) unusable;
   ```

Rebuild only the desired (sub)partitions (that are marked unusable):

```
EXECUTE dbms_pclxutil.build_part_index(4,4,<tab_name>,<idx_name>,FALSE);
```

Rebuild all (sub)partitions using force_opt = TRUE:

```
EXECUTE dbms_pclxutil.build_part_index(4,4,<tab_name>,<idx_name>,TRUE);
```

A progress report is produced, and the output appears on screen when the program is ended (because the DBMS_OUTPUT package writes messages to a buffer first, and flushes the buffer to the screen only upon termination of the program).

Operational Notes

DBMS_PCLXUTIL submits a job for each partition. It is the responsibility of the user/dba to control the number of concurrent jobs by setting the INIT.ORA parameter JOB_QUEUE_PROCESSES correctly. There is minimal error checking for correct syntax. Any errors are reported in the job queue process trace files.

Rules and Limits

> **Note:** For range partitioning, the minimum compatibility mode is 8.0; for range-hash composite partitioning, the minimum compatibility mode is 8*i*.

Because DBMS_PCLXUTIL uses the DBMS_JOB package, you must be aware of the following limitations pertaining to DBMS_JOB:

- You must decide appropriate values for the job_queue_processes initialization parameter. Clearly, if the job processes are not started before calling BUILD_PART_INDEX(), then the package will not function properly. The background processes are specified by the following init.ora parameters:

```
job_queue_processes=n   #the number of background processes = n
```

- Failure conditions are reported only in the trace files (a DBMS_JOB limitation), making it impossible to give interactive feedback to the user. This package prints a failure message, removes unfinished jobs from the queue, and requests the user to take a look at the j*.trc trace files.

Summary of DBMS_PCLXUTIL Subprograms

Table 69–1 DBMS_PCLXUTIL Package Subprograms

Subprogram	Description
BUILD_PART_INDEX Procedure on page 69-7	Provides intra-partition parallelism for creating partition-wise local indexes

BUILD_PART_INDEX Procedure

This procedure provides intra-partition parallelism for creating partition-wise local indexes.

Syntax

```
DBMS_PCLXUTIL.BUILD_PART_INDEX (
   jobs_per_batch  IN NUMBER   DEFAULT 1,
   procs_per_job   IN NUMBER   DEFAULT 1,
   tab_name        IN VARCHAR2 DEFAULT NULL,
   idx_name        IN VARCHAR2 DEFAULT NULL,
   force_opt       IN BOOLEAN  DEFAULT FALSE);
```

Parameters

Table 69–2 BUILD_PART_INDEX Procedure Parameters

Parameter	Description
jobs_per_batch	The number of concurrent partition-wise "local index builds".
procs_per_job	The number of parallel query slaves to be utilized for each local index build (1 <= procs_per_job <= max_slaves).
tab_name	The name of the partitioned table (an exception is raised if the table does not exist or not partitioned).
idx_name	The name given to the local index (an exception is raised if a local index is not created on the table tab_name).
force_opt	If TRUE, then force rebuild of all partitioned indexes; otherwise, rebuild only the partitions marked 'UNUSABLE'.

Examples

Suppose a table PROJECT is created with two partitions PROJ001 and PROJ002, along with a local index IDX.

A call to the procedure BUILD_PART_INDEX(2,4,'PROJECT','IDX',TRUE) produces the following output:

```
SQLPLUS> EXECUTE dbms_pclxutil.build_part_index(2,4,'PROJECT','IDX',TRUE);
Statement processed.
INFO: Job #21 created for partition PROJ002 with 4 slaves
INFO: Job #22 created for partition PROJ001 with 4 slaves
```

70

DBMS_PIPE

The DBMS_PIPE package lets two or more sessions in the same instance communicate. Oracle pipes are similar in concept to the pipes used in UNIX, but Oracle pipes are not implemented using the operating system pipe mechanisms.

This chapter contains the following topics:

Using DBMS_PIPE

Overview

Pipe functionality has several potential applications:

- External service interface: You can communicate with user-written services that are external to the RDBMS. This can be done effectively in a shared server process, so that several instances of the service are executing simultaneously. Additionally, the services are available asynchronously. The requestor of the service does not need to block a waiting reply. The requestor can check (with or without time out) at a later time. The service can be written in any of the 3GL languages that Oracle supports.
- Independent transactions: The pipe can communicate to a separate session which can perform an operation in an independent transaction (such as logging an attempted security violation detected by a trigger).
- Alerters (non-transactional): You can post another process without requiring the waiting process to poll. If an "after-row" or "after-statement" trigger were to alert an application, then the application would treat this alert as an indication that the data probably changed. The application would then read the data to get the current value. Because this is an "after" trigger, the application would want to do a "`SELECT FOR UPDATE`" to make sure it read the correct data.
- Debugging: Triggers and stored procedures can send debugging information to a pipe. Another session can keep reading out of the pipe and display it on the screen or write it to a file.
- Concentrator: This is useful for multiplexing large numbers of users over a fewer number of network connections, or improving performance by concentrating several user-transactions into one DBMS transaction.

Security Model

Security can be achieved by use of `GRANT EXECUTE` on the `DBMS_PIPE` package by creating a pipe using the `private` parameter in the `CREATE_PIPE` function and by writing cover packages that only expose particular features or pipenames to particular users or roles.

Depending upon your security requirements, you may choose to use either Public Pipes or Private Pipes.

Constants

```
maxwait   constant integer := 86400000; /* 1000 days */
```

This is the maximum time to wait attempting to send or receive a message.

Operational Notes

Information sent through Oracle pipes is buffered in the system global area (SGA). All information in pipes is lost when the instance is shut down.

> **Caution: Pipes are independent of transactions. Be careful using pipes when transaction control can be affected.**

The operation of DBMS_PIPE is considered with regard to the following topics:

- Public Pipes
- Writing and Reading Pipes
- Private Pipes

Public Pipes

You may create a public pipe either implicitly or explicitly. For *implicit* public pipes, the pipe is automatically created when it is referenced for the first time, and it disappears when it no longer contains data. Because the pipe descriptor is stored in the SGA, there is some space usage overhead until the empty pipe is aged out of the cache.

You create an *explicit* public pipe by calling the `CREATE_PIPE` function with the `private` flag set to `FALSE`. You must deallocate explicitly-created pipes by calling the `REMOVE_PIPE` function.

The domain of a public pipe is the schema in which it was created, either explicitly or implicitly.

Writing and Reading Pipes

Each public pipe works asynchronously. Any number of schema users can write to a public pipe, as long as they have `EXECUTE` permission on the `DBMS_PIPE` package, and they know the name of the public pipe. However, once buffered information is read by one user, it is emptied from the buffer, and is not available for other readers of the same pipe.

The sending session builds a message using one or more calls to the `PACK_MESSAGE` procedure. This procedure adds the message to the session's local message buffer. The information in this buffer is sent by calling the `SEND_MESSAGE` function, designating the pipe name to be used to send the message. When `SEND_MESSAGE` is called, all messages that have been stacked in the local buffer are sent.

A process that wants to receive a message calls the `RECEIVE_MESSAGE` function, designating the pipe name from which to receive the message. The process then calls the `UNPACK_MESSAGE` procedure to access each of the items in the message.

Private Pipes

You explicitly create a private pipe by calling the `CREATE_PIPE` function. Once created, the private pipe persists in shared memory until you explicitly deallocate it by calling the `REMOVE_PIPE` function. A private pipe is also deallocated when the database instance is shut down.

You cannot create a private pipe if an implicit pipe exists in memory and has the same name as the private pipe you are trying to create. In this case, CREATE_PIPE returns an error.

Access to a private pipe is restricted to:

- Sessions running under the same userid as the creator of the pipe
- Stored subprograms executing in the same userid privilege domain as the pipe creator
- Users connected as SYSDBA

An attempt by any other user to send or receive messages on the pipe, or to remove the pipe, results in an immediate error. Any attempt by another user to create a pipe with the same name also causes an error.

As with public pipes, you must first build your message using calls to PACK_MESSAGE before calling SEND_MESSAGE. Similarly, you must call RECEIVE_MESSAGE to retrieve the message before accessing the items in the message by calling UNPACK_MESSAGE.

Exceptions

DBMS_PIPE package subprograms can return the following errors:

Table 70–1 DBMS_PIPE Errors

Error	Description
ORA-23321:	Pipename may not be null. This can be returned by the CREATE_PIPE function, or any subprogram that takes a pipe name as a parameter.
ORA-23322:	Insufficient privilege to access pipe. This can be returned by any subprogram that references a private pipe in its parameter list.

Examples

Example 1: Debugging - PL/SQL

This example shows the procedure that a PL/SQL program can call to place debugging information in a pipe.

```
CREATE OR REPLACE PROCEDURE debug (msg VARCHAR2) AS
    status  NUMBER;
BEGIN
  DBMS_PIPE.PACK_MESSAGE(LENGTH(msg));
  DBMS_PIPE.PACK_MESSAGE(msg);
  status := DBMS_PIPE.SEND_MESSAGE('plsql_debug');
  IF status != 0 THEN
    raise_application_error(-20099, 'Debug error');
  END IF;
END debug;
```

Example 2: Debugging - Pro*C

The following Pro*C code receives messages from the PLSQL_DEBUG pipe in the previous example, and displays the messages. If the Pro*C session is run in a separate window, then it can be used to display any messages that are sent to the debug procedure from a PL/SQL program executing in a separate session.

```
#include <stdio.h>
#include <string.h>

EXEC SQL BEGIN DECLARE SECTION;
   VARCHAR username[20];
   int     status;
   int     msg_length;
   char    retval[2000];
EXEC SQL END DECLARE SECTION;

EXEC SQL INCLUDE SQLCA;

void sql_error();

main()
{

-- Prepare username:
   strcpy(username.arr, "SCOTT/TIGER");
   username.len = strlen(username.arr);

   EXEC SQL WHENEVER SQLERROR DO sql_error();
   EXEC SQL CONNECT :username;

   printf("connected\n");

-- Start an endless loop to look for and print messages on the pipe:
   FOR (;;)
```

```
   {
      EXEC SQL EXECUTE
         DECLARE
            len INTEGER;
            typ INTEGER;
            sta INTEGER;
            chr VARCHAR2(2000);
         BEGIN
            chr := '';
            sta := dbms_pipe.receive_message('plsql_debug');
            IF sta = 0 THEN
               DBMS_PIPE.UNPACK_MESSAGE(len);
               DBMS_PIPE.UNPACK_MESSAGE(chr);
            END IF;
            :status := sta;
            :retval := chr;
            IF len IS NOT NULL THEN
               :msg_length := len;
            ELSE
               :msg_length := 2000;
            END IF;
         END;
      END-EXEC;
      IF (status == 0)
         printf("\n%.*s\n", msg_length, retval);
      ELSE
         printf("abnormal status, value is %d\n", status);
   }
}

void sql_error()
{
   char msg[1024];
   int rlen, len;
   len = sizeof(msg);
   sqlglm(msg, &len, &rlen);
   printf("ORACLE ERROR\n");
   printf("%.*s\n", rlen, msg);
   exit(1);
}
```

Example 3: Execute System Commands

This example shows PL/SQL and Pro*C code let a PL/SQL stored procedure (or anonymous block) call PL/SQL procedures to send commands over a pipe to a Pro*C program that is listening for them.

The Pro*C program sleeps and waits for a message to arrive on the named pipe. When a message arrives, the Pro*C program processes it, carrying out the required action, such as executing a UNIX command through the *system*() call or executing a SQL command using embedded SQL.

`DAEMON.SQL` is the source code for the PL/SQL package. This package contains procedures that use the `DBMS_PIPE` package to send and receive message to and from the Pro*C daemon. Note that full handshaking is used. The daemon always sends a message back to the package (except in the case of the `STOP` command). This is valuable, because it allows the PL/SQL procedures to be sure that the Pro*C daemon is running.

You can call the DAEMON packaged procedures from an anonymous PL/SQL block using SQL*Plus or Enterprise Manager. For example:

```
SQLPLUS> variable rv number
SQLPLUS> execute :rv := DAEMON.EXECUTE_SYSTEM('ls -la');
```

On a UNIX system, this causes the Pro*C daemon to execute the command *system*("*ls -la*").

Remember that the daemon needs to be running first. You might want to run it in the background, or in another window beside the SQL*Plus or Enterprise Manager session from which you call it.

The DAEMON.SQL also uses the DBMS_OUTPUT package to display the results. For this example to work, you must have execute privileges on this package.

DAEMON.SQL Example. This is the code for the PL/SQL DAEMON package:

```
CREATE OR REPLACE PACKAGE daemon AS
  FUNCTION execute_sql(command VARCHAR2,
                       timeout NUMBER DEFAULT 10)
    RETURN NUMBER;

  FUNCTION execute_system(command VARCHAR2,
                          timeout NUMBER DEFAULT 10)
    RETURN NUMBER;

  PROCEDURE stop(timeout NUMBER DEFAULT 10);
END daemon;
/
CREATE OR REPLACE PACKAGE BODY daemon AS

  FUNCTION execute_system(command VARCHAR2,
                          timeout NUMBER DEFAULT 10)
  RETURN NUMBER IS

    status       NUMBER;
    result       VARCHAR2(20);
    command_code NUMBER;
    pipe_name    VARCHAR2(30);
  BEGIN
    pipe_name := DBMS_PIPE.UNIQUE_SESSION_NAME;

    DBMS_PIPE.PACK_MESSAGE('SYSTEM');
    DBMS_PIPE.PACK_MESSAGE(pipe_name);
    DBMS_PIPE.PACK_MESSAGE(command);
    status := DBMS_PIPE.SEND_MESSAGE('daemon', timeout);
    IF status <> 0 THEN
      RAISE_APPLICATION_ERROR(-20010,
        'Execute_system: Error while sending.  Status = ' ||
         status);
    END IF;

    status := DBMS_PIPE.RECEIVE_MESSAGE(pipe_name, timeout);
    IF status <> 0 THEN
      RAISE_APPLICATION_ERROR(-20011,
        'Execute_system: Error while receiving.
         Status = ' || status);
    END IF;

    DBMS_PIPE.UNPACK_MESSAGE(result);
    IF result <> 'done' THEN
```

```
    RAISE_APPLICATION_ERROR(-20012,
      'Execute_system: Done not received.');
  END IF;

  DBMS_PIPE.UNPACK_MESSAGE(command_code);
  DBMS_OUTPUT.PUT_LINE('System command executed.  result = ' ||
                       command_code);
  RETURN command_code;
END execute_system;

FUNCTION execute_sql(command VARCHAR2,
                     timeout NUMBER DEFAULT 10)
RETURN NUMBER IS

  status       NUMBER;
  result       VARCHAR2(20);
  command_code NUMBER;
  pipe_name    VARCHAR2(30);

BEGIN
  pipe_name := DBMS_PIPE.UNIQUE_SESSION_NAME;

  DBMS_PIPE.PACK_MESSAGE('SQL');
  DBMS_PIPE.PACK_MESSAGE(pipe_name);
  DBMS_PIPE.PACK_MESSAGE(command);
  status := DBMS_PIPE.SEND_MESSAGE('daemon', timeout);
  IF status <> 0 THEN
    RAISE_APPLICATION_ERROR(-20020,
      'Execute_sql: Error while sending.  Status = ' || status);
  END IF;

  status := DBMS_PIPE.RECEIVE_MESSAGE(pipe_name, timeout);

  IF status <> 0 THEN
    RAISE_APPLICATION_ERROR(-20021,
      'execute_sql: Error while receiving.
       Status = ' || status);
  END IF;

  DBMS_PIPE.UNPACK_MESSAGE(result);
  IF result <> 'done' THEN
    RAISE_APPLICATION_ERROR(-20022,
      'execute_sql: done not received.');
  END IF;

  DBMS_PIPE.UNPACK_MESSAGE(command_code);
  DBMS_OUTPUT.PUT_LINE
      ('SQL command executed.  sqlcode = ' || command_code);
  RETURN command_code;
END execute_sql;

PROCEDURE stop(timeout NUMBER DEFAULT 10) IS
  status NUMBER;
BEGIN
  DBMS_PIPE.PACK_MESSAGE('STOP');
  status := DBMS_PIPE.SEND_MESSAGE('daemon', timeout);
  IF status <> 0 THEN
    RAISE_APPLICATION_ERROR(-20030,
      'stop: error while sending.  status = ' || status);
  END IF;
```

```
  END stop;
END daemon;
```

daemon.pc Example. This is the code for the Pro*C daemon. You must precompile this using the Pro*C Precompiler, Version 1.5.x or later. You must also specify the USERID and SQLCHECK options, as the example contains embedded PL/SQL code.

> **Note:** To use a VARCHAR output host variable in a PL/SQL block, you must initialize the length component before entering the block.

```
proc iname=daemon userid=scott/tiger sqlcheck=semantics
```

Then C-compile and link in the normal way.

```
#include <stdio.h>
#include <string.h>

EXEC SQL INCLUDE SQLCA;

EXEC SQL BEGIN DECLARE SECTION;
  char *uid = "scott/tiger";
  int status;
  VARCHAR command[20];
  VARCHAR value[2000];
  VARCHAR return_name[30];
EXEC SQL END DECLARE SECTION;

void
connect_error()
{
  char msg_buffer[512];
  int msg_length;
  int buffer_size = 512;

  EXEC SQL WHENEVER SQLERROR CONTINUE;
  sqlglm(msg_buffer, &buffer_size, &msg_length);
  printf("Daemon error while connecting:\n");
  printf("%.*s\n", msg_length, msg_buffer);
  printf("Daemon quitting.\n");
  exit(1);
}

void
sql_error()
{
  char msg_buffer[512];
  int msg_length;
  int buffer_size = 512;

  EXEC SQL WHENEVER SQLERROR CONTINUE;
  sqlglm(msg_buffer, &buffer_size, &msg_length);
  printf("Daemon error while executing:\n");
  printf("%.*s\n", msg_length, msg_buffer);
  printf("Daemon continuing.\n");
}
main()
{
command.len = 20; /*initialize length components*/
```

```
value.len = 2000;
return_name.len  = 30;
  EXEC SQL WHENEVER SQLERROR DO connect_error();
  EXEC SQL CONNECT :uid;
  printf("Daemon connected.\n");

  EXEC SQL WHENEVER SQLERROR DO sql_error();
  printf("Daemon waiting...\n");
  while (1) {
    EXEC SQL EXECUTE
      BEGIN
        :status := DBMS_PIPE.RECEIVE_MESSAGE('daemon');
        IF :status = 0 THEN
          DBMS_PIPE.UNPACK_MESSAGE(:command);
        END IF;
      END;
    END-EXEC;
    IF (status == 0)
    {
      command.arr[command.len] = '\0';
      IF (!strcmp((char *) command.arr, "STOP"))
      {
        printf("Daemon exiting.\n");
        break;
      }

      ELSE IF (!strcmp((char *) command.arr, "SYSTEM"))
      {
        EXEC SQL EXECUTE
          BEGIN
            DBMS_PIPE.UNPACK_MESSAGE(:return_name);
            DBMS_PIPE.UNPACK_MESSAGE(:value);
          END;
        END-EXEC;
        value.arr[value.len] = '\0';
        printf("Will execute system command '%s'\n", value.arr);

        status = system(value.arr);
        EXEC SQL EXECUTE
          BEGIN
            DBMS_PIPE.PACK_MESSAGE('done');
            DBMS_PIPE.PACK_MESSAGE(:status);
            :status := DBMS_PIPE.SEND_MESSAGE(:return_name);
          END;
        END-EXEC;

        IF (status)
        {
          printf
           ("Daemon error while responding to system command.");
          printf("  status: %d\n", status);
        }
      }
      ELSE IF (!strcmp((char *) command.arr, "SQL")) {
        EXEC SQL EXECUTE
          BEGIN
            DBMS_PIPE.UNPACK_MESSAGE(:return_name);
            DBMS_PIPE.UNPACK_MESSAGE(:value);
          END;
        END-EXEC;
```

```
      value.arr[value.len] = '\0';
      printf("Will execute sql command '%s'\n", value.arr);

      EXEC SQL WHENEVER SQLERROR CONTINUE;
      EXEC SQL EXECUTE IMMEDIATE :value;
      status = sqlca.sqlcode;

      EXEC SQL WHENEVER SQLERROR DO sql_error();
      EXEC SQL EXECUTE
        BEGIN
          DBMS_PIPE.PACK_MESSAGE('done');
          DBMS_PIPE.PACK_MESSAGE(:status);
          :status := DBMS_PIPE.SEND_MESSAGE(:return_name);
        END;
      END-EXEC;

      IF (status)
      {
        printf("Daemon error while responding to sql command.");
        printf("  status: %d\n", status);
      }
    }
    ELSE
    {
      printf
        ("Daemon error: invalid command '%s' received.\n",
          command.arr);
    }
  }
  ELSE
  {
    printf("Daemon error while waiting for signal.");
    printf("  status = %d\n", status);
  }
}
EXEC SQL COMMIT WORK RELEASE;
exit(0);
```

Example 4: External Service Interface

Put the user-written 3GL code into an OCI or Precompiler program. The program connects to the database and executes PL/SQL code to read its request from the pipe, computes the result, and then executes PL/SQL code to send the result on a pipe back to the requestor.

Below is an example of a stock service request. The recommended sequence for the arguments to pass on the pipe for all service requests is:

```
protocol_version      VARCHAR2        - '1', 10 bytes or less
returnpipe            VARCHAR2        - 30 bytes or less
service               VARCHAR2        - 30 bytes or less
arg1                  VARCHAR2/NUMBER/DATE
   ...
argn                  VARCHAR2/NUMBER/DATE
```

The recommended format for returning the result is:

```
success               VARCHAR2        - 'SUCCESS' if OK,
                                        otherwise error message
arg1                  VARCHAR2/NUMBER/DATE
   ...
```

```
  argn                  VARCHAR2/NUMBER/DATE
```

The "stock price request server" would do, using OCI or PRO* (in pseudo-code):

```
<loop forever>
  BEGIN dbms_stock_server.get_request(:stocksymbol); END;
  <figure out price based on stocksymbol (probably from some radio
        signal), set error if can't find such a stock>
  BEGIN dbms_stock_server.return_price(:error, :price); END;
```

A client would do:

```
BEGIN :price := stock_request('YOURCOMPANY'); end;
```

The stored procedure, `dbms_stock_server`, which is called by the preceding "stock price request server" is:

```
CREATE OR REPLACE PACKAGE dbms_stock_server IS
  PROCEDURE get_request(symbol OUT VARCHAR2);
  PROCEDURE return_price(errormsg IN VARCHAR2, price IN VARCHAR2);
END;

CREATE OR REPLACE PACKAGE BODY dbms_stock_server IS
  returnpipe    VARCHAR2(30);

  PROCEDURE returnerror(reason VARCHAR2) IS
    s INTEGER;
  BEGIN
    dbms_pipe.pack_message(reason);
    s := dbms_pipe.send_message(returnpipe);
    IF s <> 0 THEN
      raise_application_error(-20000, 'Error:' || to_char(s) ||
        ' sending on pipe');
    END IF;
  END;

  PROCEDURE get_request(symbol OUT VARCHAR2) IS
    protocol_version VARCHAR2(10);
    s                INTEGER;
    service          VARCHAR2(30);
  BEGIN
    s := dbms_pipe.receive_message('stock_service');
    IF s <> 0 THEN
      raise_application_error(-20000, 'Error:' || to_char(s) ||
        'reading pipe');
    END IF;
    dbms_pipe.unpack_message(protocol_version);
    IF protocol_version <> '1' THEN
      raise_application_error(-20000, 'Bad protocol: ' ||
        protocol_version);
    END IF;
    dbms_pipe.unpack_message(returnpipe);
    dbms_pipe.unpack_message(service);
    IF service != 'getprice' THEN
      returnerror('Service ' || service || ' not supported');
    END IF;
    dbms_pipe.unpack_message(symbol);
  END;

  PROCEDURE return_price(errormsg in VARCHAR2, price in VARCHAR2) IS
    s INTEGER;
```

```
  BEGIN
    IF errormsg is NULL THEN
      dbms_pipe.pack_message('SUCCESS');
      dbms_pipe.pack_message(price);
    ELSE
      dbms_pipe.pack_message(errormsg);
    END IF;
    s := dbms_pipe.send_message(returnpipe);
    IF s <> 0 THEN
      raise_application_error(-20000, 'Error:'||to_char(s)||
        ' sending on pipe');
    END IF;
  END;
END;
```

The procedure called by the client is:

```
CREATE OR REPLACE FUNCTION stock_request (symbol VARCHAR2)
    RETURN VARCHAR2 IS
  s        INTEGER;
  price    VARCHAR2(20);
  errormsg VARCHAR2(512);
BEGIN
  dbms_pipe.pack_message('1');  -- protocol version
  dbms_pipe.pack_message(dbms_pipe.unique_session_name); -- return pipe
  dbms_pipe.pack_message('getprice');
  dbms_pipe.pack_message(symbol);
  s := dbms_pipe.send_message('stock_service');
  IF s <> 0 THEN
    raise_application_error(-20000, 'Error:'||to_char(s)||
      ' sending on pipe');
  END IF;
  s := dbms_pipe.receive_message(dbms_pipe.unique_session_name);
  IF s <> 0 THEN
    raise_application_error(-20000, 'Error:'||to_char(s)||
      ' receiving on pipe');
  END IF;
  dbms_pipe.unpack_message(errormsg);
  IF errormsg <> 'SUCCESS' THEN
    raise_application_error(-20000, errormsg);
  END IF;
  dbms_pipe.unpack_message(price);
  RETURN price;
END;
```

You would typically only `GRANT EXECUTE` on `DBMS_STOCK_SERVICE` to the stock service application server, and would only `GRANT EXECUTE` on `stock_request` to those users allowed to use the service.

See Also: Chapter 13, "DBMS_ALERT"

Summary of DBMS_PIPE Subprograms

Table 70–2 DBMS_PIPE Package Subprograms

Subprogram	Description
CREATE_PIPE Function on page 70-19	Creates a pipe (necessary for private pipes)
NEXT_ITEM_TYPE Function on page 70-21	Returns datatype of next item in buffer
PACK_MESSAGE Procedures on page 70-22	Builds message in local buffer
PURGE Procedure on page 70-24	Purges contents of named pipe
RECEIVE_MESSAGE Function on page 70-25	Copies message from named pipe into local buffer
REMOVE_PIPE Function on page 70-28	Removes the named pipe
RESET_BUFFER Procedure on page 70-27	Purges contents of local buffer
SEND_MESSAGE Function on page 70-29	Sends message on named pipe: This implicitly creates a public pipe if the named pipe does not exist
UNIQUE_SESSION_NAME Function on page 70-31	Returns unique session name
UNPACK_MESSAGE Procedures on page 70-32	Accesses next item in buffer

CREATE_PIPE Function

This function explicitly creates a public or private pipe. If the `private` flag is `TRUE`, then the pipe creator is assigned as the owner of the private pipe.

Explicitly-created pipes can only be removed by calling `REMOVE_PIPE`, or by shutting down the instance.

Syntax

```
DBMS_PIPE.CREATE_PIPE (
   pipename     IN VARCHAR2,
   maxpipesize  IN INTEGER DEFAULT 8192,
   private      IN BOOLEAN DEFAULT TRUE)
RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(create_pipe,WNDS,RNDS);
```

Parameters

Table 70–3 CREATE_PIPE Function Parameters

Parameter	Description
`pipename`	Name of the pipe you are creating. You must use this name when you call `SEND_MESSAGE` and `RECEIVE_MESSAGE`. This name must be unique across the instance. Caution: Do not use pipe names beginning with `ORA$`. These are reserved for use by procedures provided by Oracle. Pipename should not be longer than 128 bytes, and is case insensitive. At this time, the name cannot contain Globalization Support characters.
`maxpipesize`	The maximum size allowed for the pipe, in bytes. The total size of all of the messages on the pipe cannot exceed this amount. The message is blocked if it exceeds this maximum. The default `maxpipesize` is 8192 bytes. The `maxpipesize` for a pipe becomes a part of the characteristics of the pipe and persists for the life of the pipe. Callers of `SEND_MESSAGE` with larger values cause the `maxpipesize` to be increased. Callers with a smaller value use the existing, larger value.
`private`	Uses the default, `TRUE`, to create a private pipe. Public pipes can be implicitly created when you call `SEND_MESSAGE`.

Return Values

Table 70–4 CREATE_PIPE Function Return Values

Return	Description
0	Successful. If the pipe already exists and the user attempting to create it is authorized to use it, then Oracle returns 0, indicating success, and any data already in the pipe remains. If a user connected as SYSDBA/SYSOPER re-creates a pipe, then Oracle returns status 0, but the ownership of the pipe remains unchanged.
ORA-23322	Failure due to naming conflict. If a pipe with the same name exists and was created by a different user, then Oracle signals error ORA-23322, indicating the naming conflict.

Exceptions

Table 70–5 CREATE_PIPE Function Exception

Exception	Description
Null pipe name	Permission error: Pipe with the same name already exists, and you are not allowed to use it.

NEXT_ITEM_TYPE Function

This function determines the datatype of the next item in the local message buffer.

After you have called RECEIVE_MESSAGE to place pipe information in a local buffer, call NEXT_ITEM_TYPE.

Syntax

```
DBMS_PIPE.NEXT_ITEM_TYPE
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(next_item_type,WNDS,RNDS);
```

Return Values

Table 70–6 NEXT_ITEM_TYPE Function Return Values

Return	Description
0	No more items
6	NUMBER
9	VARCHAR2
11	ROWID
12	DATE
23	RAW

PACK_MESSAGE Procedures

This procedure builds your message in the local message buffer. To send a message, first make one or more calls to PACK_MESSAGE. Then, call SEND_MESSAGE to send the message in the local buffer on the named pipe.

The procedure is overloaded to accept items of type VARCHAR2, NCHAR, NUMBER, DATE., RAW and ROWID items. In addition to the data bytes, each item in the buffer requires one byte to indicate its type, and two bytes to store its length. One additional byte is needed to terminate the message.The overhead for all types other than VARCHAR is 4 bytes.

Syntax

```
DBMS_PIPE.PACK_MESSAGE (
   item  IN  VARCHAR2);

DBMS_PIPE.PACK_MESSAGE (
   item  IN  NCHAR);

DBMS_PIPE.PACK_MESSAGE (
   item  IN  NUMBER);

DBMS_PIPE.PACK_MESSAGE (
   item  IN  DATE);

DBMS_PIPE.PACK_MESSAGE_RAW (
   item  IN  RAW);

DBMS_PIPE.PACK_MESSAGE_ROWID (
   item  IN  ROWID);
```

Pragmas

```
pragma restrict_references(pack_message,WNDS,RNDS);
pragma restrict_references(pack_message_raw,WNDS,RNDS);
pragma restrict_references(pack_message_rowid,WNDS,RNDS);
```

Parameters

Table 70–7 PACK_MESSAGE Procedure Parameters

Parameter	Description
item	Item to pack into the local message buffer.

Usage Notes

In Oracle database version 8.x, the char-set-id (2 bytes) and the char-set-form (1 byte) are stored with each data item. Therefore, the overhead when using Oracle database version 8.x is 7 bytes.

When you call SEND_MESSAGE to send this message, you must indicate the name of the pipe on which you want to send the message. If this pipe already exists, then you must have sufficient privileges to access this pipe. If the pipe does not already exist, then it is created automatically.

Exceptions

ORA-06558 is raised if the message buffer overflows (currently 4096 bytes). Each item in the buffer takes one byte for the type, two bytes for the length, plus the actual data. There is also one byte needed to terminate the message.

PURGE Procedure

This procedure empties the contents of the named pipe.

An empty implicitly-created pipe is aged out of the shared global area according to the least-recently-used algorithm. Thus, calling PURGE lets you free the memory associated with an implicitly-created pipe.

Syntax

```
DBMS_PIPE.PURGE (
   pipename  IN  VARCHAR2);
```

Pragmas

```
pragma restrict_references(purge,WNDS,RNDS);
```

Parameters

Table 70–8 PURGE Procedure Parameters

Parameter	Description
pipename	Name of pipe from which to remove all messages. The local buffer may be overwritten with messages as they are discarded. Pipename should not be longer than 128 bytes, and is case-insensitive.

Usage Notes

Because PURGE calls RECEIVE_MESSAGE, the local buffer might be overwritten with messages as they are purged from the pipe. Also, you can receive an ORA-23322 (insufficient privileges) error if you attempt to purge a pipe with which you have insufficient access rights.

Exceptions

Permission error if pipe belongs to another user.

RECEIVE_MESSAGE Function

This function copies the message into the local message buffer.

Syntax

```
DBMS_PIPE.RECEIVE_MESSAGE (
   pipename     IN VARCHAR2,
   timeout      IN INTEGER      DEFAULT maxwait)
RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(receive_message,WNDS,RNDS);
```

Parameters

Table 70–9 RECEIVE_MESSAGE Function Parameters

Parameter	Description
pipename	Name of the pipe on which you want to receive a message. Names beginning with ORA$ are reserved for use by Oracle
timeout	Time to wait for a message, in seconds. The default value is the constant MAXWAIT, which is defined as 86400000 (1000 days). A timeout of 0 lets you read without blocking.

Return Values

Table 70–10 RECEIVE_MESSAGE Function Return Values

Return	Description
0	Success
1	Timed out. If the pipe was implicitly-created and is empty, then it is removed.
2	Record in the pipe is too large for the buffer. (This should not happen.)
3	An interrupt occurred.
ORA-23322	User has insufficient privileges to read from the pipe.

Usage Notes

To receive a message from a pipe, first call RECEIVE_MESSAGE. When you receive a message, it is removed from the pipe; hence, a message can only be received once. For implicitly-created pipes, the pipe is removed after the last record is removed from the pipe.

If the pipe that you specify when you call RECEIVE_MESSAGE does not already exist, then Oracle implicitly creates the pipe and waits to receive the message. If the message does not arrive within a designated timeout interval, then the call returns and the pipe is removed.

After receiving the message, you must make one or more calls to UNPACK_MESSAGE to access the individual items in the message. The UNPACK_MESSAGE procedure is

overloaded to unpack items of type DATE, NUMBER, VARCHAR2, and there are two additional procedures to unpack RAW and ROWID items. If you do not know the type of data that you are attempting to unpack, then call NEXT_ITEM_TYPE to determine the type of the next item in the buffer.

Exceptions

Table 70–11 RECEIVE_MESSAGE Function Exceptions

Exception	Description
Null pipe name	Permission error. Insufficient privilege to remove the record from the pipe. The pipe is owned by someone else.

RESET_BUFFER Procedure

This procedure resets the PACK_MESSAGE and UNPACK_MESSAGE positioning indicators to 0.

Because all pipes share a single buffer, you may find it useful to reset the buffer before using a new pipe. This ensures that the first time you attempt to send a message to your pipe, you do not inadvertently send an expired message remaining in the buffer.

Syntax

```
DBMS_PIPE.RESET_BUFFER;
```

Pragmas

```
pragma restrict_references(reset_buffer,WNDS,RNDS);
```

REMOVE_PIPE Function

This function removes explicitly-created pipes.

Pipes created implicitly by SEND_MESSAGE are automatically removed when empty. However, pipes created explicitly by CREATE_PIPE are removed only by calling REMOVE_PIPE, or by shutting down the instance. All unconsumed records in the pipe are removed before the pipe is deleted.

This is similar to calling PURGE on an implicitly-created pipe.

Syntax

```
DBMS_PIPE.REMOVE_PIPE (
   pipename  IN  VARCHAR2)
RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(remove_pipe,WNDS,RNDS);
```

Parameters

Table 70–12 REMOVE_PIPE Function Parameters

Parameter	Description
pipename	Name of pipe that you want to remove.

Return Values

Table 70–13 REMOVE_PIPE Function Return Values

Return	Description
0	Success If the pipe does not exist, or if the pipe already exists and the user attempting to remove it is authorized to do so, then Oracle returns 0, indicating success, and any data remaining in the pipe is removed.
ORA-23322	Insufficient privileges. If the pipe exists, but the user is not authorized to access the pipe, then Oracle signals error ORA-23322, indicating insufficient privileges.

Exceptions

Table 70–14 REMOVE_PIPE Function Exception

Exception	Description
Null pipe name	Permission error: Insufficient privilege to remove pipe. The pipe was created and is owned by someone else.

SEND_MESSAGE Function

This function sends a message on the named pipe.

The message is contained in the local message buffer, which was filled with calls to PACK_MESSAGE. You can create a pipe explicitly using CREATE_PIPE, otherwise, it is created implicitly.

Syntax

```
DBMS_PIPE.SEND_MESSAGE (
    pipename     IN VARCHAR2,
    timeout      IN INTEGER DEFAULT MAXWAIT,
    maxpipesize  IN INTEGER DEFAULT 8192)
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(send_message,WNDS,RNDS);
```

Parameters

Table 70–15 SEND_MESSAGE Function Parameters

Parameter	Description
pipename	Name of the pipe on which you want to place the message. If you are using an explicit pipe, then this is the name that you specified when you called CREATE_PIPE. Caution: Do not use pipe names beginning with 'ORA$'. These names are reserved for use by procedures provided by Oracle. Pipename should not be longer than 128 bytes, and is case-insensitive. At this time, the name cannot contain Globalization Support characters.
timeout	Time to wait while attempting to place a message on a pipe, in seconds. The default value is the constant MAXWAIT, which is defined as 86400000 (1000 days).
maxpipesize	Maximum size allowed for the pipe, in bytes. The total size of all the messages on the pipe cannot exceed this amount. The message is blocked if it exceeds this maximum. The default is 8192 bytes. The maxpipesize for a pipe becomes a part of the characteristics of the pipe and persists for the life of the pipe. Callers of SEND_MESSAGE with larger values cause the maxpipesize to be increased. Callers with a smaller value simply use the existing, larger value. Specifying maxpipesize as part of the SEND_MESSAGE procedure eliminates the need for a separate call to open the pipe. If you created the pipe explicitly, then you can use the optional maxpipesize parameter to override the creation pipe size specifications.

Return Values

Table 70–16 SEND_MESSAGE Function Return Values

Return	Description
0	Success. If the pipe already exists and the user attempting to create it is authorized to use it, then Oracle returns 0, indicating success, and any data already in the pipe remains. If a user connected as SYSDBS/SYSOPER re-creates a pipe, then Oracle returns status 0, but the ownership of the pipe remains unchanged.
1	Timed out. This procedure can timeout either because it cannot get a lock on the pipe, or because the pipe remains too full to be used. If the pipe was implicitly-created and is empty, then it is removed.
3	An interrupt occurred. If the pipe was implicitly created and is empty, then it is removed.
ORA-23322	Insufficient privileges. If a pipe with the same name exists and was created by a different user, then Oracle signals error ORA-23322, indicating the naming conflict.

Exceptions

Table 70–17 SEND_MESSAGE Function Exception

Exception	Description
Null pipe name	Permission error. Insufficient privilege to write to the pipe. The pipe is private and owned by someone else.

UNIQUE_SESSION_NAME Function

This function receives a name that is unique among all of the sessions that are currently connected to a database.

Multiple calls to this function from the same session always return the same value. You might find it useful to use this function to supply the `PIPENAME` parameter for your `SEND_MESSAGE` and `RECEIVE_MESSAGE` calls.

Syntax

```
DBMS_PIPE.UNIQUE_SESSION_NAME
  RETURN VARCHAR2;
```

Pragmas

```
pragma restrict_references(unique_session_name,WNDS,RNDS,WNPS);
```

Return Values

This function returns a unique name. The returned name can be up to 30 bytes.

UNPACK_MESSAGE Procedures

This procedure retrieves items from the buffer.

After you have called RECEIVE_MESSAGE to place pipe information in a local buffer, call UNPACK_MESSAGE.

> **Note:** The UNPACK_MESSAGE procedure is overloaded to return items of type VARCHAR2, NCHAR, NUMBER, or DATE. There are two additional procedures to unpack RAW and ROWID items.

Syntax

```
DBMS_PIPE.UNPACK_MESSAGE (
   item  OUT VARCHAR2);

DBMS_PIPE.UNPACK_MESSAGE (
   item  OUT NCHAR);

DBMS_PIPE.UNPACK_MESSAGE (
   item  OUT NUMBER);

DBMS_PIPE.UNPACK_MESSAGE (
   item  OUT DATE);

DBMS_PIPE.UNPACK_MESSAGE_RAW (
   item  OUT RAW);

DBMS_PIPE.UNPACK_MESSAGE_ROWID (
   item  OUT ROWID);
```

Pragmas

```
pragma restrict_references(unpack_message,WNDS,RNDS);
pragma restrict_references(unpack_message_raw,WNDS,RNDS);
pragma restrict_references(unpack_message_rowid,WNDS,RNDS);
```

Parameters

Table 70–18 UNPACK_MESSAGE Procedure Parameters

Parameter	Description
item	Argument to receive the next unpacked item from the local message buffer.

Exceptions

ORA-06556 or 06559 are generated if the buffer contains no more items, or if the item is not of the same type as that requested.

71

DBMS_PREPROCESSOR

The DBMS_PREPROCESSOR package provides an interface to print or retrieve the source text of a PL/SQL unit in its post-processed form.

This package contains the following topics

Using DBMS_PREPROCESSOR

Overview

There are three styles of subprograms.

1. Subprograms that take a schema name, a unit type name, and the unit name.
2. Subprograms that take a `VARCHAR2` string which contains the source text of an arbitrary PL/SQL compilation unit.
3. Subprograms that take a `VARCHAR2` index-by table which contains the segmented source text of an arbitrary PL/SQL compilation unit.

Subprograms of the first style are used to print or retrieve the post-processed source text of a stored PL/SQL unit. The user must have the privileges necessary to view the original source text of this unit. The user must also specify the schema in which the unit is defined, the type of the unit, and the name of the unit. If the schema is null, then the current user schema is used. If the status of the stored unit is `VALID` and the user has the required privilege, then the post-processed source text is guaranteed to be the same as that of the unit the last time it was compiled.

Subprograms of the second or third style are used to generate post-processed source text in the current user schema. The source text is passed in as a single `VARCHAR2` string in the second style, or as a `VARCHAR2` index-by table in the third style. The source text can represent an arbitrary PL/SQL compilation unit. A typical usage is to pass the source text of an anonymous block and generate its post-processed source text in the current user schema. The third style can be useful when the source text exceeds the `VARCHAR2` length limit.

Operating Notes

- For subprograms of the first style, the status of the stored PL/SQL unit does not need to be VALID. Likewise, the source text passed in as a VARCHAR2 string or a VARCHAR2 index-by table may contain compile time errors. If errors are found when generating the post-processed source, the error message text will also appear at the end of the post-processed source text. In some cases, the preprocessing can be aborted because of errors. When this happens, the post-processed source text will appear to be incomplete and the associated error message can help to indicate that an error has occurred during preprocessing.
- For subprograms of the second or third style, the source text can represent any arbitrary PL/SQL compilation unit. However, the source text of a valid PL/SQL compilation unit cannot include commonly used prefixes such as CREATE OR REPLACE. In general, the input source should be syntactically prepared in a way as if it were obtained from the ALL_SOURCE view. The following list gives some examples of valid initial syntax for some PL/SQL compilation units.

  ```
  anonymous block   (BEGIN | DECLARE) ...
  package           PACKAGE <name> ...
  package body      PACKAGE BODY <name> ...
  procedure         PROCEDURE <name> ...
  function          FUNCTION <name> ...
  type              TYPE <name> ...
  type body         TYPE BODY <name> ...
  trigger           (BEGIN | DECLARE) ...
  ```

  If the source text represents a named PL/SQL unit that is valid, that unit will not be created after its post-processed source text is generated.
- If the text of a wrapped PL/SQL unit is obtained from the ALL_SOURCE view, the keyword WRAPPED always immediately follows the name of the unit, as in this example:

  ```
  PROCEDURE "some proc" WRAPPED
  a000000
  b2
  ...
  ```

  If such source text is presented to one of the GET_POST_PROCESSED_SOURCE Functions or to one of the PRINT_POST_PROCESSED_SOURCE Procedures, the exception DBMS_PREPROCESSOR.WRAPPED_INPUT is raised.

Data Structures

The DBMS_PREPROCESSOR package defines a TABLE type.

Table Types

SOURCE_LINES_T Table Type

SOURCE_LINES_T Table Type

This table type stores lines of post-processed source text. It is used to hold PL/SQL source text both before and after it is processed. It is especially useful in cases in which the amount of text exceeds 32K.

Syntax

```
TYPE source_lines_t IS
    TABLE OF VARCHAR2(32767) INDEX BY BINARY_INTEGER;
```

Summary of DBMS_PREPROCESSOR Subprograms

Table 71–1 DBMS_PREPROCESSOR Package Subprograms

Subprogram	Description
GET_POST_PROCESSED_SOURCE Functions on page 71-8	Returns the post-processed source text
PRINT_POST_PROCESSED_SOURCE Procedures on page 71-10	Prints post-processed source text

GET_POST_PROCESSED_SOURCE Functions

This overloaded function returns the post-processed source text. The different functionality of each form of syntax is presented along with the definition.

Syntax

Returns post-processed source text of a stored PL/SQL unit:

```
DBMS_PREPROCESSOR.GET_POST_PROCESSED_SOURCE (
   object_type    VARCHAR2,
   schema_name    VARCHAR2,
   object_name    VARCHAR2);
  RETURN source_lines_t;
```

Returns post-processed source text of a compilation unit:

```
DBMS_PREPROCESSOR.GET_POST_PROCESSED_SOURCE (
   source        VARCHAR2);
  RETURN source_lines_t;
```

Returns post-processed source text of an INDEX-BY table containing the source text of the compilation unit:

```
DBMS_PREPROCESSOR.GET_POST_PROCESSED_SOURCE (
   source        source_lines_t);
  RETURN source_lines_t;
```

Parameters

Table 71–2 GET_POST_PROCESSED_SOURCE Function Parameters

Parameter	Description
object_type	Must be one of PACKAGE, PACKAGE BODY, PROCEDURE, FUNCTION, TYPE, TYPE, BODY or TRIGGER. Case sensitive.
schema_name	The schema name. Case insensitive unless a quoted identifier is used. If NULL, use current schema.
object_name	The name of the object.The object_type is always case insensitive. Case insensitive unless a quoted identifier is used.
source	The source text of the compilation unit
source_lines_t	INDEX-BY table containing the source text of the compilation unit. The source text is a concatenation of all the non-NULL INDEX-BY table elements in ascending index order.

Return Values

The function returns an INDEX-BY table containing the lines of the post-processed source text starting from index 1.

Usage Notes

- Newline characters are not removed.
- Each line in the post-processed source text is mapped to a row in the INDEX-BY table.
- In the post-processed source, unselected text will have blank lines.

Exceptions

Table 71–3 GET_POST_PROCESSED_SOURCE Function Exceptions

Exception	Description
ORA-24234	Insufficient privileges or object does not exist
ORA-24235	Bad value for object type. Should be one of PACKAGE, PACKAGE BODY, PROCEDURE, FUNCTION, TYPE, TYPE, BODY or TRIGGER.
ORA-24236	The source text is empty
ORA-00931	Missing identifier. The object_name should not be NULL.
ORA-06502	Numeric or value error: ■ Character string buffer too small ■ A line is too long (> 32767 bytes)

PRINT_POST_PROCESSED_SOURCE Procedures

This overloaded procedure calls DBMS_OUTPUT.PUT_LINE to let you view post-processed source text. The different functionality of each form of syntax is presented along with the definition.

Syntax

Prints post-processed source text of a stored PL/SQL unit:

```
DBMS_PREPROCESSOR.PRINT_POST_PROCESSED_SOURCE (
   object_type    VARCHAR2,
   schema_name    VARCHAR2,
   object_name    VARCHAR2);
```

Prints post-processed source text of a compilation unit:

```
DBMS_PREPROCESSOR.PRINT_POST_PROCESSED_SOURCE (
   source        VARCHAR2);
```

Prints post-processed source text of an INDEX-BY table containing the source text of the compilation unit:

```
DBMS_PREPROCESSOR.PRINT_POST_PROCESSED_SOURCE (
   source        source_lines_t);
```

Parameters

Table 71–4 PRINT_POST_PROCESSED_SOURCE Procedure Parameters

Parameter	Description
object_type	Must be one of PACKAGE, PACKAGE BODY, PROCEDURE, FUNCTION, TYPE, TYPE, BODY or TRIGGER. Case sensitive.
schema_name	The schema name. Case insensitive unless a quoted identifier is used. If NULL, use current schema.
object_name	The name of the object.The object_type is always case insensitive. Case insensitive unless a quoted identifier is used.
source	The source text of the compilation unit
source_lines_t	INDEX-BY table containing the source text of the compilation unit. The source text is a concatenation of all the non-NULL INDEX-BY table elements in ascending index order.

Exceptions

Table 71–5 PRINT_POST_PROCESSED_SOURCE Procedure Exceptions

Exception	Description
ORA-24234	Insufficient privileges or object does not exist
ORA-24235	Bad value for object type. Should be one of PACKAGE, PACKAGE BODY, PROCEDURE, FUNCTION, TYPE, TYPE, BODY or TRIGGER.
ORA-24236	The source text is empty
ORA-00931	Missing identifier. The object_name should not be NULL.

Table 71–5 (Cont.) PRINT_POST_PROCESSED_SOURCE Procedure Exceptions

Exception	Description
ORA-06502	Numeric or value error: ■ Character string buffer too small ■ A line is too long (> 32767 bytes)

Usage Notes

The index-by table may contain holes. NULL elements are ignored when doing the concatenation.

72

DBMS_PREDICTIVE_ANALYTICS

Data mining can discover useful information buried in vast amounts of data. However, it is often the case that both the programming interfaces and the data mining expertise required to obtain these results are too complex for use by the wide audiences that can obtain benefits from using Oracle Data Mining (ODM).

The `DBMS_PREDICTIVE_ANALYTICS` package addresses both of these complexities by automating the entire data mining process from data preprocessing through model building to scoring new data. This package provides an important tool that makes data mining possible for a broad audience of users, in particular, business analysts.

Data used by ODM consists of tables or views stored in an Oracle database. Each column in a record (row) holds an item of information. Data mining models are often used to identify important columns or to predict column values.

The `DBMS_PREDICTIVE_ANALYTICS` package supports the following functionality:

- `EXPLAIN` - Ranks attributes in order of influence in explaining a target column.
- `PREDICT` - Predict the value of a column.

This chapter contains the following topics:

Using DBMS_PREDICTIVE_ANALYTICS

This section contains topics which relate to using the `DBMS_PREDICTIVE_ANALYTICS` package.

Overview

Data mining, according to a commonly used process model, requires the following steps:

1. Understand the business problem.
2. Understand the data.
3. Prepare the data for mining.
4. Create models using the prepared data.
5. Evaluate the models.
6. Deploy and use the model to score new data.

`DBMS_PREDICTIVE_ANALYTICS` automates parts of step 3 and steps 4 and 5 of this process.

The user provides input data in a single table or view. For `EXPLAIN`, the user identifies a column to explain; for `PREDICT`, the user identifies a column to predict and also identifies a case id column. The procedure accepts the input, analyzes the data, performs suitable preprocessing, builds and tests models, selects the best model, and applies the model to data.

Input for `DBMS_PREDICTIVE_ANALYTICS` is the name of a single table or view in an Oracle database. Each column in the table must have one of the following data types:

- `NUMBER`
- `FLOAT`
- `CHAR`
- `VARCHAR2`
- `DATE`
- `TIMESTAMP`

Summary of DBMS_PREDICTIVE_ANALYTICS Subprograms

Table 72–1 DBMS_PREDICTIVE_ANALYTICS Package Subprograms

Subprogram	Purpose
EXPLAIN Procedure on page 72-5	Ranks attributes in order of influence in explaining a target column
PREDICT Procedure on page 72-7	Predicts the value of a column based on values in the input data

EXPLAIN Procedure

The procedure ranks attributes in order of influence in explaining a target column.

The procedure analyzes the input table, performs data preprocessing, builds a model, analyzes the model to identify key columns, and creates a result table listing the important columns and quantifying the explanatory power of each important column.

Syntax

```
DBMS_PREDICTIVE_ANALYTICS.EXPLAIN (
     data_table_name     IN VARCHAR2,
     explain_column_name IN VARCHAR2,
     result_table_name   IN VARCHAR2,
     data_schema_name    IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 72–2 EXPLAIN Procedure Parameters

Parameter	Description
`data_table_name`	Name of input table or view
`explain_column_name`	Name of column to be explained
`result_table_name`	Name of table where results are saved
`data_schema_name`	Name of schema where the input table or view resides. Default: the current schema.

Usage Notes

The result table has the following definition:

```
column_name              VARCHAR2(30)
explanatory_value        NUMBER
rank                     NUMBER
```

Table 72–3 describes the columns in the result table.

Table 72–3 EXPLAIN Procedure Result Table

Column Name	Meaning
`column_name`	Name of a column in the input data; all columns except the explained column are listed in the result table.
`explanatory_value`	Value indicating how useful the column is for determining the value of the explained column. Higher values indicate greater explanatory power. Value can range from 0 to 1. An individual column's explanatory value is independent of other columns in the input table. Instead, the values are based on how strong each individual column correlates with the explained column. The value is affected by the number of records in the input table, and the relations of the values of the column to the values of the explain column. An explanatory power value of 0 implies there is no useful correlation between the column's values and the explain column's values. An explanatory power of 1 implies perfect correlation; such columns should be eliminated from consideration for `PREDICT`. In practice, an explanatory power equal to 1 is rarely returned.

Table 72–3 (Cont.) EXPLAIN Procedure Result Table

Column Name	Meaning
rank	Ranking of explanatory power. Rows with equal values for explanatory_power have the same rank. Rank values are not skipped in the event of ties.

Example

The following example performs an EXPLAIN operation and views the results:

```
--Perform EXPLAIN operation
BEGIN
    DBMS_PREDICTIVE_ANALYTICS.EXPLAIN(
        data_table_name      => 'census_dataset',
        explain_column_name  => 'class',
        result_table_name    => 'census_explain_result');
END;
/
--View results
SELECT * FROM census_explain_result;

COLUMN_NAME     EXPLANATORY_VALUE  RANK
-----------     -----------------  ----
RELATIONSHIP    .21234788             1
MARITAL_STATUS  .195201808            2
CAPITAL_GAIN    .102951498            3
OCCUPATION      .06883765             4
EDUCATION       .067517394            5
EDUCATION_NUM   .067517394            5
SEX             .055541542            6
HOURS_PER_WEEK  .032476973            7
AGE             .021933245            8
CAPITAL_LOSS    .013083265            9
RACE            .009670242           10
WORKCLASS       0                    11
NATIVE_COUNTRY  0                    11
WEIGHT          0                    11
PERSON_ID       0                    11

15 rows selected.
```

PREDICT Procedure

This procedure is used to predict values of a specific column. The input consists of a table and a target column, the target column containing the values to predict. The input data must contain some cases in which the target value is known (that is, is not NULL). Cases where the target values are known are used to train models.

PREDICT returns a predicted value for every case, including those where the value is known.

Syntax

```
DBMS_PREDICTIVE_ANALYTICS.PREDICT (
    accuracy                OUT NUMBER,
    data_table_name         IN VARCHAR2,
    case_id_column_name     IN VARCHAR2,
    target_column_name      IN VARCHAR2,
    result_table_name       IN VARCHAR2,
    data_schema_name        IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 72–4 PREDICT Procedure Parameters

Parameter	Description
data_table_name	Name of input table or view
case_id_column_name	Name of column that uniquely identifies each case in the input data (for example, the column containing the customer id or case id)
target_column_name	Name of the column containing the value to predict (the target)
result_table_name	Name of table where results will be saved
data_schema_name	Name of schema where the input table or view resides and where the result table is written. Default: the current schema.

Usage Notes

The result table has the following definition:

```
case_id_column_name      VARCHAR2 or NUMBER
prediction               VARCHAR2 or NUMBER
probability_number       NUMBER
```

Table 72–5 describes the result table of the PREDICT procedure.

Table 72–5 PREDICT Procedure Result Table

Column Name	Meaning
case_id_column_name	Each of the cases identified in the case_id column. This is the same as the name that was passed in. The data type is the same as input case_id type.
prediction	The predicted value of the target column for the given case. The data type is the same as the input target_column_name type.
probability	For classification (categorical target), the probability of the prediction. For regression problems (numerical target), this column contains NULL.

Predictions are returned for all cases in the input data.

Predicted values for known cases may be interesting in some situations, for example, to perform deviation analysis, that is, to compare predicted values and actual values.

Example

The following example performs a `PREDICT` operation and display the first 10 predictions. In this example, since the target column `class` is categorical, a probability is returned for each prediction:

```
--Perform PREDICT operation
DECLARE
    v_accuracy NUMBER(10,9);
BEGIN
    DBMS_PREDICTIVE_ANALYTICS.PREDICT(
        accuracy             => v_accuracy,
        data_table_name      => 'census_dataset',
        case_id_column_name  => 'person_id',
        target_column_name   => 'class',
        result_table_name    => 'census_predict_result');
    DBMS_OUTPUT.PUT_LINE('Accuracy = ' || v_accuracy);
END;
/
--View first 10 predictions
SELECT * FROM census_predict_result where rownum < 10;

PERSON_ID    PREDICTION      PROBABILITY
----------   ----------      -----------
  2                   1      .418787003
  7                   0      .922977991
  8                   0      .99869723
  9                   0      .999999605
 10                   0      .9999009
 11                   0      .999999996
 12                   1      .953949094
 15                   0      .99999997
 16                   0      .999968961

9 rows selected.
```

73

DBMS_PROFILER

The DBMS_PROFILER package provides an interface to profile existing PL/SQL applications and identify performance bottlenecks. You can then collect and persistently store the PL/SQL profiler data.

This chapter contains the following topics:

Using DBMS_PROFILER

Overview

This package enables the collection of profiler (perfoprmance) data for performance improvement or for determining code coverage for PL/SQL applications. Application developers can use code coverage data to focus their incremental testing efforts.

With this interface, you can generate profiling information for all named library units that are executed in a session. The profiler gathers information at the PL/SQL virtual machine level. This information includes the total number of times each line has been executed, the total amount of time that has been spent executing that line, and the minimum and maximum times that have been spent on a particular execution of that line.

> **Note:** It is possible to infer the code coverage figures for PL/SQL units for which data has been collected.

The profiling information is stored in database tables. This enables querying on the data: you can build customizable reports (summary reports, hottest lines, code coverage data, and so on. And you can analyze the data.

The `PROFTAB.SQL` script creates tables with the columns, datatypes, and definitions as shown in Table 73–1, Table 73–2, and Table 73–3.

Table 73–1 Columns in Table PLSQL_PROFILER_RUNS

Column	Datatype	Definition
runid	NUMBER PRIMARY KEY	Unique run identifier from plsql_profiler_ runnumber
related_run	NUMBER	Runid of related run (for client/server correlation)
run_owner	VARCHAR2(32),	User who started run
run_date	DATE	Start time of run
run_comment	VARCHAR2(2047)	User provided comment for this run
run_total_ time	NUMBER	Elapsed time for this run in nanoseconds
run_system_ info	VARCHAR2(2047)	Currently unused
run_comment1	VARCHAR2(2047)	Additional comment
spare1	VARCHAR2(256)	Unused

Table 73–2 Columns in Table PLSQL_PROFILER_UNITS

Column	Datatype	Definition
runid	NUMBER	Primary key, references plsql_profiler_runs,
unit_number	NUMBER	Primary key, internally generated library unit #
unit_type	VARCHAR2(32)	Library unit type
unit_owner	VARCHAR2(32)	Library unit owner name
unit_name	VARCHAR2(32)	Library unit name timestamp on library unit
unit_ timestamp	DATE	In the future will be used to detect changes to unit between runs

Table 73–2 (Cont.) Columns in Table PLSQL_PROFILER_UNITS

Column	Datatype	Definition
total_time	NUMBER	Total time spent in this unit in nanoseconds. The profiler does not set this field, but it is provided for the convenience of analysis tools.
spare1	NUMBER	Unused
spare2	NUMBER	Unused

Table 73–3 Columns in Table PLSQL_PROFILER_DATA

Column	Datatype	Definition
runid	NUMBER	Primary key, unique (generated) run identifier
unit_number	NUMBER	Primary key, internally generated library unit number
line#	NUMBER	Primary key, not null, line number in unit
total_occur	NUMBER	Number of times line was executed
total_time	NUMBER	Total time spent executing line in nanoseconds
min_time	NUMBER	Minimum execution time for this line in nanoseconds
max_time	NUMBER	Maximum execution time for this line in nanoseconds
spare1	NUMBER	Unused
spare2	NUMBER	Unused
spare3	NUMBER	Unused
spare4	NUMBER	Unused

With Oracle database version 8.x, a sample textual report writer(profrep.sql) is provided with the PL/SQL demo scripts.

Security Model

The profiler only gathers data for units for which a user has CREATE privilege; you cannot use the package to profile units for which EXECUTE ONLY access has been granted. In general, if a user can debug a unit, the same user can profile it. However, a unit can be profiled whether or not it has been compiled DEBUG. Oracle advises that modules that are being profiled should be compiled DEBUG, since this provides additional information about the unit in the database.

Note: DBMS_PROFILER treats any program unit that is compiled in NATIVE mode as if you do not have CREATE privilege, that is, you will not get any output.

Operational Notes

Typical Run

Improving application performance is an iterative process. Each iteration involves the following steps:

1. Running the application with one or more benchmark tests with profiler data collection enabled.
2. Analyzing the profiler data and identifying performance problems.
3. Fixing the problems.

The PL/SQL profiler supports this process using the concept of a *"run"*. A run involves running the application through benchmark tests with profiler data collection enabled. You can control the beginning and the ending of a run by calling the `START_PROFILER` and `STOP_PROFILER` functions.

A typical run involves:

- Starting profiler data collection in the run.
- Executing PL/SQL code for which profiler and code coverage data is required.
- Stopping profiler data collection, which writes the collected data for the run into database tables

> **Note:** The collected profiler data is not automatically stored when the user disconnects. You must issue an explicit call to the `FLUSH_DATA` or the `STOP_PROFILER` function to store the data at the end of the session. Stopping data collection stores the collected data.

As the application executes, profiler data is collected in memory data structures that last for the duration of the run. You can call the `FLUSH_DATA` function at intermediate points during the run to get incremental data and to free memory for allocated profiler data structures.

Flushing the collected data involves storing collected data in database tables. The tables should already exist in the profiler user's schema. The `PROFTAB.SQL` script creates the tables and other data structures required for persistently storing the profiler data.

Note that running `PROFTAB.SQL` drops the current tables. The `PROFTAB.SQL` script is in the `RDBMS/ADMIN` directory. Some PL/SQL operations, such as the first execution of a PL/SQL unit, may involve I/O to catalog tables to load the byte code for the PL/SQL unit being executed. Also, it may take some time executing package initialization code the first time a package procedure or function is called.

To avoid timing this overhead, *"warm up"* the database before collecting profile data. To do this, run the application once without gathering profiler data.

You can allow profiling across all users of a system, for example, to profile all users of a package, independent of who is using it. In such cases, the SYSADMIN should use a modified `PROFLOAD.SQL` script which:

- Creates the profiler tables and sequence
- Grants SELECT/INSERT/UPDATE on those tables and sequence to all users
- Defines public synonyms for the tables and sequence

Note: Do not alter the actual fields of the tables.

See Also: "FLUSH_DATA Function and Procedure" on page 73-10.

Two Methods of Exception Generation

Each routine in this package has two versions that allow you to determine how errors are reported.

- A function that returns success/failure as a status value and will never raise an exception
- A procedure that returns normally if it succeeds and raises an exception if it fails

In each case, the parameters of the function and procedure are identical. Only the method by which errors are reported differs. If there is an error, there is a correspondence between the error codes that the functions return, and the exceptions that the procedures raise.

To avoid redundancy, the following section only provides details about the functional form.

Exceptions

Table 73–4 DBMS_PROFILER Exceptions

Exception	Description
version_mismatch	Corresponds to error_version.
profiler_error	Corresponds to either "error_param" or "error_io".

A 0 return value from any function denotes successful completion; a nonzero return value denotes an error condition. The possible errors are as follows:

- 'A subprogram was called with an incorrect parameter.'

  ```
  error_param constant binary_integer := 1;
  ```

- 'Data flush operation failed. Check whether the profiler tables have been created, are accessible, and that there is adequate space.'

  ```
  error_io    constant binary_integer := 2;
  ```

- There is a mismatch between package and database implementation. Oracle returns this error if an incorrect version of the DBMS_PROFILER package is installed, and if the version of the profiler package cannot work with this database version. The only recovery is to install the correct version of the package.

  ```
  error_version constant binary_integer := -1;
  ```

Summary of DBMS_PROFILER Subprograms

Table 73–5 DBMS_PROFILER Package Subprograms

Subprogram	Description
FLUSH_DATA Function and Procedure on page 73-10	Flushes profiler data collected in the user's session
GET_VERSION Procedure on page 73-11	Gets the version of this API
INTERNAL_VERSION_CHECK Function on page 73-12	Verifies that this version of the `DBMS_PROFILER` package can work with the implementation in the database
PAUSE_PROFILER Function and Procedure on page 73-13	Pauses profiler data collection
RESUME_PROFILER Function and Procedure on page 73-14	Resumes profiler data collection
START_PROFILER Functions and Procedures on page 73-15	Starts profiler data collection in the user's session
STOP_PROFILER Function and Procedure on page 73-16	Stops profiler data collection in the user's session

FLUSH_DATA Function and Procedure

This function flushes profiler data collected in the user's session. The data is flushed to database tables, which are expected to preexist.

> **Note:** Use the PROFTAB.SQL script to create the tables and other data structures required for persistently storing the profiler data.

Syntax

```
DBMS_PROFILER.FLUSH_DATA
  RETURN BINARY_INTEGER;

DBMS_PROFILER.FLUSH_DATA;
```

GET_VERSION Procedure

This procedure gets the version of this API.

Syntax

```
DBMS_PROFILER.GET_VERSION (
   major  OUT BINARY_INTEGER,
   minor  OUT BINARY_INTEGER);
```

Parameters

Table 73–6 GET_VERSION Procedure Parameters

Parameter	Description
major	Major version of DBMS_PROFILER.
minor	Minor version of DBMS_PROFILER.

INTERNAL_VERSION_CHECK Function

This function verifies that this version of the DBMS_PROFILER package can work with the implementation in the database.

Syntax

```
DBMS_PROFILER.INTERNAL_VERSION_CHECK
  RETURN BINARY_INTEGER;
```

PAUSE_PROFILER Function and Procedure

This function pauses profiler data collection.

Syntax

```
DBMS_PROFILER.PAUSE_PROFILER
  RETURN BINARY_INTEGER;

DBMS_PROFILER.PAUSE_PROFILER;
```

RESUME_PROFILER Function and Procedure

This function resumes profiler data collection.

Syntax

```
DBMS_PROFILER.RESUME_PROFILER
  RETURN BINARY_INTEGER;

DBMS_PROFILER.RESUME_PROFILER;
```

START_PROFILER Functions and Procedures

This function starts profiler data collection in the user's session.

There are two overloaded forms of the START_PROFILER function; one returns the run number of the started run, as well as the result of the call. The other does not return the run number. The first form is intended for use with GUI-based tools controlling the profiler.

Syntax

```
DBMS_PROFILER.START_PROFILER(
   run_comment   IN VARCHAR2 := sysdate,
   run_comment1  IN VARCHAR2 :='',
   run_number    OUT BINARY_INTEGER)
 RETURN BINARY_INTEGER;

DBMS_PROFILER.START_PROFILER(
   run_comment IN VARCHAR2 := sysdate,
   run_comment1 IN VARCHAR2 :='')
RETURN BINARY_INTEGER;

DBMS_PROFILER.START_PROFILER(
   run_comment   IN VARCHAR2 := sysdate,
   run_comment1  IN VARCHAR2 :='',
   run_number    OUT BINARY_INTEGER);

DBMS_PROFILER.START_PROFILER(
   run_comment IN VARCHAR2 := sysdate,
   run_comment1 IN VARCHAR2 :='');
```

Parameters

Table 73–7 START_PROFILER Function Parameters

Parameter	Description
run_comment	Each profiler run can be associated with a comment. For example, the comment could provide the name and version of the benchmark test that was used to collect data.
run_number	Stores the number of the run so you can store and later recall the run's data.
run_comment1	Allows you to make interesting comments about the run.

STOP_PROFILER Function and Procedure

This function stops profiler data collection in the user's session.

This function has the side effect of flushing data collected so far in the session, and it signals the end of a run.

Syntax

```
DBMS_PROFILER.STOP_PROFILER
  RETURN BINARY_INTEGER;

DBMS_PROFILER.STOP_PROFILER;
```

74

DBMS_PROPAGATION_ADM

The DBMS_PROPAGATION_ADM package, one of a set of Streams packages, provides administrative interfaces for configuring a propagation from a source queue to a destination queue.

> **See Also:** *Oracle Streams Concepts and Administration* and *Oracle Streams Replication Administrator's Guide* for more information about this package and propagations

This chapter contains the following topic:

Summary of DBMS_PROPAGATION_ADM Subprograms

Table 74–1 DBMS_PROPAGATION_ADM Package Subprograms

Subprogram	Description
ALTER_PROPAGATION Procedure on page 74-3	Adds, alters, or removes a rule set for a propagation
CREATE_PROPAGATION Procedure on page 74-5	Creates a propagation and specifies the source queue, destination queue, and rule set for the propagation
DROP_PROPAGATION Procedure on page 74-8	Drops a propagation
START_PROPAGATION Procedure on page 74-10	Starts a propagation
STOP_PROPAGATION Procedure on page 74-11	Stops a propagation

Note: All subprograms commit unless specified otherwise.

ALTER_PROPAGATION Procedure

This procedure adds, alters, or removes a rule set for a propagation.

> **See Also:** *Oracle Streams Concepts and Administration* and Chapter 92, "DBMS_RULE_ADM" for more information about rules and rule sets

Syntax

```
DBMS_PROPAGATION_ADM.ALTER_PROPAGATION(
   propagation_name          IN  VARCHAR2,
   rule_set_name             IN  VARCHAR2  DEFAULT NULL,
   remove_rule_set           IN  BOOLEAN   DEFAULT FALSE,
   negative_rule_set_name    IN  VARCHAR2  DEFAULT NULL,
   remove_negative_rule_set  IN  BOOLEAN   DEFAULT FALSE);
```

Parameters

Table 74–2 ALTER_PROPAGATION Procedure Parameters

Parameter	Description
`propagation_name`	The name of the propagation you are altering. You must specify an existing propagation name. Do not specify an owner.
`rule_set_name`	The name of the positive rule set for the propagation. The positive rule set contains the rules that instruct the propagation to propagate messages. If you want to use a positive rule set for the propagation, then you must specify an existing rule set in the form `[`*`schema_name`*`.]`*`rule_set_name`*. For example, to specify a positive rule set in the `hr` schema named `prop_rules`, enter `hr.prop_rules`. If the schema is not specified, then the current user is the default. An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the `DBMS_STREAMS_ADM` package or the `DBMS_RULE_ADM` package. If you specify `NULL` and the `remove_rule_set` parameter is set to `FALSE`, then the procedure retains any existing positive rule set. If you specify `NULL` and the `remove_rule_set` parameter is set to `TRUE`, then the procedure removes any existing positive rule set.
`remove_rule_set`	If `TRUE`, then the procedure removes the positive rule set for the specified propagation. If you remove a positive rule set for a propagation, and the propagation does not have a negative rule set, then the propagation propagates all messages. If you remove a positive rule set for a propagation, and a negative rule set exists for the propagation, then the propagation propagates all messages in its queue that are not discarded by the negative rule set. If `FALSE`, then the procedure retains the positive rule set for the specified propagation. If the `rule_set_name` parameter is non-`NULL`, then this parameter should be set to `FALSE`.

Table 74–2 (Cont.) ALTER_PROPAGATION Procedure Parameters

Parameter	Description
negative_rule_set_name	The name of the negative rule set for the propagation. The negative rule set contains the rules that instruct the propagation to discard messages. If you want to use a negative rule set for the propagation, then you must specify an existing rule set in the form [*schema_name*.]*rule_set_name*. For example, to specify a negative rule set in the hr schema named neg_rules, enter hr.neg_rules. If the schema is not specified, then the current user is the default. An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the DBMS_STREAMS_ADM package or the DBMS_RULE_ADM package. If you specify NULL and the remove_negative_rule_set parameter is set to FALSE, then the procedure retains any existing negative rule set. If you specify NULL and the remove_negative_rule_set parameter is set to TRUE, then the procedure removes any existing negative rule set. If you specify both a positive and a negative rule set for a propagation, then the negative rule set is always evaluated first.
remove_negative_rule_set	If TRUE, then the procedure removes the negative rule set for the specified propagation. If you remove a negative rule set for a propagation, and the propagation does not have a positive rule set, then the propagation propagates all messages. If you remove a negative rule set for a propagation, and a positive rule set exists for the propagation, then the propagation propagates all messages in its queue that are not discarded by the positive rule set. If FALSE, then the procedure retains the negative rule set for the specified propagation. If the negative_rule_set_name parameter is non-NULL, then this parameter should be set to FALSE.

CREATE_PROPAGATION Procedure

This procedure creates a propagation and specifies the source queue, destination queue, and any rule set for the propagation. A propagation propagates messages in a local source queue to a destination queue. The destination queue might or might not be in the same database as the source queue.

Syntax

```
DBMS_PROPAGATION_ADM.CREATE_PROPAGATION(
   propagation_name        IN  VARCHAR2,
   source_queue            IN  VARCHAR2,
   destination_queue       IN  VARCHAR2,
   destination_dblink      IN  VARCHAR2  DEFAULT NULL,
   rule_set_name           IN  VARCHAR2  DEFAULT NULL,
   negative_rule_set_name  IN  VARCHAR2  DEFAULT NULL,
   queue_to_queue          IN  BOOLEAN   DEFAULT NULL);
```

Parameters

Table 74–3 CREATE_PROPAGATION Procedure Parameters

Parameter	Description
propagation_name	The name of the propagation you are creating. A NULL setting is not allowed. Do not specify an owner. **Note:** The propagation_name setting cannot be altered after the propagation is created.
source_queue	The name of the source queue, specified as [*schema_name.*]*queue_name*. The current database must contain the source queue. For example, to specify a source queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter. If the schema is not specified, then the current user is the default.
destination_queue	The name of the destination queue, specified as [*schema_name.*]*queue_name*. For example, to specify a destination queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter. If the schema is not specified, then the current user is the default.
destination_dblink	The name of the database link that will be used by the propagation. The database link is from the database that contains the source queue to the database that contains the destination queue. If NULL, then the source queue and destination queue must be in the same database. **Note:** Connection qualifiers are not allowed.

Table 74–3 (Cont.) CREATE_PROPAGATION Procedure Parameters

Parameter	Description
rule_set_name	The name of the positive rule set for the propagation. The positive rule set contains the rules that instruct the propagation to propagate messages. If you want to use a positive rule set for the propagation, then you must specify an existing rule set in the form [*schema_name*.]*rule_set_name*. For example, to specify a positive rule set in the hr schema named prop_rules, enter hr.prop_rules. If the schema is not specified, then the current user is the default. An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the DBMS_STREAMS_ADM package or the DBMS_RULE_ADM package. If you specify NULL, and no negative rule set exists for the propagation, then the propagation propagates all messages in its queue. If you specify NULL, and a negative rule set exists for the propagation, then the propagation propagates all messages in its queue that are not discarded by the negative rule set.
negative_rule_set_name	The name of the negative rule set for the propagation. The negative rule set contains the rules that instruct the propagation to discard messages. If you want to use a negative rule set for the propagation, then you must specify an existing rule set in the form [*schema_name*.]*rule_set_name*. For example, to specify a negative rule set in the hr schema named neg_rules, enter hr.neg_rules. If the schema is not specified, then the current user is the default. An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the DBMS_STREAMS_ADM package or the DBMS_RULE_ADM package. If you specify NULL, and no positive rule set exists for the propagation, then the propagation propagates all messages in its queue. If you specify NULL, and a positive rule set exists for the propagation, then the propagation propagates all messages in its queue that are not discarded by the positive rule set. If you specify both a positive and a negative rule set for a propagation, then the negative rule set is always evaluated first.
queue_to_queue	If TRUE, then the propagation is a queue to queue propagation. A queue-to-queue propagation always has its own propagation job and uses a service for automatic failover when the destination queue is a buffered queue in a Real Application Clusters (RAC) database. If FALSE or NULL, then the propagation is a queue-to-dblink propagation. A queue-to-dblink propagation can share a propagation job with other propagations that use the same database link and does not support automatic failover in a RAC environment. **See Also:** *Oracle Streams Concepts and Administration* for more information about queue-to-queue propagations

Usage Notes

This procedure starts propagation and might create a propagation job. If this procedure creates a propagation job, then it establishes a default schedule for the propagation job.

The user who owns the source queue is the user who propagates messages. This user must have the necessary privileges to propagate messages.

See Also:

- Chapter 92, "DBMS_RULE_ADM"
- *Oracle Streams Concepts and Administration* for more information about propagations, the privileges required to propagate messages, propagation jobs, and propagation schedules

DROP_PROPAGATION Procedure

This procedure drops a propagation and deletes all captured and user-enqueued messages for the destination queue in the source queue. This procedure also removes the schedule for propagation from the source queue to the destination queue.

Syntax

```
DBMS_PROPAGATION_ADM.DROP_PROPAGATION(
   propagation_name       IN  VARCHAR2,
   drop_unused_rule_sets  IN  BOOLEAN  DEFAULT FALSE);
```

Parameters

Table 74–4 DROP_PROPAGATION Procedure Parameters

Parameter	Description
propagation_name	The name of the propagation you are dropping. You must specify an existing propagation name. Do not specify an owner.
drop_unused_rule_sets	If TRUE, then the procedure drops any rule sets, positive and negative, used by the specified propagation if these rule sets are not used by any other Streams client, which includes capture processes, propagations, apply processes, and messaging clients. If this procedure drops a rule set, then this procedure also drops any rules in the rule set that are not in another rule set. If FALSE, then the procedure does not drop the rule sets used by the specified propagation, and the rule sets retain their rules.

Usage Notes

When you use this procedure to drop a propagation, information about rules created for the propagation using the DBMS_STREAMS_ADM package is removed from the data dictionary views for Streams rules. Information about such a rule is removed even if the rule is not in either rule set for the propagation.

> **See Also:** *Oracle Streams Concepts and Administration* for more information about Streams data dictionary views

The following are the data dictionary views for Streams rules:

- ALL_STREAMS_GLOBAL_RULES
- DBA_STREAMS_GLOBAL_RULES
- ALL_STREAMS_MESSAGE_RULES
- DBA_STREAMS_MESSAGE_RULES
- ALL_STREAMS_SCHEMA_RULES
- DBA_STREAMS_SCHEMA_RULES
- ALL_STREAMS_TABLE_RULES
- DBA_STREAMS_TABLE_RULES

Note: When you drop a propagation, the propagation job used by the propagation is dropped automatically, if no other propagations are using the propagation job.

START_PROPAGATION Procedure

This procedure starts a propagation.

Syntax

```
DBMS_PROPAGATION_ADM.START_PROPAGATION(
   propagation_name  IN  VARCHAR2);
```

Parameter

Table 74–5 START_PROPAGATION Procedure Parameter

Parameter	Description
`propagation_name`	The name of the propagation you are starting. You must specify an existing propagation name. Do not specify an owner.

Usage Notes

The propagation status is persistently recorded. Hence, if the status is `ENABLED`, then the propagation is started upon database instance startup.

STOP_PROPAGATION Procedure

This procedure stops a propagation.

Syntax

```
DBMS_PROPAGATION_ADM.STOP_PROPAGATION(
   propagation_name  IN  VARCHAR2,
   force             IN  BOOLEAN  DEFAULT FALSE);
```

Parameter

Table 74–6 STOP_PROPAGATION Procedure Parameter

Parameter	Description
propagation_name	The name of the propagation you are stopping. You must specify an existing propagation name. Do not specify an owner.
force	If TRUE, then the procedure stops the propagation and clears the statistics for the propagation. If FALSE, then the procedure stops the propagation without clearing the statistics for the propagation.

Usage Notes

The propagation status is persistently recorded. Hence, if the status is DISABLED or ABORTED, then the propagation is not started upon database instance startup.

75

DBMS_RANDOM

The DBMS_RANDOM package provides a built-in random number generator. DBMS_RANDOM is not intended for cryptography.

This chapter contains the following topics:

Using DBMS_RANDOM

Security Model

This package should be installed as `SYS`. By default, the package is initialized with the current user name, current time down to the second, and the current session.

Operational Notes

- `DBMS_RANDOM.RANDOM` produces integers in [-2^^31, 2^^31).
- `DBMS_RANDOM.VALUE` produces numbers in [0,1) with 38 digits of precision.

DBMS_RANDOM can be explicitly initialized, but does not need to be initialized before calling the random number generator. It will automatically initialize with the date, userid, and process id if no explicit initialization is performed.

If this package is seeded twice with the same seed, then accessed in the same way, it will produce the same results in both cases.

In some cases, such as when testing, you may want the sequence of random numbers to be the same on every run. In that case, you seed the generator with a constant value by calling one of the overloads of `DBMS_RANDOM.SEED`. To produce different output for every run, simply to omit the call to "Seed" and the system will choose a suitable seed for you.

Summary of DBMS_RANDOM Subprograms

Table 75–1 DBMS_RANDOM Package Subprograms

Subprogram	Description
INITIALIZE Procedure on page 75-6	Initializes the package with a seed value
NORMAL Function on page 75-7	Returns random numbers in a normal distribution
RANDOM Procedure on page 75-8	Generates a random number
SEED Procedures on page 75-9	Resets the seed
STRING Function on page 75-10	Gets a random string
TERMINATE Procedure on page 75-11	Terminates package
VALUE Functions on page 75-12	This function gets a random number, greater than or equal to 0 and less than 1, with 38 digits to the right of the decimal (38-digit precision), while the overloaded function gets a random Oracle number x, where x is greater than or equal to `low` and less than `high`

Note: The INITIALIZE Procedure, RANDOM Procedure and the TERMINATE Procedure are all obsolete and, while currently supported, are included in this release for legacy reasons only.

INITIALIZE Procedure

This procedure initializes the generator (but see Usage Notes).

Syntax

```
DBMS_RANDOM.INITIALIZE (
   val  IN  BINARY_INTEGER);
```

Pragmas

```
PRAGMA restrict_references (initialize, WNDS)
```

Parameters

Table 75–2 INITIALIZE Procedure Parameters

Parameter	Description
val	The seed number used to generate a random number.

Usage Notes

This procedure is obsolete as it simply calls the SEED Procedures on page 75-9.

NORMAL Function

This function returns random numbers in a standard normal distribution.

Syntax

```
DBMS_RANDOM.NORMAL
  RETURN NUMBER;
```

Pragmas

```
PRAGMA restrict_references (normal, WNDS)
```

Return Values

Table 75–3 NORMAL Procedure Parameters

Parameter	Description
number	Returns a random number.

RANDOM Procedure

This procedure generates a random number (but see Usage Notes).

Syntax

```
DBMS_RANDOM.RANDOM
   RETURN binary_integer;
```

Pragmas

```
PRAGMA restrict_references (random, WNDS)
```

Return Values

Table 75–4 RANDOM Procedure Parameters

Parameter	Description
`binary_integer`	Returns a random integer greater or equal to -power(2,31) and less than power(2,31).

Usage Notes

This procedure is obsolete and, although it is currently supported, it should not be used.

SEED Procedures

This procedure resets the seed.

Syntax

```
DBMS_RANDOM.SEED (
   seed  IN  BINARY_INTEGER);

DBMS_RANDOM.SEED (
   seed  IN  VARCHAR2);
```

Pragmas

```
PRAGMA restrict_references (seed, WNDS);
```

Parameters

Table 75–5 SEED Procedure Parameters

Parameter	Description
seed	Seed number or string used to generate a random number.

Usage Notes

The seed can be a string up to length 2000.

STRING Function

This function gets a random string.

Syntax

```
DBMS_RANDOM.STRING
   opt  IN  CHAR,
   len  IN  NUMBER)
  RETURN VARCHAR2;
```

Pragmas

```
PRAGMA restrict_references (string, WNDS)
```

Parameters

Table 75–6 STRING Function Parameters

Parameter	Description
opt	Specifies what the returning string looks like: ■ 'u', 'U' - returning string in uppercase alpha characters ■ 'l', 'L' - returning string in lowercase alpha characters ■ 'a', 'A' - returning string in mixed case alpha characters ■ 'x', 'X' - returning string in uppercase alpha-numeric characters ■ 'p', 'P' - returning string in any printable characters. Otherwise the returning string is in uppercase alpha characters.
len	The length of the returning string.

Return Values

Table 75–7 STRING Function Return Values

Parameter	Description
VARCHAR2	Returns a VARCHAR2.

TERMINATE Procedure

When you are finished with the package, call the TERMINATE procedure (but see Usage Notes)

Syntax

```
DBMS_RANDOM.TERMINATE
```

Usage Notes

This procedure performs no function and, although it is currently supported, it is obsolete and should not be used.

VALUE Functions

The basic function gets a random number, greater than or equal to 0 and less than 1, with 38 digits to the right of the decimal (38-digit precision). Alternatively, you can get a random Oracle number x, where x is greater than or equal to `low` and less than `high`.

Syntax

```
DBMS_RANDOM.VALUE
  RETURN NUMBER;

DBMS_RANDOM.VALUE(
  low  IN  NUMBER,
  high IN  NUMBER)
RETURN NUMBER;
```

Parameters

Table 75–8 VALUE Function Parameters

Parameter	Description
`low`	The lowest number in a range from which to generate a random number. The number generated may be equal to `low`.
`high`	The highest number below which to generate a random number. The number generated will be less than `high`.

Return Values

Table 75–9 VALUE Function Return Values

Parameter	Description
`NUMBER`	Returns an Oracle Number.

76

DBMS_RECTIFIER_DIFF

The `DBMS_RECTIFIER_DIFF` package provides an interface used to detect and resolve data inconsistencies between two replicated sites.

Documentation of DBMS_RECTIFIER_DIFF

For a complete description of this package within the context of Replication, see DBMS_RECTIFIER_DIFF in the *Oracle Database Advanced Replication Management API Reference*.

77

DBMS_REDEFINITION

The DBMS_REDEFINITION package provides an interface to perform an online redefinition of tables.

> **See Also:** *Oracle Database Administrator's Guide* for more information about online redefinition of tables

This chapter contains the following topics:

Using DBMS_REDEFINITION

Overview

To achieve online redefinition, incrementally maintainable local materialized views are used. These logs keep track of the changes to the master tables and are used by the materialized views during refresh synchronization.

Constants

The DBMS_REDEFINITION package uses the constants shown in Table 77–1, " DBMS_REDEFINITION Constants":

Table 77–1 DBMS_REDEFINITION Constants

Constant	Type	Value	Description
CONS_CONSTRAINT	PLS_INTEGER	3	Used to specify that dependent object type is a constraint
CONS_INDEX	PLS_INTEGER	2	Used to specify that dependent object type is a index
CONS_ORIG_PARAMS	PLS_INTEGER	1	Used to specify that indexes should be cloned with their original storage parameters
CONS_TRIGGER	PLS_INTEGER	4	Used to specify that dependent object type is a trigger
CONS_USE_PK	BINARY_INTEGER	1	Used to indicate that the redefinition should be done using primary keys or pseudo-primary keys (unique keys with all component columns having not-NULL constraints)
CONS_USE_ROWID	BINARY_INTEGER	2	Used to indicate that the redefinition should be done using rowids

Operational Notes

- CONS_USE_PK and CONS_USE_ROWID are constants used as input to the "options_flag" parameter in both the START_REDEF_TABLE Procedure and CAN_REDEF_TABLE Procedure. CONS_USE_ROWID is used to indicate that the redefinition should be done using rowids while CONS_USE_PK implies that the redefinition should be done using primary keys or pseudo-primary keys (which are unique keys with all component columns having NOT NULL constraints).
- CONS_INDEX, CONS_TRIGGER and CONS_CONSTRAINT are used to specify the type of the dependent object being (un)registered in REGISTER_DEPENDENT_OBJECT Procedure and UNREGISTER_DEPENDENT_OBJECT Procedure (parameter "dep_type").

  CONS_INDEX ==> dependent object is of type INDEX

  CONS_TRIGGER ==> dependent object is of type TRIGGER

  CONS_CONSTRAINT==> dependent object type is of type CONSTRAINT
- CONS_ORIG_PARAMS as used as input to the "copy_indexes" parameter in COPY_TABLE_DEPENDENTS Procedure. Using this parameter implies that the indexes on the original table be copied onto the interim table using the same storage parameters as that of the original index.

Rules and Limits

For information about various rules and limits that apply to implementation of this package, see the *Oracle Database Administrator's Guide*.

Summary of DBMS_REDEFINITION Subprograms

Table 77–2 DBMS_REDEFINITION Package Subprograms

Subprogram	Description
ABORT_REDEF_TABLE Procedure on page 77-8	Cleans up errors that occur during the redefinition process and removes all temporary objects created by the reorganization process
CAN_REDEF_TABLE Procedure on page 77-9	Determines if a given table can be redefined online
COPY_TABLE_DEPENDENTS Procedure on page 77-10	Copies the dependent objects of the original table onto the interim table
FINISH_REDEF_TABLE Procedure on page 77-12	Completes the redefinition process.
REGISTER_DEPENDENT_OBJECT Procedure on page 77-13	Registers a dependent object (index, trigger or constraint) on the table being redefined and the corresponding dependent object on the interim table
START_REDEF_TABLE Procedure on page 77-14	Initiates the redefinition process
SYNC_INTERIM_TABLE Procedure on page 77-15	Keeps the interim table synchronized with the original table
UNREGISTER_DEPENDENT_OBJECT Procedure on page 77-16	Unregisters a dependent object (index, trigger or constraint) on the table being redefined and the corresponding dependent object on the interim table

ABORT_REDEF_TABLE Procedure

This procedure cleans up errors that occur during the redefinition process. This procedure can also be used to terminate the redefinition process any time after the START_REDEF_TABLE Procedure has been called and before the FINISH_REDEF_TABLE Procedure is called. This process will remove the temporary objects that are created by the redefinition process such as materialized view logs.

Syntax

```
DBMS_REDEFINITION.ABORT_REDEF_TABLE (
   uname       IN VARCHAR2,
   orig_table  IN VARCHAR2,
   int_table   IN VARCHAR2,
  part_name    IN  VARCHAR2 := NULL);
```

Parameters

Table 77–3 ABORT_REDEF_TABLE Procedure Parameters

Parameter	Description
uname	The schema name of the tables.
orig_table	The name of the table to be redefined.
int_table	The name of the interim table.
part_name	The name of the partition being redefined. If redefining only a single partition of a table, specify the partition name in this parameter. NULL implies the entire table is being redefined.

CAN_REDEF_TABLE Procedure

This procedure determines if a given table can be redefined online. This is the first step of the online redefinition process. If the table is not a candidate for online redefinition, an error message is raised.

Syntax

```
DBMS_REDEFINITION.CAN_REDEF_TABLE (
   uname         IN  VARCHAR2,
   tname        IN  VARCHAR2,
   options_flag  IN  PLS_INTEGER := 1,
   part_name     IN  VARCHAR2 := NULL);
```

Parameters

Table 77–4 CAN_REDEF_TABLE Procedure Parameters

Parameter	Description
uname	The schema name of the table
tname	The name of the table to be re-organized
options_flag	Indicates the type of redefinition method to use. ■ If dbms_redefinition.cons_use_pk, the redefinition is done using primary keys or pseudo-primary keys (unique keys with all component columns having NOT NULL constraints). The default method of redefinition is using primary keys. ■ If dbms_redefinition.cons_use_rowid, the redefinition is done using rowids.
part_name	The name of the partition being redefined. If redefining only a single partition of a table, specify the partition name in this parameter. NULL implies the entire table is being redefined.

Exceptions

If the table is not a candidate for online redefinition, an error message is raised.

COPY_TABLE_DEPENDENTS Procedure

This procedure clones the dependent objects of the table being redefined onto the interim table and registers the dependent objects. This procedure does not clone the already registered dependent objects.

This subprogram is used to clone the dependent objects like grants, triggers, constraints and privileges from the table being redefined to the interim table (which represents the post-redefinition table).

Syntax

```
DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS(
   uname                   IN  VARCHAR2,
   orig_table              IN  VARCHAR2,
   int_table               IN  VARCHAR2,
   copy_indexes            IN  PLS_INTEGER := 1,
   copy_triggers           IN  BOOLEAN     := TRUE,
   copy_constraints        IN  BOOLEAN     := TRUE,
   copy_privileges         IN  BOOLEAN     := TRUE,
   ignore_errors           IN  BOOLEAN     := FALSE,
   num_errors              OUT PLS_INTEGER,
   copy_statistics         IN  BOOLEAN     := FALSE);
```

Parameters

Table 77–5 COPY_TABLE_DEPENDENTS Procedure Parameters

Parameter	Description
uname	The schema name of the tables.
orig_table	The name of the table being redefined.
int_table	The name of the interim table.
copy_indexes	A flag indicating whether to copy the indexes ■ 0 - do not copy any index ■ dbms_redefinition.cons_orig_params – copy the indexes using the physical parameters of the source indexes
copy_triggers	TRUE = clone triggers, FALSE = do nothing
copy_constraints	TRUE = clone constraints, FALSE = do nothing. If compatibility setting is 10.2 or higher, then clone CHECK and NOT NULL constraints
copy_privileges	TRUE = clone privileges, FALSE = do nothing
ignore_errors	TRUE = if an error occurs while cloning a particular dependent object, then skip that object and continue cloning other dependent objects. FALSE = that the cloning process should stop upon encountering an error.
num_errors	The number of errors that occurred while cloning dependent objects
copy_statistics	TRUE = copy statistics, FALSE = do nothing

Usage Notes

- The user must check the column `num_errors` before proceeding to ensure that no errors occurred during the cloning of the objects.
- In case of an error, the user should fix the cause of the error and call the COPY_TABLE_DEPENDENTS Procedure again to clone the dependent object. Alternatively the user can manually clone the dependent object and then register the manually cloned dependent object using the REGISTER_DEPENDENT_OBJECT Procedure.
- All cloned referential constraints involving the interim tables will be created disabled (they will be automatically enabled after the redefinition) and all triggers on interim tables will not fire till the redefinition is completed. After the redefinition is complete, the cloned objects will be renamed to the corresponding pre-redefinition names of the objects (from which they were cloned from).
- It is the user's responsibility that the cloned dependent objects are unaffected by the redefinition. All the triggers will be cloned and it is the user's responsibility that the cloned triggers are unaffected by the redefinition.

FINISH_REDEF_TABLE Procedure

This procedure completes the redefinition process. Before this step, you can create new indexes, triggers, grants, and constraints on the interim table. The referential constraints involving the interim table must be disabled. After completing this step, the original table is redefined with the attributes and data of the interim table. The original table is locked briefly during this procedure.

Syntax

```
DBMS_REDEFINITION.FINISH_REDEF_TABLE (
   uname       IN VARCHAR2,
   orig_table  IN VARCHAR2,
   int_table   IN VARCHAR2,
   part_name   IN  VARCHAR2 := NULL);
```

Parameters

Table 77–6 FINISH_REDEF_TABLE Procedure Parameters

Parameters	Description
uname	The schema name of the tables.
orig_table	The name of the table to be redefined.
int_table	The name of the interim table.
part_name	The name of the partition being redefined. If redefining only a single partition of a table, specify the partition name in this parameter. NULL implies the entire table is being redefined.

REGISTER_DEPENDENT_OBJECT Procedure

This procedure registers a dependent object (index, trigger or constraint) on the table being redefined and the corresponding dependent object on the interim table.

This can be used to have the same object on each table but with different attributes. For example: for an index, the storage and tablespace attributes could be different but the columns indexed remain the same

Syntax

```
DBMS_REDEFINITION.REGISTER_DEPEPENDENT_OBJECT(
   uname             IN VARCHAR2,
   orig_table        IN VARCHAR2,
   int_table         IN VARCHAR2,
   dep_type          IN  PLS_INTEGER,
   dep_owner         IN VARCHAR2,
   dep_orig_name     IN VARCHAR2,
   dep_int_name      IN  VARCHAR2);
```

Parameters

Table 77–7 REGISTER_DEPENDENT_OBJECT Procedure Parameters

Parameters	Description
uname	The schema name of the tables.
orig_table	The name of the table to be redefined.
int_table	The name of the interim table.
dep_type	The type of the dependent object.
dep_owner	The owner of the dependent object.
dep_orig_name	The name of the original dependent object.
dep_int_name	The name of the interim dependent object.

Usage Notes

- Attempting to register an already registered object will raise an error.
- Registering a dependent object will automatically remove that object from DBA_REDEFINITION_ERRORS if an entry exists for that object.

START_REDEF_TABLE Procedure

Prior to calling this procedure, you must manually create an empty interim table (in the same schema as the table to be redefined) with the desired attributes of the post-redefinition table, and then call this procedure to initiate the redefinition.

Syntax

```
DBMS_REDEFINITION.START_REDEF_TABLE (
   uname          IN VARCHAR2,
   orig_table     IN VARCHAR2,
   int_table      IN VARCHAR2,
   col_mapping    IN VARCHAR2 := NULL,
   options_flag   IN BINARY_INTEGER := 1,
   orderby_cols   IN VARCHAR2 := NULL,
   part_name     IN  VARCHAR2 := NULL);
```

Parameters

Table 77–8 START_REDEF_TABLE Procedure Parameters

Parameter	Description
uname	The schema name of the tables.
orig_table	The name of the table to be redefined.
int_table	The name of the interim table.
col_mapping	The mapping information from the columns in the original table to the columns in the interim table. (This is similar to the column list on the SELECT clause of a query.) If NULL, all the columns in the original table are selected and have the same name after redefinition.
options_flag	Indicates the type of redefinition method to use. ■ If dbms_redefinition.cons_use_pk, the redefinition is done using primary keys or pseudo-primary keys (unique keys with all component columns having NOT NULL constraints). The default method of redefinition is using primary keys. ■ If dbms_redefinition.cons_use_rowid, the redefinition is done using rowids.
orderby_cols	This optional parameter accepts the list of columns (along with the optional keyword(s) ascending/descending) with which to order by the rows during the initial instantiation of the interim table (the order by is only done for the initial instantiation and not for subsequent synchronizations)
part_name	The name of the partition being redefined. If redefining only a single partition of a table, specify the partition name in this parameter. NULL implies the entire table is being redefined.

SYNC_INTERIM_TABLE Procedure

This procedure keeps the interim table synchronized with the original table.

Syntax

```
DBMS_REDEFINITION.SYNC_INTERIM_TABLE (
   uname          IN  VARCHAR2,
   orig_table     IN  VARCHAR2,
   int_table      IN  VARCHAR2,
   part_name      IN  VARCHAR2 := NULL);
```

Parameters

Table 77–9 SYNC_INTERIM_TABLE Procedure Parameters

Parameter	Description
uname	The schema name of the table.
orig_table	The name of the table to be redefined.
int_table	The name of the interim table.
part_name	The name of the partition being redefined. If redefining only a single partition of a table, specify the partition name in this parameter. NULL implies the entire table is being redefined.

Usage Notes

- This step is useful in minimizing the amount of synchronization needed to be done by the FINISH_REDEF_TABLE Procedure before completing the online redefinition.
- This procedure can be called between long running operations (such as CREATE INDEX) on the interim table to sync it up with the data in the original table and speed up subsequent operations.

UNREGISTER_DEPENDENT_OBJECT Procedure

This procedure unregisters a dependent object (index, trigger or constraint) on the table being redefined and the corresponding dependent object on the interim table.

Syntax

```
DBMS_REDEFINITION.UNREGISTER_DEPEPENDENT_OBJECT(
   uname              IN VARCHAR2,
   orig_table        IN VARCHAR2,
   int_table         IN VARCHAR2,
   dep_type          IN PLS_INTEGER,
   dep_owner         IN VARCHAR2,
   dep_orig_name     IN VARCHAR2,
   dep_int_name      IN  VARCHAR2);
```

Parameters

Table 77–10 UNREGISTER_DEPENDENT_OBJECT Procedure Parameters

Parameters	Description
uname	The schema name of the tables.
orig_table	The name of the table to be redefined.
int_table	The name of the interim table.
dep_type	The type of the dependent object.
dep_owner	The owner of the dependent object.
dep_orig_name	The name of the original dependent object.
dep_int_name	The name of the interim dependent object.

78

DBMS_REFRESH

The DBMS_REFRESH package enables you to create groups of materialized views that can be refreshed together to a transactionally consistent point in time.

- Documentation of DBMS_REFRESH

Documentation of DBMS_REFRESH

For a complete description of this package within the context of Replication, see `DBMS_REFRESH` in the *Oracle Database Advanced Replication Management API Reference*.

79

DBMS_REPAIR

The DBMS_REPAIR package contains data corruption repair procedures that enable you to detect and repair corrupt blocks in tables and indexes. You can address corruptions where possible and continue to use objects while you attempt to rebuild or repair them.

> **See Also:** For detailed information about using the DBMS_REPAIR package, see *Oracle Database Administrator's Guide.*

This chapter contains the following topics:

Using DBMS_REPAIR

Overview

Note: The DBMS_REPAIR package is intended for use by database administrators only. It is not intended for use by application developers.

Security Model

The package is owned by `SYS`. Execution privilege is not granted to other users.

Constants

The DBMS_REPAIR package defines several enumerated constants that should be used for specifying parameter values. Enumerated constants must be prefixed with the package name. For example, DBMS_REPAIR.TABLE_OBJECT.

Table 79–1 lists the parameters and the enumerated constants.

Table 79–1 DBMS_REPAIR Parameters with Enumerated Constants

Parameter	Option	Type	Description
object_type	■ TABLE_OBJECT ■ INDEX_OBJECT ■ CLUSTER_OBJECT	BINARY_ INTEGER	
action	■ CREATE_ACTION ■ DROP_ACTION ■ PURGE_ACTION	BINARY_ INTEGER	
table_type	■ REPAIR_TABLE ■ ORPHAN_TABLE	BINARY_ INTEGER	
flags	■ SKIP_FLAG ■ NOSKIP_FLAG	BINARY_ INTEGER	
object_id	■ ALL_INDEX_ID := 0	BINARY_ INTEGER	Clean up all objects that qualify
wait_for_lock	■ LOCK_WAIT := 1 ■ LOCK_NOWAIT := 0	BINARY_ INTEGER	Specifies whether to try getting DML locks on underlying table [[sub]partition] object

Note: The default table_name will be REPAIR_TABLE when table_type is REPAIR_TABLE, and will be ORPHAN_KEY_TABLE when table_type is ORPHAN_TABLE.

Operating Notes

The procedure to create the ORPHAN_KEYS_TABLE is similar to the one used to create the REPAIR_TABLE.

```
CONNECT / AS SYSDBA;
EXEC DBMS_REPAIR.ADMIN_TABLES('ORPHAN_KEYS_TABLE', DBMS_REPAIR.ORPHAN_TABLE,
                              DBMS_REPAIR.CREATE_ACTION);
EXEC DBMS_REPAIR.ADMIN_TABLES('REPAIR_TABLE', DBMS_REPAIR.REPAIR_TABLE,
                              DBMS_REPAIR.CREATE_ACTION);
DESCRIBE ORPHAN_KEYS_TABLE;
DESCRIBE REPAIR_TABLE;
SELECT * FROM ORPHAN_KEYS_TABLE;
SELECT * FROM REPAIR_TABLE;
```

The DBA would create the repair and orphan keys tables once. Subsequent executions of the CHECK_OBJECT Procedure would add rows into the appropriate table indicating the types of errors found.

The name of the repair and orphan keys tables can be chosen by the user, with the following restriction: the name of the repair table must begin with the 'REPAIR_' prefix, and the name of the orphan keys table must begin with the 'ORPHAN_' prefix. The following code is also legal:

```
CONNECT / AS SYSDBA;
EXEC DBMS_REPAIR.ADMIN_TABLES('ORPHAN_FOOBAR', DBMS_REPAIR.ORPHAN_TABLE,
                              DBMS_REPAIR.CREATE_ACTION);
EXEC DBMS_REPAIR.ADMIN_TABLES('REPAIR_ABCD', DBMS_REPAIR.REPAIR_TABLE,
                              DBMS_REPAIR.CREATE_ACTION);
DESCRIBE ORPHAN_FOOBAR;
DESCRIBE REPAIR_ABCD;
SELECT * FROM ORPHAN_FOOBAR;
SELECT * FROM REPAIR_ABCD;
```

When invoking the CHECK_OBJECT Procedure the name of the repair and orphan keys tables that were created should be specified correctly, especially if the default values were not used in the ADMIN_TABLES Procedure or CREATE_ACTION.

Other actions in the ADMIN_TABLES Procedure can be used to purge/delete the REPAIR_TABLE and the ORPHAN_KEYS_TABLE.

Exceptions

Table 79–2 DBMS_REPAIR Exceptions

Exception	Description	Action
942	Reported by DBMS_REPAIR.ADMIN_TABLES during a DROP_ACTION when the specified table doesn't exist.	
955	Reported by DBMS_REPAIR.CREATE_ACTION when the specified table already exists.	
24120	An invalid parameter was passed to the specified DBMS_REPAIR procedure.	Specify a valid parameter value or use the parameter's default.
24122	An incorrect block range was specified.	Specify correct values for the BLOCK_START and BLOCK_END parameters.
24123	An attempt was made to use the specified feature, but the feature is not yet implemented.	Do not attempt to use the feature.
24124	An invalid ACTION parameter was specified.	Specify CREATE_ACTION, PURGE_ACTION or DROP_ACTION for the ACTION parameter.
24125	An attempt was made to fix corrupt blocks on an object that has been dropped or truncated since DBMS_REPAIR.CHECK_OBJECT was run.	Use DBMS_REPAIR.ADMIN_TABLES to purge the repair table and run DBMS_REPAIR.CHECK_OBJECT to determine whether there are any corrupt blocks to be fixed.
24127	TABLESPACE parameter specified with an ACTION other than CREATE_ACTION.	Do not specify TABLESPACE when performing actions other than CREATE_ACTION.
24128	A partition name was specified for an object that is not partitioned.	Specify a partition name only if the object is partitioned.
24129	An attempt was made to pass a table name parameter without the specified prefix.	Pass a valid table name parameter.
24130	An attempt was made to specify a repair or orphan table that does not exist.	Specify a valid table name parameter.
24131	An attempt was made to specify a repair or orphan table that does not have a correct definition.	Specify a table name that refers to a properly created table.
24132	An attempt was made to specify a table name is greater than 30 characters long.	Specify a valid table name parameter.

Examples

```
/* Fix the bitmap status for all the blocks in table mytab in schema sys */

EXECUTE DBMS_REPAIR.SEGMENT_FIX_STATUS('SYS', 'MYTAB');

/* Mark block number 45, filenumber 1 for table mytab in sys schema as FULL.*/

EXECUTE DBMS_REPAIR.SEGMENT_FIX_STATUS('SYS', 'MYTAB', TABLE_OBJECT,1, 45, 1);
```

Summary of DBMS_REPAIR Subprograms

Table 79–3 DBMS_REPAIR Package Subprograms

Subprogram	Description
ADMIN_TABLES Procedure on page 79-10	Provides administrative functions for the DBMS_REPAIR package repair and orphan key tables, including create, purge, and drop functions
CHECK_OBJECT Procedure on page 79-11	Detects and reports corruptions in a table or index
DUMP_ORPHAN_KEYS Procedure on page 79-13	Reports on index entries that point to rows in corrupt data blocks
FIX_CORRUPT_BLOCKS Procedure on page 79-14	Marks blocks software corrupt that have been previously detected as corrupt by CHECK_OBJECT
ONLINE_INDEX_CLEAN Function on page 79-15	Performs a manual cleanup of failed or interrupted online index builds or rebuilds
REBUILD_FREELISTS Procedure on page 79-16	Rebuilds an object's freelists
SEGMENT_FIX_STATUS Procedure on page 79-17	Fixes the corrupted state of a bitmap entry
SKIP_CORRUPT_BLOCKS Procedure on page 79-18	Sets whether to ignore blocks marked corrupt during table and index scans or to report ORA-1578 when blocks marked corrupt are encountered

ADMIN_TABLES Procedure

This procedure provides administrative functions for the DBMS_REPAIR package repair and orphan key tables.

Syntax

```
DBMS_REPAIR.ADMIN_TABLES (
   table_name  IN   VARCHAR2,
   table_type  IN   BINARY_INTEGER,
   action      IN   BINARY_INTEGER,
   tablespace  IN   VARCHAR2  DEFAULT NULL);
```

Parameters

Table 79–4 ADMIN_TABLES Procedure Parameters

Parameter	Description
table_name	Name of the table to be processed. Defaults to ORPHAN_KEY_TABLE or REPAIR_TABLE based on the specified table_type. When specified, the table name must have the appropriate prefix: ORPHAN_ or REPAIR_.
table_type	Type of table; must be either ORPHAN_TABLE or REPAIR_TABLE. See "Constants" on page 79-5.
action	Indicates what administrative action to perform. Must be either CREATE_ACTION, PURGE_ACTION, or DROP_ACTION. If the table already exists, and if CREATE_ACTION is specified, then an error is returned. PURGE_ACTION indicates to delete all rows in the table that are associated with non-existent objects. If the table does not exist, and if DROP_ACTION is specified, then an error is returned. When CREATE_ACTION and DROP_ACTION are specified, an associated view named DBA_<table_name> is created and dropped respectively. The view is defined so that rows associated with non-existent objects are eliminated. Created in the SYS schema. See "Constants" on page 79-5.
tablespace	Indicates the tablespace to use when creating a table. By default, the SYS default tablespace is used. An error is returned if the tablespace is specified and if the action is not CREATE_ACTION.

CHECK_OBJECT Procedure

This procedure checks the specified objects and populates the repair table with information about corruptions and repair directives.

Validation consists of block checking all blocks in the object.

Syntax

```
DBMS_REPAIR.CHECK_OBJECT (
   schema_name       IN  VARCHAR2,
   object_name       IN  VARCHAR2,
   partition_name    IN  VARCHAR2       DEFAULT NULL,
   object_type       IN  BINARY_INTEGER DEFAULT TABLE_OBJECT,
   repair_table_name IN  VARCHAR2       DEFAULT 'REPAIR_TABLE',
   flags             IN  BINARY_INTEGER DEFAULT NULL,
   relative_fno      IN  BINARY_INTEGER DEFAULT NULL,
   block_start       IN  BINARY_INTEGER DEFAULT NULL,
   block_end         IN  BINARY_INTEGER DEFAULT NULL,
   corrupt_count     OUT BINARY_INTEGER);
```

Parameters

Table 79–5 CHECK_OBJECT Procedure Parameters

Parameter	Description
schema_name	Schema name of the object to be checked.
object_name	Name of the table or index to be checked.
partition_name	Partition or subpartition name to be checked. If this is a partitioned object, and if partition_name is not specified, then all partitions and subpartitions are checked. If this is a partitioned object, and if the specified partition contains subpartitions, then all subpartitions are checked.
object_type	Type of the object to be processed. This must be either TABLE_OBJECT (default) or INDEX_OBJECT. See "Constants" on page 79-5.
repair_table_name	Name of the repair table to be populated. The table must exist in the SYS schema. Use the ADMIN_TABLES Procedure to create a repair table. The default name is REPAIR_TABLE.
flags	Reserved for future use.
relative_fno	Relative file number: Used when specifying a block range.
block_start	First block to process if specifying a block range. May be specified only if the object is a single table, partition, or subpartition.
block_end	Last block to process if specifying a block range. May be specified only if the object is a single table, partition, or subpartition. If only one of block_start or block_end is specified, then the other defaults to the first or last block in the file respectively.
corrupt_count	Number of corruptions reported.

Usage Notes

You may optionally specify a DBA range, partition name, or subpartition name when you want to check a portion of an object.

DUMP_ORPHAN_KEYS Procedure

This procedure reports on index entries that point to rows in corrupt data blocks. For each such index entry encountered, a row is inserted into the specified orphan table.

If the repair table is specified, then any corrupt blocks associated with the base table are handled in addition to all data blocks that are marked software corrupt. Otherwise, only blocks that are marked corrupt are handled.

This information may be useful for rebuilding lost rows in the table and for diagnostic purposes.

Syntax

```
DBMS_REPAIR.DUMP_ORPHAN_KEYS (
   schema_name       IN  VARCHAR2,
   object_name       IN  VARCHAR2,
   partition_name    IN  VARCHAR2       DEFAULT NULL,
   object_type       IN  BINARY_INTEGER DEFAULT INDEX_OBJECT,
   repair_table_name IN  VARCHAR2       DEFAULT 'REPAIR_TABLE',
   orphan_table_name IN  VARCHAR2       DEFAULT 'ORPHAN_KEYS_TABLE',
   flags             IN  BINARY_INTEGER DEFAULT NULL,
   key_count         OUT BINARY_INTEGER);
```

Parameters

Table 79–6 DUMP_ORPHAN_KEYS Procedure Parameters

Parameter	Description
schema_name	Schema name.
object_name	Object name.
partition_name	Partition or subpartition name to be processed. If this is a partitioned object, and if partition_name is not specified, then all partitions and subpartitions are processed. If this is a partitioned object, and if the specified partition contains subpartitions, then all subpartitions are processed.
object_type	Type of the object to be processed. The default is INDEX_OBJECT See "Constants" on page 79-5.
repair_table_name	Name of the repair table that has information regarding corrupt blocks in the base table. The specified table must exist in the SYS schema. The ADMIN_TABLES Procedure is used to create the table.
orphan_table_name	Name of the orphan key table to populate with information regarding each index entry that refers to a row in a corrupt data block. The specified table must exist in the SYS schema. The ADMIN_TABLES Procedure is used to create the table.
flags	Reserved for future use.
key_count	Number of index entries processed.

FIX_CORRUPT_BLOCKS Procedure

This procedure fixes the corrupt blocks in specified objects based on information in the repair table that was previously generated by the CHECK_OBJECT Procedure.

Prior to effecting any change to a block, the block is checked to ensure the block is still corrupt. Corrupt blocks are repaired by marking the block software corrupt. When a repair is effected, the associated row in the repair table is updated with a fix timestamp.

Syntax

```
DBMS_REPAIR.FIX_CORRUPT_BLOCKS (
   schema_name       IN  VARCHAR2,
   object_name       IN  VARCHAR2,
   partition_name    IN  VARCHAR2       DEFAULT NULL,
   object_type       IN  BINARY_INTEGER DEFAULT TABLE_OBJECT,
   repair_table_name IN  VARCHAR2       DEFAULT 'REPAIR_TABLE',
   flags             IN  BINARY_INTEGER DEFAULT NULL,
   fix_count         OUT BINARY_INTEGER);
```

Parameters

Table 79–7 FIX_CORRUPT_BLOCKS Procedure Parameters

Parameter	Description
schema_name	Schema name.
object_name	Name of the object with corrupt blocks to be fixed.
partition_name	Partition or subpartition name to be processed. If this is a partitioned object, and if partition_name is not specified, then all partitions and subpartitions are processed. If this is a partitioned object, and if the specified partition contains subpartitions, then all subpartitions are processed.
object_type	Type of the object to be processed. This must be either TABLE_OBJECT (default) or INDEX_OBJECT. See "Constants" on page 79-5.
repair_table_name	Name of the repair table with the repair directives. Must exist in the SYS schema.
flags	Reserved for future use.
fix_count	Number of blocks fixed.

ONLINE_INDEX_CLEAN Function

This function performs a manual cleanup of failed or interrupted online index builds or rebuilds. This action is also performed periodically by SMON, regardless of user-initiated cleanup.

This function returns `TRUE` if all indexes specified were cleaned up and `FALSE` if one or more indexes could not be cleaned up.

Syntax

```
DBMS_REPAIR.ONLINE_INDEX_CLEAN (
   object_id      IN BINARY_INTEGER DEFAULT ALL_INDEX_ID,
   wait_for_lock  IN BINARY_INTEGER DEFAULT LOCK_WAIT)
 RETURN BOOLEAN;
```

Parameters

Table 79–8 ONLINE_INDEX_CLEAN Function Parameters

Parameter	Description
`object_id`	Object id of index to be cleaned up. The default cleans up all object ids that qualify.
`wait_for_lock`	This parameter specifies whether to try getting DML locks on underlying table [[sub]partition] object. The default retries up to an internal retry limit, after which the lock get will give up. If `LOCK_NOWAIT` is specified, then the lock get does not retry.

REBUILD_FREELISTS Procedure

This procedure rebuilds the freelists for the specified object. All free blocks are placed on the master freelist. All other freelists are zeroed.

If the object has multiple freelist groups, then the free blocks are distributed among all freelists, allocating to the different groups in round-robin fashion.

Syntax

```
DBMS_REPAIR.REBUILD_FREELISTS (
   schema_name    IN VARCHAR2,
   object_name    IN  VARCHAR2,
   partition_name IN VARCHAR2 DEFAULT NULL,
   object_type    IN BINARY_INTEGER DEFAULT TABLE_OBJECT);
```

Parameters

Table 79–9 REBUILD_FREELISTS Procedure Parameters

Parameter	Description
schema_name	Schema name.
object_name	Name of the object whose freelists are to be rebuilt.
partition_name	Partition or subpartition name whose freelists are to be rebuilt. If this is a partitioned object, and partition_name is not specified, then all partitions and subpartitions are processed. If this is a partitioned object, and the specified partition contains subpartitions, then all subpartitions are processed.
object_type	Type of the object to be processed. This must be either TABLE_OBJECT (default) or INDEX_OBJECT. See"Constants" on page 79-5.

SEGMENT_FIX_STATUS Procedure

With this procedure you can fix the corrupted state of a bitmap entry. The procedure either recalculates the state based on the current contents of the corresponding block or sets the state to a specific value.

Syntax

```
DBMS_REPAIR.SEGMENT_FIX_STATUS (
   segment_owner   IN VARCHAR2,
   segment_name    IN VARCHAR2,
   segment_type    IN BINARY_INTEGER DEFAULT TABLE_OBJECT,
   file_number     IN BINARY_INTEGER DEFAULT NULL,
   block_number    IN BINARY_INTEGER DEFAULT NULL,
   status_value    IN BINARY_INTEGER DEFAULT NULL,
   partition_name  IN VARCHAR2 DEFAULT NULL,);
```

Parameters

Table 79–10 SEGMENT_FIX_STATUS Procedure Parameters

Parameter	Description
schema_owner	Schema name of the segment.
segment_name	Segment name.
partition_name	Optional. Name of an individual partition. NULL for nonpartitioned objects. Default is NULL.
segment_type	Optional Type of the segment (for example, TABLE_OBJECT or INDEX_OBJECT). Default is NULL.
file_number	(optional) The tablespace-relative file number of the data block whose status has to be fixed. If omitted, all the blocks in the segment will be checked for state correctness and fixed.
block_number	(optional) The file-relative block number of the data block whose status has to be fixed. If omitted, all the blocks in the segment will be checked for state correctness and fixed.
status_value	(optional) The value to which the block status described by the file_number and block_number will be set. If omitted, the status will be set based on the current state of the block. This is almost always the case, but if there is a bug in the calculation algorithm, the value can be set manually. Status values: ■ 1 = block is full ■ 2 = block is 0-25% free ■ 3 = block is 25-50% free ■ 4 = block is 50-75% free ■ 5 = block is 75-100% free The status for bitmap blocks, segment headers, and extent map blocks cannot be altered. The status for blocks in a fixed hash area cannot be altered. For index blocks, there are only two possible states: 1 = block is full and 3 = block has free space.

SKIP_CORRUPT_BLOCKS Procedure

This procedure enables or disables the skipping of corrupt blocks during index and table scans of the specified object.

When the object is a table, skip applies to the table and its indexes. When the object is a cluster, it applies to all of the tables in the cluster, and their respective indexes.

> **Note:** When Oracle performs an index range scan on a corrupt index after DBMS_REPAIR.SKIP_CORRUPT_BLOCKS has been set for the base table, corrupt branch blocks and root blocks are not skipped. Only corrupt non-root leaf blocks are skipped.

Syntax

```
DBMS_REPAIR.SKIP_CORRUPT_BLOCKS (
   schema_name  IN VARCHAR2,
   object_name  IN VARCHAR2,
   object_type  IN BINARY_INTEGER DEFAULT TABLE_OBJECT,
   flags        IN BINARY_INTEGER DEFAULT SKIP_FLAG);
```

Parameters

Table 79–11 SKIP_CORRUPT_BLOCKS Procedure Parameters

Parameter	Description
`schema_name`	Schema name of the object to be processed.
`object_name`	Name of the object.
`object_type`	Type of the object to be processed. This must be either `TABLE_OBJECT` (default) or `CLUSTER_OBJECT`. See "Constants" on page 79-5.
`flags`	If `SKIP_FLAG` is specified, then it turns on the skip of software corrupt blocks for the object during index and table scans. If `NOSKIP_FLAG` is specified, then scans that encounter software corrupt blocks return an `ORA-1578`. See"Constants" on page 79-5.

80

DBMS_REPCAT

The DBMS_REPCAT package provides routines to administer and update the replication catalog and environment.

- Documentation of DBMS_REPCAT

Documentation of DBMS_REPCAT

For a complete description of this package within the context of Replication, see `DBMS_REPCAT` in the *Oracle Database Advanced Replication Management API Reference.*

81

DBMS_REPCAT_ADMIN

The DBMS_REPCAT_ADMIN package enables you to create users with the privileges needed by the symmetric replication facility.

Documentation of DBMS_REPCAT_ADMIN

For a complete description of this package within the context of Replication, see `DBMS_REPCAT_ADMIN` in the *Oracle Database Advanced Replication Management API Reference*.

82

DBMS_REPCAT_INSTANTIATE

The `DBMS_REPCAT_INSTANTIATE` package instantiates deployment templates.

- Documentation of DBMS_REPCAT_INSTANTIATE

Documentation of DBMS_REPCAT_INSTANTIATE

For a complete description of this package within the context of Replication, see `DBMS_REPCAT_INSTANTIATE` in the *Oracle Database Advanced Replication Management API Reference*.

83

DBMS_REPCAT_RGT

The DBMS_REPCAT_RGT package controls the maintenance and definition of refresh group templates.

- Documentation of DBMS_REPCAT_RGT

Documentation of DBMS_REPCAT_RGT

For a complete description of this package within the context of Replication, see `DBMS_REPCAT_RGT` in the *Oracle Database Advanced Replication Management API Reference*.

84

DBMS_REPUTIL

The `DBMS_REPUTIL` package contains subprograms to generate shadow tables, triggers, and packages for table replication, as well as subprograms to generate wrappers for replication of standalone procedure invocations and packaged procedure invocations. This package is referenced only by the generated code.

Documentation of DBMS_REPUTIL

For a complete description of this package within the context of Replication, see `DBMS_REPUTIL` in the *Oracle Database Advanced Replication Management API Reference*.

85

DBMS_RESOURCE_MANAGER

The DBMS_RESOURCE_MANAGER package maintains plans, consumer groups, and plan directives. It also provides semantics so that you may group together changes to the plan schema.

> **See Also:** For more information on using the Database Resource Manager, see *Oracle Database Administrator's Guide.*

This chapter contains the following topics:

Using DBMS_RESOURCE_MANAGER

Security Model

The invoker must have the `ADMINISTER_RESOURCE_MANAGER` system privilege to execute these procedures. The procedures to grant and revoke this privilege are in the package Chapter 86, "DBMS_RESOURCE_MANAGER_PRIVS".

Constants

Table 85–1 Constants - Names and Oracle Enterprise Manager Abbreviations

Constant	Definition
CLIENT_MACHINE	CONSTANT VARCHAR2(30) := 'CLIENT_MACHINE';
CLIENT_OS_USER	CONSTANT VARCHAR2(30) := 'CLIENT_OS_USER';
CLIENT_PROGRAM	CONSTANT VARCHAR2(30) := 'CLIENT_PROGRAM';
MODULE_NAME	CONSTANT VARCHAR2(30) := 'MODULE_NAME';
MODULE_NAME_ACTION	CONSTANT VARCHAR2(30) := 'MODULE_NAME_ACTION';
ORACLE_USER	CONSTANT VARCHAR2(30) := 'ORACLE_USER'
SERVICE_MODULE	CONSTANT VARCHAR2(30) := 'SERVICE_MODULE';
SERVICE_MODULE_ ACTION	CONSTANT VARCHAR2(30) := 'SERVICE_MODULE_ ACTION';
SERVICE_NAME	CONSTANT VARCHAR2(30) := 'SERVICE_NAME';

Examples

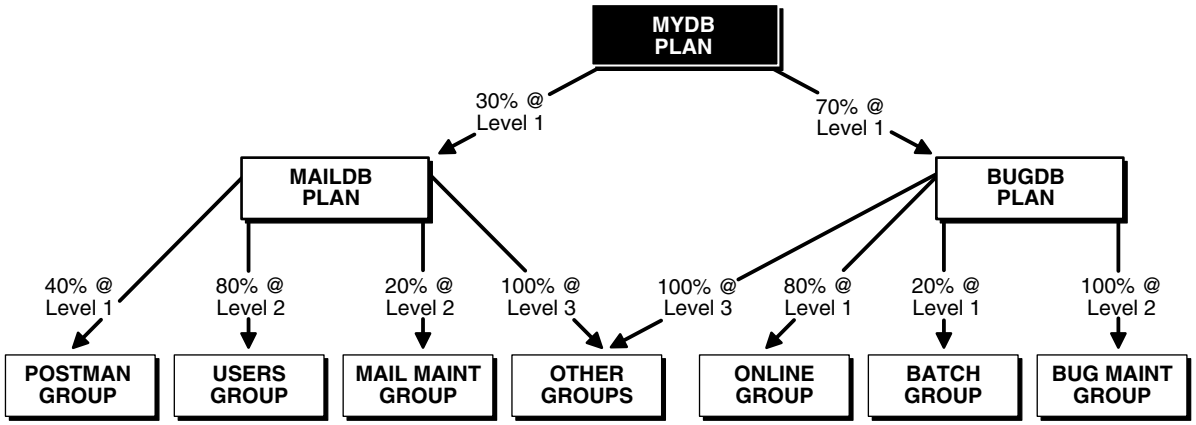
One of the advantages of plans is that they can refer to each other. The entries in a plan can either be consumer groups or subplans. For example, the following is also a set of valid CPU plan directives:

Table 85–2 MYDB PLAN CPU Plan Directives

Subplan/Group	CPU_Level 1
MAILDB Plan	30%
BUGDB Plan	70%

If these plan directives were in effect and there were an infinite number of runnable sessions in all consumer groups, then the MAILDB plan would be assigned 30% of the available CPU resources, while the BUGDB plan would be assigned 70% of the available CPU resources. Breaking this further down, sessions in the "Postman" consumer group would be run 12% (40% of 30%) of the time, while sessions in the "Online" consumer group would be run 56% (80% of 70%) of the time. Figure 85–1 diagram depicts this scenario:

Figure 85–1 Resource Manager Scenario



Conceptually the active sessions are underneath the consumer groups. In other words, a session belongs to a resource consumer group, and this consumer group is used by a plan to determine allocation of processing resources.

A multiplan (plan with one or more subplans) definition of CPU plan directives cannot be collapsed into a single plan with one set of plan directives, because each plan is its own entity. The CPU quanta that is allotted to a plan or subplan gets used only within that plan, unless that plan contains no consumer groups with active sessions. Therefore, in this example, if the Bug Maintenance Group did not use any of its quanta, then it would get recycled within that plan, thus going back to level 1 within the BUGDB PLAN. If the multiplan definition in the preceding example got collapsed into a single plan with multiple consumer groups, then there would be no way to explicitly recycle the Bug Maintenance Group's unused quanta. It would have to be recycled globally, thus giving the mail sessions an opportunity to use it.

The resources for a database can be partitioned at a high level among multiple applications and then repartitioned within an application. If a given group within an

application does not need all the resources it is assigned, then the resource is only repartitioned within the same application.

The following example uses the default plan and consumer group allocation methods:

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'bugdb_plan',
   COMMENT => 'Resource plan/method for bug users sessions');
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'maildb_plan',
   COMMENT => 'Resource plan/method for mail users sessions');
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'mydb_plan',
   COMMENT => 'Resource plan/method for bug and mail users sessions');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Bug_Online_group',
   COMMENT => 'Resource consumer group/method for online bug users sessions');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Bug_Batch_group',
COMMENT => 'Resource consumer group/method for bug users sessions who run batch
jobs');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Bug_Maintenance_
group',
   COMMENT => 'Resource consumer group/method for users sessions who maintain
   the bug  db');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Mail_users_group',
   COMMENT => 'Resource consumer group/method for mail users sessions');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Mail_Postman_
group',
   COMMENT => 'Resource consumer group/method for mail postman');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Mail_Maintenance_
group',
   COMMENT => 'Resource consumer group/method for users sessions who maintain the
mail
   db');
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan', GROUP_OR_SUBPLAN
=> 'Bug_Online_group',
   COMMENT => 'online bug users sessions at level 1', CPU_P1 => 80, CPU_P2=> 0,
   PARALLEL_DEGREE_LIMIT_P1 => 8);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan', GROUP_OR_SUBPLAN
=> 'Bug_Batch_group',
   COMMENT => 'batch bug users sessions at level 1', CPU_P1 => 20, CPU_P2 => 0,
   PARALLEL_DEGREE_LIMIT_P1 => 2);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan', GROUP_OR_SUBPLAN
=> 'Bug_Maintenance_group',
   COMMENT => 'bug maintenance users sessions at level 2', CPU_P1 => 0, CPU_P2 =>
100, PARALLEL_DEGREE_LIMIT_P1 => 3);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan', GROUP_OR_SUBPLAN
=> 'OTHER_GROUPS',
   COMMENT => 'all other users sessions at level 3', CPU_P1 => 0, CPU_P2 => 0,
CPU_P3 => 100);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan', GROUP_OR_
SUBPLAN => 'Mail_Postman_group',
   COMMENT => 'mail postman at level 1', CPU_P1 => 40, CPU_P2 => 0,
   PARALLEL_DEGREE_LIMIT_P1 => 4);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan', GROUP_OR_
SUBPLAN => 'Mail_users_group',
   COMMENT => 'mail users sessions at level 2', CPU_P1 => 0, CPU_P2 => 80,
   PARALLEL_DEGREE_LIMIT_P1 => 4);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan', GROUP_OR_
SUBPLAN => 'Mail_Maintenance_group',
   COMMENT => 'mail maintenance users sessions at level 2', CPU_P1 => 0, CPU_P2 =>
20,
   PARALLEL_DEGREE_LIMIT_P1 => 2);
```

```
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan', GROUP_OR_
SUBPLAN => 'OTHER_GROUPS',
   COMMENT => 'all other users sessions at level 3', CPU_P1 => 0, CPU_P2 => 0,
CPU_P3 => 100);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'mydb_plan', GROUP_OR_SUBPLAN
=> 'maildb_plan',
   COMMENT=> 'all mail users sessions at level 1', CPU_P1 => 30);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'mydb_plan', GROUP_OR_SUBPLAN
=> 'bugdb_plan',
   COMMENT => 'all bug users sessions at level 1', CPU_P1 => 70);
DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;

-- The preceding call to VALIDATE_PENDING_AREA
-- is optional, because the validation is implicitly done in SUBMIT_PENDING_AREA.

BEGIN
DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();

DBMS_RESOURCE_MANAGER.CREATE_PLAN(
 PLAN => 'bugdb_plan',
 COMMENT => 'Resource plan/method for bug users sessions');
DBMS_RESOURCE_MANAGER.CREATE_PLAN(
 PLAN => 'maildb_plan',
 COMMENT => 'Resource plan/method for mail users sessions');

DBMS_RESOURCE_MANAGER.CREATE_PLAN(
 PLAN => 'mydb_plan',
 COMMENT => 'Resource plan/method for bug and mail users sessions');

DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(
 CONSUMER_GROUP => 'Bug_Online_group',
 COMMENT => 'Resource consumer group/method for online bug users sessions');

DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(
 CONSUMER_GROUP => 'Bug_Batch_group',
 COMMENT => 'Resource consumer group/method for bug users sessions who run
batch jobs');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(
 CONSUMER_GROUP => 'Bug_Maintenance_group',
 COMMENT => 'Resource consumer group/method for users sessions who maintain
the bug db');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(
 CONSUMER_GROUP => 'Mail_users_group',
 COMMENT => 'Resource consumer group/method for mail users sessions');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(
 CONSUMER_GROUP => 'Mail_Postman_group',
 COMMENT => 'Resource consumer group/method for mail postman');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(
 CONSUMER_GROUP => 'Mail_Maintenance_group',
 COMMENT => 'Resource consumer group/method for users sessions who maintain
the mail
db');

DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(
 PLAN => 'bugdb_plan', GROUP_OR_SUBPLAN => 'Bug_Online_group',
 COMMENT => 'online bug users sessions at level 1',
 CPU_P1 => 80, CPU_P2=> 0,
 PARALLEL_DEGREE_LIMIT_P1 => 8);
```

```
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(
 PLAN => 'bugdb_plan', GROUP_OR_SUBPLAN => 'Bug_Batch_group',
 COMMENT => 'batch bug users sessions at level 1',
 CPU_P1 => 20, CPU_P2 => 0,
 PARALLEL_DEGREE_LIMIT_P1 => 2);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(
 PLAN => 'bugdb_plan', GROUP_OR_SUBPLAN => 'Bug_Maintenance_group',
 COMMENT => 'bug maintenance users sessions at level 2',
 CPU_P1 => 0, CPU_P2 => 100,
 PARALLEL_DEGREE_LIMIT_P1 => 3);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(
 PLAN => 'bugdb_plan', GROUP_OR_SUBPLAN => 'OTHER_GROUPS',
 COMMENT => 'all other users sessions at level 3',
 CPU_P1 => 0, CPU_P2 => 0, CPU_P3 => 100);

DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(
 PLAN => 'maildb_plan', GROUP_OR_SUBPLAN => 'Mail_Postman_group',
 COMMENT => 'mail postman at level 1',
 CPU_P1 => 40, CPU_P2 => 0,
 PARALLEL_DEGREE_LIMIT_P1 => 4);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(
 PLAN => 'maildb_plan', GROUP_OR_SUBPLAN => 'Mail_users_group',
 COMMENT => 'mail users sessions at level 2',
 CPU_P1 => 0, CPU_P2 => 80,
 PARALLEL_DEGREE_LIMIT_P1 => 4);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(
 PLAN => 'maildb_plan', GROUP_OR_SUBPLAN => 'Mail_Maintenance_group',
 COMMENT => 'mail maintenance users sessions at level 2',
 CPU_P1 => 0, CPU_P2 => 20,
 PARALLEL_DEGREE_LIMIT_P1 => 2);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(
 PLAN => 'maildb_plan', GROUP_OR_SUBPLAN => 'OTHER_GROUPS',
 COMMENT => 'all other users sessions at level 3',
 CPU_P1 => 0, CPU_P2 => 0, CPU_P3 => 100);

DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(
 PLAN => 'mydb_plan', GROUP_OR_SUBPLAN => 'maildb_plan',
 COMMENT=> 'all mail users sessions at level 1',
 CPU_P1 => 30);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(
 PLAN => 'mydb_plan', GROUP_OR_SUBPLAN => 'bugdb_plan',
 COMMENT => 'all bug users sessions at level 1',
 CPU_P1 => 70);

DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
/
```

Summary of DBMS_RESOURCE_MANAGER Subprograms

Table 85–3 DBMS_RESOURCE_MANAGER Package Subprograms

Subprogram	Description
CLEAR_PENDING_AREA Procedure on page 85-11	Clears the work area for the resource manager
CREATE_CONSUMER_ GROUP Procedure on page 85-12	Creates entries which define resource consumer groups
CREATE_PENDING_AREA Procedure on page 85-13	Creates a work area for changes to resource manager objects
CREATE_PLAN Procedure on page 85-15	Creates entries which define resource plans
CREATE_PLAN_DIRECTIVE Procedure on page 85-16	Creates resource plan directives
CREATE_SIMPLE_PLAN Procedure on page 85-18	Creates a single-level resource plan containing up to eight consumer groups in one step
DELETE_CONSUMER_ GROUP Procedure on page 85-19	Deletes entries which define resource consumer groups
DELETE_PLAN Procedure on page 85-20	Deletes the specified plan as well as all the plan directives it refers to
DELETE_PLAN_CASCADE Procedure on page 85-21	Deletes the specified plan as well as all its descendants (plan directives, subplans, consumer groups)
DELETE_PLAN_DIRECTIVE Procedure on page 85-22	Deletes resource plan directives
SET_CONSUMER_GROUP_ MAPPING Procedure on page 85-23	Adds, deletes, or modifies pairs for the login and run-time attribute mappings
SET_CONSUMER_GROUP_ MAPPING_PRI Procedure on page 85-24	Creates the session attribute mapping priority list
SET_INITIAL_CONSUMER_ GROUP Procedure on page 85-25	Assigns the initial resource consumer group for a user
SUBMIT_PENDING_AREA Procedure on page 85-26	Submits pending changes for the resource manager
SWITCH_CONSUMER_ GROUP_FOR_SESS Procedure on page 85-27	Changes the resource consumer group of a specific session
SWITCH_CONSUMER_ GROUP_FOR_USER Procedure on page 85-28	Changes the resource consumer group for all sessions with a given user name
SWITCH_PLAN Procedure on page 85-29	Sets the current resource manager plan
UPDATE_CONSUMER_ GROUP Procedure on page 85-30	Updates entries which define resource consumer groups

Table 85–3 (Cont.) DBMS_RESOURCE_MANAGER Package Subprograms

Subprogram	Description
UPDATE_PLAN Procedure on page 85-31	Updates entries which define resource plans
UPDATE_PLAN_DIRECTIVE Procedure on page 85-32	Updates resource plan directives
VALIDATE_PENDING_AREA Procedure on page 85-34	Validates pending changes for the resource manage

CLEAR_PENDING_AREA Procedure

This procedure lets you clear pending changes for the resource manager.

Syntax

```
DBMS_RESOURCE_MANAGER.CLEAR_PENDING_AREA;
```

CREATE_CONSUMER_GROUP Procedure

This procedure lets you create entries which define resource consumer groups.

Syntax

```
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
   consumer_group  IN VARCHAR2,
   comment         IN VARCHAR2,
   cpu_mth         IN VARCHAR2 DEFAULT 'ROUND-ROBIN');
```

Parameters

Table 85–4 CREATE_CONSUMER_GROUP Procedure Parameters

Parameter	Description
consumer_group	The name of the consumer group.
comment	The user's comment.
cpu_mth	The resource allocation method for distributing CPU among sessions in the consumer group. The default is ROUND-ROBIN, which uses a round-robin scheduler to ensure sessions are fairly executed. RUN-TO-COMPLETION specifies that sessions with the largest active time are scheduled ahead of other sessions

CREATE_PENDING_AREA Procedure

This procedure lets you make changes to resource manager objects.

All changes to the plan schema must be done within a pending area. The pending area can be thought of as a "scratch" area for plan schema changes. The administrator creates this pending area, makes changes as necessary, possibly validates these changes, and only when the submit is completed do these changes become active.

Syntax

```
DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA;
```

Usage Notes

You may, at any time while the pending area is active, view the current plan schema with your changes by selecting from the appropriate user views.

At any time, you may clear the pending area if you want to stop the current changes. You may also call the `VALIDATE` procedure to confirm whether the changes you has made are valid. You do not have to do your changes in a given order to maintain a consistent group of entries. These checks are also implicitly done when the pending area is submitted.

> **Note:** Oracle allows "orphan" consumer groups (in other words, consumer groups that have no plan directives that refer to them). This is in anticipation that an administrator may want to create a consumer group that is not currently being used, but will be used in the future.

The following rules must be adhered to, and they are checked whenever the validate or submit procedures are executed:

- No plan schema may contain any loops.
- All plans and consumer groups referred to by plan directives must exist.
- All plans must have plan directives that refer to either plans or consumer groups.
- All percentages in any given level must not add up to greater than 100 for the emphasis resource allocation method.
- No plan may be deleted that is currently being used as a top plan by an active instance.
- For Oracle8*i*, the plan directive parameter, `parallel_degree_limit_p1`, may only appear in plan directives that refer to consumer groups (that is, not at subplans).
- There cannot be more than 32 plan directives coming from any given plan (that is, no plan can have more than 32 children).
- There cannot be more than 32 consumer groups in any active plan schema.
- Plans and consumer groups use the same namespace; therefore, no plan can have the same name as any consumer group.
- There must be a plan directive for `OTHER_GROUPS` somewhere in any active plan schema.This ensures that a session not covered by the currently active plan is allocated resources as specified by the `OTHER_GROUPS` directive.

If any of the preceding rules are broken when checked by the `VALIDATE` or `SUBMIT` procedures, then an informative error message is returned. You may then make changes to fix the problem(s) and reissue the validate or submit procedures.

CREATE_PLAN Procedure

This procedure creates entries which define resource plans.

Syntax

```
DBMS_RESOURCE_MANAGER.CREATE_PLAN (
   plan                      IN VARCHAR2,
   comment                   IN VARCHAR2,
   cpu_mth                   IN VARCHAR2 DEFAULT 'EMPHASIS',
   active_sess_pool_mth       IN VARCHAR2 DEFAULT 'ACTIVE_SESS_POOL_ABSOLUTE',
   parallel_degree_limit_mth  IN VARCHAR2 DEFAULT
                             'PARALLEL_DEGREE_LIMIT_ABSOLUTE',
   queueing_mth              IN VARCHAR2 DEFAULT 'FIFO_TIMEOUT',);
```

Parameters

Table 85–5 CREATE_PLAN Procedure Parameters

Parameter	Description
plan	The name of the resource plan.
comment	User's comment.
cpu_mth	The resource allocation method for specifying how much CPU each consumer group or sub plan gets. EMPHASIS, the default method, is for multilevel plans that use percentages to specify how CPU is distributed among consumer groups. RATIO is for single-level plans that use ratios to specify how CPU is distributed.
active_sess_pool_mth	The Active session pool resource allocation method. Limits the number of active sessions. All other sessions are inactive and wait in a queue to be activated. ACTIVE_SESS_POOL_ABSOLUTE is the default and only method available.
parallel_degree_limit_mth	The resource allocation method for specifying a limit on the degree of parallelism of any operation. PARALLEL_DEGREE_LIMIT_ABSOLUTE is the default and only method available.
queueing_mth	The Queuing resource allocation method. Controls order in which queued inactive sessions will execute. FIFO_TIMEOUT is the default and only method available

Usage Notes

If you want to use any default resource allocation method, then you do not need not specify it when creating or updating a plan.

CREATE_PLAN_DIRECTIVE Procedure

This procedure lets you create resource plan directives.

Syntax

```
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
   plan                      IN VARCHAR2,
   group_or_subplan          IN VARCHAR2,
   comment                   IN VARCHAR2,
   cpu_p1                    IN NUMBER   DEFAULT NULL,
   cpu_p2                    IN NUMBER   DEFAULT NULL,
   cpu_p3                    IN NUMBER   DEFAULT NULL,
   cpu_p4                    IN NUMBER   DEFAULT NULL,
   cpu_p5                    IN NUMBER   DEFAULT NULL,
   cpu_p6                    IN NUMBER   DEFAULT NULL,
   cpu_p7                    IN NUMBER   DEFAULT NULL,
   cpu_p8                    IN NUMBER   DEFAULT NULL,
   active_sess_pool_p1       IN NUMBER   DEFAULT NULL,
   queueing_p1               IN NUMBER   DEFAULT NULL,
   parallel_degree_limit_p1  IN NUMBER   DEFAULT NULL,
   switch_group              IN VARCHAR2 DEFAULT NULL,
   switch_time               IN NUMBER   DEFAULT NULL,
   switch_estimate           IN BOOLEAN  DEFAULT FALSE,
   max_est_exec_time         IN NUMBER   DEFAULT NULL,
   undo_pool                 IN NUMBER   DEFAULT NULL,
   max_idle_time             IN NUMBER   DEFAULT NULL,
               max_idle_blocker_time     IN NUMBER   DEFAULT NULL,
               switch_time_in_call       IN NUMBER   DEFAULT NULL);
```

Parameters

Table 85–6 CREATE_PLAN_DIRECTIVE Procedure Parameters

Parameter	Description
plan	The name of the resource plan.
group_or_subplan	The name of the consumer group or subplan.
comment	Comment for the plan directive.
cpu_p1	For EMPHASIS, specifies the CPU percentage at the first level. For RATIO, specifies the weight of CPU usage. Default is NULL for all CPU parameters.
cpu_p2	For EMPHASIS, specifies the CPU percentage at the second level. Not applicable for RATIO.
cpu_p3	For EMPHASIS, specifies the CPU percentage at the third level. Not applicable for RATIO.
cpu_p4	For EMPHASIS, specifies the CPU percentage at the fourth level. Not applicable for RATIO.
cpu_p5	For EMPHASIS, specifies the CPU percentage at the fifth level. Not applicable for RATIO.
cpu_p6	For EMPHASIS, specifies the CPU percentage at the sixth level. Not applicable for RATIO.
cpu_p7	For EMPHASIS, specifies the CPU percentage at the seventh level. Not applicable for RATIO.

Table 85–6 (Cont.) CREATE_PLAN_DIRECTIVE Procedure Parameters

Parameter	Description
cpu_p8	For EMPHASIS, specifies the CPU percentage at the eighth level. Not applicable for RATIO.
active_sess_pool_p1	Specifies maximum number of concurrently active sessions for a consumer group. Default is NULL, which means unlimited.
queueing_p1	Specified time (in seconds) after which a job in the inactive session queue (waiting for execution) will time out. Default is NULL, which means unlimited.
parallel_degree_ limit_p1	Specifies a limit on the degree of parallelism for any operation. Default is NULL, which means unlimited.
switch_group	Specifies consumer group to which this session is switched if other switch criteria is met. Default is NULL. If the group name is 'CANCEL_SQL', the current call will be canceled when other switch criteria are met. If the group name is 'KILL_SESSION', the session will be killed when other switch criteria are met.
switch_time	Specifies time (in seconds) that a session can execute before an action is taken. Default is NULL, which means unlimited.
switch_estimate	If TRUE, tells Oracle to use its execution time estimate to automatically switch the consumer group of an operation before beginning its execution. Default is FALSE.
max_est_exec_time	Specifies the maximum execution time (in seconds) allowed for a session. If the optimizer estimates that an operation will take longer than MAX_EST_EXEC_TIME, the operation is not started and ORA-07455 is issued. If the optimizer does not provide an estimate, this directive has no effect. Default is NULL, which means unlimited.
undo_pool	Sets a maximum in kilobytes (K) on the total amount of undo generated by a consumer group. Default is NULL, which means unlimited.
max_idle_time	Indicates the maximum session idle time. Default is NULL, which means unlimited.
max_idle_blocker_ time	The maximum amount of time in seconds that a session can be idle while blocking another session's acquisition of a resource.
switch_time_in_call	Specifies time (in seconds) that a session can execute before an action is taken. At the end of the top call, the consumer group of the session is restored to its original consumer group. Default is NULL, which means unlimited. Both SWITCH_TIME_ IN_CALL and SWITCH_TIME cannot be specified.

Usage Notes

- All parameters default to NULL. However, for the EMPHASIS CPU resource allocation method, this case would starve all the users.
- For max_idle_time and max_idle_blocker_time, PMON will check these limits once a minute. If it finds a session that has exceeded one of the limits, it will forcibly kill the session and clean up all its state.
- The parameter switch_time_in_call is mostly useful for three-tier applications where the mid-tier server is implementing session pooling. By using switch_time_in_call, the resource usage of one client will not affect a future client that happens to be executed on the same session.

CREATE_SIMPLE_PLAN Procedure

This procedure creates a single-level resource plan containing up to eight consumer groups in one step. You do not need to create a pending area manually before creating a resource plan, or use the CREATE_CONSUMER_GROUP and CREATE_RESOURCE_PLAN_DIRECTIVES procedures separately.

Syntax

```
DBMS_RESOURCE_MANAGER.CREATE_SIMPLE_PLAN (
   SIMPLE_PLAN      IN  VARCHAR2  DEFAULT,
   CONSUMER_GROUP1  IN  VARCHAR2  DEFAULT,
   GROUP1_CPU       IN  NUMBER    DEFAULT,
   CONSUMER_GROUP2  IN  VARCHAR2  DEFAULT,
   GROUP2_CPU       IN  NUMBER    DEFAULT,
   CONSUMER_GROUP3  IN  VARCHAR2  DEFAULT,
   GROUP3_CPU       IN  NUMBER    DEFAULT,
   CONSUMER_GROUP4  IN  VARCHAR2  DEFAULT,
   GROUP4_CPU       IN  NUMBER    DEFAULT,
   CONSUMER_GROUP5  IN  VARCHAR2  DEFAULT,
   GROUP5_CPU       IN  NUMBER    DEFAULT,
   CONSUMER_GROUP6  IN  VARCHAR2  DEFAULT,
   GROUP6_CPU       IN  NUMBER    DEFAULT,
   CONSUMER_GROUP7  IN  VARCHAR2  DEFAULT,
   GROUP7_CPU       IN  NUMBER    DEFAULT,
   CONSUMER_GROUP8  IN  VARCHAR2  DEFAULT,
   GROUP8_CPU       IN  NUMBER    DEFAULT);
```

DELETE_CONSUMER_GROUP Procedure

This procedure lets you delete entries which define resource consumer groups.

Syntax

```
DBMS_RESOURCE_MANAGER.DELETE_CONSUMER_GROUP (
   consumer_group IN VARCHAR2);
```

Parameters

Table 85–7 DELETE_CONSUMER_GROUP Procedure Parameters

Parameters	Description
consumer_group	The name of the consumer group to be deleted.

DELETE_PLAN Procedure

This procedure deletes the specified plan as well as all the plan directives to which it refers.

Syntax

```
DBMS_RESOURCE_MANAGER.DELETE_PLAN (
   plan IN VARCHAR2);
```

Parameters

Table 85–8 DELETE_PLAN Procedure Parameters

Parameter	Description
plan	The name of the resource plan to delete.

DELETE_PLAN_CASCADE Procedure

This procedure deletes the specified plan and all of its descendants (plan directives, subplans, consumer groups). Mandatory objects and directives are not deleted.

Syntax

```
DBMS_RESOURCE_MANAGER.DELETE_PLAN_CASCADE (
   plan IN VARCHAR2);
```

Parameters

Table 85–9 DELETE_PLAN_CASCADE Procedure Parameters

Parameters	Description
plan	The name of the plan.

Usage Notes

If DELETE_PLAN_CASCADE encounters any error, then it rolls back, and nothing is deleted.

DELETE_PLAN_DIRECTIVE Procedure

This procedure lets you delete resource plan directives.

Syntax

```
DBMS_RESOURCE_MANAGER.DELETE_PLAN_DIRECTIVE (
   plan              IN VARCHAR2,
   group_or_subplan  IN VARCHAR2);
```

Parameters

Table 85–10 DELETE_PLAN_DIRECTIVE Procedure Parameters

Parameter	Description
`plan`	The name of the resource plan.
`group_or_subplan`	The name of the group or subplan.

SET_CONSUMER_GROUP_MAPPING Procedure

This procedure adds, deletes, or modifies entries that map sessions to consumer groups, based on the session's login and runtime attributes.

Syntax

```
DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING(
   attribute       IN VARCHAR2,
   value            IN VARCHAR2,
   consumer_group  IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 85–11 SET_CONSUMER_GROUP_MAPPING Procedure Parameters

Parameters	Description
`attribute`	The mapping attribute to add/modify. It can be one of the Constants listed.
`value`	The attribute value to match.
`consumer_group`	The name of the mapped consumer group, or `NULL` to delete a mapping.

Usage Notes

If no mapping exists for the given attribute and value, a mapping to the given consumer group will be created. If a mapping already exists for the given attribute and value, the mapped consumer group will be updated to the one given. If the `consumer_group` argument is `NULL`, then any mapping from the given attribute and value will be deleted.

SET_CONSUMER_GROUP_MAPPING_PRI Procedure

Multiple attributes of a session can be used to map the session to a consumer group. This procedure prioritizes the attribute mappings.

Syntax

```
DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING_PRI(
   explicit               IN NUMBER,
   oracle_user            IN NUMBER,
   service_name           IN NUMBER,
   client_os_user         IN NUMBER,
   client_program         IN NUMBER,
   client_machine         IN NUMBER,
   module_name            IN NUMBER,
   module_name_action     IN NUMBER,
   service_module         IN NUMBER,
   service_module_action  IN NUMBER);
```

Parameters

Table 85–12 SET_CONSUMER_GROUP_MAPPING_PRI Procedure Parameters

Parameters	Description
`explicit`	The priority of the explicit mapping.
`oracle_user`	The priority of the Oracle user name mapping.
`service_name`	The priority of the client service name mapping.
`client_os_user`	The priority of the client operating system user name mapping.
`client_program`	The priority of the client program mapping.
`client_machine`	The priority of the client machine mapping.
`module_name`	The priority of the application module name mapping.
`module_name_action`	The priority of the application module name and action mapping.
`service_module`	The priority of the service name and application module name mapping.
`module_name_action`	The priority of the service name, application module name, and application action mapping.

Usage Notes

- This procedure requires that you include the pseudo-attribute `explicit` as an argument. It must be set to 1. It indicates that explicit consumer group switches have the highest priority. You explicitly switch consumer groups with these package procedures:
  - `DBMS_SESSION.SWITCH_CURRENT_CONSUMER_GROUP`
  - `DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_SESS`
  - `DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_USER`
- Each priority value must be a unique integer from 1 to 10. Together, they establish an ordering where 1 is the highest priority and 10 is the lowest.

SET_INITIAL_CONSUMER_GROUP Procedure

The initial consumer group of a user is the consumer group to which any session created by that user initially belongs. This procedure sets the initial resource consumer group for a user.

Syntax

```
DBMS_RESOURCE_MANAGER.SET_INITIAL_CONSUMER_GROUP (
   user           IN VARCHAR2,
   consumer_group IN VARCHAR2);
```

Parameters

Table 85–13 SET_INITIAL_CONSUMER_GROUP Procedure Parameters

Parameters	Description
user	The name of the user.
consumer_group	The user's initial consumer group.

Usage Notes

The ADMINISTER_RESOURCE_MANAGER or the ALTER USER system privilege are required to be able to execute this procedure. The user, or PUBLIC, must be directly granted switch privilege to a consumer group before it can be set to be the user's initial consumer group. Switch privilege for the initial consumer group cannot come from a role granted to that user.

Note: These semantics are similar to those for ALTER USER DEFAULT ROLE.

If the initial consumer group for a user has never been set, then the user's initial consumer group is automatically the consumer group: DEFAULT_CONSUMER_GROUP.

DEFAULT_CONSUMER_GROUP has switch privileges granted to PUBLIC; therefore, all users are automatically granted switch privilege for this consumer group. Upon deletion of a consumer group, all users having the deleted group as their initial consumer group now have DEFAULT_CONSUMER_GROUP as their initial consumer group. All currently active sessions belonging to a deleted consumer group are switched to DEFAULT_CONSUMER_GROUP.

SUBMIT_PENDING_AREA Procedure

This procedure lets you submit pending changes for the resource manager. It clears the pending area after validating and committing the changes (if valid).

> **Note:** A call to SUBMIT_PENDING_AREA may fail even if VALIDATE_PENDING_AREA succeeds. This may happen if a plan being deleted is loaded by an instance after a call to VALIDATE_PENDING_AREA, but before a call to SUBMIT_PENDING_AREA.

Syntax

```
DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA;
```

SWITCH_CONSUMER_GROUP_FOR_SESS Procedure

This procedure lets you change the resource consumer group of a specific session. It also changes the consumer group of any (PQ) slave sessions that are related to the top user session.

Syntax

```
DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_SESS (
   session_id      IN NUMBER,
   session_serial  IN NUMBER,
   consumer_group  IN VARCHAR2);
```

Parameters

Table 85–14 SWITCH_CONSUMER_GROUP_FOR_SESS Procedure Parameters

Parameter	Description
session_id	SID column from the view V$SESSION.
session_serial	SERIAL# column from view V$SESSION.
consumer_group	The name of the consumer group to switch to.

SWITCH_CONSUMER_GROUP_FOR_USER Procedure

This procedure lets you change the resource consumer group for all sessions with a given user ID. It also change the consumer group of any (PQ) slave sessions that are related to the top user session.

Syntax

```
DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_USER (
   user            IN VARCHAR2,
   consumer_group  IN VARCHAR2);
```

Parameters

Table 85–15 SWITCH_CONSUMER_GROUP_FOR_USER Procedure Parameters

Parameter	Description
`user`	The name of the user.
`consumer_group`	The name of the consumer group to switch to.

Usage Notes

The SWITCH_CONSUMER_GROUP_FOR_SESS Procedure and `SWITCH_CONSUMER_GROUP_FOR_USER` procedures let you to raise or lower the allocation of CPU resources of certain sessions or users. This provides a functionality similar to the `nice` command on UNIX.

These procedures cause the session to be moved into the newly specified consumer group immediately.

SWITCH_PLAN Procedure

This procedure sets the current resource manager plan.

Syntax

```
DBMS_RESOURCE_MANAGER.SWITCH_PLAN(
   plan_name                     IN   VARCHAR2,
   sid                           IN   VARCHAR2 DEFAULT '*',
   allow_scheduler_plan_switches IN   BOOLEAN DEFAULT TRUE);
```

Parameters

Table 85–16 SWITCH_PLAN Procedure Parameters

Parameter	Description
plan_name	The name of the plan to which to switch. Passing in an empty string ('') for the plan_name, disables the resource manager
sid	The sid parameter is relevant only in a Real Application Clusters environment. This parameter lets you change the plan for a particular instance. Specify the sid of the instance where you want to change the plan. Or specify '*' if you want Oracle to change the plan for all instances.
allow_scheduler_ plan_switches	FALSE - disables automated plan switches by the job scheduler at window boundaries. To re-enable automated plan switches, switch_plan must be called again by the administrator with allow_scheduler_plan_switches set to TRUE. By default automated plan switches by the job scheduler are enabled.

UPDATE_CONSUMER_GROUP Procedure

This procedure lets you update entries which define resource consumer groups.

Syntax

```
DBMS_RESOURCE_MANAGER.UPDATE_CONSUMER_GROUP (
   consumer_group  IN VARCHAR2,
   new_comment     IN VARCHAR2 DEFAULT NULL,
   new_cpu_mth     IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 85–17 UPDATE_CONSUMER_GROUP Procedure Parameter

Parameter	Description
consumer_group	The name of consumer group.
new_comment	New user's comment.
new_cpu_mth	The name of new method for CPU resource allocation.

Usage Notes

If the parameters to the UPDATE_CONSUMER_GROUP procedure are not specified, then they remain unchanged in the data dictionary.

UPDATE_PLAN Procedure

This procedure updates entries which define resource plans.

Syntax

```
DBMS_RESOURCE_MANAGER.UPDATE_PLAN (
   plan                                IN VARCHAR2,
   new_comment                         IN VARCHAR2 DEFAULT NULL,
   new_cpu_mth                         IN VARCHAR2 DEFAULT NULL,
   new_active_sess_pool_mth            IN VARCHAR2 DEFAULT NULL,
   new_parallel_degree_limit_mth       IN VARCHAR2 DEFAULT NULL,
   new_queueing_mth                    IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 85–18 UPDATE_PLAN Procedure Parameters

Parameter	Description
plan	The name of resource plan.
new_comment	New user's comment.
new_cpu_mth	The name of new allocation method for CPU resources.
new_active_sess_ pool_mth	The name of new method for maximum active sessions.
new_parallel_degree_ limit_mth	The name of new method for degree of parallelism.
new_queueing_mth	Specifies type of queuing policy to use with active session pool feature.

Usage Notes

- If the parameters to UPDATE_PLAN Procedure are not specified, then they remain unchanged in the data dictionary.
- If you want to use any default resource allocation method, then you do not need not specify it when creating or updating a plan.

UPDATE_PLAN_DIRECTIVE Procedure

This procedure lets you update resource plan directives.

Syntax

```
DBMS_RESOURCE_MANAGER.UPDATE_PLAN_DIRECTIVE (
   plan                          IN VARCHAR2,
   group_or_subplan              IN VARCHAR2,
   new_comment                   IN VARCHAR2 DEFAULT NULL,
   new_cpu_p1                    IN NUMBER   DEFAULT NULL,
   new_cpu_p2                    IN NUMBER   DEFAULT NULL,
   new_cpu_p3                    IN NUMBER   DEFAULT NULL,
   new_cpu_p4                    IN NUMBER   DEFAULT NULL,
   new_cpu_p5                    IN NUMBER   DEFAULT NULL,
   new_cpu_p6                    IN NUMBER   DEFAULT NULL,
   new_cpu_p7                    IN NUMBER   DEFAULT NULL,
   new_cpu_p8                    IN NUMBER   DEFAULT NULL,
   new_active_sess_pool_p1       IN NUMBER   DEFAULT NULL,
   new_queueing_p1               IN NUMBER   DEFAULT NULL,
   new_parallel_degree_limit_p1  IN NUMBER   DEFAULT NULL,
   new_switch_group              IN VARCHAR2 DEFAULT NULL,
   new_switch_time               IN NUMBER   DEFAULT NULL,
   new_switch_estimate           IN BOOLEAN  DEFAULT FALSE,
   new_max_est_exec_time         IN NUMBER   DEFAULT NULL,
   new_undo_pool                 IN NUMBER   DEFAULT NULL,
   new_max_idle_time             IN NUMBER   DEFAULT NULL,
   new_max_idle_blocker_time     IN NUMBER   DEFAULT NULL,
   new_switch_time_in_call       IN NUMBER   DEFAULT NULL);
```

Parameters

Table 85–19 UPDATE_PLAN_DIRECTIVE Procedure Parameters

Parameter	Description
plan	The name of the resource plan.
group_or_subplan	The name of the consumer group or subplan.
new_comment	Comment for the plan directive.
new_cpu_p1	For EMPHASIS, specifies the CPU percentage at the first level. For RATIO, specifies the weight of CPU usage. Default is NULL for all CPU parameters.
new_cpu_p2	For EMPHASIS, specifies the CPU percentage at the second level. Not applicable for RATIO.
new_cpu_p3	For EMPHASIS, specifies the CPU percentage at the third level. Not applicable for RATIO.
new_cpu_p4	For EMPHASIS, specifies the CPU percentage at the fourth level. Not applicable for RATIO.
new_cpu_p5	For EMPHASIS, specifies the CPU percentage at the fifth level. Not applicable for RATIO.
new_cpu_p6	For EMPHASIS, specifies the CPU percentage at the sixth level. Not applicable for RATIO.
new_cpu_p7	For EMPHASIS, specifies the CPU percentage at the seventh level. Not applicable for RATIO.

Table 85–19 (Cont.) UPDATE_PLAN_DIRECTIVE Procedure Parameters

Parameter	Description
new_cpu_p8	For EMPHASIS, specifies the CPU percentage at the eighth level. Not applicable for RATIO.
new_active_sess_pool_p1	Specifies maximum number of concurrently active sessions for a consumer group. Default is NULL, which means unlimited.
new_queueing_p1	Specified time (in seconds) after which a job in the inactive session queue (waiting for execution) will time out. Default is NULL, which means unlimited.
new_switch_group	Specifies a limit on the degree of parallelism for any operation. Default is NULL, which means unlimited.
new_switch_time	Specifies consumer group to which this session is switched if other switch criteria is met. Default is NULL. If the group name is 'CANCEL_SQL', the current call will be canceled when other switch criteria are met. If the group name is 'KILL_SESSION', the session will be killed when other switch criteria are met.
new_switch_estimate	Specifies time (in seconds) that a session can execute before an action is taken. Default is NULL, which means unlimited.
new_max_est_exec_time	If TRUE, tells Oracle to use its execution time estimate to automatically switch the consumer group of an operation before beginning its execution. Default is FALSE.
new_undo_pool	Specifies the maximum execution time (in seconds) allowed for a session. If the optimizer estimates that an operation will take longer than MAX_EST_EXEC_TIME, the operation is not started and ORA-07455 is issued. If the optimizer does not provide an estimate, this directive has no effect. Default is NULL, which means unlimited.
new_parallel_degree_limit_p1	Sets a maximum in kilobytes (K) on the total amount of undo generated by a consumer group. Default is NULL, which means unlimited.
new_max_idle_time	Indicates the maximum session idle time. Default is NULL, which means unlimited.
new_max_idle_blocker_time	The maximum amount of time in seconds that a session can be idle while blocking another session's acquisition of a resource.
new_switch_time_in_call	Specifies time (in seconds) that a session can execute before an action is taken. At the end of the top call, the consumer group of the session is restored to its original consumer group. Default is NULL, which means unlimited. Both SWITCH_TIME_IN_CALL and SWITCH_TIME cannot be specified.

Usage Notes

- If the parameters for UPDATE_PLAN_DIRECTIVE are left unspecified, then they remain unchanged in the data dictionary.
- For new_max_idle_time and new_max_idle_blocker_time, PMON will check these limits once a minute. If it finds a session that has exceeded one of the limits, it will forcibly kill the session and clean up all its state.
- The parameter new_switch_time_in_call is mostly useful for three-tier applications where the mid-tier server is implementing session pooling. By turning on new_switch_time_in_call, the resource usage of one client will not affect the consumer group of a future client that happens to be executed on the same session.

VALIDATE_PENDING_AREA Procedure

This procedure lets you validate pending changes for the resource manager.

Syntax

```
DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA;
```

86

DBMS_RESOURCE_MANAGER_PRIVS

The DBMS_RESOURCE_MANAGER_PRIVS package maintains privileges associated with the Resource Manager.

> **See Also:** For more information on using the Database Resource Manager, see *Oracle Database Administrator's Guide.*

This chapter contains the following topics:

Summary of DBMS_RESOURCE_MANAGER_PRIVS Subprograms

Table 86–1 DBMS_RESOURCE_MANAGER_PRIVS Package Subprograms

Subprogram	Description
GRANT_SWITCH_ CONSUMER_GROUP Procedure on page 86-3	Grants the privilege to switch to resource consumer groups
GRANT_SYSTEM_PRIVILEGE Procedure on page 86-4	Performs a grant of a system privilege
REVOKE_SWITCH_ CONSUMER_GROUP Procedure on page 86-5	Revokes the privilege to switch to resource consumer groups.
REVOKE_SYSTEM_PRIVILEGE Procedure on page 86-6	Performs a revoke of a system privilege

GRANT_SWITCH_CONSUMER_GROUP Procedure

This procedure grants the privilege to switch to a resource consumer group.

Syntax

```
DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SWITCH_CONSUMER_GROUP (
   grantee_name   IN VARCHAR2,
   consumer_group IN VARCHAR2,
   grant_option   IN BOOLEAN);
```

Parameters

Table 86–2 GRANT_SWITCH_CONSUMER_GROUP Procedure Parameters

Parameter	Description
grantee_name	Name of the user or role to whom privilege is to be granted.
consumer_group	Name of consumer group.
grant_option	TRUE if grantee should be allowed to grant access, FALSE otherwise.

Usage Notes

If you grant permission to switch to a particular consumer group to a user, then that user can immediately switch their current consumer group to the new consumer group.

If you grant permission to switch to a particular consumer group to a role, then any users who have been granted that role and have enabled that role can immediately switch their current consumer group to the new consumer group.

If you grant permission to switch to a particular consumer group to PUBLIC, then any user can switch to that consumer group.

If the grant_option parameter is TRUE, then users granted switch privilege for the consumer group may also grant switch privileges for that consumer group to others.

In order to set the initial consumer group of a user, you must grant the switch privilege for that group to the user.

See Also: Chapter 85, "DBMS_RESOURCE_MANAGER"

Examples

```
BEGIN
DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SWITCH_CONSUMER_GROUP (
 'scott', 'mail_maintenance_group', true);
DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
DBMS_RESOURCE_MANAGER.set_consumer_group_mapping(
  dbms_resource_manager.oracle_user, 'scott','mail_maintenance_group');
DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
/
```

GRANT_SYSTEM_PRIVILEGE Procedure

This procedure performs a grant of a system privilege to a user or role.

Syntax

```
DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SYSTEM_PRIVILEGE (
   grantee_name   IN VARCHAR2,
   privilege_name  IN VARCHAR2 DEFAULT 'ADMINISTER_RESOURCE_MANAGER',
   admin_option   IN BOOLEAN);
```

Parameters

Table 86–3 GRANT_SYSTEM_PRIVILEGE Procedure Parameters

Parameter	Description
grantee_name	Name of the user or role to whom privilege is to be granted.
privilege_name	Name of the privilege to be granted.
admin_option	TRUE if the grant is with admin_option, FALSE otherwise.

Usage Notes

Currently, Oracle provides only one system privilege for the Resource Manager: ADMINISTER_RESOURCE_MANAGER. Database administrators have this system privilege with the ADMIN option. The grantee and the revokee can either be a user or a role. Users that have been granted the system privilege with the ADMIN option can also grant this privilege to others.

Examples

The following call grants this privilege to a user called scott without the ADMIN option:

```
BEGIN
DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SYSTEM_PRIVILEGE (
 grantee_name => 'scott',
 privilege_name => 'ADMINISTER_RESOURCE_MANAGER',
 admin_option => FALSE);
END;
/
```

REVOKE_SWITCH_CONSUMER_GROUP Procedure

This procedure revokes the privilege to switch to a resource consumer group.

Syntax

```
DBMS_RESOURCE_MANAGER_PRIVS.REVOKE_SWITCH_CONSUMER_GROUP (
   revokee_name   IN VARCHAR2,
   consumer_group IN VARCHAR2);
```

Parameters

Table 86–4 REVOKE_SWITCH_CONSUMER_GROUP Procedure Parameter

Parameter	Description
revokee_name	Name of user/role from which to revoke access.
consumer_group	Name of consumer group.

Usage Notes

If you revoke a user's switch privilege for a particular consumer group, then any subsequent attempts by that user to switch to that consumer group will fail.

If you revoke the initial consumer group from a user, then that user will automatically be part of the DEFAULT_CONSUMER_GROUP consumer group when logging in.

If you revoke the switch privilege for a consumer group from a role, then any users who only had switch privilege for the consumer group through that role will not be able to switch to that consumer group.

If you revoke the switch privilege for a consumer group from PUBLIC, then any users who could previously only use the consumer group through PUBLIC will not be able to switch to that consumer group.

Examples

The following example revokes the privileges to switch to mail_maintenance_group from Scott:

```
BEGIN
DBMS_RESOURCE_MANAGER_PRIVS.REVOKE_SWITCH_CONSUMER_GROUP (
 'scott', 'mail_maintenance_group');
END;
/
```

REVOKE_SYSTEM_PRIVILEGE Procedure

This procedure performs a revoke of a system privilege from a user or role.

Syntax

```
DBMS_RESOURCE_MANAGER_PRIVS.REVOKE_SYSTEM_PRIVILEGE (
   revokee_name   IN VARCHAR2,
   privilege_name IN VARCHAR2 DEFAULT 'ADMINISTER_RESOURCE_MANAGER');
```

Parameters

Table 86–5 REVOKE_SYSTEM_PRIVILEGE Procedure Parameters

Parameter	Description
`revokee_name`	Name of the user or role from whom privilege is to be revoked.
`privilege_name`	Name of the privilege to be revoked.

Examples

The following call revokes the `ADMINISTER_RESOURCE_MANAGER` from user scott:

```
BEGIN
DBMS_RESOURCE_MANAGER_PRIVS.REVOKE_SYSTEM_PRIVILEGE ('scott');
END;
/
```

87

DBMS_RESUMABLE

With the DBMS_RESUMABLE package, you can suspend large operations that run out of space or reach space limits after executing for a long time, fix the problem, and make the statement resume execution. In this way you can write applications without worrying about running into space-related errors.

This chapter contains the following topics:

Using DBMS_RESUMABLE

Operational Notes

When you suspend a statement, you should log the suspension in the alert log. You should also register a procedure to be executed when the statement is suspended. Using a view, you can monitor the progress of the statement and indicate whether the statement is currently executing or suspended.

Suspending a statement automatically results in suspending the transaction. Thus all transactional resources are held during a statement suspend and resume. When the error condition disappears, the suspended statement automatically resumes execution. A resumable space allocation can be suspended and resumed multiple times during execution.

A suspension timeout interval is associated with resumable space allocations. A resumable space allocation that is suspended for the timeout interval (the default is two hours) wakes up and returns an exception to the user. A suspended statement may be forced to throw an exception using the `DMBS_RESUMABLE.ABORT()` procedure.

Summary of DBMS_RESUMABLE Subprograms

Table 87–1 DBMS_RESUMABLE Package Subprograms

Subprogram	Description
ABORT Procedure on page 87-5	Aborts a suspended resumable space allocation
GET_SESSION_TIMEOUT Function on page 87-6	Returns the current timeout value of the resumable space allocations for a session with `session_id`
GET_TIMEOUT Function on page 87-7	Returns the current timeout value of resumable space allocations for the current session
SET_SESSION_TIMEOUT Procedure on page 87-8	Sets the timeout of resumable space allocations for a session with `session_id`
SET_TIMEOUT Procedure on page 87-9	Sets the timeout of resumable space allocations for the current session
SPACE_ERROR_INFO Function on page 87-10	Looks for space-related errors in the error stack, otherwise returning `FALSE`

ABORT Procedure

This procedure aborts a suspended resumable space allocation. The parameter `session_id` is the session ID in which the statement is executed. For a parallel DML/DDL, `session_id` is any session ID that participates in the parallel DML/DDL. This operation is guaranteed to succeed. The procedure can be called either inside or outside of the `AFTER SUSPEND` trigger.

Syntax

```
DBMS_RESUMABLE.ABORT (
   session_id  IN NUMBER);
```

Parameters

Table 87–2 ABORT Procedure Parameters

Parameter	Description
`session_id`	The session identifier of the resumable space allocation.

Usage Notes

To call an `ABORT` procedure, you must be the owner of the session with `session_id`, have `ALTER SYSTEM` privileges, or be a DBA.

GET_SESSION_TIMEOUT Function

This function returns the current timeout value of resumable space allocations for a session with `session_id`.

Syntax

```
DBMS_RESUMABLE.GET_SESSION_TIMEOUT (
   session_id  IN NUMBER)
RETURN NUMBER;
```

Parameters

Table 87–3 GET_SESSION_TIMEOUT Function Parameters

Parameter	Description
`session_id`	The session identifier of the resumable space allocation.

Return Values

Table 87–4 GET_SESSION_TIMEOUT Function Return Values

Return Value	Description
`NUMBER`	The current timeout value of resumable space allocations for a session with session_id.The timeout is returned in seconds.

Usage Notes

If `session_id` does not exist, the `GET_SESSION_TIMEOUT` function returns -1.

GET_TIMEOUT Function

This function returns the current timeout value of resumable space allocations for the current session.

Syntax

```
DBMS_RESUMABLE.GET_TIMEOUT
 RETURN NUMBER;
```

Return Values

Table 87–5 GET_TIMEOUT Function Return Values

Return Value	Description
NUMBER	The current timeout value of resumable space allocations for the current session. The returned value is in seconds.

Usage Notes

If the current session is not resumable enabled, the GET_TIMEOUT function returns -1.

SET_SESSION_TIMEOUT Procedure

This procedure sets the timeout of resumable space allocations for a session with `session_id`. The new timeout setting applies to the session immediately. If `session_id` does not exist, no operation occurs.

Syntax

```
DBMS_RESUMABLE.SET_SESSION_TIMEOUT (
   session_id  IN NUMBER,
   timeout     IN NUMBER);
```

Parameters

Table 87–6 SET_SESSION_TIMEOUT Procedure Parameters

Parameter	Description
`session_id`	The session identifier of the resumable space allocation.
`timeout`	The timeout of the resumable space allocation.

SET_TIMEOUT Procedure

This procedure sets the timeout of resumable space allocations for the current session. The new timeout setting applies to the session immediately.

Syntax

```
DBMS_RESUMABLE.SET_TIMEOUT (
   timeout  IN NUMBER);
```

Parameters

Table 87–7 SET_TIMEOUT Procedure Parameters

Parameter	Description
timeout	The timeout of the resumable space allocation.

SPACE_ERROR_INFO Function

This function looks for space-related errors in the error stack. If it cannot find a space related error, it will return `FALSE`. Otherwise, `TRUE` is returned and information about the particular object that causes the space error is returned.

Syntax

```
DBMS_RESUMABLE.SPACE_ERROR_INFO
   error_type        OUT VARCHAR2,
   object_type       OUT VARCHAR2,
   object_owner      OUT VARCHAR2,
   table_space_name  OUT VARCHAR2,
   object_name       OUT VARCHAR2,
   sub_object_name   OUT VARCHAR2)
RETURN BOOLEAN;
```

Parameters

Table 87–8 SPACE_ERROR_INFO Function Parameters

Parameter	Description
`error_type`	The space error type. It will be one of the following: ■ `NO MORE SPACE` ■ `MAX EXTENTS REACHED` ■ `SPACE QUOTA EXCEEDED`
`object_type`	The object type. It will be one of the following: ■ TABLE ■ INDEX ■ CLUSTER ■ TABLE SPACE ■ ROLLBACK SEGMENT ■ UNDO SEGMENT ■ LOB SEGMENT ■ TEMP SEGMENT ■ INDEX PARTITION ■ TABLE PARTITION ■ LOB PARTITION ■ TABLE SUBPARTITION ■ INDEX SUBPARTITION ■ LOB SUBPARTITION The type can also be `NULL` if it does not apply.
`object_owner`	The owner of the object. `NULL` if it cannot be determined.
`table_space_name`	The table space where the object resides. `NULL` if it cannot be determined.
`object_name`	The name of rollback segment, temp segment, table, index, or cluster.
`sub_object_name`	The partition name or sub-partition name of `LOB, TABLE,` or `INDEX. NULL` if it cannot be determined.

88

DBMS_RLMGR

The DBMS_RLMGR package contains various procedures to create and manage rules and rule sessions by the Rules Manager.

> **See Also:** *Oracle Database Application Developer's Guide - Rules Manager and Expression Filter* for more information.

This chapter contains the following topic:

- Summary of Rules Manager Subprograms

Summary of Rules Manager Subprograms

Table 88–1 describes the subprograms in the DBMS_RLMGR package.

All the values and names passed to the procedures defined in the DBMS_RLMGR package are case insensitive unless otherwise mentioned. In order to preserve the case, double quotation marks should be used around the values.

Table 88–1 DBMS_RLMGR Package Subprograms

Subprogram	Description
ADD_ELEMENTARY_ATTRIBUTE Procedures	Adds the specified attribute to the event structure (also the Expression Filter attribute set)
ADD_EVENT Procedures	Adds an event to a rule class in an active session
ADD_FUNCTIONS Procedure	Adds a Function, a Type, or a Package to the approved list of functions with an event structure (also the Expression Filter attribute set)
ADD_RULE Procedure	Adds a rule to the rule class
CONSUME_EVENT Function	Consumes an event using its identifiers and prepares the corresponding rule for action execution
CONSUME_PRIM_EVENTS Function	Consumes one or more primitive events with all or none semantics
CREATE_EVENT_STRUCTURE Procedure	Creates an event structure
CREATE_RULE_CLASS Procedure	Creates a rule class
DELETE_RULE Procedure	Deletes a rule from a rule class
DROP_EVENT_STRUCTURE Procedure	Drops an event structure
DROP_RULE_CLASS Procedure	Drops a rule class
GRANT_PRIVILEGE Procedure	Grants a privilege on a rule class to another user
PROCESS_RULES Procedure	Process the rules for a given event
RESET_SESSION Procedure	Starts a new rule session within a database session
REVOKE_PRIVILEGE Procedure	Revokes a privilege on a rule class from a user

ADD_ELEMENTARY_ATTRIBUTE Procedures

This procedure adds the specified attribute to an event structure, which is also the Expression Filter attribute set. The procedure is overloaded. The different functionality of each form of syntax is presented along with the definitions.

Syntax

Adds the specified elementary attribute to the attribute set:

```
DBMS_RLMGR.ADD_ELEMENTARY_ATTRIBUTE (
              event_struct   IN   VARCHAR2,
              attr_name      IN   VARCHAR2,
              attr_type      IN   VARCHAR2,
              attr_defvl     IN   VARCHAR2 default NULL);
```

Identifies the elementary attributes that are table aliases and adds them to the event structure:

```
DBMS_RLMGR.ADD_ELEMENTARY_ATTRIBUTE (
              event_struct   IN   VARCHAR2,
              attr_name      IN   VARCHAR2,
              tab_alias      IN   rlm$table_alias);
```

Parameters

Table 88–2 ADD_ELEMENTARY_ATTRIBUTE Procedure Parameters

Parameter	Description
`event_struct`	Name of the event structure or attribute set to which this attribute is added
`attr_name`	Name of the elementary attribute to be added. No two attributes in a set can have the same name.
`attr_type`	Datatype of the attribute. This argument accepts any standard SQL datatype or the name of an object type that is accessible to the current user.
`tab_alias`	The type that identifies the database table to which the attribute is aliased
`attr_defv1`	Default value for the elementary attribute

Usage Notes

- This procedure adds an elementary attribute to an event structure. The event structure is internally managed as the Expression Filter attribute set. If the event structure was originally created from an existing object type, then additional attributes cannot be added.

  Elementary attributes cannot be added to an attribute set that is already assigned to a column storing expressions, which is equivalent to an event structure that is used for a rule class.

- One or more, or all elementary attributes in an attribute set can be table aliases. If an elementary attribute is a table alias, then the value assigned to the elementary attribute is a `ROWID` from the corresponding table. An attribute set with one or more table alias attributes cannot be created from an existing object type. For more information about table aliases, see Appendix A in *Oracle Database Application Developer's Guide - Rules Manager and Expression Filter*.

- Elementary attributes cannot be added to an attribute set that is already assigned to a column storing expressions.
- See "Defining Attribute Sets" in *Oracle Database Application Developer's Guide - Rules Manager and Expression Filter* for more information about adding elementary attributes.
- Related views: USER_EXPFIL_ATTRIBUTE_SETS and USER_EXPFIL_ATTRIBUTES.

Examples

The following commands add two elementary attributes to an attribute set:

```
BEGIN
  DBMS_RLMGR.ADD_ELEMENTARY_ATTRIBUTE (
                               event_struct  => 'HRAttrSet',
                               attr_name => 'HRREP',
                               attr_type => 'VARCHAR2(30)');
  DBMS_RLMGR.ADD_ELEMENTARY_ATTRIBUTE (
                               event_struct  => 'HRAttrSet',
                               attr_name => 'DEPT',
                               tab_alias => exf$table_alias('DEPT'));
END;
```

ADD_EVENT Procedures

This procedure adds a primitive event to a rule class in an active rule session. The procedure is overloaded. The different functionality of each form of syntax is presented along with the definitions.

Syntax

Adds a string representation of the primitive event instance to a rule class:

```
DBMS_RLMGR.ADD_EVENT (
   rule_class      IN VARCHAR2,
   event_inst      IN VARCHAR2,
   event_type      IN VARCHAR2 default null);
```

Adds an AnyData representation of the primitive event instance to a rule class:

```
DBMS_RLMGR.ADD_EVENT (
   rule_class      IN VARCHAR2,
   event_inst      IN sys.AnyData);
```

Parameters

Table 88–3 ADD_EVENT Procedure Parameters

Parameter	Description
rule_class	Name of the rule class. A schema extended rule class name can be used to refer to a rule class that does not belong to the current schema.
event_inst	String or AnyData representation of the event instance being added to the rule class
event_type	Type of event instance assigned to the event_inst argument when the string representation of the event instance is used for a rule class configured for composite events

Usage Notes

- This procedure is used to add a primitive or a simple event to a rule class within an active rule session. By default, a rule session is the same as the database session. Optionally, multiple (sequential) rule sessions can be started within a database session by using the RESET_SESSION or PROCESS_RULES procedures.
- When the rule class is configured for simple events (consisting of only one primitive event structure), the event_type argument for the ADD_EVENT procedure can be ignored. Also, when the AnyData format of the event instance is passed, the event type information is embedded in the AnyData instance. In all other cases, the name of the primitive event structure being added to the rule class should be assigned to the event_type argument.
- For a valid event instance, the ADD_EVENT procedure processes the rules in the rule class and captures the results in the rule class results view (configured at the time of rule class creation). These results are preserved until the end of the rule session.
- When schema extended name is used for the rule class, you should have PROCESS RULES privilege on the rule class. See the GRANT_PRIVILEGE Procedure for additional information. The value specified for the event_type argument is always resolved in the rule class owner's schema and should not use schema extended names. When a composite event structure is configured with a table alias

primitive event type, the name of the corresponding table should be assigned to the `event_type` argument.

Examples

The following commands add two events to the CompTravelPromo rule class that is configured for two types of primitive events (AddFlight and AddRentalCar).

```
BEGIN
 DBMS_RLMGR.ADD_EVENT(rule_class => 'CompTravelPromo',
                      event_inst =>
                         AddFlight.getVarchar(987, 'Abcair', 'Boston',
                                              'Orlando', '01-APR-2003', '08-APR-2003'),
                      event_type => 'AddFlight');

DBMS_RLMGR.ADD_EVENT(rule_class => 'Scott.CompTravelPromo',
                      event_inst =>
                         AnyData.convertObject(
                                        AddRentalCar(987, 'Luxury', '03-APR-2003',
                                                     '08-APR-2003', NULL)));
END;
```

ADD_FUNCTIONS Procedure

This procedure adds a user-defined function, package, or type representing a set of functions to the event structure, which is also the Expression Filter attribute set.

Syntax

```
DBMS_RLMGR.ADD_FUNCTIONS (
   event_struct   IN   VARCHAR2,
   funcs_name     IN   VARCHAR2);
```

Parameters

Table 88–4 ADD_FUNCTIONS Procedure Parameters

Parameter	Description
`event_struct`	Name of the event structure to which the functions are added
`funcs_name`	Name of a function, package, or type (representing a function set) or its synonyms

Usage Notes

- By default, an attribute set implicitly allows references to all Oracle supplied SQL functions for use in the rule conditions. If the expression set refers to a user-defined function, the expression set must be explicitly added to the attribute set.
- The `ADD_FUNCTIONS` procedure adds a user-defined function or a package (or type) representing a set of functions to the attribute set. Any new or modified expressions are validated using this list.
- The function or the package name can be specified with a schema extension. If a function name is specified without a schema extension, only such references in the rule condition are considered valid. The conditional expression can be restricted to use a synonym to a function or a package by adding the corresponding synonym to the attribute set. This preserves the portability of the expression set to other schemas.
- See "Defining Attribute Sets" in *Oracle Database Application Developer's Guide - Rules Manager and Expression Filter* for more information about adding functions to an attribute set.
- Related views: `USER_EXPFIL_ATTRIBUTE_SETS` and `USER_EXPFIL_ASET_FUNCTIONS`

Examples

The following commands add two functions to the attribute set:

```
BEGIN
  DBMS_RLMGR.ADD_FUNCTIONS (attr_set   => 'Car4Sale',
                            funcs_name => 'HorsePower');
  DBMS_RLMGR.ADD_FUNCTIONS (attr_set   => 'Car4Sale',
                            funcs_name => 'Scott.CrashTestRating');
END;
```

ADD_RULE Procedure

This procedure adds new rules to a rule class.

Syntax

```
DBMS_RLMGR.ADD_RULE (
   rule_class      IN  VARCHAR2,
   rule_id         IN  VARCHAR2,
   rule_cond       IN  VARCHAR2,
   actprf_nml      IN  VARCHAR2 DEFAULT NULL,
   actprf_vall     IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 88–5 ADD_RULE Procedure Parameters

Parameter	Description
`rule_class`	Name of the rule class. A schema extended rule class name can be used to refer to a rule class that does not belong to the current schema.
`rule_id`	Unique identifier for the rule within the rule class
`rule_cond`	The condition for the rule. The condition uses the variables defined in the rule class's event structure.
`actprf_nml`	The list of action preference names for which values will be assigned through the `actprf_vall` argument
`actprf_vall`	The list of action preference values for the names list assigned to the `actprf_nml` argument

Usage Notes

- This procedure is used to add new rules to the rule class. The rule condition passed to the `ADD_RULE` procedure is validated using the event structure associated with the rule class. The action preferences names list is a subset of action preference categories configured during rule class creation.
- When schema extended name is used for the rule class, you should have `ADD RULE` privilege on the rule class. See the GRANT_PRIVILEGE Procedure for more information.
- Alternately, the owner of the rule class can add the rules using SQL `INSERT` statement on the rule class table (that shares the same name as the rule class). Note that the owner of the rule class can also grant direct DML privileges on the rule class table to other users.

  Note: The AUTOCOMMIT property of the rule class is ignored if the new rules are added using the SQL INSERT statement instead of the ADD_RULE procedure.

- See the CREATE_RULE_CLASS Procedure procedure for the structure of the rule class table.

Examples

The following command adds a rule to the rule class.

```
BEGIN
DBMS_RLMGR.ADD_RULE (
           rule_class => 'CompTravelPromo',
           rule_id => 'AB_AV_FL',
           rule_cond =>
             '<condition>
                 <and join="Flt.CustId = Car.CustId">
                    <object name="Flt">
                      Airline=''Abcair'' and ToCity=''Orlando''
                    </object>
                    <object name="Car">
                      CarType = ''Luxury''
                    </object>
                  </and>
               </condition>' ,
             actprf_nml => 'PromoType, OfferedBy',
             actprf_vall => '''RentalCar'', ''Acar''');
END;
```

With proper privileges, the following SQL INSERT statement can be used to add the rule to the rule class.

```
INSERT INTO CompTravelPromo (rlm$ruleid, rlm$rulecond, PromoType, OfferedBy)
  VALUES ('AB_AV_FL',
          '<condition>
             <and join="Flt.CustId = Car.CustId">
               <object name="Flt">
                   Airline=''Abcair'' and ToCity=''Orlando''
               </object>
               <object name="Car">
                   CarType = ''Luxury''
               </object>
             </and>
           </condition>',
          'RentalCar','Acar');
```

CONSUME_EVENT Function

This function consumes an event and prepares the corresponding rule for action execution. This is required only when the action (or rule execution) is carried by the user's application and not in the callback.

Syntax

```
DBMS_RLMGR.CONSUME_EVENT (
   rule_class       IN VARCHAR2,
   event_ident      IN VARCHAR2)
 RETURN NUMBER;
```

Parameters

Table 88–6 CONSUME_EVENT Function Parameters

Parameter	Description
rule_class	Name of the rule class. A schema extended rule class name can be used to refer to a rule class that does not belong to the current schema.
event_ident	Event identifier obtained from the corresponding rule class results view (or arguments of the action callback procedure in the case of rule class configured for RULE based consumption policy)

Returns

The function returns:

- 1 -- If the event is successfully consumed.
- 0 -- If the event is expired (owing to duration policy) or consumed by another session prior to this call.

Usage Notes

- When an EXCLUSIVE consumption policy is set for the events in a rule class, an event must be deleted from the system immediately after the rule it matched is executed (action is executed). When the rule action is carried in the rule class callback procedure by calling the PROCESS_RULES procedure, the rule manager automatically handles the consumption of the events. However, when you request the results from matching events with rules in a rule class results view using the ADD_EVENT procedure, you should take appropriate action to indicate the exact rule-event combination that is to be used for rule execution. The CONSUME_EVENT procedure performs the required housekeeping services when the unique identifier for the event used in a rule execution is passed in.
- Since there could be a time lag between fetching the rule class matching results and the execution of the user initiated action, the application should execute the action only if the CONSUME_EVENT call succeeds in consuming the event. This avoids any race condition with parallel sessions trying to consume the same events. When the event is successfully consumed, this call returns 1. In all other cases, it returns 0. A return value of 0 implies that the event is already consumed by another session and hence it is not available for this session.
- The CONSUME_EVENT procedure deletes the events configured with EXCLUSIVE consumption policy and does nothing for events configured for 4 consumption policy.

- Unlike the EXCLUSIVE and SHARED consumption policies, which are determined at the rule class level, a RULE consumption policy can be used to determine the consumption of an event on a rule by rule basis. That is a subset of the rules in a rule class may be configured such that when they are matched, the event is deleted from the system. At the same time the other set of rules could leave the event in the system even after executing the corresponding action. In this scenario, the action callback procedure implemented by the application developer can call CONSUME_EVENT function (with appropriate arguments) to conditionally consume the event for certain rules. Also see the use of CONSUME_PRIM_EVENTS Function for rule classes configured for RULE consumption policy

Examples

The following commands identify an event that is used for a rule execution and consumes it using its identifier.

```
var eventid VARCHAR(40);
var evtcnsmd NUMBER;

BEGIN
  SELECT rlm$eventid INTO :eventid FROM MatchingPromos WHERE rownum < 2;

  -- carry the required action for a rule matched by the above event --
  :evtcnsmd := DBMS_RLMGR.CONSUME_EVENT(rule_class  => 'TravelPromotion',
                                        event_ident => :eventid);
END;
```

CONSUME_PRIM_EVENTS Function

This function consumes a set of primitive events with all or nothing semantics in the case of a rule class configured with RULE based consumption policy.

Syntax

```
DBMS_RLMGR.CONSUME_PRIM_EVENTS (
   rule_class       IN VARCHAR2,
   event_idents     IN RLM$EVENTIDS)
 RETURN NUMBER;
```

Parameters

Table 88–7 CONSUME_PRIM_EVENTS Function Parameters

Parameter	Description
rule_class	Name of the rule class. A schema extended rule class name can be used to refer to a rule class that does not belong to the current schema.
event_ident	Event identifiers obtained from the corresponding rule class results view or the arguments of the action callback procedure

Returns

The function returns:

- 1 -- If all the events, the identifiers for which are passed in, are successfully consumed.
- 0 -- If one or more primitive event could not be consumed.

Usage Notes

When the rule class is configured for RULE based consumption policy, the CONSUME_PRIM_EVENTS function can be used to consume one or more primitive events that constitute a composite event. This operation will succeed only when all the events passed in are still valid and are available for consumption. Any user initiated action should be implemented after checking the return value of the CONSUME_PRIM_EVENTS call.

Examples

The following commands show the body of the action callback procedure for a rule class configured for RULE consumption policy. This demonstrates the use of CONSUME_PRIM_EVENTS procedure to consume the events before executing the action for the matched rules.

```
create or replace procedure PromoAction (
      Flt        AddFlight,
      Flt_EvtId  ROWID,    --- rowid for the flight primitive event
      Car        AddRentalCar,
      Car_EvtId  ROWID,
      rlm$rule   TravelPromotions%ROWTYPE) is
  evtcnsmd   NUMBER;
BEGIN
  evtcnsmd := DBMS_RLMGR.CONSUME_PRIM_EVENTS(
                   rule_class   => 'TravelPromotions',
                   event_idents => RLM$EVENTIDS(Flt_EvtId, Car_EvtId));
```

```
  if (evtcnsmd = 1) then
    -- consume operation was successful; perform the action ---
    OfferPromotion (Flt.CustId, rlm$rule.PromoType, rlm$rule.OfferedBy);
  end if;
END;
```

CREATE_EVENT_STRUCTURE Procedure

This procedure creates an event structure.

Syntax

```
DBMS_RLMGR.CREATE_EVENT_STRUCTURE  (
   event_struct  IN  VARCHAR2);
```

Parameters

Table 88–8 CREATE_EVENT_ STRUCTURE Procedure Parameter

Parameter	Description
event_struct	Name of the event structure to be created in the current schema

Usage Notes

This procedure creates a dummy the event structure in the current schema. One or more attributes can be added to this event structure using the ADD_ELEMENTARY_ATTRIBUTE procedure.

Examples

The following command creates the event structure.

```
BEGIN
  DBMS_RLMGR.CREATE_EVENT_STRUCT(event_struct => 'AddFlight');
END;
```

CREATE_RULE_CLASS Procedure

This procedure creates a rule class.

Syntax

```
DBMS_RLMGR.CREATE_RULE_CLASS  (
   rule_class      IN  VARCHAR2,
   event_struct    IN  VARCHAR2,
   action_cbk      IN  VARCHAR2,
   actprf_spec     IN  VARCHAR2  default null,
   rslt_viewnm     IN  VARCHAR2  default null,
   rlcls_prop      IN  VARCHAR2  default <simple/>);
```

Parameters

Table 88–9 CREATE_RULE_CLASS Procedure Parameters

Parameter	Description
rule_class	The name of the rule class to be created in the current schema
event_struct	The name of the object type or an Expression Filter attribute set in the current schema that represents the event structure for the rule class
action_cbk	The name of the action callback procedure to be created for the rule class
actprf_spec	The specification (name and SQL datatype pairs) for the action preferences associated with the rule class
rlst_viewnm	The name of the rule class results view that lists the matching events and rules within a session. A view with this name is created in the current schema.
rlcls_prop	The XML document for setting the rule class properties. By default, the rule class created is for simple events (non-composite).

Usage Notes

- For successful creation of a rule class, you should have sufficient privileges to create views, object types, tables, packages, and procedures.
- This command creates the rule class and its dependent objects in the user's schema. For this operation to succeed the name specified for the event structure should refer to an existing object type or an Expression Filter attribute set in the user's schema. When an object type is used for an event structure, the CREATE_RULE_CLASS procedure implicitly creates an attribute set for the object type. In the case of a rule class configured for composite events, the previous procedure also creates attribute sets for the object types that are directly embedded in the event structure's object type (or the attribute set). A maximum of 32 embedded objects (and/or table aliases) can be specified with an event structure that is used for a composite rule class. The types of dependent objects created with this procedure and their structure depend on the properties of the rule class and its event structure. The minimum set of dependent objects created for a rule class is as follows:
  - Rule class table – A rule class table that shares the name of the rule class is created in the user's schema to store the rule definitions (rule identifiers, rule conditions, rule descriptions, and action preferences). This table implicitly has three columns, rlm$ruleid, rlm$rulecond, and rlm$ruledesc to store the rule identifiers, rule conditions, and rule descriptions respectively. In addition to

these three columns, the rule class table has few columns according to the rule classes' access preference specification. For example, if a TravelPromotion rule class uses 'PromoType VARCHAR(20), OfferedBy VARCHAR(20)' as its access preference specification (assigned to actpref_spec argument), the rule class table is created with the following structure.

```
TABLE TravelPromotion (
      rlm$ruleid VARCHAR(100),     -- rule identifier column --
      PromoType VARCHAR(20),       -- access preference 1 --
      OfferedBy VARCHAR(20),       -- access preference 2 --
      rlm$rulecond VARCHAR(4000),  -- rule condition --
      rlm$ruledesc VARCHAR(1000)); -- rule description --
```

The rule class table structure varies from one rule class to another based on the exact list of action preference categories specified for the rule class.

– Action Callback Procedure – The skeleton for the action callback procedure with the given name is created in the user's schema and it is associated with the rule class. During rule evaluation, the callback procedure is called for each matching rule and event. You should implement the body of the action callback procedure to perform the appropriate action for each rule. The exact action for a rule can be determined based on the event that matched the rule and rule definition along with its action preferences. This information is passed to the action callback procedure through its arguments. Hence, the argument list for the action callback procedure depends on the event structure associated with the rule class and the rule class itself.

  In the case of a rule class configured for simple events (<simple/> assigned to the properties of the rule class), the event that matches a rule is passed through a rlm$event argument that is declared to be of the same type as the event structure. Additionally, the rule definitions are passed to the action callback procedure using an rlm$rule argument that is declared as ROWTYPE of the corresponding rule class table. For example, the structure of the PromoAction action callback procedure created for a TravelPromotion rule class configured for a simple (non-composite) AddFlight event structure is as follows:

```
PROCEDURE PromoAction (rlm$event AddFlight,
                       rlm$rule TravlePromotion%ROWTYPE);
```

  In the case of a rule class created for composite events (<composite/> assigned to the properties of the rule class), the action callback procedure is created to pass each primitive event as a separate argument. For example, the CompPromoAction action callback procedure created for a rule class CompTravelPromo configured for a composite event with AddFlight and AddRentalCar primitive events are shown as follows:

```
-- composite event structure --
TYPE TSCompEvent (Flt AddFlight,
                  Car AddRentalCar);
-- corresponding action callback procedure --
PROCEDURE PromoAction (Flt AddFlight,
                       Car AddRentalCar,
                       rlm$rule CompTravelPromo%ROWTYPE)
```

– Rule class results view – A view to display the results from matching some events with rules is created in the same schema as the rule class. By default, this view is created with a system-generated name. Optionally, the rule class creator can specify a name for this view with the rlst_viewnm argument of

the CREATE_RULE_CLASS procedure. When the events are added to the rule manager within a rule session using the ADD_EVENT procedure, the list of matching events and rules are displayed in the rule class results view.

The structure of the view defined for the rule class results depends on the event structure and the action preferences configured with the rule class. Minimally, the view has three columns to display the system generated event identifier (rlm$evenetid), the identifier of the rule it matches (rlm$ruleid), and the rule condition (rlm$rulecond). Additionally, it has columns to display the event information and the rule action preferences.

In the case of a rule class configured for simple events, the event information is displayed as rlm$event that is declared to be of the event structure type. So, a MatchingPromos view created for the TravelPromotion rule class configured for a simple AddFlight event structure is as follows:

```
VIEW MatchingPromos (
    rlm$eventid ROWID,
    rlm$event AddFlight,
    rlm$ruleid VARCHAR(100),
    PromoType VARCHAR(30), -- action preference 1 --
    OffredBy VARCHAR(30), -- action preference 2 --
    rlm$rulecond VARCHAR(4000),
    rlm$ruledesc VARCHAR(1000)
);
```

In the case of a rule class configured for composite events, the primitive events matching a rule are displayed separately using corresponding columns. For the above CompTravelPromo rule class, a MatchingCompPromos view is created with the following structure.

```
VIEW MatchingCompPromos (
    rlm$eventid ROWID,
    Flt AddFlight,
    Car AddRentalCar,
    rlm$ruleid VARCHAR(100),
    PromoType VARCHAR(30),   -- action preference 1 --
    OffredBy VARCHAR(30),    -- action preference 2 --
    rlm$rulecond VARCHAR(4000),
    rlm$ruledesc VARCHAR(1000)
);
```

The values from the rlm$eventid column are used to enforce rule class consumption policies when the corresponding rule is executed. See the CONSUME_EVENT Function procedure for more information.

Examples

The following commands create a rule class for simple events (of AddFlight type).

```
CREATE or REPLACE TYPE AddFlight AS OBJECT (
                  CustId NUMBER,
                  Airline VARCHAR(20),
                  FromCity VARCHAR(30),
                  ToCity VARCHAR(30),
                  Depart DATE,
                  Return DATE);
BEGIN
  DBMS_RLMGR.CREATE_RULE_CLASS (
             rule_class   => 'TravelPromotion', -- rule class name --
             event_struct => 'AddFlight', -- event struct name --
```

```
              action_cbk    => 'PromoAction', -- callback proc name --
              rslt_viewnm   => 'MatchingPromos', -- results view --
              actprf_spec   => 'PromoType VARCHAR(20),
                                OfferedBy VARCHAR(20)');
END;
```

The following commands create a rule class for composite events consisting of two primitive events (AddFlight and AddRentalCar).

```
CREATE or REPLACE TYPE TSCompEvent (Flt AddFlight,
                                    Car AddRentalCar);
BEGIN
  DBMS_RLMGR.CREATE_RULE_CLASS (
              rule_class    => 'CompTravelPromo', -- rule class name --
              event_struct  => 'TSCompEvent', -- event struct name --
              action_cbk    => 'CompPromoAction', -- callback proc name --
              rslt_viewnm   => 'MatchingCompPromos', -- results view --
              actprf_spec   => 'PromoType VARCHAR(20),
                                OfferedBy VARCHAR(20)',
              properties    => '<composite/>');
END;
```

DELETE_RULE Procedure

This procedure deletes a rule from a rule class.

Syntax

```
DBMS_RLMGR.DELETE_RULE (
   rule_class    IN    VARCHAR2,
   rule_id       IN    VARCHAR2);
```

Parameters

Table 88–10 DELETE_RULE Procedure Parameters

Parameter	Description
rule_class	Name of the rule class. A schema extended rule class name can be used to refer to a rule class that does not belong to the current schema.
rule_id	Identifier for the rule to be deleted

Usage Notes

- This procedure is used to delete a rule from the rule class. The identifier for the rule to be deleted can be obtained by querying the rule class table (that shares the same name as the rule class). Alternately, the owner of the rule class can use a SQL DELETE statement on one rule class table to delete a rule.
- When schema extended name is used for the rule class, you should have DELETE RULE privilege on the rule class. See the GRANT_PRIVILEGE Procedure procedure for more information.

  Note: AUTOCOMMIT property of the rule class is ignored if the rules are deleted with the SQL DELETE statement instead of the DELETE_RULE procedure.

- See the CREATE_RULE_CLASS Procedure procedure for the structure of the rule class table.

Examples

The following command deletes a rule from the rule class.

```
BEGIN
  DBMS_RLMGR.DELETE_RULE (
           rule_class  => 'CompTravelPromo',
           rule_id     => 'AB_AV_FL');
END;
```

Alternately, the following SQL DELETE statement can be issued to delete the above rule from the rule class.

```
DELETE FROM CompTravelPromo WHERE rlm$ruleid = 'AB_AV_FL';
```

DROP_EVENT_STRUCTURE Procedure

This procedure drops an event structure.

Syntax

```
DBMS_RLMGR.DROP_EVENT_STRUCTURE  (
   event_struct  IN  VARCHAR2);
```

Parameters

Table 88–11 DROP_EVENT_ STRUCTURE Procedure Parameter

Parameter	Description
event_struct	Name of the event structure in the current schema

Usage Notes

This procedure drops the event structure from the current schema. This drops all the dependent objects created to manage the event structure.

Examples

The following command drops the event structure.

```
BEGIN
  DBMS_RLMGR.DROP_EVENT_STRUCT(event_struct => 'AddFlight');
END;
```

DROP_RULE_CLASS Procedure

This procedure drops a rule class.

Syntax

```
DBMS_RLMGR.DROP_RULE_CLASS  (
   rule_class  IN  VARCHAR2);
```

Parameters

Table 88–12 DROP_RULE_CLASS Procedure Parameter

Parameter	Description
rule_class	The name of the rule class in the current schema

Usage Notes

This procedure drops the rule class from the current schema. This drops all the dependent objects created to manage the rule class. Because an event structure in a user's schema can be shared across multiple rule classes, the event structure is not dropped with this command. The DROP_EVENT_STRUCTURE API should be used for the composite event as well as the individual primitive events to cleanup unused event structures.

Examples

The following command drops the rule class.

```
BEGIN
  DBMS_RLMGR.DROP_RULE_CLASS(rule_class => 'CompTravelPromo');
END;
```

GRANT_PRIVILEGE Procedure

This procedure grants privileges on a rule class to another user.

Syntax

```
DBMS_RLMGR.GRANT_PRIVILEGE  (
   rule_class      IN  VARCHAR2,
   priv_type       IN  VARCHAR2,
   to_user         IN  VARCHAR2);
```

Parameters

Table 88–13 GRANT_PRIVILEGE Procedure Parameters

Parameter	Description
rule_class	The name of the rule class in the current schema
priv_type	Type of rule class privilege to be granted
to_user	The user to whom the privilege is to be granted

Usage Notes

- This procedure grants appropriate privileges to a user who is not the owner of the rule class. The types of privileges that can be granted to a user are:
  - PROCESS RULES: A user with PROCESS RULES privilege on a rule class can process the rules in the rule class using the PROCESS_RULES procedure or the ADD_EVENT procedure. Also, the user with this privilege can select from the corresponding rule class results view.
  - ADD RULE: A user with ADD RULE privilege on a rule class can add rules to a rule class. Alternatively, the owner of the rule class can grant INSERT privileges on one rule class table to other users.
  - DELETE RULE: A user with DELETE RULE privilege on a rule class can delete rules from a rule class. Alternatively, the owner of the rule class can grant DELETE privileges on one rule class table to other users.
  - ALL: Granting the ALL privilege on a rule class is equivalent to granting all the above privileges on the rule class to the user.
- The owner of the rule class always has privileges to drop a rule class, process rules in a rule class, add rules and delete rules from a rules class. Only the owner of the rule class can drop a rule class and this privilege cannot be granted to another user.
- A user should also have EXECUTE privileges on the primitive event types associated with a rule class in order to make use of the corresponding rule class results view

Examples

The following command grants PROCESS RULES privilege on TravelPromo rule class to the user SCOTT.

```
BEGIN
  DBMS_RLMGR.GRANT_PRIVILEGE(rule_class => 'TravelPromo',
                             priv_type => 'PROCESS RULES',
```

```
                                    to_user => 'SCOTT');
END;
```

PROCESS_RULES Procedure

This procedure processes the rules for a given event. The procedure is overloaded. The different functionality of each form of syntax is presented along with the definitions.

Syntax

Processes the rules for a string representation of the event instance being added to the rule class:

```
DBMS_RLMGR.PROCESS_RULES  (
   rule_class    IN  VARCHAR2,
   event_inst    IN  VARCHAR2,
   event_type    IN  VARCHAR2 default null);
```

Processes the rules for an AnyData representation of the event instance being added to the rule class:

```
DBMS_RLMGR.PROCESS_RULES  (
   rule_class    IN  VARCHAR2,
   event_inst    IN  sys.AnyData);
```

Parameters

Table 88–14 PROCESS_RULES Procedure Parameters

Parameter	Description
`rule_class`	The name of the rule class. A schema extended rule class name can be used to refer to a rule class that does not belong to the current schema.
`event_inst`	String or AnyData representation of the event instance being added to the rule class
`event_type`	Type of event instance assigned to the `event_inst` argument when the string representation of the event instance is used for a rule class configured for composite events

Usage Notes

- This procedure is used to process the rules in a rule class for an event instance assigned to the `event_inst` argument.
- In the case of a rule class configured for simple events (non-composite), the event instance is an instantiation of the corresponding event structure. The rules are evaluated (conclusively) for this event and the corresponding action callback procedure is called for each matching rule. If the event does not match any rule, no further action is performed. If the event matches two or more rules, the ordering clause configured for the rule class is used to order them accordingly to invoke the action callback procedure. If the rule class is configured for `EXCLUSIVE` consumption policy, once the first rule in this order is executed (and the corresponding action callback procedure is called), the rest of the rules that matched the event are ignored.
- In the case of a rule class configured for composite events, the event instance assigned to the `event_inst` argument is an instantiation of one of the primitive type within the composite event. When the instance is represented as a string, the corresponding type name should be assigned to the `event_type` argument. The `PROCESS_RULES` call on a rule class configured for composite events performs

various actions depending on the state of the rule class and the kind of rules in the rule class.

- The rules operating only on the primitive event passed in are evaluated conclusively and the action callback procedure is called for the matching rules, as described in previous paragraph.
- In the case of a rule operating on more than one primitive event, the event instance passed through PROCESS_RULES procedure could match only a part of the rule.
  * If there is (are) another primitive event instance(s) that matches the rest of the rule, the current event instance is combined with the other instance(s) to form a complete composite event that matches a rule in the rule class. So, the event instance assigned to the event_inst argument of the PROCESS_RULES procedure could be combined with various other primitive events (previously processed) to evaluate one or more rules conclusively. The rule classes' action callback procedure is called for each such combination of primitive events (composite event) and the rule. The ordering clause for the rule class and the consumption policy for the primitive events in taken into account while invoking the action callback procedure.
  * If there is no other primitive event that matches the rest of the rule, the current event instance and its (incremental) evaluation results are recorded in the database. These results are preserved until either the event is consumed or deleted from the system owing to the duration policy used for the rule class.

Examples

The following command processes the rules in the TravelPromotion rule class for the given events.

```
BEGIN
  DBMS_RLMGR.PROCESS_RULES (
               rule_class => 'TravelPromotion',
               event_inst =>
                 AddFlight.getVarchar(987, 'Abcair', 'Boston', 'Orlando',
                                      '01-APR-2003', '08-APR-2003'));
END;
```

The following commands process the rules in the CompTravelPromo rule class for the two primitive events shown.

```
BEGIN
  DBMS_RLMGR.PROCESS_RULES(
               rule_class => 'CompTravelPromo',
               event_inst =>
                 AddFlight.getVarchar(987, 'Abcair', 'Boston', 'Orlando',
                                      '01-APR-2003', '08-APR-2003'),
               event_type => 'AddFlight');
  DBMS_RLMGR.PROCESS_RULES(
               rule_class => 'Scott.CompTravelPromo',
               event_inst =>
                  AnyData.convertObject(AddRentalCar(987, 'Luxury', '03-APR-2003',
                                        '08-APR-2003', NULL)));
END;
```

RESET_SESSION Procedure

This procedure starts a new session and thus discards the results in the rule class results view.

Syntax

```
DBMS_RLMGR.RESET_SESSION  (
   rule_class  IN  VARCHAR2);
```

Parameters

Table 88–15 RESET_SESSION Procedure Parameter

Parameter	Description
`rule_class`	The name of the rule class. A schema extended rule class name can be used to refer to a rule class that does not belong to the current schema.

Usage Notes

- When the `ADD_EVENT` procedure is used to add events to the rule class, the results from matching rules with events are recorded in the rule class results view. By default, these results are reset at the end of the database session. Alternately, the RESET_SESSION Procedure can be used to reset and start a new rule session within a database session.
- This procedure is only applicable while using ADD_EVENT Procedures to evaluate the rules.

Examples

The following command resets a rule class session.

```
BEGIN
  DBMS_RLMGR.RESET_SESSION(
             rule_class => 'CompTravelPromo');
END;
```

REVOKE_PRIVILEGE Procedure

This procedure revokes privileges on a rule class from another user.

Syntax

```
DBMS_RLMGR.REVOKE_PRIVILEGE  (
   rule_class      IN  VARCHAR2,
   priv_type       IN  VARCHAR2,
   from_user       IN  VARCHAR2);
```

Parameters

Table 88–16 REVOKE_PRIVILEGE Procedure Parameters

Parameter	Description
`rule_class`	The name of the rule class in the current schema
`priv_type`	Type of rule class privilege to be revoked
`from_user`	The user from whom the privilege is to be revoked

Usage Notes

This procedure revokes appropriate privileges from a user. The types of privileges that can be revoked are the same as the types listed in the GRANT_PRIVILEGE Procedure description. Rule class privileges cannot be revoked from the owner of the rule class.

Examples

The following command revokes `PROCESS RULES` privilege on `TravelPromo` rule class from the user `SCOTT`.

```
BEGIN
  DBMS_RLMGR.REVOKE_PRIVILEGE(rule_class  => 'TravelPromo',
                              priv_type   => 'PROCESS RULES',
                              from_user   => 'SCOTT');
END;
```

89

DBMS_RLS

The DBMS_RLS package contains the fine-grained access control administrative interface, which is used to implement Virtual Private Database (VPD). DBMS_RLS is available with the Enterprise Edition only.

> **See Also:** *Oracle Database Security Guide* for usage information on DBMS_RLS.

This chapter contains the following topics:

Using DBMS_RLS

Overview

The functionality to support fine-grained access control is based on dynamic predicates, where security rules are not embedded in views, but are acquired at the statement parse time, when the base table or view is referenced in a DML statement.

A dynamic predicate for a table, view, or synonym is generated by a PL/SQL function, which is associated with a security policy through a PL/SQL interface. For example:

```
DBMS_RLS.ADD_POLICY (
   'hr', 'employees', 'emp_policy', 'hr', 'emp_sec', 'select');
```

Whenever the EMPLOYEES table, under the HR schema, is referenced in a query or subquery (SELECT), the server calls the EMP_SEC function (under the HR schema). This function returns a predicate specific to the current user for the EMP_POLICY policy. The policy function may generate the predicates based on the session environment variables available during the function call. These variables usually appear in the form of application contexts. The policy can specify any combination of security-relevant columns and of these statement types: INDEX, SELECT, INSERT, UPDATE, or DELETE.

The server then produces a transient view with the text:

```
SELECT * FROM hr.employees WHERE P1
```

Here, P1 (for example, where SAL > 10000, or even a subquery) is the predicate returned from the EMP_SEC function. The server treats the EMPLOYEES table as a view and does the view expansion just like the ordinary view, except that the view text is taken from the transient view instead of the data dictionary.

If the predicate contains subqueries, then the owner (definer) of the policy function is used to resolve objects within the subqueries and checks security for those objects. In other words, users who have access privilege to the policy-protected objects do not need to know anything about the policy. They do not need to be granted object privileges for any underlying security policy. Furthermore, the users do not require EXECUTE privilege on the policy function, because the server makes the call with the function definer's right.

> **Note:** The transient view can preserve the updatability of the parent object because it is derived from a single table or view with predicate only; that is, no JOIN, ORDER BY, GROUP BY, and so on.

DBMS_RLS also provides the interface to drop or enable security policies. For example, you can drop or enable the EMP_POLICY with the following PL/SQL statements:

```
DBMS_RLS.DROP_POLICY('hr', 'employees', 'emp_policy');
DBMS_RLS.ENABLE_POLICY('hr', 'employees', 'emp_policy', FALSE);
```

Security Model

A security check is performed when the transient view is created with a subquery. The schema owning the policy function, which generates the dynamic predicate, is the transient view's definer for security check and object lookup.

Operational Notes

The DBMS_RLS procedures cause current DML transactions, if any, to commit before the operation. However, the procedures do not cause a commit first if they are inside a DDL event trigger. With DDL transactions, the DBMS_RLS procedures are part of the DDL transaction.

For example, you may create a trigger for CREATE TABLE. Inside the trigger, you may add a column through ALTER TABLE, and you can add a policy through DBMS_RLS. All these operations are in the same transaction as CREATE TABLE, even though each one is a DDL statement. The CREATE TABLE succeeds only if the trigger is completed successfully.

Views of current cursors and corresponding predicates are available from v$vpd_policies.

A synonym can reference only a view or a table.

Summary of DBMS_RLS Subprograms

Table 89–1 DBMS_RLS Subprograms Package Subprograms

Subprogram	Description
ADD_GROUPED_POLICY Procedure on page 89-7	Adds a policy associated with a policy group
ADD_POLICY Procedure on page 89-9	Adds a fine-grained access control policy to a table, view, or synonym
ADD_POLICY_CONTEXT Procedure on page 89-13	Adds the context for the active application
CREATE_POLICY_GROUP Procedure on page 89-14	Creates a policy group
DELETE_POLICY_GROUP Procedure on page 89-15	Deletes a policy group
DISABLE_GROUPED_POLICY Procedure on page 89-16	Disables a row-level group security policy
DROP_GROUPED_POLICY Procedure on page 89-17	Drops a policy associated with a policy group
DROP_POLICY Procedure on page 89-18	Drops a fine-grained access control policy from a table, view, or synonym
DROP_POLICY_CONTEXT Procedure on page 89-19	Drops a driving context from the object so that it will have one less driving context
ENABLE_GROUPED_POLICY Procedure on page 89-20	Enables or disables a row-level group security policy
ENABLE_POLICY Procedure on page 89-21	Enables or disables a fine-grained access control policy
REFRESH_GROUPED_POLICY Procedure on page 89-22	Reparses the SQL statements associated with a refreshed policy
REFRESH_POLICY Procedure on page 89-23	Causes all the cached statements associated with the policy to be reparsed

ADD_GROUPED_POLICY Procedure

This procedure adds a policy associated with a policy group.

Syntax

```
DBMS_RLS.ADD_GROUPED_POLICY(
   object_schema   VARCHAR2,
   object_name     VARCHAR2,
   policy_group    VARCHAR2,
   policy_name     VARCHAR2,
   function_schema VARCHAR2,
   policy_function VARCHAR2,
   statement_types VARCHAR2,
   update_check    BOOLEAN,
   enabled         BOOLEAN,
   static_policy   IN BOOLEAN  FALSE,
   policy_type     IN BINARY_INTEGER NULL,
   long_predicate  IN BOOLEAN  FALSE,
   sec_relevant_cols IN VARCHAR2);
```

Parameters

Table 89–2 ADD_GROUPED_POLICY Procedure Parameters

Parameter	Description
object_schema	The schema containing the table, view, or synonym.
object_name	The name of the table, view, or synonym to which the policy is added.
policy_group	The name of the policy group that the policy belongs to.
policy_name	The name of the policy; must be unique for the same table or view.
function_schema	The schema owning the policy function.
policy_function	The name of the function that generates a predicate for the policy. If the function is defined within a package, the name of the package must be present.
statement_types	Statement types to which the policy applies. It can be any combination of INDEX, SELECT, INSERT, UPDATE, or DELETE. The default is to apply to all of these types except INDEX.
update_check	For INSERT and UPDATE statements only, setting update_check to TRUE causes the server to check the policy against the value after INSERT or UPDATE.
enable	Indicates if the policy is enable when it is added. The default is TRUE.
static_policy	The default is FALSE. If it is set to TRUE, the server assumes that the policy function for the static policy produces the same predicate string for anyone accessing the object, except for SYS or the privilege user who has the EXEMPT ACCESS POLICY privilege.
policy_type	Default is NULL, which means policy_type is decided by the value of static_policy. The available policy types are listed in Table 89–4. Specifying any of these policy types overrides the value of static_policy.

Table 89–2 (Cont.) ADD_GROUPED_POLICY Procedure Parameters

Parameter	Description
long_predicate	Default is FALSE, which means the policy function can return a predicate with a length of up to 4000 bytes. TRUE means the predicate text string length can be up to 32K bytes.Policies existing prior to the availability of this parameter retain a 32K limit.
sec_relevant_cols	Enables column-level Virtual Private Database (VPD), which enforces security policies when a column containing sensitive information is referenced in a query. Applies to tables and views, but not to synonyms. Specify a list of comma- or space-separated valid column names of the policy-protected object. The policy is enforced only if a specified column is referenced (or, for an abstract datatype column, its attributes are referenced) in the user SQL statement or its underlying view definition. Default is all the user-defined columns for the object.
sec_relevant_cols_opt	Use with sec_relevant_cols to display all rows for column-level VPD filtered queries (SELECT only), but where sensitive columns appear as NULL. Default is set to NULL, which allows the filtering defined with sec_relevant_cols to take effect. Set to dbms_rls.ALL_ROWS to display all rows, but with sensitive column values, which are filtered by sec_relevant_cols, displayed as NULL. See "Usage Notes" on page 89-11 for restrictions and additional information about this option.

Usage Notes

- This procedure adds a policy to the specified table, view, or synonym and associates the policy with the specified policy group.
- The policy group must have been created by using the CREATE_POLICY_GROUP Procedure on page 89-14.
- The policy name must be unique within a policy group for a specific object.
- Policies from the default policy group, SYS_DEFAULT, are always executed regardless of the active policy group; however, fine-grained access control policies do not apply to users with EXEMPT ACCESS POLICY system privilege.
- If no object_schema is specified, the current log-on user schema is assumed.

ADD_POLICY Procedure

This procedure adds a fine-grained access control policy to a table, view, or synonym.

The procedure causes the current transaction, if any, to commit before the operation is carried out. However, this does not cause a commit first if it is inside a DDL event trigger.

See Also: Operational Notes on page 89-5

A COMMIT is also performed at the end of the operation.

Syntax

```
DBMS_RLS.ADD_POLICY (
   object_schema            IN VARCHAR2 NULL,
   object_name              IN VARCHAR2,
   policy_name              IN VARCHAR2,
   function_schema          IN VARCHAR2 NULL,
   policy_function          IN VARCHAR2,
   statement_types          IN VARCHAR2 NULL,
   update_check             IN BOOLEAN  FALSE,
   enable                   IN BOOLEAN  TRUE,
   static_policy            IN BOOLEAN  FALSE,
   policy_type              IN BINARY_INTEGER NULL,
   long_predicate           IN BOOLEAN  FALSE,
   sec_relevant_cols        IN VARCHAR2,
   sec_relevant_cols_opt    IN BINARY_INTEGER NULL);
```

Parameters

Table 89–3 ADD_POLICY Procedure Parameters

Parameter	Description
object_schema	Schema containing the table, view, or synonym. If no object_ schema is specified, the current log-on user schema is assumed.
object_name	Name of table, view, or synonym to which the policy is added.
policy_name	Name of policy to be added. It must be unique for the same table or view.
function_schema	Schema of the policy function (current default schema, if NULL).
policy_function	Name of a function which generates a predicate for the policy. If the function is defined within a package, then the name of the package must be present.
statement_types	Statement types to which the policy applies. It can be any combination of INDEX, SELECT, INSERT, UPDATE, or DELETE. The default is to apply to all of these types except INDEX.
update_check	Optional argument for INSERT or UPDATE statement types. The default is FALSE. Setting update_check to TRUE causes the server to also check the policy against the value after insert or update.
enable	Indicates if the policy is enabled when it is added. The default is TRUE.

Table 89–3 (Cont.) ADD_POLICY Procedure Parameters

Parameter	Description
static_policy	The default is FALSE. If it is set to TRUE, the server assumes that the policy function for the static policy produces the same predicate string for anyone accessing the object, except for SYS or the privilege user who has the EXEMPT ACCESS POLICY privilege.
policy_type	Default is NULL, which means policy_type is decided by the value of static_policy. The available policy types are listed in Table 89–4. Specifying any of these policy types overrides the value of static_policy.
long_predicate	Default is FALSE, which means the policy function can return a predicate with a length of up to 4000 bytes. TRUE means the predicate text string length can be up to 32K bytes.Policies existing prior to the availability of this parameter retain a 32K limit.
sec_relevant_cols	Enables column-level Virtual Private Database (VPD), which enforces security policies when a column containing sensitive information is referenced in a query. Applies to tables and views, but not to synonyms. Specify a list of comma- or space-separated valid column names of the policy-protected object. The policy is enforced only if a specified column is referenced (or, for an abstract datatype column, its attributes are referenced) in the user SQL statement or its underlying view definition. Default is all the user-defined columns for the object.
sec_relevant_cols_opt	Use with sec_relevant_cols to display all rows for column-level VPD filtered queries (SELECT only), but where sensitive columns appear as NULL. Default is set to NULL, which allows the filtering defined with sec_relevant_cols to take effect. Set to dbms_rls.ALL_ROWS to display all rows, but with sensitive column values, which are filtered by sec_relevant_cols, displayed as NULL. See "Usage Notes" on page 89-11 for restrictions and additional information about this option.

Table 89–4 DBMS_RLS.ADD_POLICY Policy Types

Policy Type	Description
STATIC	Predicate is assumed to be the same regardless of the runtime environment. Static policy functions are executed once and then cached in SGA. Statements accessing the same object do not reexecute the policy function. However, each execution of the same cursor could produce a different row set even for the same predicate because the predicate may filter the data differently based on attributes such as SYS_CONTEXT or SYSDATE. Applies to only one object.
SHARED_STATIC	Same as STATIC except that the server first looks for a cached predicate generated by the same policy function of the same policy type. Shared across multiple objects.
CONTEXT_SENSITIVE	Server re-evaluates the policy function at statement execution time if it detects context changes since the last use of the cursor. For session pooling where multiple clients share a database session, the middle tier must reset context during client switches. Note that the server does not cache the value returned by the function for this policy type; it always executes the policy function on statement parsing. Applies to only one object.
SHARED_CONTEXT_SENSITIVE	Same as CONTEXT_SENSITIVE except that the server first looks for a cached predicate generated by the same policy function of the same policy type within the same database session. If the predicate is found in the session memory, the policy function is not reexecuted and the cached value is valid until session private application context changes occur. Shared across multiple objects.
DYNAMIC	The default policy type. Server assumes the predicate may be affected by any system or session environment at any time, and so always reexecutes the policy function on each statement parsing or execution. Applies to only one object.

Usage Notes

- SYS is free of any security policy.
- If no object_schema is specified, the current log-on user schema is assumed.
- The policy functions which generate dynamic predicates are called by the server. Following is the interface for the function:

```
FUNCTION policy_function (object_schema IN VARCHAR2, object_name VARCHAR2)
    RETURN VARCHAR2
--- object_schema is the schema owning the table of view.
--- object_name is the name of table, view, or synonym to which the policy
applies.
```

- The policy functions must have the purity level of WNDS (write no database state).

  See Also: The *Oracle Database Application Developer's Guide - Fundamentals* has more details about the RESTRICT_REFERENCES pragma.

- Dynamic predicates generated out of different policies for the same object have the combined effect of a conjunction (ANDed) of all the predicates.
- The security check and object lookup are performed against the owner of the policy function for objects in the subqueries of the dynamic predicates.
- If the function returns a zero length predicate, then it is interpreted as no restriction being applied to the current user for the policy.
- When a table alias is required (for example, parent object is a type table) in the predicate, the name of the table or view itself must be used as the name of the alias. The server constructs the transient view as something like

```
"select c1, c2, ... from tab tab where <predicate>"
```

- Validity of the function is checked at runtime for ease of installation and other dependency issues during import and export.
- Column-level VPD column masking behavior (specified with sec_relevant_cols_opt => dbms_rls.ALL_ROWS) is fundamentally different from all other VPD policies, which return only a subset of rows. Instead the column masking behavior returns all rows specified by the user's query, but the sensitive column values display as NULL. The restrictions for this option are as follows:
  - Only applies to SELECT statements
  - Unlike regular VPD predicates, the masking condition that is generated by the policy function must be a simple boolean expression.
  - If your application performs calculations, or does not expect NULL values, then you should use the default behavior of column-level VPD, which is specified with the sec_relevant_cols parameter.
  - If you use UPDATE AS SELECT with this option, then only the values in the columns you are allowed to see will be updated.
  - This option may prevent some rows from displaying. For example:

    ```
    select * from employees
    where salary = 10
    ```

    This query may not return rows if the salary column returns a NULL value because the column masking option has been set.

Examples

As the first of two examples, the following creates a policy that applies to the `hr.employee` table. This is a column-level VPD policy that will be enforced only if a `SELECT` or an `INDEX` statement refers to the `salary`, `birthdate`, or `SSN` columns of the table explicitly, or implicitly through a view. It is also a `CONTEXT_SENSITIVE` policy, so the server will invoke the policy function `hr.hrfun` at parse time. During execution, it will only invoke the function if there has been any session private context change since the last use of the statement cursor. The predicate generated by the policy function must not exceed 4000 bytes, the default length limit, since the `long_predicate` parameter is omitted from the call.

```
BEGIN
dbms_rls.add_policy(object_schema => 'hr',
object_name => 'employee',
policy_name => 'hr_policy',
function_schema =>'hr',
policy_function => 'hrfun',
statement_types =>'select,index',
policy_type => dbms_rls.CONTEXT_SENSITIVE,
sec_relevant_cols=>'salary,birthdate,ssn');
END;
/
```

As the second example, the following command creates another policy that applies to the same object for hosting, so users can access only data based on their subscriber ID. Since it is defined as a `SHARED_STATIC` policy type, the server will first try to find the predicate in the SGA cache. The server will only invoke the policy function, `subfun`, if that search fails.

```
BEGIN
dbms_rls.add_policy(object_schema => 'hr',
object_name => 'employee',
policy_name => 'hosting_policy',
function_schema =>'hr',
policy_function => 'subfun',
policy_type => dbms_rls.SHARED_STATIC);
END;
/
```

ADD_POLICY_CONTEXT Procedure

This procedure adds the context for the active application.

Syntax

```
DBMS_RLS.ADD_POLICY_CONTEXT (
   object_schema   VARCHAR2,
   object_name     VARCHAR2,
   namespace       VARCHAR2,
   attribute       VARCHAR2);
```

Parameters

Table 89–5 ADD_POLICY_CONTEXT Procedure Parameters

Parameter	Description
`object_schema`	The schema containing the table, view, or synonym.
`object_name`	The name of the table, view, or synonym to which the policy is added.
`namespace`	The namespace of the driving context
`attribute`	The attribute of the driving context.

Usage Notes

Note the following:

- This procedure indicates the application context that drives the enforcement of policies; this is the context that determines which application is running.
- If no object_schema is specified, the current log-on user schema is assumed.
- The driving context can be session or global.
- At execution time, the server retrieves the name of the active policy group from the value of this context.
- There must be at least one driving context defined for each object that has fine-grained access control policies; otherwise, all policies for the object will be executed.
- Adding multiple context to the same object will cause policies from multiple policy groups to be enforced.
- If the driving context is `NULL`, policies from all policy groups are used.
- If the driving context is a policy group with policies, all enabled policies from that policy group will be applied, along with all policies from the `SYS_DEFAULT` policy group.
- To add a policy to table `hr.employees` in group `access_control_group`, the following command is issued:

```
DBMS_RLS.ADD_GROUPED_POLICY('hr','employees','access_control_
group','policy1','SYS', 'HR.ACCESS');
```

CREATE_POLICY_GROUP Procedure

This procedure creates a policy group.

Syntax

```
DBMS_RLS.CREATE_POLICY_GROUP (
   object_schema   VARCHAR2,
   object_name     VARCHAR2,
   policy_group    VARCHAR2);
```

Parameters

Table 89–6 CREATE_POLICY_GROUP Procedure Parameters

Parameter	Description
`object_schema`	Schema containing the table, view, or synonym.
`object_name`	Name of the table, view, or synonym to which the policy is added.
`policy_group`	Name of the policy group that the policy belongs to.

Usage Notes

The group must be unique for each table or view.

DELETE_POLICY_GROUP Procedure

This procedure deletes a policy group.

Syntax

```
DBMS_RLS.DELETE_POLICY_GROUP (
  object_schema   VARCHAR2,
  object_name     VARCHAR2,
  policy_group    VARCHAR2);
```

Parameters

Table 89–7 DELETE_POLICY_GROUP Procedure Parameters

Parameter	Description
object_schema	The schema containing the table, view, or synonym.
object_name	The name of the table, view, or synonym to which the policy is added.
policy_group	The name of the policy group that the policy belongs to.

Usage Notes

Note the following:

- This procedure deletes a policy group for the specified table, view, or synonym.
- No policy can be in the policy group.

DISABLE_GROUPED_POLICY Procedure

This procedure disables a row-level group security policy.

Syntax

```
DBMS_RLS.DISABLE_GROUPED_POLICY (
   object_schema   VARCHAR2,
   object_name     VARCHAR2,
   group_name      VARCHAR2,
   policy_name     VARCHAR2);
```

Parameters

Table 89–8 ENABLE_GROUPED_POLICY Procedure Parameters

Parameter	Description
`object_schema`	The schema containing the table, view, or synonym.
`object_name`	The name of the table, view, or synonym with which the policy is associated.
`group_name`	The name of the group of the policy.
`policy_name`	The name of the policy to be enabled or disabled.

Usage Notes

- The procedure causes the current transaction, if any, to commit before the operation is carried out.
- A commit is performed at the end of the operation.
- A policy is disabled when this procedure is executed or when the ENABLE_GROUPED_POLICY procedure is executed with "enable" set to FALSE.

DROP_GROUPED_POLICY Procedure

This procedure drops a policy associated with a policy group.

Syntax

```
DBMS_RLS.DROP_GROUPED_POLICY (
   object_schema   VARCHAR2,
   object_name     VARCHAR2,
   policy_group    VARCHAR2,
   policy_name     VARCHAR2);
```

Parameters

Table 89–9 DROP_GROUPED_POLICY Procedure Parameters

Parameter	Description
`object_schema`	The schema containing the table, view, or synonym.
`object_name`	The name of the table, view, or synonym to which the policy is dropped.
`policy_group`	The name of the policy group that the policy belongs to.
`policy_name`	The name of the policy.

DROP_POLICY Procedure

This procedure drops a fine-grained access control policy from a table, view, or synonym.

The procedure causes the current transaction, if any, to commit before the operation is carried out. However, this does not cause a commit first if it is inside a DDL event trigger.

See Also: Operational Notes on page 89-5

A `COMMIT` is also performed at the end of the operation.

Syntax

```
DBMS_RLS.DROP_POLICY (
   object_schema   IN VARCHAR2 NULL,
   object_name     IN VARCHAR2,
   policy_name     IN VARCHAR2);
```

Parameters

Table 89–10 DROP_GROUPED_POLICY Procedure Parameters

Parameter	Description
`object_schema`	Schema containing the table, view or synonym (current default schema if `NULL`)..
`object_name`	Name of the table, view, or synonym for which the policy is dropped.
`policy_name`	Name of policy to be dropped from table, view, or synonym.

DROP_POLICY_CONTEXT Procedure

This procedure drops a driving context from the object so that it will have one less driving context.

Syntax

```
DBMS_RLS.DROP_POLICY_CONTEXT (
   object_schema   VARCHAR2,
   object_name     VARCHAR2,
   namespace       VARCHAR2,
   attribute       VARCHAR2);
```

Parameters

Table 89–11 DROP_POLICY_CONTEXT Procedure Parameters

Parameter	Description
`object_schema`	The schema containing the table, view, or synonym.
`object_name`	The name of the table, view, or synonym to which the policy is dropped.
`namespace`	The namespace of the driving context.
`attribute`	The attribute of the driving context.

ENABLE_GROUPED_POLICY Procedure

This procedure enables or disables a row-level group security policy.

Syntax

```
DBMS_RLS.ENABLE_GROUPED_POLICY (
   object_schema   VARCHAR2,
   object_name     VARCHAR2,
   group_name      VARCHAR2,
   policy_name     VARCHAR2,
   enable          BOOLEAN);
```

Parameters

Table 89–12 ENABLE_GROUPED_POLICY Procedure Parameters

Parameter	Description
`object_schema`	The schema containing the table, view, or synonym.
`object_name`	The name of the table, view, or synonym with which the policy is associated.
`group_name`	The name of the group of the policy.
`policy_name`	The name of the policy to be enabled or disabled.
`enable`	`TRUE` enables the policy; `FALSE` disables the policy.

Usage Notes

- The procedure causes the current transaction, if any, to commit before the operation is carried out.
- A commit is performed at the end of the operation.
- A policy is enabled when it is created.

ENABLE_POLICY Procedure

This procedure enables or disables a fine-grained access control policy. A policy is enabled when it is created.

The procedure causes the current transaction, if any, to commit before the operation is carried out. However, this does not cause a commit first if it is inside a DDL event trigger.

See Also: Operational Notes on page 89-5

A `COMMIT` is also performed at the end of the operation.

Syntax

```
DBMS_RLS.ENABLE_POLICY (
   object_schema IN VARCHAR2 NULL,
   object_name   IN VARCHAR2,
   policy_name   IN VARCHAR2,
   enable        IN BOOLEAN);
```

Parameters

Table 89–13 ENABLE_POLICY Procedure Parameters

Parameter	Description
`object_schema`	Schema containing table, view, or synonym (current default schema if `NULL`).
`object_name`	Name of table, view, or synonym with which the policy is associated.
`policy_name`	Name of policy to be enabled or disabled.
`enable`	`TRUE` to enable the policy, `FALSE` to disable the policy.

REFRESH_GROUPED_POLICY Procedure

This procedure reparses the SQL statements associated with a refreshed policy.

Syntax

```
DBMS_RLS.REFRESH_GROUPED_POLICY (
   object_schema   VARCHAR2,
   object_name     VARCHAR2,
   group_name      VARCHAR2,
   policy_name     VARCHAR2);
```

Parameters

Table 89–14 REFRESH_GROUPED_POLICY Procedure Parameters

Parameter	Description
`object_schema`	The schema containing the table, view, or synonym.
`object_name`	The name of the table, view, or synonym with which the policy is associated.
`group_name`	The name of the group of the policy.
`policy_name`	The name of the policy.

Usage Notes

- This procedure causes all the cached statements associated with the policy to be reparsed. This guarantees that the latest change to the policy has immediate effect after the procedure is executed.
- The procedure causes the current transaction, if any, to commit before the operation is carried out.
- A commit is performed at the end of the operation.
- The procedure returns an error if it tries to refresh a disabled policy.

REFRESH_POLICY Procedure

This procedure causes all the cached statements associated with the policy to be reparsed. This guarantees that the latest change to this policy will have immediate effect after the procedure is executed.

The procedure causes the current transaction, if any, to commit before the operation is carried out. However, this does not cause a commit first if it is inside a DDL event trigger.

See Also: Operational Notes on page 89-5

A COMMIT is also performed at the end of the operation.

Syntax

```
DBMS_RLS.REFRESH_POLICY (
   object_schema IN VARCHAR2 NULL,
   object_name   IN VARCHAR2 NULL,
   policy_name   IN VARCHAR2 NULL);
```

Parameters

Table 89–15 REFRESH_POLICY Procedure Parameters

Parameter	Description
`object_schema`	Schema containing the table, view, or synonym.
`object_name`	Name of table, view, or synonym with which the policy is associated.
`policy_name`	Name of policy to be refreshed.

Usage Notes

The procedure returns an error if it tries to refresh a disabled policy.

90

DBMS_ROWID

The DBMS_ROWID package lets you create ROWIDs and obtain information about ROWIDs from PL/SQL programs and SQL statements. You can find the data block number, the object number, and other ROWID components without writing code to interpret the base-64 character external ROWID. DBMS_ROWID is intended for upgrading from Oracle database version 7 to Oracle database version 8.X.

> **Note:** DBMS_ROWID is not to be used with universal ROWIDs (UROWIDs).

This chapter contains the following topics:

Using DBMS_ROWID

Security Model

This package runs with the privileges of calling user, rather than the package owner `SYS`.

Types

Extension and Restriction Types

The types are as follows:

- RESTRICTED—restricted ROWID
- EXTENDED—extended ROWID

For example:

```
rowid_type_restricted constant integer := 0;
rowid_type_extended   constant integer := 1;
```

Note: Extended ROWIDs are only used in Oracle database version 8.X*i* and higher.

Verification Types

Table 90–1 Verification Types

Result	Description
VALID	Valid ROWID
INVALID	Invalid ROWID

For example:

```
rowid_is_valid   constant integer := 0;
rowid_is_invalid constant integer := 1;
```

Object Types

Table 90–2 Object Types

Result	Description
UNDEFINED	Object Number not defined (for restricted ROWIDs)

For example:

```
rowid_object_undefined constant integer := 0;
```

Conversion Types

Table 90–3 Conversion Types

Result	Description
INTERNAL	Convert to/from column of ROWID type

Table 90–3 (Cont.) Conversion Types

Result	Description
EXTERNAL	Convert to/from string format

For example:

```
rowid_convert_internal constant integer := 0;
rowid_convert_external constant integer := 1;
```

Exceptions

Table 90–4 Exceptions

Exception	Description
ROWID_INVALID	Invalid rowid format
ROWID_BAD_BLOCK	Block is beyond end of file

For example:

```
ROWID_INVALID exception;
   pragma exception_init(ROWID_INVALID, -1410);

ROWID_BAD_BLOCK exception;
   pragma exception_init(ROWID_BAD_BLOCK, -28516);
```

Operational Notes

- Some of the functions in this package take a single parameter, such as a ROWID. This can be a character or a PL/SLQ ROWID, either restricted or extended, as required.
- You can call the DBMS_ROWID functions and procedures from PL/SQL code, and you can also use the functions in SQL statements.

  Note: ROWID_INFO is a procedure. It can only be used in PL/SQL code.

- You can use functions from the DBMS_ROWID package just like built-in SQL functions; in other words, you can use them wherever you can use an expression. In this example, the ROWID_BLOCK_NUMBER function is used to return just the block number of a single row in the EMP table:

```
SELECT DBMS_ROWID.ROWID_BLOCK_NUMBER(rowid)
   FROM emp
   WHERE ename = 'KING';
```

- If Oracle returns the error "ORA:452, 0, 'Subprogram '%s' violates its associated pragma' for pragma restrict_references, it could mean the violation is due to:
  - A problem with the current procedure or function
  - Calling a procedure or function without a pragma or due to calling one with a less restrictive pragma
  - Calling a package procedure or function that touches the initialization code in a package or that sets the default values

Examples

This example returns the ROWID for a row in the EMP table, extracts the data object number from the ROWID, using the ROWID_OBJECT function in the DBMS_ROWID package, then displays the object number:

```
DECLARE
  object_no   INTEGER;
  row_id      ROWID;
  ...
BEGIN
  SELECT ROWID INTO row_id FROM emp
    WHERE empno = 7499;
  object_no := DBMS_ROWID.ROWID_OBJECT(row_id);
  DBMS_OUTPUT.PUT_LINE('The obj. # is '|| object_no);
  ...
```

Summary of DBMS_ROWID Subprograms

Table 90–5 DBMS_ROWID Package Subprograms

Subprogram	Description
ROWID_BLOCK_NUMBER Function on page 90-10	Returns the block number of a `ROWID`
ROWID_CREATE Function on page 90-11	Creates a `ROWID`, for testing only
ROWID_INFO Procedure on page 90-12	Returns the type and components of a `ROWID`
ROWID_OBJECT Function on page 90-13	Returns the object number of the extended `ROWID`
ROWID_RELATIVE_FNO Function on page 90-14	Returns the file number of a `ROWID`
ROWID_ROW_NUMBER Function on page 90-15	Returns the row number
ROWID_TO_ABSOLUTE_FNO Function on page 90-16	Returns the absolute file number associated with the `ROWID` for a row in a specific table
ROWID_TO_EXTENDED Function on page 90-17	Converts a `ROWID` from restricted format to extended
ROWID_TO_RESTRICTED Function on page 90-19	Converts an extended `ROWID` to restricted format
ROWID_TYPE Function on page 90-20	Returns the `ROWID` type: 0 is restricted, 1 is extended
ROWID_VERIFY Function on page 90-21	Checks if a `ROWID` can be correctly extended by the `ROWID_TO_EXTENDED` function

ROWID_BLOCK_NUMBER Function

This function returns the database block number for the input `ROWID`.

Syntax

```
DBMS_ROWID.ROWID_BLOCK_NUMBER (
   row_id      IN   ROWID,
   ts_type_in  IN   VARCHAR2 DEFAULT 'SMALLFILE')
  RETURN NUMBER;
```

Pragmas

```
   pragma RESTRICT_REFERENCES(rowid_block_number,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 90–6 ROWID_BLOCK_NUMBER Function Parameters

Parameter	Description
`row_id`	`ROWID` to be interpreted.
`ts_type_in`	The type of the tablespace (bigfile/smallfile) to which the row belongs.

Examples

The example SQL statement selects the block number from a `ROWID` and inserts it into another table:

```
INSERT INTO T2 (SELECT dbms_rowid.rowid_block_number(ROWID, 'BIGFILE')
   FROM some_table
   WHERE key_value = 42);
```

ROWID_CREATE Function

This function lets you create a ROWID, given the component parts as parameters.

This is useful for testing ROWID operations, because only the Oracle Server can create a valid ROWID that points to data in a database.

Syntax

```
DBMS_ROWID.ROWID_CREATE (
   rowid_type    IN NUMBER,
   object_number IN NUMBER,
   relative_fno  IN NUMBER,
   block_number  IN NUMBER,
   row_number    IN NUMBER)
  RETURN ROWID;
```

Pragmas

```
pragma RESTRICT_REFERENCES(rowid_create,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 90–7 ROWID_CREATE Function Parameters

Parameter	Description
rowid_type	Type (restricted or extended). Set the rowid_type parameter to 0 for a restricted ROWID. Set it to 1 to create an extended ROWID. If you specify rowid_type as 0, then the required object_number parameter is ignored, and ROWID_CREATE returns a restricted ROWID.
object_number	Data object number (rowid_object_undefined for restricted).
relative_fno	Relative file number.
block_number	Block number in this file.
row_number	Returns row number in this block.

Examples

Create a dummy extended ROWID:

```
my_rowid := DBMS_ROWID.ROWID_CREATE(1, 9999, 12, 1000, 13);
```

Find out what the rowid_object function returns:

```
obj_number := DBMS_ROWID.ROWID_OBJECT(my_rowid);
```

The variable obj_number now contains 9999.

ROWID_INFO Procedure

This procedure returns information about a ROWID, including its type (restricted or extended), and the components of the ROWID. This is a procedure, and it cannot be used in a SQL statement.

Syntax

```
DBMS_ROWID.ROWID_INFO (
   rowid_in         IN   ROWID,
   ts_type_in       IN   VARCHAR2 DEFAULT 'SMALLFILE',
   rowid_type       OUT  NUMBER,
   object_number    OUT  NUMBER,
   relative_fno     OUT  NUMBER,
   block_number     OUT  NUMBER,
   row_number       OUT  NUMBER);
```

Pragmas

```
   pragma RESTRICT_REFERENCES(rowid_info,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 90–8 ROWID_INFO Procedure Parameters

Parameter	Description
rowid_in	ROWID to be interpreted. This determines if the ROWID is a restricted (0) or extended (1) ROWID.
ts_type_in	The type of the tablespace (bigfile/smallfile) to which the row belongs.
rowid_type	Returns type (restricted/extended).
object_number	Returns data object number (rowid_object_undefined for restricted).
relative_fno	Returns relative file number.
block_number	Returns block number in this file.
row_number	Returns row number in this block.

See Also: "ROWID_TYPE Function" on page 90-20

Examples

This example reads back the values for the ROWID that you created in the ROWID_CREATE:

```
DBMS_ROWID.ROWID_INFO(my_rowid, 'BIGFILE', rid_type, obj_num, file_num, block_num,
row_num);

DBMS_OUTPUT.PUT_LINE('The type is ' || rid_type);
DBMS_OUTPUT.PUT_LINE('Data object number is ' || obj_num);
-- and so on...
```

ROWID_OBJECT Function

This function returns the data object number for an extended ROWID. The function returns zero if the input ROWID is a restricted ROWID.

Syntax

```
DBMS_ROWID.ROWID_OBJECT (
   rowid_id IN ROWID)
  RETURN NUMBER;
```

Pragmas

```
   pragma RESTRICT_REFERENCES(rowid_object,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 90–9 ROWID_OBJECT Function Parameters

Parameter	Description
row_id	ROWID to be interpreted.

Note: The ROWID_OBJECT_UNDEFINED constant is returned for restricted ROWIDs.

Examples

```
SELECT dbms_rowid.rowid_object(ROWID)
   FROM emp
   WHERE empno = 7499;
```

ROWID_RELATIVE_FNO Function

This function returns the relative file number of the ROWID specified as the IN parameter. (The file number is relative to the tablespace.)

Syntax

```
DBMS_ROWID.ROWID_RELATIVE_FNO (
   rowid_id      IN   ROWID,
   ts_type_in    IN   VARCHAR2 DEFAULT 'SMALLFILE')
  RETURN NUMBER;
```

Pragmas

```
   pragma RESTRICT_REFERENCES(rowid_relative_fno,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 90–10 ROWID_RELATIVE_FNO Function Parameters

Parameter	Description
row_id	ROWID to be interpreted.
ts_type_in	The type of the tablespace (bigfile/smallfile) to which the row belongs.

Examples

The example PL/SQL code fragment returns the relative file number:

```
DECLARE
   file_number    INTEGER;
   rowid_val      ROWID;
BEGIN
   SELECT ROWID INTO rowid_val
     FROM dept
     WHERE loc = 'Boston';
   file_number :=
     dbms_rowid.rowid_relative_fno(rowid_val, 'SMALLFILE');
    ...
```

ROWID_ROW_NUMBER Function

This function extracts the row number from the `ROWID IN` parameter.

Syntax

```
DBMS_ROWID.ROWID_ROW_NUMBER (
   row_id IN ROWID)
  RETURN NUMBER;
```

Pragmas

```
   PRAGMA RESTRICT_REFERENCES(rowid_row_number,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 90–11 ROWID_ROW_NUMBER Function Parameters

Parameter	Description
`row_id`	`ROWID` to be interpreted.

Examples

Select a row number:

```
SELECT dbms_rowid.rowid_row_number(ROWID)
   FROM emp
   WHERE ename = 'ALLEN';
```

ROWID_TO_ABSOLUTE_FNO Function

This function extracts the absolute file number from a ROWID, where the file number is absolute for a row in a given schema and table. The schema name and the name of the schema object (such as a table name) are provided as IN parameters for this function.

Syntax

```
DBMS_ROWID.ROWID_TO_ABSOLUTE_FNO (
   row_id      IN ROWID,
   schema_name IN VARCHAR2,
   object_name IN VARCHAR2)
  RETURN NUMBER;
```

Pragmas

```
   pragma RESTRICT_REFERENCES(rowid_to_absolute_fno,WNDS,WNPS,RNPS);
```

Parameters

Table 90–12 ROWID_TO_ABSOLUTE_FNO Function Parameters

Parameter	Description
row_id	ROWID to be interpreted.
schema_name	Name of the schema which contains the table.
object_name	Table name.

Examples

```
DECLARE
   abs_fno        INTEGER;
   rowid_val      CHAR(18);
   object_name    VARCHAR2(20) := 'EMP';
BEGIN
    SELECT ROWID INTO rowid_val
     FROM emp
     WHERE empno = 9999;
     abs_fno := dbms_rowid.rowid_to_absolute_fno(
     rowid_val, 'SCOTT', object_name);
```

Note: For partitioned objects, the name must be a table name, not a partition or a sub/partition name.

ROWID_TO_EXTENDED Function

This function translates a restricted ROWID that addresses a row in a schema and table that you specify to the extended ROWID format. Later, it may be removed from this package into a different place.

Syntax

```
DBMS_ROWID.ROWID_TO_EXTENDED (
   old_rowid       IN ROWID,
   schema_name     IN VARCHAR2,
   object_name     IN VARCHAR2,
   conversion_type IN INTEGER)
  RETURN ROWID;
```

Pragmas

```
   pragma RESTRICT_REFERENCES(rowid_to_extended,WNDS,WNPS,RNPS);
```

Parameters

Table 90–13 ROWID_TO_EXTENDED Function Parameters

Parameter	Description
old_rowid	ROWID to be converted.
schema_name	Name of the schema which contains the table (optional).
object_name	Table name (optional).
conversion_type	The following constants are defined: rowid_convert_internal (:=0) rowid_convert_external (:=1)

Return Values

ROWID_TO_EXTENDED returns the ROWID in the extended character format. If the input ROWID is NULL, then the function returns NULL. If a zero-valued ROWID is supplied (00000000.0000.0000), then a zero-valued restricted ROWID is returned.

Examples

Assume that there is a table called RIDS in the schema SCOTT, and that the table contains a column ROWID_COL that holds ROWIDs (restricted), and a column TABLE_COL that point to other tables in the SCOTT schema. You can convert the ROWIDs to extended format with the statement:

```
UPDATE SCOTT.RIDS
   SET rowid_col =
   dbms_rowid.rowid_to_extended (
      rowid_col, 'SCOTT", TABLE_COL, 0);
```

Usage Notes

If the schema and object names are provided as IN parameters, then this function verifies SELECT authority on the table named, and converts the restricted ROWID provided to an extended ROWID, using the data object number of the table. That ROWID_TO_EXTENDED returns a value, however, does not guarantee that the

converted ROWID actually references a valid row in the table, either at the time that the function is called, or when the extended ROWID is actually used.

If the schema and object name are not provided (are passed as NULL), then this function attempts to fetch the page specified by the restricted ROWID provided. It treats the file number stored in this ROWID as the absolute file number. This can cause problems if the file has been dropped, and its number has been reused prior to the migration. If the fetched page belongs to a valid table, then the data object number of this table is used in converting to an extended ROWID value. This is very inefficient, and Oracle recommends doing this only as a last resort, when the target table is not known. The user must still know the correct table name at the time of using the converted value.

If an extended ROWID value is supplied, the data object number in the input extended ROWID is verified against the data object number computed from the table name parameter. If the two numbers do not match, the INVALID_ROWID exception is raised. If they do match, the input ROWID is returned.

> **See Also:** The ROWID_VERIFY Function has a method to determine if a given ROWID can be converted to the extended format.

ROWID_TO_RESTRICTED Function

This function converts an extended ROWID into restricted ROWID format.

Syntax

```
DBMS_ROWID.ROWID_TO_RESTRICTED (
   old_rowid       IN ROWID,
   conversion_type IN INTEGER)
  RETURN ROWID;
```

Pragmas

```
   pragma RESTRICT_REFERENCES(rowid_to_restricted,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 90–14 ROWID_TO_RESTRICTED Function Parameters

Parameter	Description
old_rowid	ROWID to be converted.
conversion_type	The following constants are defined: rowid_convert_internal (:=0) rowid_convert_external (:=1)

ROWID_TYPE Function

This function returns 0 if the ROWID is a restricted ROWID, and 1 if it is extended.

Syntax

```
DBMS_ROWID.ROWID_TYPE (
   rowid_id IN ROWID)
  RETURN NUMBER;
```

Pragmas

```
   pragma RESTRICT_REFERENCES(rowid_type,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 90–15 ROWID_TYPE Function Parameters

Parameter	Description
row_id	ROWID to be interpreted.

Examples

```
IF DBMS_ROWID.ROWID_TYPE(my_rowid) = 1 THEN
   my_obj_num := DBMS_ROWID.ROWID_OBJECT(my_rowid);
```

ROWID_VERIFY Function

This function verifies the ROWID. It returns 0 if the input restricted ROWID can be converted to extended format, given the input schema name and table name, and it returns 1 if the conversion is not possible.

> **Note:** You can use this function in a WHERE clause of a SQL statement, as shown in the example.

Syntax

```
DBMS_ROWID.ROWID_VERIFY (
   rowid_in        IN ROWID,
   schema_name     IN VARCHAR2,
   object_name     IN VARCHAR2,
   conversion_type IN INTEGER
  RETURN NUMBER;
```

Pragmas

```
   pragma RESTRICT_REFERENCES(rowid_verify,WNDS,WNPS,RNPS);
```

Parameters

Table 90–16 ROWID_VERIFY Function Parameters

Parameter	Description
rowid_in	ROWID to be verified.
schema_name	Name of the schema which contains the table.
object_name	Table name.
conversion_type	The following constants are defined: rowid_convert_internal (:=0) rowid_convert_external (:=1)

Examples

Considering the schema in the example for the ROWID_TO_EXTENDED function, you can use the following statement to find bad ROWIDs prior to conversion. This enables you to fix them beforehand.

```
SELECT ROWID, rowid_col
   FROM SCOTT.RIDS
   WHERE dbms_rowid.rowid_verify(rowid_col, NULL, NULL, 0) =1;
```

See Also: Chapter 174, "UTL_RAW", Chapter 176, "UTL_REF"

91

DBMS_RULE

The DBMS_RULE package contains subprograms that enable the evaluation of a rule set for a specified event.

See Also:

- Chapter 196, "Rule TYPEs" for more information about the types used with the DBMS_RULE package
- Chapter 92, "DBMS_RULE_ADM" and *Oracle Streams Concepts and Administration* for more information about this package and rules

This chapter contains the following topics:

Using DBMS_RULE

This section contains topics which relate to using the `DBMS_RULE` package.

Security Model

PUBLIC is granted EXECUTE privilege on this package.

Summary of DBMS_RULE Subprograms

Table 91–1 DBMS_RULE Package Subprograms

Subprogram	Description
CLOSE_ITERATOR Procedure on page 91-5	Closes an open iterator
EVALUATE Procedures on page 91-6	Evaluates the rules in the specified rule set that use the evaluation context specified
GET_NEXT_HIT Function on page 91-10	Returns the next rule that evaluated to `TRUE` from a true rules iterator, or returns the next rule that evaluated to `MAYBE` from a maybe rules iterator; returns `NULL` if there are no more rules that evaluated to `TRUE` or `MAYBE`.

CLOSE_ITERATOR Procedure

This procedure closes an open iterator.

Syntax

```
DBMS_RULE.CLOSE_ITERATOR(
  iterator  IN  NUMBER);
```

Parameter

Table 91–2 CLOSE_ITERATOR Procedure Parameter

Parameter	Description
iterator	The iterator to be closed

Usage Notes

This procedure requires an open iterator that was returned by an earlier call to DBMS_RULE.EVALUATE in the same session. The user who runs this procedure does not require any privileges on the rule set being evaluated.

Closing an iterator frees resources, such as memory, associated with the iterator. Therefore, Oracle recommends that you close an iterator when it is no longer needed.

See Also: "EVALUATE Procedures" on page 91-6

EVALUATE Procedures

This procedure evaluates the rules in the specified rule set that use the evaluation context specified for a specified event.

This procedure is overloaded. The `true_rules` and `maybe_rules` parameters are mutually exclusive with the `true_rules_iterator` and `maybe_rules_iterator` parameters. In addition, the procedure with the `true_rules` and `maybe_rules` parameters includes the `stop_on_first_hit` parameter, but the other procedure does not.

Syntax

```
DBMS_RULE.EVALUATE(
  rule_set_name         IN   VARCHAR2,
  evaluation_context    IN   VARCHAR2,
  event_context         IN   SYS.RE$NV_LIST               DEFAULT NULL,
  table_values          IN   SYS.RE$TABLE_VALUE_LIST      DEFAULT NULL,
  column_values         IN   SYS.RE$COLUMN_VALUE_LIST     DEFAULT NULL,
  variable_values       IN   SYS.RE$VARIABLE_VALUE_LIST   DEFAULT NULL,
  attribute_values      IN   SYS.RE$ATTRIBUTE_VALUE_LIST  DEFAULT NULL,
  stop_on_first_hit     IN   BOOLEAN                      DEFAULT FALSE,
  simple_rules_only     IN   BOOLEAN                      DEFAULT FALSE,
  true_rules            OUT  SYS.RE$RULE_HIT_LIST,
  maybe_rules           OUT  SYS.RE$RULE_HIT_LIST);

DBMS_RULE.EVALUATE(
  rule_set_name          IN   VARCHAR2,
  evaluation_context     IN   VARCHAR2,
  event_context          IN   SYS.RE$NV_LIST               DEFAULT NULL,
  table_values           IN   SYS.RE$TABLE_VALUE_LIST      DEFAULT NULL,
  column_values          IN   SYS.RE$COLUMN_VALUE_LIST     DEFAULT NULL,
  variable_values        IN   SYS.RE$VARIABLE_VALUE_LIST   DEFAULT NULL,
  attribute_values       IN   SYS.RE$ATTRIBUTE_VALUE_LIST  DEFAULT NULL,
  simple_rules_only      IN   BOOLEAN                      DEFAULT FALSE,
  true_rules_iterator    OUT  BINARY_INTEGER,
  maybe_rules_iterator   OUT  BINARY_INTEGER);
```

Parameters

Table 91–3 EVALUATE Procedure Parameters

Parameter	Description
`rule_set_name`	Name of the rule set in the form `[`*`schema_name`*`.]`*`rule_set_name`*. For example, to evaluate all of the rules in a rule set named `hr_rules` in the `hr` schema, enter `hr.hr_rules` for this parameter. If the schema is not specified, then the schema of the current user is used.
`evaluation_context`	An evaluation context name in the form `[`*`schema_name`*`.]`*`evaluation_context_name`*. If the schema is not specified, then the name of the current user is used. Only rules that use the specified evaluation context are evaluated.
`event_context`	A list of name-value pairs that identify events that cause evaluation

Table 91–3 (Cont.) EVALUATE Procedure Parameters

Parameter	Description
table_values	Contains the data for table rows using the table aliases specified when the evaluation context was created. Each table alias in the list must be unique.
column_values	Contains the partial data for table rows. It must not contain column values for tables, whose values are already specified in table_values.
variable_values	A list containing the data for variables. The only way for an explicit variable value to be known is to specify its value in this list. If an implicit variable value is not specified in the list, then the function used to obtain the value of the implicit variable is invoked. If an implicit variable value is specified in the list, then this value is used and the function is not invoked.
attribute_values	Contains the partial data for variables. It must not contain attribute values for variables whose values are already specified in variable_values.
stop_on_first_hit	If TRUE, then the rules engine stops evaluation as soon as it finds a TRUE rule. If TRUE and there are no TRUE rules, then the rules engine stops evaluation as soon as it finds a rule that may evaluate to TRUE given more data. If FALSE, then the rules engine continues to evaluate rules even after it finds a TRUE rule.
simple_rules_only	If TRUE, then only those rules that are simple enough to be evaluated fast (without issuing SQL) are considered for evaluation. If FALSE, then evaluates all rules.
true_rules	Receives the output of the EVALUATE procedure into a varray of RE$RULE_HIT_LIST type. If no rules evaluate to TRUE, then true_rules is empty. If at least one rule evaluates to TRUE and stop_on_first_hit is TRUE, then true_rules contains one rule that evaluates to TRUE. If stop_on_first_hit is FALSE, then true_rules contains all rules that evaluate to TRUE.
maybe_rules	If all rules can be evaluated completely, without requiring any additional data, then maybe_rules is empty. If stop_on_first_hit is TRUE, then if there is at least one rule that may evaluate to TRUE given more data, and no rules evaluate to TRUE, then maybe_rules contains one rule that may evaluate to TRUE. If stop_on_first_hit is FALSE, then maybe_rules contains all rules that may evaluate to TRUE given more data.
true_rules_iterator	Contains the iterator for accessing rules that are TRUE
maybe_rules_iterator	Contains the iterator for accessing rules that may be TRUE given additional data or the ability to issue SQL

Usage Notes

> **Note:** Rules in the rule set that use an evaluation context different from the one specified are not considered for evaluation.

The rules in the rule set are evaluated using the data specified for `table_values`, `column_values`, `variable_values`, and `attribute_values`. These values must refer to tables and variables in the specified evaluation context. Otherwise, an error is raised.

The caller may specify, using `stop_on_first_hit`, if evaluation must stop as soon as the first `TRUE` rule or the first `MAYBE` rule (if there are no `TRUE` rules) is found.

The caller may also specify, using `simple_rules_only`, if only rules that are simple enough to be evaluated fast (which means without SQL) should be considered for evaluation. This makes evaluation faster, but causes rules that cannot be evaluated without SQL to be returned as `MAYBE` rules.

Partial evaluation is supported. The `EVALUATE` procedure can be called with data for only some of the tables, columns, variables, or attributes. In such a case, rules that cannot be evaluated because of a lack of data are returned as `MAYBE` rules, unless they can be determined to be `TRUE` or `FALSE` based on the values of one or more simple expressions within the rule. For example, given a value of `1` for attribute `"a.b"` of variable `"x"`, a rule with the following rule condition can be returned as `TRUE`, without a value for table `"tab"`:

```
(:x.a.b = 1) or (tab.c > 10)
```

The results of an evaluation are the following:

- `TRUE` rules, which is the list of rules that evaluate to `TRUE` based on the given data. These rules are returned either in the `OUT` parameter `true_rules`, which returns all of the rules that evaluate to `TRUE`, or in the `OUT` parameter `true_rules_iterator`, which returns each rule that evaluates to `TRUE` one at a time.
- `MAYBE` rules, which is the list of rules that could not be evaluated for one of the following reasons:
  - The rule refers to data that was unavailable. For example, a variable attribute `"x.a.b"` is specified, but no value is specified for the variable `"x"`, the attribute `"a"`, or the attribute `"a.b"`.
  - The rule is not simple enough to be evaluated fast (without SQL) and `simple_rules_only` is specified as `TRUE`, or partial data is available.

  Maybe rules are returned either in the `OUT` parameter `maybe_rules`, which returns all of the rules that evaluate to `MAYBE`, or in the `OUT` parameter `maybe_rules_iterator`, which returns each rule that evaluates to `MAYBE` one at a time.

The caller may specify whether the procedure returns all of the rules that evaluate to `TRUE` and `MAYBE` for the event or an iterator for rules that evaluate to `TRUE` and `MAYBE`. A true rules iterator enables the client to fetch each rule that evaluates to `TRUE` one at a time, and a maybe rules iterator enables the client to fetch each rule that evaluates to `MAYBE` one at a time.

If you use an iterator, then you use the `GET_NEXT_HIT` function in the `DBMS_RULE` package to retrieve the next rule that evaluates to `TRUE` or `MAYBE` from an iterator. Oracle recommends that you close an iterator if it is no longer needed to free

resources, such as memory, used by the iterator. An iterator can be closed in the following ways:

- The `CLOSE_ITERATOR` procedure in the `DBMS_RULE` package is run with the iterator specified.
- The iterator returns `NULL` because no more rules evaluate to `TRUE` or `MAYBE`.
- The session in which the iterator is running ends.

To run the `DBMS_RULE.EVALUATE` procedure, a user must meet at least one of the following requirements:

- Have `EXECUTE_ON_RULE_SET` privilege on the rule set
- Have `EXECUTE_ANY_RULE_SET` system privilege
- Be the rule set owner

Note: The rules engine does not invoke any actions. An action context can be returned with each returned rule, but the client of the rules engine must invoke any necessary actions.

See Also:

- Chapter 196, "Rule TYPEs" for more information about the types used with the `DBMS_RULE` package
- "GET_NEXT_HIT Function" on page 91-10
- "CLOSE_ITERATOR Procedure" on page 91-5

GET_NEXT_HIT Function

This function returns the next rule that evaluated to TRUE from a true rules iterator, or returns the next rule that evaluated to MAYBE from a maybe rules iterator. The function returns NULL if there are no more rules that evaluated to TRUE or MAYBE.

Syntax

```
DBMS_RULE.GET_NEXT_HIT(
  iterator  IN  NUMBER)
RETURN SYS.RE$RULE_HIT;
```

Parameter

Table 91–4 GET_NEXT_HIT Function Parameter

Parameter	Description
iterator	The iterator from which the rule that evaluated to TRUE or MAYBE is retrieved

Usage Notes

This procedure requires an open iterator that was returned by an earlier call to DBMS_RULE.EVALUATE in the same session. The user who runs this procedure does not require any privileges on the rule set being evaluated.

When an iterator returns NULL, it is closed automatically. If an open iterator is no longer needed, then use the CLOSE_ITERATOR procedure in the DBMS_RULE package to close it.

Note: This function raises an error if the rule set being evaluated was modified after the call to the DBMS_RULE.EVALUATE procedure that returned the iterator. Modifications to a rule set include added rules to the rule set, changing existing rules in the rule set, dropping rules from the rule set, and dropping the rule set.

See Also:

- Chapter 196, "Rule TYPEs" for more information about the types used with the DBMS_RULE package
- "EVALUATE Procedures" on page 91-6
- "CLOSE_ITERATOR Procedure" on page 91-5

92

DBMS_RULE_ADM

The DBMS_RULE_ADM package provides the subprograms for creating and managing rules, rule sets, and rule evaluation contexts.

See Also:

- Chapter 196, "Rule TYPEs" for more information about the types used with the DBMS_RULE_ADM package
- Chapter 91, "DBMS_RULE" and *Oracle Streams Concepts and Administration* for more information about this package and rules

This chapter contains the following topics:

Using DBMS_RULE_ADM

This section contains topics which relate to using the `DBMS_RULE_ADM` package.

Security Model

User group PUBLIC is granted EXECUTE privilege on this package.

See Also: *Oracle Database Security Guide* for more information about user group PUBLIC

Summary of DBMS_RULE_ADM Subprograms

Table 92–1 DBMS_RULE_ADM Package Subprograms

Subprogram	Description
ADD_RULE Procedure on page 92-5	Adds the specified rule to the specified rule set
ALTER_EVALUATION_CONTEXT Procedure on page 92-7	Alters a rule evaluation context
ALTER_RULE Procedure on page 92-10	Changes one or more aspects of the specified rule
CREATE_EVALUATION_CONTEXT Procedure on page 92-12	Creates a rule evaluation context
CREATE_RULE Procedure on page 92-14	Creates a rule with the specified name
CREATE_RULE_SET Procedure on page 92-16	Creates a rule set with the specified name
DROP_EVALUATION_CONTEXT Procedure on page 92-17	Drops the rule evaluation context with the specified name
DROP_RULE Procedure on page 92-18	Drops the rule with the specified name
DROP_RULE_SET Procedure on page 92-19	Drops the rule set with the specified name
GRANT_OBJECT_PRIVILEGE Procedure on page 92-20	Grants the specified object privilege on the specified object to the specified user or role
GRANT_SYSTEM_PRIVILEGE Procedure on page 92-22	Grants the specified system privilege to the specified user or role
REMOVE_RULE Procedure on page 92-24	Removes the specified rule from the specified rule set
REVOKE_OBJECT_PRIVILEGE Procedure on page 92-26	Revokes the specified object privilege on the specified object from the specified user or role
REVOKE_SYSTEM_PRIVILEGE Procedure on page 92-27	Revokes the specified system privilege from the specified user or role

Note: All subprograms commit unless specified otherwise.

ADD_RULE Procedure

This procedure adds the specified rule to the specified rule set.

Syntax

```
DBMS_RULE_ADM.ADD_RULE(
   rule_name           IN  VARCHAR2,
   rule_set_name       IN  VARCHAR2,
   evaluation_context  IN  VARCHAR2   DEFAULT NULL,
   rule_comment        IN  VARCHAR2   DEFAULT NULL);
```

Parameters

Table 92–2 ADD_RULE Procedure Parameters

Parameter	Description
rule_name	The name of the rule you are adding to the rule set, specified as [*schema_name.*]*rule_name*. For example, to add a rule named all_a in the hr schema, enter hr.all_a for this parameter. If the schema is not specified, then the current user is the default.
rule_set_name	The name of the rule set to which you are adding the rule, specified as [*schema_name.*]*rule_set_name*. For example, to add the rule to a rule set named apply_rules in the hr schema, enter hr.apply_rules for this parameter. If the schema is not specified, then the current user is the default.
evaluation_context	An evaluation context name in the form [*schema_name.*]*evaluation_context_name*. If the schema is not specified, then the current user is the default. Only specify an evaluation context if the rule itself does not have an evaluation context and you do not want to use the rule set's evaluation context for the rule.
rule_comment	Optional description, which can contain the reason for adding the rule to the rule set

Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Have ALTER_ON_RULE_SET privilege on the rule set
- Have ALTER_ANY_RULE_SET system privilege
- Be the owner of the rule set

Also, the rule set owner must meet at least one of the following requirements:

- Have EXECUTE_ON_RULE privilege on the rule
- Have EXECUTE_ANY_RULE system privilege
- Be the rule owner

If the rule has no evaluation context and no evaluation context is specified when you run this procedure, then the rule uses the evaluation context associated with the rule set. In such a case, the rule owner must have the necessary privileges on all the base objects accessed by the rule using the evaluation context.

If an evaluation context is specified, then the rule set owner must meet at least one of the following requirements:

- Have `EXECUTE_ON_EVALUATION_CONTEXT` privilege on the evaluation context
- Have `EXECUTE_ANY_EVALUATION_CONTEXT` system privilege, and the owner of the evaluation context must not be `SYS`
- Be the evaluation context owner

Also, the rule owner must have the necessary privileges on all the base objects accessed by the rule using the evaluation context.

ALTER_EVALUATION_CONTEXT Procedure

This procedure alters a rule evaluation context. A rule evaluation context defines external data that can be referenced in rule conditions. The external data can either exist as variables or as table data.

Syntax

```
DBMS_RULE_ADM.ALTER_EVALUATION_CONTEXT(
  evaluation_context_name      IN  VARCHAR2,
  table_aliases                IN  SYS.RE$TABLE_ALIAS_LIST    DEFAULT NULL,
  remove_table_aliases         IN  BOOLEAN                    DEFAULT FALSE,
  variable_types               IN  SYS.RE$VARIABLE_TYPE_LIST  DEFAULT NULL,
  remove_variable_types        IN  BOOLEAN                    DEFAULT FALSE,
  evaluation_function          IN  VARCHAR2                   DEFAULT NULL,
  remove_evaluation_function   IN  BOOLEAN                    DEFAULT FALSE,
  evaluation_context_comment   IN  VARCHAR2                   DEFAULT NULL,
  remove_eval_context_comment  IN  BOOLEAN                    DEFAULT FALSE);
```

Parameters

Table 92–3 ALTER_EVALUATION_CONTEXT Procedure Parameters

Parameter	Description
evaluation_context_name	The name of the evaluation context you are altering, specified as [*schema_name.*]*evaluation_context_name*. For example, to alter an evaluation context named dept_eval_context in the hr schema, enter hr.dept_eval_context for this parameter. If the schema is not specified, then the current user is the default.
table_aliases	If NULL and remove_table_aliases is FALSE, then the procedure retains the existing table aliases. If NULL and remove_table_aliases is TRUE, then the procedure removes the existing table aliases. If non-NULL, then the procedure replaces the existing table aliases for the evaluation context with the specified table aliases. Table aliases specify the tables in an evaluation context. The table aliases can be used to reference tables in rule conditions.
remove_table_aliases	If TRUE and table_aliases is NULL, then the procedure removes the existing table aliases for the evaluation context. If TRUE and table_aliases is non-NULL, then the procedure raises an error. If FALSE, then the procedure does not remove table aliases.
variable_types	If NULL and remove_variable_types is FALSE, then the procedure retains the variable types. If NULL and remove_variable_types is TRUE, then the procedure removes the existing variable types. If non-NULL, then the procedure replaces the existing variable types for the evaluation context with the specified variable types.

Table 92–3 (Cont.) ALTER_EVALUATION_CONTEXT Procedure Parameters

Parameter	Description
remove_variable_types	If TRUE and variable_types is NULL, then the procedure removes the existing variable types for the evaluation context. If TRUE and variable_types is non-NULL, then the procedure raises an error. If FALSE, then the procedure does not remove the variable types.
evaluation_function	If NULL and remove_evaluation_function is FALSE, then the procedure retains the existing evaluation function. If NULL and remove_evaluation_function is TRUE, then the procedure removes the existing evaluation function. If non-NULL, then the procedure replaces the existing evaluation function for the evaluation context with the specified evaluation function. An evaluation function is an optional function that will be called to evaluate rules that use the evaluation context. It must have the same form as the DBMS_RULE.EVALUATE procedure. If the schema is not specified, then the current user is the default. See "CREATE_EVALUATION_CONTEXT Procedure" on page 92-12for more information about evaluation functions.
remove_evaluation_function	If TRUE and evaluation_function is NULL, then the procedure removes the existing evaluation function for the evaluation context. If TRUE and evaluation_function is non-NULL, then the procedure raises an error. If FALSE, then the procedure does not remove the evaluation function.
evaluation_context_comment	If NULL and remove_eval_context_comment is FALSE, then the procedure retains the existing evaluation context comment. If NULL and remove_evaluation_function is TRUE, then the procedure removes the existing evaluation context comment. If non-NULL, then the procedure replaces the existing comment for the evaluation context with the specified comment. An evaluation context comment is an optional description of the rule evaluation context.
remove_eval_context_comment	If TRUE and evaluation_context_comment is NULL, then the procedure removes the existing comment for the evaluation context. If TRUE and evaluation_context_comment is non-NULL, then the procedure raises an error. If FALSE, then the procedure does not remove the evaluation context comment.

Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Be the owner of the evaluation context being altered
- Have ALL_ON_EVALUATION_CONTEXT or ALTER_ON_EVALUATION_CONTEXT object privilege on an evaluation context owned by another user

- Have `ALTER_ANY_EVALUATION_CONTEXT` system privilege

See Also: Chapter 196, "Rule TYPEs" for more information about the types used with the `DBMS_RULE_ADM` package

ALTER_RULE Procedure

This procedure changes one or more aspects of the specified rule.

Syntax

```
DBMS_RULE_ADM.ALTER_RULE(
   rule_name                  IN  VARCHAR2,
   condition                  IN  VARCHAR2        DEFAULT NULL,
   evaluation_context         IN  VARCHAR2        DEFAULT NULL,
   remove_evaluation_context  IN  BOOLEAN         DEFAULT FALSE,
   action_context             IN  SYS.RE$NV_LIST  DEFAULT NULL,
   remove_action_context      IN  BOOLEAN         DEFAULT FALSE,
   rule_comment               IN  VARCHAR2        DEFAULT NULL,
   remove_rule_comment        IN  BOOLEAN         DEFAULT FALSE);
```

Parameters

Table 92–4 ALTER_RULE Procedure Parameters

Parameter	Description
rule_name	The name of the rule you are altering, specified as [*schema_name.*]*rule_name*. For example, to alter a rule named all_a in the hr schema, enter hr.all_a for this parameter. If the schema is not specified, then the current user is the default.
condition	The condition to be associated with the rule. If non-NULL, then the procedure replaces the existing condition of the rule with the specified condition.
evaluation_context	An evaluation context name in the form [*schema_name.*]*evaluation_context_name*. If the schema is not specified, then the current user is the default. If non-NULL, then the procedure replaces the existing evaluation context of the rule with the specified evaluation context.
remove_evaluation_context	If TRUE, then the procedure sets the evaluation context for the rule to NULL, which effectively removes the evaluation context from the rule. If FALSE, then the procedure retains any evaluation context for the specified rule. If the evaluation_context parameter is non-NULL, then this parameter should be set to FALSE.
action_context	If non-NULL, then the procedure changes the action context associated with the rule. A rule action context is information associated with a rule that is interpreted by the client of the rules engine when the rule is evaluated.
remove_action_context	If TRUE, then the procedure sets the action context for the rule to NULL, which effectively removes the action context from the rule. If FALSE, then the procedure retains any action context for the specified rule. If the action_context parameter is non-NULL, then this parameter should be set to FALSE.
rule_comment	If non-NULL, then the existing comment of the rule is replaced by the specified comment.

Table 92–4 (Cont.) ALTER_RULE Procedure Parameters

Parameter	Description
remove_rule_comment	If TRUE, then the procedure sets the comment for the rule to NULL, which effectively removes the comment from the rule. If FALSE, then the procedure retains any comment for the specified rule. If the rule_comment parameter is non-NULL, then this parameter should be set to FALSE.

Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Have ALTER_ON_RULE privilege on the rule
- Have ALTER_ANY_RULE system privilege
- Be the owner of the rule being altered

If an evaluation context is specified, then the rule owner must meet at least one of the following requirements:

- Have EXECUTE_ON_EVALUATION_CONTEXT privilege on the evaluation context
- Have EXECUTE_ANY_EVALUATION_CONTEXT system privilege, and the owner of the evaluation context must not be SYS
- Be the evaluation context owner

Also, the rule owner must have the necessary privileges on all the base objects accessed by the rule using the evaluation context.

> **See Also:** Chapter 196, "Rule TYPEs" for more information about the types used with the DBMS_RULE_ADM package

CREATE_EVALUATION_CONTEXT Procedure

This procedure creates a rule evaluation context. A rule evaluation context defines external data that can be referenced in rule conditions. The external data can either exist as variables or as table data.

Syntax

```
DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT(
  evaluation_context_name      IN  VARCHAR2,
  table_aliases                IN  SYS.RE$TABLE_ALIAS_LIST    DEFAULT NULL,
  variable_types               IN  SYS.RE$VARIABLE_TYPE_LIST  DEFAULT NULL,
  evaluation_function          IN  VARCHAR2                   DEFAULT NULL,
  evaluation_context_comment   IN  VARCHAR2                   DEFAULT NULL);
```

Parameters

Table 92–5 CREATE_EVALUATION_CONTEXT Procedure Parameters

Parameter	Description
evaluation_context_name	The name of the evaluation context you are creating, specified as [*schema_name.*]*evaluation_context_name*. For example, to create an evaluation context named dept_eval_context in the hr schema, enter hr.dept_eval_context for this parameter. If the schema is not specified, then the current user is the default.
table_aliases	Table aliases that specify the tables in an evaluation context. The table aliases can be used to reference tables in rule conditions.
variable_types	A list of variables for the evaluation context
evaluation_function	An optional function that will be called to evaluate rules using the evaluation context. It must have the same form as the DBMS_RULE.EVALUATE procedure. If the schema is not specified, then the current user is the default. See "Usage Notes" for more information about the evaluation function.
evaluation_context_comment	An optional description of the rule evaluation context.

Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Be the owner of the evaluation context being created and have CREATE_EVALUATION_CONTEXT_OBJ system privilege
- Have CREATE_ANY_EVALUATION_CONTEXT system privilege

> **See Also:** Chapter 196, "Rule TYPEs" for more information about the types used with the DBMS_RULE_ADM package

The evaluation function must have the following signature:

```
FUNCTION evaluation_function_name(
  rule_set_name       IN   VARCHAR2,
```

```
  evaluation_context  IN   VARCHAR2,
  event_context       IN   SYS.RE$NV_LIST               DEFAULT NULL,
  table_values        IN   SYS.RE$TABLE_VALUE_LIST      DEFAULT NULL,
  column_values       IN   SYS.RE$COLUMN_VALUE_LIST     DEFAULT NULL,
  variable_values     IN   SYS.RE$VARIABLE_VALUE_LIST   DEFAULT NULL,
  attribute_values    IN   SYS.RE$ATTRIBUTE_VALUE_LIST  DEFAULT NULL,
  stop_on_first_hit   IN   BOOLEAN                      DEFAULT FALSE,
  simple_rules_only   IN   BOOLEAN                      DEFAULT FALSE,
  true_rules          OUT  SYS.RE$RULE_HIT_LIST,
  maybe_rules         OUT  SYS.RE$RULE_HIT_LIST);
RETURN BINARY_INTEGER;
```

Note: Each parameter is required and must have the specified datatype. However, you can change the names of the parameters.

The return value of the function must be one of the following:

- `DBMS_RULE_ADM.EVALUATION_SUCCESS`: The user specified evaluation function completed the rule set evaluation successfully. The rules engine returns the results of the evaluation obtained by the evaluation function to the rules engine client using the `DBMS_RULE.EVALUATE` procedure.
- `DBMS_RULE_ADM.EVALUATION_CONTINUE`: The rules engine evaluates the rule set as if there were no evaluation function. The evaluation function is not used, and any results returned by the evaluation function are ignored.
- `DBMS_RULE_ADM.EVALUATION_FAILURE`: The user specified evaluation function failed. Rule set evaluation stops, and an error is raised.

CREATE_RULE Procedure

This procedure creates a rule.

Syntax

```
DBMS_RULE_ADM.CREATE_RULE(
   rule_name           IN  VARCHAR2,
   condition           IN  VARCHAR2,
   evaluation_context  IN  VARCHAR2        DEFAULT NULL,
   action_context      IN  SYS.RE$NV_LIST  DEFAULT NULL,
   rule_comment        IN  VARCHAR2        DEFAULT NULL);
```

Parameters

Table 92–6 CREATE_RULE Procedure Parameters

Parameter	Description
rule_name	The name of the rule you are creating, specified as [*schema_name.*]*rule_name*. For example, to create a rule named all_a in the hr schema, enter hr.all_a for this parameter. If the schema is not specified, then the current user is the default.
condition	The condition to be associated with the rule. A condition evaluates to TRUE or FALSE and can be any condition allowed in the WHERE clause of a SELECT statement. For example, the following is a valid rule condition: department_id = 30 **Note:** Do not include the word "WHERE" in the condition.
evaluation_context	An optional evaluation context name in the form [*schema_name.*]*evaluation_context_name*, which is associated with the rule. If the schema is not specified, then the current user is the default. If evaluation_context is not specified, then the rule inherits the evaluation context from its rule set.
action_context	The action context associated with the rule. A rule action context is information associated with a rule that is interpreted by the client of the rules engine when the rule is evaluated.
rule_comment	An optional description of the rule

Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Be the owner of the rule being created and have the CREATE_RULE_OBJ system privilege
- Have CREATE_ANY_RULE system privilege

If an evaluation context is specified, then the rule owner must meet at least one of the following requirements:

- Have EXECUTE_ON_EVALUATION_CONTEXT privilege on the evaluation context
- Have EXECUTE_ANY_EVALUATION_CONTEXT system privilege, and the owner of the evaluation context must not be SYS.
- Be the evaluation context owner

Also, the rule owner must have the necessary privileges on all the base objects accessed by the rule using the evaluation context.

> **See Also:** Chapter 196, "Rule TYPEs" for more information about the types used with the `DBMS_RULE_ADM` package

CREATE_RULE_SET Procedure

This procedure creates a rule set.

Syntax

```
DBMS_RULE_ADM.CREATE_RULE_SET(
   rule_set_name       IN  VARCHAR2,
   evaluation_context  IN  VARCHAR2  DEFAULT NULL,
   rule_set_comment    IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 92–7 CREATE_RULE_SET Procedure Parameters

Parameter	Description
`rule_set_name`	The name of the rule set you are creating, specified as `[`*`schema_name.`*`]`*`rule_set_name`*. For example, to create a rule set named `apply_rules` in the `hr` schema, enter `hr.apply_rules` for this parameter. If the schema is not specified, then the current user is the default.
`evaluation_context`	An optional evaluation context name in the form `[`*`schema_name.`*`]`*`evaluation_context_name`*, which applies to all rules in the rule set that are not associated with an evaluation context explicitly. If the schema is not specified, then the current user is the default.
`rule_set_comment`	An optional description of the rule set

Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Be the owner of the rule set being created and have `CREATE_RULE_SET_OBJ` system privilege
- Have `CREATE_ANY_RULE_SET` system privilege

If an evaluation context is specified, then the rule set owner must meet at least one of the following requirements:

- Have `EXECUTE_ON_EVALUATION_CONTEXT` privilege on the evaluation context
- Have `EXECUTE_ANY_EVALUATION_CONTEXT` system privilege, and the owner of the evaluation context must not be `SYS`
- Be the evaluation context owner

DROP_EVALUATION_CONTEXT Procedure

This procedure drops a rule evaluation context.

Syntax

```
DBMS_RULE_ADM.DROP_EVALUATION_CONTEXT(
   evaluation_context_name  IN  VARCHAR2,
   force                    IN  BOOLEAN   DEFAULT FALSE);
```

Parameters

Table 92–8 DROP_EVALUATION_CONTEXT Procedure Parameters

Parameter	Description
`evaluation_context_name`	The name of the evaluation context you are dropping, specified as `[`*`schema_name`*`.]`*`evaluation_context_name`*.
	For example, to drop an evaluation context named `dept_eval_context` in the `hr` schema, enter `hr.dept_eval_context` for this parameter. If the schema is not specified, then the current user is the default.
`force`	If `TRUE`, then the procedure removes the rule evaluation context from all rules and rule sets that use it.
	If `FALSE` and no rules or rule sets use the rule evaluation context, then the procedure drops the rule evaluation context.
	If `FALSE` and one or more rules or rule sets use the rule evaluation context, then the procedure raises an exception.
	Caution: Setting `force` to `TRUE` can result in rules and rule sets that do not have an evaluation context. If neither a rule nor the rule set it is in has an evaluation context, and no evaluation context was specified for the rule by the `ADD_RULE` procedure, then the rule cannot be evaluated.

Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Be the owner of the evaluation context
- Have `DROP_ANY_EVALUATION_CONTEXT` system privilege

DROP_RULE Procedure

This procedure drops a rule.

Syntax

```
DBMS_RULE_ADM.DROP_RULE(
   rule_name  IN  VARCHAR2,
   force      IN  BOOLEAN   DEFAULT FALSE);
```

Parameters

Table 92–9 DROP_RULE Procedure Parameters

Parameter	Description
rule_name	The name of the rule you are dropping, specified as [*schema_name.*]*rule_name*. For example, to drop a rule named all_a in the hr schema, enter hr.all_a for this parameter. If the schema is not specified, then the current user is the default.
force	If TRUE, then the procedure removes the rule from all rule sets that contain it. If FALSE and no rule sets contain the rule, then the procedure drops the rule. If FALSE and one or more rule sets contain the rule, then the procedure raises an exception.

Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Be the owner of the rule
- Have DROP_ANY_RULE system privilege

Note:

- To remove a rule from a rule set without dropping the rule from the database, use the REMOVE_RULE procedure.
- The rule evaluation context associated with the rule, if any, is not dropped when you run this procedure.

DROP_RULE_SET Procedure

This procedure drops a rule set.

Syntax

```
DBMS_RULE_ADM.DROP_RULE_SET(
   rule_set_name  IN  VARCHAR2,
   delete_rules   IN  BOOLEAN   DEFAULT FALSE);
```

Parameters

Table 92–10 DROP_RULE_SET Procedure Parameters

Parameter	Description
rule_set_name	The name of the rule set you are dropping, specified as [*schema_name.*]*rule_set_name*. For example, to drop a rule set named apply_rules in the hr schema, enter hr.apply_rules for this parameter. If the schema is not specified, then the current user is the default.
delete_rules	If TRUE, then the procedure drops any rules that are in the rule set. If any of the rules in the rule set are also in another rule set, then these rules are not dropped. If FALSE, then the procedure does not drop the rules in the rule set.

Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Have DROP_ANY_RULE_SET system privilege
- Be the owner of the rule set

> **Note:** The rule evaluation context associated with the rule set, if any, is not dropped when you run this procedure.

GRANT_OBJECT_PRIVILEGE Procedure

This procedure grants the specified object privilege on the specified object to the specified user or role. If a user owns the object, then the user automatically is granted all privileges on the object, with grant option.

Syntax

```
DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
   privilege     IN  BINARY_INTEGER,
   object_name   IN  VARCHAR2,
   grantee       IN  VARCHAR2,
   grant_option  IN  BOOLEAN   DEFAULT FALSE);
```

Parameters

Table 92–11 GRANT_OBJECT_PRIVILEGE Procedure Parameters

Parameter	Description
`privilege`	The name of the object privilege to grant to the grantee on the object. See "Usage Notes" on page 92-20 for the available object privileges.
`object_name`	The name of the object for which you are granting the privilege to the grantee, specified as `[`*`schema_name`*`.]`*`object_name`*. For example, to grant the privilege on a rule set named `apply_rules` in the `hr` schema, enter `hr.apply_rules` for this parameter. If the schema is not specified, then the current user is the default. The object must be an existing rule, rule set, or evaluation context.
`grantee`	The name of the user or role for which the privilege is granted. The specified user cannot be the owner of the object.
`grant_option`	If `TRUE`, then the specified user or users granted the specified privilege can grant this privilege to others. If `FALSE`, then the specified user or users granted the specified privilege cannot grant this privilege to others.

Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Be the owner of the object on which the privilege is granted
- Have the same privilege as the privilege being granted with the grant option

In addition, if the object is a rule set, then the user must have `EXECUTE` privilege on all the rules in the rule set with grant option or must own the rules in the rule set.

Table 92–12 lists the object privileges.

Table 92–12 Object Privileges for Evaluation Contexts, Rules, and Rule Sets

Privilege	Description
SYS.DBMS_RULE_ADM.ALL_ON_EVALUATION_CONTEXT	Alter and execute a particular evaluation context in another user's schema
SYS.DBMS_RULE_ADM.ALL_ON_RULE	Alter and execute a particular rule in another user's schema
SYS.DBMS_RULE_ADM.ALL_ON_RULE_SET	Alter and execute a particular rule set in another user's schema
SYS.DBMS_RULE_ADM.ALTER_ON_EVALUATION_CONTEXT	Alter a particular evaluation context in another user's schema
SYS.DBMS_RULE_ADM.ALTER_ON_RULE	Alter a particular rule in another user's schema
SYS.DBMS_RULE_ADM.ALTER_ON_RULE_SET	Alter a particular rule set in another user's schema
SYS.DBMS_RULE_ADM.EXECUTE_ON_EVALUATION_CONTEXT	Execute a particular evaluation context in another user's schema
SYS.DBMS_RULE_ADM.EXECUTE_ON_RULE	Execute a particular rule in another user's schema
SYS.DBMS_RULE_ADM.EXECUTE_ON_RULE_SET	Execute a particular rule set in another user's schema

Examples

For example, to grant the HR user the privilege to alter a rule named hr_dml in the strmadmin schema, enter the following:

```
BEGIN
  DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
    privilege    => SYS.DBMS_RULE_ADM.ALTER_ON_RULE,
    object_name  => 'strmadmin.hr_dml',
    grantee      => 'hr',
    grant_option => FALSE);
END;
/
```

GRANT_SYSTEM_PRIVILEGE Procedure

This procedure grant the specified system privilege to the specified user or role.

Syntax

```
DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
   privilege     IN  BINARY_INTEGER,
   grantee       IN  VARCHAR2,
   grant_option  IN  BOOLEAN   DEFAULT FALSE);
```

Parameters

Table 92–13 GRANT_SYSTEM_PRIVILEGE Procedure Parameters

Parameter	Description
privilege	The name of the system privilege to grant to the grantee.
grantee	The name of the user or role for which the privilege is granted
grant_option	If TRUE, then the specified user or users granted the specified privilege can grant the system privilege to others. If FALSE, then the specified user or users granted the specified privilege cannot grant the system privilege to others.

Usage Notes

Table 92–14 lists the system privileges.

Table 92–14 System Privileges for Evaluation Contexts, Rules, and Rule Sets

Privilege	Description
SYS.DBMS_RULE_ADM.ALTER_ANY_EVALUATION_CONTEXT	Alter any evaluation context owned by any user
SYS.DBMS_RULE_ADM.ALTER_ANY_RULE	Alter any rule owned by any user
SYS.DBMS_RULE_ADM.ALTER_ANY_RULE_SET	Alter any rule set owned by any user
SYS.DBMS_RULE_ADM.CREATE_ANY_EVALUATION_CONTEXT	Create a new evaluation context in any schema
SYS.DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT_OBJ	Create a new evaluation context in the grantee's schema
SYS.DBMS_RULE_ADM.CREATE_ANY_RULE	Create a new rule in any schema
SYS.DBMS_RULE_ADM.CREATE_RULE_OBJ	Create a new rule in the grantee's schema
SYS.DBMS_RULE_ADM.CREATE_ANY_RULE_SET	Create a new rule set in any schema
SYS.DBMS_RULE_ADM.CREATE_RULE_SET_OBJ	Create a new rule set in the grantee's schema
SYS.DBMS_RULE_ADM.DROP_ANY_EVALUATION_CONTEXT	Drop any evaluation context in any schema
SYS.DBMS_RULE_ADM.DROP_ANY_RULE	Drop any rule in any schema
SYS.DBMS_RULE_ADM.DROP_ANY_RULE_SET	Drop any rule set in any schema

Table 92–14 (Cont.) System Privileges for Evaluation Contexts, Rules, and Rule Sets

Privilege	Description
SYS.DBMS_RULE_ADM.EXECUTE_ANY_EVALUATION_CONTEXT	Execute any evaluation context owned by any user
SYS.DBMS_RULE_ADM.EXECUTE_ANY_RULE	Execute any rule owned by any user
SYS.DBMS_RULE_ADM.EXECUTE_ANY_RULE_SET	Execute any rule set owned by any user

For example, to grant the strmadmin user the privilege to create a rule set in any schema, enter the following:

```
BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => SYS.DBMS_RULE_ADM.CREATE_ANY_RULE_SET,
    grantee      => 'strmadmin',
    grant_option => FALSE);
END;
/
```

Note: When you grant a privilege on "ANY" object (for example, ALTER_ANY_RULE), and the initialization parameter O7_DICTIONARY_ACCESSIBILITY is set to FALSE, you give the user access to that type of object in all schemas except the SYS schema. By default, the initialization parameter O7_DICTIONARY_ACCESSIBILITY is set to FALSE.

If you want to grant access to an object in the SYS schema, then you can grant object privileges explicitly on the object. Alternatively, you can set the O7_DICTIONARY_ACCESSIBILITY initialization parameter to TRUE. Then privileges granted on "ANY" object will allow access to any schema, including SYS.

REMOVE_RULE Procedure

This procedure removes the specified rule from the specified rule set.

Syntax

```
DBMS_RULE_ADM.REMOVE_RULE(
   rule_name                IN  VARCHAR2,
   rule_set_name            IN  VARCHAR2,
   evaluation_context       IN  VARCHAR2  DEFAULT NULL,
   all_evaluation_contexts  IN  BOOLEAN   DEFAULT FALSE);
```

Parameters

Table 92–15 REMOVE_RULE Procedure Parameters

Parameter	Description
rule_name	The name of the rule you are removing from the rule set, specified as [*schema_name.*]*rule_name*. For example, to remove a rule named all_a in the hr schema, enter hr.all_a for this parameter. If the schema is not specified, then the current user is the default.
rule_set_name	The name of the rule set from which you are removing the rule, specified as [*schema_name.*]*rule_set_name*. For example, to remove the rule from a rule set named apply_rules in the hr schema, enter hr.apply_rules for this parameter. If the schema is not specified, then the current user is the default.
evaluation_context_name	The name of the evaluation context associated with the rule you are removing, specified as [*schema_name.*]*evaluation_context_name*. For example, to specify an evaluation context named dept_eval_context in the hr schema, enter hr.dept_eval_context for this parameter. If the schema is not specified, then the current user is the default. If an evaluation context was specified for the rule you are removing when you added the rule to the rule set using the ADD_RULE procedure, then specify the same evaluation context. If you added the same rule more than once with different evaluation contexts, then specify the rule with the evaluation context you want to remove. If you specify an evaluation context that is not associated with the rule, then the procedure raises an error. Specify NULL if you did not specify an evaluation context when you added the rule to the rule set. If you specify NULL and there are one or more evaluation contexts associated with the rule, then the procedure raises an error.
all_evaluation_contexts	If TRUE, then the procedure removes the rule from the rule set with all of its associated evaluation contexts. If FALSE, then the procedure only removes the rule with the specified evaluation context. This parameter is relevant only if the same rule is added more than once to the rule set with different evaluation contexts.

Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Have ALTER_ON_RULE_SET privilege on the rule set
- Have ALTER_ANY_RULE_SET system privilege
- Be the owner of the rule set

Note: This procedure does not drop a rule from the database. To drop a rule from the database, use the DROP_RULE procedure.

REVOKE_OBJECT_PRIVILEGE Procedure

This procedure revokes the specified object privilege on the specified object from the specified user or role.

Syntax

```
DBMS_RULE_ADM.REVOKE_OBJECT_PRIVILEGE(
   privilege    IN  BINARY_INTEGER,
   object_name  IN  VARCHAR2,
   revokee      IN  VARCHAR2);
```

Parameters

Table 92–16 REVOKE_OBJECT_PRIVILEGE Procedure Parameters

Parameter	Description
`privilege`	The name of the object privilege on the object to revoke from the revokee. See "GRANT_OBJECT_PRIVILEGE Procedure" on page 92-20 for a list of the object privileges.
`object_name`	The name of the object for which you are revoking the privilege from the revokee, specified as `[`*`schema_name.`*`]`*`object_name`*. For example, to revoke an object privilege on a rule set named `apply_rules` in the `hr` schema, enter `hr.apply_rules` for this parameter. If the schema is not specified, then the current user is the default. The object must be an existing rule, rule set, or evaluation context.
`revokee`	The name of the user or role from which the privilege is revoked. The user who owns the object cannot be specified.

REVOKE_SYSTEM_PRIVILEGE Procedure

This procedure revokes the specified system privilege from the specified user or role.

Syntax

```
DBMS_RULE_ADM.REVOKE_SYSTEM_PRIVILEGE(
   privilege  IN  BINARY_INTEGER,
   revokee    IN  VARCHAR2);
```

Parameters

Table 92–17 REVOKE_SYSTEM_PRIVILEGE Procedure Parameters

Parameter	Description
`privilege`	The name of the system privilege to revoke from the revokee. See "GRANT_SYSTEM_PRIVILEGE Procedure" on page 92-22 for a list of the system privileges.
`revokee`	The name of the user or role from which the privilege is revoked

93

DBMS_SCHEDULER

The DBMS_SCHEDULER package provides a collection of scheduling functions and procedures that are callable from any PL/SQL program.

> **See Also:** *Oracle Database Administrator's Guide* for more information regarding how to use DBMS_SCHEDULER

This chapter contains the following topics:

Using DBMS_SCHEDULER

This section contains topics which relate to using the `DBMS_SCHEDULER` package.

Rules and Limits

The following rules apply when using the DBMS_SCHEDULER package:

- Only SYS can do anything in SYS schema.
- Several of the procedures accept comma-delimited lists of object names. When a list of names is provided, the Scheduler will stop executing the list on the very first object that returns an error. This means that the Scheduler will not perform the task on the objects in the list after the one that caused the error. For example, consider the statement DBMS_SCHEDULER.STOP_JOB ('job1, job2, job3, sys.jobclass1, sys.jobclass2, sys.jobclass3'); If job3 could not be stopped, then job1 and job2 will be stopped, but the jobs in jobclass1, jobclass2, and jobclass3 will not be stopped.
- Performing an action on an object that does not exist returns a PL/SQL exception stating that the object does not exist.

Operational Notes

The Scheduler uses a rich **calendaring syntax** to enable you to define repeating schedules, such as "every Tuesday and Friday at 4:00 p.m." or "the second Wednesday of every month." This calendaring syntax is used in calendaring expressions in the `repeat_interval` argument of a number of package subprograms. Evaluating a calendaring expression results in a set of discrete timestamps.

See *Oracle Database Administrator's Guide* for examples of the calendaring syntax.

Calendaring Syntax

The calendaring syntax is as follows:

```
repeat_interval = regular_schedule | combined_schedule

regular_schedule = frequency_clause
[";" interval_clause] [";" bymonth_clause] [";" byweekno_clause]
[";" byyearday_clause] [";" bydate_clause] [";" bymonthday_clause]
[";" byday_clause] [";" byhour_clause] [";" byminute_clause]
[";" bysecond_clause] [";" bysetpos_clause] [";" include_clause]
[";" exclude_clause] [";" intersect_clause][";" periods_clause]
[";" byperiod_clause]

combined_schedule = schedule_list [";" include_clause]
[";" exclude_clause] [";" intersect_clause]

frequency_clause = "FREQ" "=" ( predefined_frequency | user_defined_frequency )
predefined_frequency = "YEARLY" | "MONTHLY" | "WEEKLY" | "DAILY" |
   "HOURLY" | "MINUTELY" | "SECONDLY"
user_defined_frequency = named_schedule

interval_clause = "INTERVAL" "=" intervalnum
   intervalnum = 1 through 99
bymonth_clause = "BYMONTH" "=" monthlist
   monthlist = monthday ( "," monthday)*
   month = numeric_month | char_month
   numeric_month = 1 | 2 | 3 ...  12
   char_month = "JAN" | "FEB" | "MAR" | "APR" | "MAY" | "JUN" |
   "JUL" | "AUG" | "SEP" | "OCT" | "NOV" | "DEC"
byweekno_clause = "BYWEEKNO" "=" weeknumber_list
   weeknumber_list = weeknumber ( "," weeknumber)*
   weeknumber = [minus] weekno
   weekno = 1 through 53
byyearday_clause = "BYYEARDAY" "=" yearday_list
   yearday_list = yearday ( "," yearday)*
   yearday = [minus] yeardaynum
   yeardaynum = 1 through 366
bydate_clause = "BYDATE" "=" date_list
   date_list = date ( "," date)*
   date = [YYYY]MMDD [ offset | span ]
bymonthday_clause = "BYMONTHDAY" "=" monthday_list
   monthday_list = monthday ( "," monthday)*
   monthday = [minus] monthdaynum
   monthdaynum = 1 through 31
byday_clause = "BYDAY" "=" byday_list
   byday_list = byday ( "," byday)*
   byday = [weekdaynum] day
   weekdaynum = [minus] daynum
   daynum = 1 through 53 /* if frequency is yearly */
```

```
   daynum = 1 through 5  /* if frequency is monthly */
   day = "MON" | "TUE" | "WED" | "THU" | "FRI" | "SAT" | "SUN"
byhour_clause = "BYHOUR" "=" hour_list
   hour_list = hour ( "," hour)*
   hour = 0 through 23
byminute_clause = "BYMINUTE" "=" minute_list
   minute_list = minute ( "," minute)*
   minute = 0 through 59
bysecond_clause = "BYSECOND" "=" second_list
   second_list = second ( "," second)*
   second = 0 through 59
bysetpos_clause = "BYSETPOS" "=" setpos_list
   setpos_list = setpos ("," setpos)*
   setpos = [minus] setpos_num
   setpos_num = 1 through 9999

include_clause = "INCLUDE" "=" schedule_list
exclude_clause = "EXCLUDE" "=" schedule_list
intersect_clause = "INTERSECT" "=" schedule_list
schedule_list = schedule_clause ("," schedule_clause)*
schedule_clause = named_schedule [ offset ]
named_schedule = [schema "."] schedule
periods_clause = "PERIODS" "=" periodnum
byperiod_clause = "BYPERIOD" "=" period_list
period_list = periodnum ("," periodnum)*
periodnum = 1 through 100

offset = ("+" | "-") ["OFFSET:"] duration_val
span = ("+" | "-" | "^") "SPAN:" duration_val
duration_val = dur-weeks | dur_days
dur_weeks = numofweeks "W"
dur_days = numofdays "D"
numofweeks = 1 through 53
numofdays = 1 through 376
minus = "-"
```

In calendaring syntax, * means 0 or more.

Table 93–1 Values for repeat_interval

Name	Description
FREQ	This specifies the type of recurrence. It must be specified. The possible predefined frequency values are YEARLY, MONTHLY, WEEKLY, DAILY, HOURLY, MINUTELY, and SECONDLY. Alternatively, specifies an existing schedule to use as a user-defined frequency.
INTERVAL	This specifies a positive integer representing how often the recurrence repeats. The default is 1, which means every second for secondly, every day for daily, and so on. The maximum value is 999.
BYMONTH	This specifies which month or months you want the job to execute in. You can use numbers such as 1 for January and 3 for March, as well as three-letter abbreviations such as FEB for February and JUL for July.

Table 93–1 (Cont.) Values for repeat_interval

Name	Description
BYWEEKNO	This specifies the week of the year as a number. It follows ISO-8601, which defines the week as starting with Monday and ending with Sunday; and the first week of a year as the first week, which is mostly within the Gregorian year. That last definition is equivalent to the following two variants: the week that contains the first Thursday of the Gregorian year; and the week containing January 4th. The ISO-8601 week numbers are integers from 1 to 52 or 53; parts of week 1 may be in the previous calendar year; parts of week 52 may be in the following calendar year; and if a year has a week 53, parts of it must be in the following calendar year. As an example, in the year 1998 the ISO week 1 began on Monday December 29th, 1997; and the last ISO week (week 53) ended on Sunday January 3rd, 1999. So December 29th, 1997, is in the ISO week 1998-01; and January 1st, 1999, is in the ISO week 1998-53. byweekno is only valid for YEARLY. Examples of invalid specifications are "FREQ=YEARLY; BYWEEKNO=1; BYMONTH=12" and "FREQ=YEARLY;BYWEEKNO=53;BYMONTH=1".
BYYEARDAY	This specifies the day of the year as a number. Valid values are 1 to 366. An example is 69, which is March 10 (31 for January, 28 for February, and 10 for March). 69 evaluates to March 10 for non-leap years and March 9 in leap years. -2 will always evaluate to December 30th independent of whether it is a leap year.
BYDATE	This specifies a list of dates, where each date is of the form [YYYY]MMDD. A list of consecutive dates can be generated by using the SPAN modifier, and a date can be adjusted with the OFFSET modifier. An example of a simple BYDATE clause is the following: BYDATE=0115,0315,0615,0915,1215,20060115 The following SPAN example is equivalent to BYDATE=0110,0111,0112,0113,0114, which is a span of 5 days starting at 1/10: BYDATE=0110+SPAN:5D The plus sign in front of the SPAN keyword indicates a span starting at the supplied date. The minus sign indicates a span ending at the supplied date, and the "^" sign indicates a span of n days or weeks centered around the supplied date. If n is an even number, it is adjusted up to the next odd number. Offsets adjust the supplied date by adding or subtracting n days or weeks. BYDATE=0205-OFFSET:2W is equivalent to BYDATE=0205-14D (the OFFSET: keyword is optional), which is also equivalent to BYDATE=0122.
BYMONTHDAY	This specifies the day of the month as a number. Valid values are 1 to 31. An example is 10, which means the 10th day of the selected month. You can use the minus sign (-) to count backward from the last day, so, for example, BYMONTHDAY=-1 means the last day of the month and BYMONTHDAY=-2 means the next to last day of the month.
BYDAY	This specifies the day of the week from Monday to Sunday in the form MON, TUE, and so on. Using numbers, you can specify the 26th Friday of the year, if using a YEARLY frequency, or the 4th THU of the month, using a MONTHLY frequency. Using the minus sign, you can say the second to last Friday of the month. For example, -1 FRI is the last Friday of the month.
BYHOUR	This specifies the hour on which the job is to run. Valid values are 0 to 23. As an example, 10 means 10 a.m.
BYMINUTE	This specifies the minute on which the job is to run. Valid values are 0 to 59. As an example, 45 means 45 minutes past the chosen hour.

Table 93–1 (Cont.) Values for repeat_interval

Name	Description
BYSECOND	This specifies the second on which the job is to run. Valid values are 0 to 59. As an example, 30 means 30 seconds past the chosen minute.
BYSETPOS	This selects one or more items by position in the list of timestamps that result after the whole calendaring expression is evaluated. It is useful for requirements such as running a job on the last workday of the month. Rather than attempting to express this with the other BY clauses, you can code the calendaring expression to evaluate to a list of every workday of the month, and then add the BYSETPOS clause to select only the last item of that list. Assuming that workdays are Monday through Friday, the syntax would then be: `FREQ=MONTHLY; BYDAY=MON,TUE,WED,THU,FRI; BYSETPOS=-1` Valid values are 1 through 9999. A negative number selects an item from the end of the list (-1 is the last item, -2 is the next to last item, and so on) and a positive number selects from the front of the list. The BYSETPOS clause is always evaluated last. BYSETPOS is only supported with the MONTHLY and YEARLY frequencies. The BYSETPOS clause is applied to the list of timestamps once per frequency period. For example, when the frequency is defined as MONTHLY, the Scheduler determines all valid timestamps for the month, orders that list, and then applies the BYSETPOS clause. The Scheduler then moves on to the next month and repeats the procedure. Assuming a start date of Jun 10, 2004, the example evaluates to: Jun 30, Jul 30, Aug 31, Sep 30, Oct 29, and so on.
INCLUDE	This includes one or more named schedules in the calendaring expression. That is, the set of timestamps defined by each included named schedule is added to the results of the calendaring expression. If an identical timestamp is contributed by both an included schedule and the calendaring expression, it is included in the resulting set of timestamps only once. The named schedules must have been defined with the CREATE_SCHEDULE procedure.
EXCLUDE	This excludes one or more named schedules from the calendaring expression. That is, the set of timestamps defined by each excluded named schedule is removed from the results of the calendaring expression. The named schedules must have been defined with the CREATE_SCHEDULE procedure.

INTERSECT	This specifies an intersection between the calendaring expression results and the set of timestamps defined by one or more named schedules. Only the timestamps that appear both in the calendaring expression and in one of the named schedules are included in the resulting set of timestamps.
	For example, assume that the named schedule last_sat indicates the last Saturday in every month, and that for the year 2005, the only months where the last day of the month is also a Saturday are April and December. Assume also that the named schedule end_qtr indicates the last day of each quarter in 2005:
	`3/31/2005, 6/30/2005, 9/30/2005, 12/31/2005`
	The following calendaring expression results in the these dates:
	`3/31/2005, 4/30/2005, 6/30/2005, 9/30/2005, 12/31/2005`
	`FREQ=MONTHLY; BYMONTHDAY=-1; INTERSECT=last_sat,end_qtr`
	In this example, the terms FREQ=MONTHLY; BYMONTHDAY=-1 indicate the last day of each month.

Table 93–1 (Cont.) Values for repeat_interval

Name	Description
PERIODS	This identifies the number of periods that together form one cycle of a user defined frequency. It is used in the repeat_interval expression of the schedule that defines the user defined frequency. It is mandatory when the repeat_interval expression in the main schedule contains a BYPERIOD clause. The following example defines the quarters of a fiscal year. FREQ=YEARLY;BYDATE=0301,0601,0901,1201;PERIODS=4
BYPERIOD	This selects periods from a user defined frequency. For example, if a main schedule names a user defined frequency schedule that defines the fiscal quarters shown in the previous example, the clause BYPERIOD=2,4 in the main schedule selects the 2nd and 4th fiscal quarters.

Combining Schedules

There are two ways to combine schedules:

- Using a combined schedule expression, which is a list of individual schedules

  For example, to create a schedule for all company holidays, you provide a list of individual schedules, where each schedule in the list defines a single holiday. The Scheduler evaluates each individual schedule, and then returns a union of the timestamps returned by each individual schedule. You can follow the initial list of individual schedules with include, exclude, and intersect clauses to create more complex combinations.

- Embedding other schedules into the main schedule using include, exclude, and intersect clauses

  With this method, the embedded schedules inherit certain attributes from the main schedule.

  – Timestamps generated by the INCLUDE clause that fall into periods that are skipped by the main schedule are ignored. This is the case when the main schedule skips periods due to the INTERVAL clause, the BYPERIOD clause, or the BYMONTH clause for freq=monthly.

  – Days that are added by the INCLUDE clause follow the hourly/minutely/secondly execution pattern of the main schedule.

  – When the INCLUDE clause is present, no date-specific defaults are retrieved from the start date (but time-specific defaults can be). (See "Start Dates and Repeat Intervals", later in this section.) For example, a repeat_interval of FREQ=MONTHLY;INCLUDE=HOLIDAY executes only on holidays and not on the month/day defaults retrieved from the start date.

  The following is an example:

```
BEGIN
dbms_scheduler.create_schedule('embed_sched', repeat_interval =>
  'FREQ=YEARLY;BYDATE=0130,0220,0725');
dbms_scheduler.create_schedule('main_sched', repeat_interval =>
  'FREQ=MONTHLY;INTERVAL=2;BYMONTHDAY=15;BYHOUR=9,17;INCLUDE=embed_sched');
END;
/
```

  In this example, the dates 1/30, 2/20, and 7/25 are added to the main schedule. However, the Scheduler does not include dates that fall in months that are skipped by the INTERVAL clause. If the start date of the main schedule is 1/1/2005, then 2/20 isn't added. On the dates that are added, the embedded schedule follows the

execution pattern of the main schedule: jobs are executed at 9:00 a.m. and 5:00 p.m. on 1/30 and 7/25. If the embedded schedule does not itself have a start date, it inherits the start date from the main schedule.

User Defined Frequencies

Instead of using predefined frequencies like DAILY, WEEKLY, MONTHLY, and so on, you can create your own frequencies by creating a schedule that returns the start date of each period. For example, the following repeat_interval expression is used in a schedule named fiscal_year that defines the start of each quarter in a fiscal year:

```
FREQ=YEARLY;BYDATE=0301,0601,0901,1201;PERIODS=4
```

To return the last Wednesday of every quarter, you create a schedule (the "main schedule") that uses the fiscal_year schedule as a user defined frequency:

```
FREQ=fiscal_year;BYDAY=-1WED
```

Periods in a user defined frequency do not have to be equal in length. In the main schedule, the BYSETPOS clause and numbered weekdays are recalculated based on the size of each period. To select dates in specific periods, you must use the BYPERIOD clause in the main schedule. To enable this, the schedule that is used as the user defined frequency must include a PERIODS clause, and it must set its start date appropriately. The first date returned by this schedule is used as the starting point of period 1.

As another example, assuming work days are Monday through Friday, to get the last work day of the 2nd and 4th quarters of the fiscal year, the repeat_interval clause in the main schedule is the following:

```
FREQ=fiscal_year;BYDAY=MON,TUE,WED,THU,FRI;BYPERIOD=2,4;BYSETPOS=-1
```

Start Dates and Repeat Intervals

The Scheduler retrieves the date and time from the job or schedule start date and incorporates them as defaults into the repeat_interval. For example, if the specified frequency is yearly and there is no BYMONTH or BYMONTHDAY clause in the repeat interval, the month and day on which to run the job are retrieved from the start date. Similarly, if frequency is monthly but there is no BYMONTHDAY clause in the repeat interval, the day of the month on which to run the job is retrieved from the start date. If present, BYHOUR, BYMINUTE, and BYSECOND defaults are also retrieved from the start date, and used if those clauses are not specified. Note that if the INCLUDE, EXCLUDE, or INTERSECT clauses are present, no date-related defaults are retrieved from the start date, but time-related defaults are.

The following are some examples:

```
start_date:      4/15/05 9:00:00
repeat_interval: freq=yearly

is expanded internally to:

freq=yearly;bymonth=4;bymonthday=15;byhour=9;byminute=0;bysecond=0
```

The preceding schedule executes on 04/15/05 9:00:00, 04/15/06 9:00:00, 04/15/07 9:00:00, and so on.

For the next example, assume that schedule S1 has a repeat_interval of FREQ=YEARLY;BYDATE=0701.

```
start_date:      01/20/05 9:00:00
repeat_interval: freq=yearly;include=S1
```

```
is expanded internally to:

freq=yearly;byhour=9;byminute=0;bysecond=0;include=S1
```

Because an INCLUDE clause is present, date-related information is not retrieved from the start date. However, time-specific information is, so the preceding schedule executes on 07/01/05 9:00:00, 07/01/06 9:00:00, 07/01/08 9:00:00, and so on.

General Rules

When using a calendaring expression, consider the following rules:

- For a regular schedule (as opposed to a combined schedule), the calendar string must start with the frequency clause. All other clauses are optional and can be put in any order.
- All clauses are separated by a semi-colon, and each clause can be present at most once, with the exception of the include, exclude, and intersect clauses.
- Spaces are allowed between syntax elements and the strings are case insensitive.
- The list of values for a specific BY clause do not need to be ordered.
- When not enough BY clauses are present to determine what the next date is, this information is retrieved from the start date. For example, "FREQ=YEARLY" with a start date of 02/15/2003 becomes "FREQ=YEARLY;BYMONTH=FEB; BYMONTHDAY=15", which means every year on the 15th of February.

  "FREQ=YEARLY;BYMONTH=JAN,JUL" with start date 01/21/2003 becomes "FREQ=YEARLY;BYMONTH=JAN,JUL;BYMONTHDAY=21", which means every year on January 21 and July 21.
- The byweekno clause is only allowed if the frequency is YEARLY. It cannot be used with other frequencies. When it is present, it will return all days in that week number. If you want to limit it to specific days within the week, you have to add a BYDAY clause. For example, "FREQ=YEARLY;BYWEEKNO=2" with a start date of 01/01/2003 will return:

  ```
  01/06/2003, 01/07/2003, 01/08/2003, 01/09/2003, 01/10/2003, 01/11/2003,
  01/12/2003, 01/05/2004, 01/06/2004, 01/07/2004, .... and so on.
  ```

  Note that when the byweekno clause is used, it is possible that the dates returned are from a year other than the current year. For example, if returning dates for the year 2004 and the calendar string is "FREQ=YEARLY;BYWEEKNO=1,53" for the specified week numbers in 2004, it will return the dates:

  ```
  12/29/03, 12/30/03, 12/31/03, 01/01/04, 01/02/04, 01/03/04, 01/04/04, 12/27/04,
  12/28/04, 12/29/04, 12/30/04, 12/31/04, 01/01/05, 01/02/05
  ```

- For those BY clauses that do not have a consistent range of values, you can count backward by putting a "-" in front of the numeric value. For example, specifying BYMONTHDAY=31 will not give you the last day of every month, because not every month has 31 days. Instead, BYMONTHDAY=-1 will give you the last day of the month.

  This is not supported for BY clauses that are fixed in size. In other words, BYMONTH, BYHOUR, BYMINUTE, and BYSECOND are not supported.
- The basic values for the BYDAY clause are the days of the week. When the frequency is YEARLY, or MONTHLY, you are allowed to specify a positive or negative number in front of each day of the week. In the case of YEARLY,

BYDAY=40MON, indicates the 40th Monday of the year. In the case of MONTHLY, BYDAY=-2SAT, indicates the second to last Saturday of the month.

Note that positive or negative numbers in front of the weekdays are not supported for other frequencies and that in the case of yearly, the number ranges from -53 ... -1, 1 ... 53, whereas for the monthly frequency it is limited to -5 ... -1, 1... 5.

If no number is present in front of the weekday it specifies, every occurrence of that weekday in the specified frequency.

- The first day of the week is Monday.
- Repeating jobs with frequencies smaller than daily follow their frequencies exactly across daylight savings adjustments. For example, suppose that a job is scheduled to repeat every 3 hours, the clock is moved forward from 1:00 a.m. to 2:00 a.m., and the last time the job ran was midnight. Its next scheduled time will be 4:00 a.m. Thus, the 3 hour period between subsequent job runs is retained. The same applies when the clock is moved back. This behavior is not the case for repeating jobs that have frequencies of daily or larger. For example, if a repeating job is supposed to be executed on a daily basis at midnight, it will continue to run at midnight if the clock is moved forward or backward. When the execution time of such a daily (or larger frequency) job happens to fall inside a window where the clock is moved forward, the job executes at the end of the window.
- The calendaring syntax does not allow you to specify a time zone. Instead the Scheduler retrieves the time zone from the start_date argument. If jobs must follow daylight savings adjustments you must make sure that you specify a region name for the time zone of the start_date. For example specifying the start_date time zone as 'US/Eastern' in New York will make sure that daylight saving adjustments are automatically applied. If instead the time zone of the start_date is set to an absolute offset, such as '-5:00', daylight savings adjustments are not followed and your job execution will be off by an hour half of the year.
- When start_date is NULL, the Scheduler will determine the time zone for the repeat interval as follows:

  1. It will check whether the session time zone is a region name. The session time zone can be set by either:
     - Issuing an ALTER SESSION statement, for example:

       ```
       SQL> ALTER SESSION SET time_zone = 'Asia/Shanghai';
       ```

     - Setting the ORA_SDTZ environment variable.
  2. If the session time zone is an absolute offset instead of a region name, the Scheduler will use the value of the DEFAULT_TIMEZONE Scheduler attribute. For more information, see the SET_SCHEDULER_ATTRIBUTE Procedure.
  3. If the DEFAULT_TIMEZONE attribute is NULL, the Scheduler will use the time zone of systimestamp when the job or window is enabled.

BYSETPOS Clause Rules

- The BYSETPOS clause is the last clause to be evaluated. It is processed after all other BY clauses and the INCLUDE, EXCLUDE and INTERSECT clauses have been evaluated.
- The INTERVAL clause does not change the size of the period to which the BYSETPOS clause is applied. For example, when the frequency is set to monthly and interval is set to 3, the list of timestamps to which BYSETPOS is applied is generated from a month, not a quarter. The only impact of the INTERVAL clause is

to cause months to be skipped. However, you can still select the second to last workday of the quarter like this:

```
FREQ=MONTHLY;INTERVAL=3;BYDAY=MON,TUE,WED,THU,FRI;BYSETPOS=-2
```

provided that you set the start date in the right month. This example returns the next to last workday of a month, and repeats once a quarter.

- To get consistent results, the set to which `BYSETPOS` is applied is determined from the beginning of the frequency period independently of when the evaluation occurs. Whether the Scheduler evaluates

  ```
  FREQ=MONTHLY;BYDAY=MON,TUE,FRI;BYSETPOS=1,3
  ```

  on 01/01/2004 or 01/15/2004, in both cases the expression evaluates to Friday 01/02/2004, and Tuesday 01/06/2004. The only difference is that when the expression is evaluated on 01/15/2004, the Scheduler determines that there are no matches in January because the timestamps found are in the past, and it moves on to the matches in the next month, February.

BYDATE Clause Rules

- If dates in the `BYDATE` clause do not have their optional year component, the job runs on those dates every year.
- The job execution times on the included dates are derived from the `BY` clauses in the calendaring expression. For example, if `repeat_interval` is defined as

  ```
  freq=daily;byhour=8,13,18;byminute=0;bysecond=0;bydate=0502,0922
  ```

  then the execution times on 05/02 and 09/22 are 8:00 a.m., 1:00 p.m., and 6:00 p.m.

EXCLUDE Clause Rules

- Excluded dates without a time component are 24 hour periods. All timestamps that fall on an excluded date are removed. In the following example, `jan_fifteen` is a named schedule that resolves to the single date of 01/15:

  ```
  freq=monthly;bymonthday=15,30;byhour=8,13,18;byminute=0;bysecond=0;
       exclude=jan_fifteenth
  ```

  In this case, all three instances of the job are removed for 01/15.

OFFSET Rules

- You can adjust the dates of individual named schedules by adding positive offsets to them. For example, to execute `JOB2` exactly 15 days after every occurrence of `JOB1`, add `+OFFSET:15D` to the schedule of `JOB1`, as follows:

  ```
  BEGIN
  dbms_scheduler.create_schedule('job2_schedule', repeat_interval =>
    'job1_schedule+OFFSET:15D');
  END;
  /
  ```

  Note that negative offsets to named schedules are not supported.

Example: Putting It All Together

This example demonstrates the use of user defined frequencies, spans, offsets, and the `BYSETPOS` and `INCLUDE` clauses. (Note that the `OFFSET:` keyword in an offset clause is optional.)

Many companies in the retail industry share the same fiscal year. The fiscal year starts on the Sunday closest to February 1st, and subsequent quarters start exactly 13 weeks later. The fiscal year schedule for the retail industry can be defined as the following:

```
begin
 dbms_scheduler.create_schedule('year_start', repeat_interval=>
       'FREQ=YEARLY;BYDATE=0201^SPAN:1W;BYDAY=SUN');
 dbms_scheduler.create_schedule('retail_fiscal_year',
        to_timestamp_tz('15-JAN-2005 12:00:00','DD-MON-YYYY HH24:MI:SS'),
         'year_start,year_start+13w,year_start+26w,year_start+39w;periods=4');
end;
/
```

The following schedule can be used to execute a job on the 5th day off in the 2nd and the 4th quarters of the retail industry. This assumes that Saturday and Sunday are off days as well as the days in the existing `holiday` schedule.

```
begin
 dbms_scheduler.create_schedule('fifth_day_off', repeat_interval=>
  'FREQ=retail_fiscal_year;BYDAY=SAT,SUN;INCLUDE=holiday;
    BYPERIOD=2,4;BYSETPOS=5');
end;
/
```

Summary of DBMS_SCHEDULER Subprograms

Table 93–2 DBMS_SCHEDULER Package Subprograms

Subprogram	Description
ADD_EVENT_QUEUE_SUBSCRIBER Procedure on page 93-17	Adds a user as a subscriber to the Scheduler event queue `SYS.SCHEDULER$_EVENT_QUEUE`
ADD_WINDOW_GROUP_MEMBER Procedure on page 93-18	Adds a window to an existing window group
ALTER_CHAIN Procedure on page 93-19	Alters specified steps of a chain
ALTER_RUNNING_CHAIN Procedure on page 93-20	Alters specified steps of a running chain
CLOSE_WINDOW Procedure on page 93-22	Closes an open window prematurely
COPY_JOB Procedure on page 93-23	Copies an existing job
CREATE_CHAIN Procedure on page 93-24	Creates a chain, which is a named series of programs that are linked together for a combined objective
CREATE_EVENT_SCHEDULE Procedure on page 93-25	Creates an event schedule, which is a schedule that starts a job based on the detection of an event
CREATE_JOB Procedures on page 93-27	Creates a job
CREATE_JOB_CLASS Procedure on page 93-32	Creates a job class, which provides a way to group jobs for resource allocation and prioritization
CREATE_PROGRAM Procedure on page 93-34	Creates a program
CREATE_SCHEDULE Procedure on page 93-36	Creates a schedule
CREATE_WINDOW Procedures on page 93-37	Creates a window, which provides a way to automatically activate different resource plans at different times
CREATE_WINDOW_GROUP Procedure on page 93-39	Creates a window group
DEFINE_ANYDATA_ARGUMENT Procedure on page 93-40	Defines a program argument whose value is of a complex type and must be passed encapsulated in an `AnyData` object
DEFINE_CHAIN_EVENT_STEP Procedure on page 93-41	Adds or replaces a chain step and associates it with an event schedule or inline event. See also: `DEFINE_CHAIN_STEP`.
DEFINE_CHAIN_RULE Procedure on page 93-42	Adds a rule to an existing chain
DEFINE_CHAIN_STEP Procedure on page 93-45	Defines a chain step, which can be a program or another (nested) chain. See also: `DEFINE_CHAIN_EVENT_STEP`.

Table 93–2 (Cont.) DBMS_SCHEDULER Package Subprograms

Subprogram	Description
DEFINE_METADATA_ ARGUMENT Procedure on page 93-46	Defines a special metadata argument for the program. You can retrieve specific metadata through this argument
DEFINE_PROGRAM_ ARGUMENT Procedures on page 93-48	Defines a program argument whose value can be passed as a string literal to the program
DISABLE Procedure on page 93-50	Disables a program, job, chain, window, or window group
DROP_CHAIN Procedure on page 93-52	Drops an existing chain
DROP_CHAIN_RULE Procedure on page 93-53	Removes a rule from an existing chain
DROP_CHAIN_STEP Procedure on page 93-54	Drops a chain step
DROP_JOB Procedure on page 93-55	Drops a job or all jobs in a job class
DROP_JOB_CLASS Procedure on page 93-56	Drops a job class
DROP_PROGRAM Procedure on page 93-57	Drops a program
DROP_PROGRAM_ ARGUMENT Procedures on page 93-58	Drops a program argument
DROP_SCHEDULE Procedure on page 93-59	Drops a schedule
DROP_WINDOW Procedure on page 93-60	Drops a window
DROP_WINDOW_ GROUP Procedure on page 93-61	Drops a window group
ENABLE Procedure on page 93-62	Enables a program, job, chain, window, or window group
EVALUATE_ CALENDAR_STRING Procedure on page 93-63	Evaluates the calendar string and tells you what the next execution date of a job or window will be
EVALUATE_RUNNING_ CHAIN Procedure on page 93-65	Forces reevaluation of the rules of a running chain to trigger any rules for which the conditions have been satisfied
GENERATE_JOB_NAME Function on page 93-66	Generates a unique name for a job. This enables you to identify jobs by adding a prefix, so, for example, Sally's jobs would be named `sally1`, `sally2`, and so on
GET_ATTRIBUTE Procedure on page 93-67	Retrieves the value of an attribute of an object
GET_SCHEDULER_ ATTRIBUTE Procedure on page 93-68	Retrieves the value of a Scheduler attribute
OPEN_WINDOW Procedure on page 93-69	Opens a window prematurely. The window is opened immediately for the duration

Table 93–2 (Cont.) DBMS_SCHEDULER Package Subprograms

Subprogram	Description
PURGE_LOG Procedure on page 93-70	Purges specific rows from the job and window logs
REMOVE_EVENT_QUEUE_SUBSCRIBER Procedure on page 93-71	Unsubscribes a user from the Scheduler event queue `SYS.SCHEDULER$_EVENT_QUEUE`
REMOVE_WINDOW_GROUP_MEMBER Procedure on page 93-72	Removes a window from an existing window group. This fails if the specified window is not a member of the given group
RESET_JOB_ARGUMENT_VALUE Procedures on page 93-73	Resets the current value assigned to an argument defined with the associated program
RUN_CHAIN Procedures on page 93-74	Immediately runs a chain by creating a run-once job
RUN_JOB Procedure on page 93-76	Runs a job immediately
SET_ATTRIBUTE Procedure on page 93-78	Changes an attribute of a job, schedule, or other Scheduler object
SET_ATTRIBUTE_NULL Procedure on page 93-87	Changes an attribute of an object to `NULL`
SET_JOB_ANYDATA_VALUE Procedures on page 93-88	Sets the value of a job argument encapsulated in an `AnyData` object
SET_JOB_ARGUMENT_VALUE Procedures on page 93-89	Sets the value of a job argument
SET_SCHEDULER_ATTRIBUTE Procedure on page 93-90	Sets the value of a Scheduler attribute
STOP_JOB Procedure on page 93-92	Stops a currently running job or all jobs in a job class

ADD_EVENT_QUEUE_SUBSCRIBER Procedure

This procedure adds a user as a subscriber to the Scheduler event queue SYS.SCHEDULER$_EVENT_QUEUE, and grants the user permission to dequeue from this queue using the designated agent.

Syntax

```
DBMS_SCHEDULER.ADD_EVENT_QUEUE_SUBSCRIBER (
   subscriber_name         IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 93–3 ADD_EVENT_QUEUE_SUBSCRIBER Procedure Parameters

Parameter	Description
subscriber_name	Name of the Oracle Streams Advanced Queuing (AQ) agent to be used to subscribe to the Scheduler event queue. If NULL, an agent is created and assigned the user name of the calling user.

Usage Notes

The subscription is rule-based. The rule permits the user to see only events raised by jobs that the user owns, and filters out all other messages. If an AQ agent with the same name already exists, an error is raised.

ADD_WINDOW_GROUP_MEMBER Procedure

This procedure adds one or more windows to an existing window group.

Syntax

```
DBMS_SCHEDULER.ADD_WINDOW_GROUP_MEMBER (
   group_name              IN VARCHAR2,
   window_list             IN VARCHAR2);
```

Parameters

Table 93–4 ADD_WINDOW_GROUP_MEMBER Procedure Parameters

Parameter	Description
group_name	The name of the window group
window_list	The name of the window or windows

Usage Notes

If an already open window is added to a window group, the Scheduler will not pick up jobs that point to this window group until the next window in the window group opens.

Adding a window to a group requires the MANAGE SCHEDULER privilege.

Note that a window group cannot be a member of another window group.

ALTER_CHAIN Procedure

This procedure alters specified steps of a chain. This affects all future runs of the specified steps.

Syntax

```
DBMS_SCHEDULER.ALTER_CHAIN (
   chain_name              IN VARCHAR2,
   step_name               IN VARCHAR2,
   attribute               IN VARCHAR2,
   value                   IN BOOLEAN);
```

Parameters

Table 93–5 ALTER_CHAIN Procedure Parameters

Parameter	Description
chain_name	The name of the chain to alter
step_name	The name of the step or a comma-separated list of steps to alter. This cannot be NULL.
attribute	The attribute of the steps to change. This must be one of: 'PAUSE', 'SKIP', or 'RESTART_ON_RECOVERY' ■ 'PAUSE'—If the PAUSE attribute is set TRUE for a step, after the step has run, its state will be changed to PAUSED (and the completed attribute will remain FALSE). If PAUSE is reset to FALSE for a paused chain step (using ALTER_RUNNING_CHAIN), the state will be set to its completion state (SUCCEEDED, FAILED, or STOPPED) and the completed attribute will be set to TRUE. Setting PAUSE will have no effect on steps that have already run. This allows execution of a chain to be suspended after the execution of certain steps. ■ 'SKIP'—If the SKIP attribute is set TRUE for a step, when the step condition is met, instead of being run, the step is treated as if it has immediately succeeded. Setting SKIP to TRUE has no effect for a step that is running, scheduled to run after a delay, or has already run. If SKIP is set TRUE for a step that PAUSE is also set for, when the step condition is met, the step immediately changes to state PAUSED. ■ 'RESTART_ON_RECOVERY'—If the RESTART_ON_RECOVERY attribute is set to TRUE for a step, then if the step is stopped by a database shutdown, it is restarted when the database is recovered. If this attribute is set to FALSE, then if the step is stopped by a database shutdown, the step is marked as stopped when the database is recovered and the chain continues.
value	The value to set for the attribute. This must be one of: TRUE, FALSE.

Usage Notes

Altering a chain requires ALTER privileges on the chain either by being the owner of the chain, or by having the ALTER object privilege on the chain or by having the CREATE ANY JOB system privilege.

ALTER_RUNNING_CHAIN Procedure

This procedure alters specified steps of a running chain. This will affect only steps of this running instance of the chain.

Syntax

```
DBMS_SCHEDULER.ALTER_RUNNING_CHAIN (
   job_name                IN VARCHAR2,
   step_name               IN VARCHAR2,
   attribute               IN VARCHAR2,
   value                   IN [BOOLEAN|VARCHAR2]);
```

Parameters

Table 93–6 ALTER_RUNNING_CHAIN Procedure Parameters

Parameter	Description
job_name	The name of the job that is running the chain
step_name	The name of the step or a comma-separated list of steps to alter. If this is set to NULL and attribute is PAUSE or SKIP, then all steps of the running chain will be altered.
attribute	The attribute of the steps to change. This must be one of: 'PAUSE', 'SKIP', 'RESTART_ON_RECOVERY', or 'STATE'. ■ 'PAUSE'—If the PAUSE attribute is set TRUE for a step, after the step has run, its state will be changed to PAUSED (and the completed attribute will remain false). If PAUSE is reset to FALSE for a paused chain step (using ALTER_RUNNING_CHAIN), the state will be set to its completion state (SUCCEEDED, FAILED, or STOPPED) and the completed attribute will be set to TRUE. Setting PAUSE will have no effect on steps that have already run. This allows execution of a chain to be suspended after the execution of certain steps. If step_name is set to NULL, PAUSE will be set to TRUE for all steps of this running chain. ■ 'SKIP'—If the SKIP attribute is set to TRUE for a step, when the step condition is met, instead of being run, the step is treated as if it has immediately succeeded. Setting SKIP to TRUE has no effect for a step that is running, scheduled to run after a delay, or has already run. If step_name is set to NULL, SKIP will be set TRUE for all steps of this running chain. If SKIP is set TRUE for a step that PAUSE is also set for, when the step condition is met the step will immediately change to state PAUSED. ■ 'RESTART_ON_RECOVERY'—If the RESTART_ON_RECOVERY attribute is set to TRUE for a step, then if the step is stopped by a database shutdown, it is restarted when the database is recovered. If this attribute is set to FALSE, then if the step is stopped by a database shutdown, the step is marked as stopped when the database is recovered and the chain continues. ■ 'STATE'—This changes the state of the steps. The state can only be changed if the step is not running. The state can only be changed to one of the following: 'NOT_STARTED', 'SUCCEEDED', 'FAILED *error_code*' If the state is being changed to FAILED, an error code must be included (this must be a positive integer).
value	The value to set for the attribute. This must be one of: TRUE, FALSE, 'NOT_STARTED', 'SUCCEEDED', or 'FAILED *error_code*'

Usage Notes

Altering a running chain requires alter privileges on the job which is running (either by being the owner, or by having `ALTER` privileges on the job or by having the `CREATE ANY JOB` system privilege).

CLOSE_WINDOW Procedure

This procedure closes an open window prematurely. A closed window means that it is no longer in effect. When a window is closed, the Scheduler will switch the resource plan to the one that was in effect outside the window or in the case of overlapping windows to another window.

Syntax

```
DBMS_SCHEDULER.CLOSE_WINDOW (
   window_name             IN VARCHAR2);
```

Parameters

Table 93–7 CLOSE_WINDOW Procedure Parameters

Parameter	Description
window_name	The name of the window

Usage Notes

If you try to close a window that does not exist or is not open, an error is generated.

A job that is running will not stop when the window it is running in closes unless the attribute stop_on_window_close was set to TRUE for the job. However, the resources allocated to the job may change because the resource plan may change.

When a running job has a window group as its schedule, the job will not be stopped when its window is closed if another window that is also a member of the same window group then becomes active. This is the case even if the job has the attribute stop_on_window_close set to TRUE.

Closing a window requires the MANAGE SCHEDULER privilege.

COPY_JOB Procedure

This procedure copies all attributes of an existing job to a new job. The new job is created disabled, while the state of the existing job is unaltered.

Syntax

```
DBMS_SCHEDULER.COPY_JOB (
   old_job                IN VARCHAR2,
   new_job                IN VARCHAR2);
```

Parameters

Table 93–8 COPY_JOB Procedure Parameters

Parameter	Description
old_job	The name of the existing job
new_job	The name of the new job

Usage Notes

Copying a job requires privileges to create a job in the schema of the new job (the CREATE JOB system privilege if it is in the user's own schema, and the CREATE ANY JOB system privilege otherwise). If the old job is not in the user's own schema, then he needs to additionally have ALTER privileges on the old job or the CREATE ANY JOB system privilege.

CREATE_CHAIN Procedure

This procedure creates a new chain. The chain name can be optionally qualified with a schema name (for example, `myschema.myname`).

A chain is always created disabled and must be enabled with the ENABLE Procedure before it can be used.

Syntax

```
DBMS_SCHEDULER.CREATE_CHAIN (
   chain_name              IN VARCHAR2,
   rule_set_name           IN VARCHAR2 DEFAULT NULL,
   evaluation_interval     IN INTERVAL DAY TO SECOND DEFAULT NULL,
   comments                IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 93–9 CREATE_CHAIN Procedure Parameters

Parameter	Description
`chain_name`	The name of the new chain, which can optionally be qualified with a schema. This must be unique in the SQL namespace, so there must not already be a table or other object with this name and schema.
`rule_set_name`	In the normal case, no rule set should be passed in. The Scheduler will automatically create a rule set and associated empty evaluation context. You then use `DEFINE_CHAIN_RULE` to add rules and `DROP_CHAIN_RULE` to remove them. Advanced users can create a rule set that describes their chain dependencies and pass it in here. This allows greater flexibility in defining rules. For example, conditions can refer to external variables, and tables can be exposed through the evaluation context. If you pass in a rule set, you must ensure that it is in the format of a chain rule set. (For example, all steps must be listed as variables in the evaluation context). If no rule set is passed in, the rule set created will be of the form `SCHED_RULESET${N}` and the evaluation context created will be of the form `SCHED_EVCTX${N}` See *Oracle Streams Concepts and Administration* for information on rules and rule sets.
`evaluation_interval`	If this is `NULL`, reevaluation of the rules of a running chain are performed only when the job starts and when a step completes. A non-`NULL` value causes rule evaluations to also occur periodically at the specified interval. Because evaluation may be CPU-intensive, this should be conservatively set to the highest possible value or left at `NULL` if possible. `evaluation_interval` cannot be less than a minute or greater than a day.
`comments`	An optional comment describing the purpose of the chain

Usage Notes

Creating a chain in one's own schema requires the `CREATE JOB` system privilege. Creating a chain in a different schema requires the `CREATE ANY JOB` system privilege. If no `rule_set_name` is given, a rule set and evaluation context will be created in the schema that the chain is being created in, so the user will need to have the privileges required to create these objects. See the `DBMS_RULE_ADM.CREATE_RULE_SET` and `DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT` procedures for more information.

CREATE_EVENT_SCHEDULE Procedure

This procedure creates an event schedule, which is used to start a job when a particular event is raised.

Syntax

```
DBMS_SCHEDULER.CREATE_EVENT_SCHEDULE (
   schedule_name           IN VARCHAR2,
   start_date              IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
   event_condition         IN VARCHAR2,
   queue_spec              IN VARCHAR2,
   end_date                IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
   comments                IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 93–10 CREATE_EVENT_SCHEDULE Parameters

Parameter	Description
schedule_name	This.attribute specifies a unique identifier for the schedule. The name has to be unique in the SQL namespace. For example, a schedule cannot have the same name as a table in a schema. If no name is specified, then an error occurs.
start_date	This attribute specifies the date on which this schedule becomes valid. Occurrences of the event before this date are ignored in the context of this schedule.
event_condition	This is a conditional expression based on the columns of the event source queue table. The expression must have the syntax of an Advanced Queuing rule. Accordingly, you can include user data properties in the expression provided that the message payload is an object type, and that you prefix object attributes in the expression with tab.user_data. For more information on rules, see the DBMS_AQADM.ADD_SUBSCRIBER procedure.
queue_spec	This argument specifies the queue into which events that start this particular job will be enqueued (the **source queue**). If the source queue is a secure queue, the queue_spec argument is a string containing a pair of values of the form *queue_name*, *agent name*. For non-secure queues, only the queue name need be provided. If a fully qualified queue name is not provided, the queue is assumed to be in the job owner's schema. In the case of secure queues, the agent name provided should belong to a valid agent that is currently subscribed to the queue.
end_date	The date after which jobs will not run and windows will not open. An event schedule that has no end_date is valid forever. end_date has to be after the start_date. If this is not the case, then an error will be generated when the schedule is created.
comments	This attribute specifies an optional comment about the schedule. By default, this attribute is NULL.

Usage Notes

This procedure requires the CREATE JOB privilege to create a schedule in one's own schema or the CREATE ANY JOB privilege to create a schedule in someone else's schema by specifying schema.schedule_name. Once a schedule has been created, it

can be used by other users. The schedule is created with access to `PUBLIC`. Therefore, there is no need to explicitly grant access to the schedule.

CREATE_JOB Procedures

This procedure creates a job.

The procedure is overloaded. The different functionality of each form of syntax is presented along with the syntax declaration.

Syntax

Creates a job in a single call without using an existing program or schedule:

```
DBMS_SCHEDULER.CREATE_JOB (
   job_name             IN VARCHAR2,
   job_type             IN VARCHAR2,
   job_action           IN VARCHAR2,
   number_of_arguments  IN PLS_INTEGER              DEFAULT 0,
   start_date           IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
   repeat_interval      IN VARCHAR2                 DEFAULT NULL,
   end_date             IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
   job_class            IN VARCHAR2                 DEFAULT 'DEFAULT_JOB_CLASS',
   enabled              IN BOOLEAN                  DEFAULT FALSE,
   auto_drop            IN BOOLEAN                  DEFAULT TRUE,
   comments             IN VARCHAR2                 DEFAULT NULL);
```

Creates a job using a named schedule object and a named program object:

```
DBMS_SCHEDULER.CREATE_JOB (
   job_name               IN VARCHAR2,
   program_name           IN VARCHAR2,
   schedule_name          IN VARCHAR2,
   job_class              IN VARCHAR2             DEFAULT 'DEFAULT_JOB_CLASS',
   enabled                IN BOOLEAN              DEFAULT FALSE,
   auto_drop              IN BOOLEAN              DEFAULT TRUE,
   comments               IN VARCHAR2             DEFAULT NULL);
```

Creates a job using a named program object and an inlined schedule:

```
DBMS_SCHEDULER.CREATE_JOB (
   job_name             IN VARCHAR2,
   program_name         IN VARCHAR2,
   start_date           IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
   repeat_interval      IN VARCHAR2                 DEFAULT NULL,
   end_date             IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
   job_class            IN VARCHAR2                 DEFAULT 'DEFAULT_JOB_CLASS',
   enabled              IN BOOLEAN                  DEFAULT FALSE,
   auto_drop            IN BOOLEAN                  DEFAULT TRUE,
   comments             IN VARCHAR2                 DEFAULT NULL);
```

Creates a job using a named schedule object and an inlined program:

```
DBMS_SCHEDULER.CREATE_JOB (
   job_name               IN VARCHAR2,
   schedule_name          IN VARCHAR2,
   job_type               IN VARCHAR2,
   job_action             IN VARCHAR2,
   number_of_arguments    IN PLS_INTEGER       DEFAULT 0,
   job_class              IN VARCHAR2          DEFAULT 'DEFAULT_JOB_CLASS',
   enabled                IN BOOLEAN           DEFAULT FALSE,
   auto_drop              IN BOOLEAN           DEFAULT TRUE,
   comments               IN VARCHAR2          DEFAULT NULL);
```

Creates a job using an inlined program and an event:

```
DBMS_SCHEDULER.CREATE_JOB (
   job_name                IN VARCHAR2,
   job_type                IN VARCHAR2,
   job_action              IN VARCHAR2,
   number_of_arguments     IN PLS_INTEGER       DEFAULT 0,
   start_date              IN TIMESTAMP WITH TIME ZONE,
   event_condition         IN VARCHAR2,
   event_queue             IN VARCHAR2,
   end_date                IN TIMESTAMP WITH TIME ZONE,
   job_class               IN VARCHAR2          DEFAULT 'DEFAULT_JOB_CLASS',
   enabled                 IN BOOLEAN           DEFAULT FALSE,
   auto_drop               IN BOOLEAN           DEFAULT TRUE,
   comments                IN VARCHAR2          DEFAULT NULL);
```

Creates a job using a named program object and an event:

```
DBMS_SCHEDULER.CREATE_JOB (
   job_name                IN VARCHAR2,
   program_name            IN VARCHAR2,
   start_date              IN TIMESTAMP WITH TIME ZONE,
   event_condition         IN VARCHAR2,
   queue_spec              IN VARCHAR2,
   end_date                IN TIMESTAMP WITH TIME ZONE,
   job_class               IN VARCHAR2          DEFAULT 'DEFAULT_JOB_CLASS',
   enabled                 IN BOOLEAN           DEFAULT FALSE,
   auto_drop               IN BOOLEAN           DEFAULT TRUE,
   comments                IN VARCHAR2          DEFAULT NULL);
```

Parameters

Table 93–11 CREATE_JOB Procedure Parameters

Parameter	Description
`job_name`	This attribute specifies the name of the job and uniquely identifies the job. The name has to be unique in the SQL namespace. For example, a job cannot have the same name as a table in a schema. If the job being created will reside in another schema, it must be qualified with the schema name. If `job_name` is not specified, an error is generated. If you want to have a name generated by the Scheduler, you can use the `GENERATE_JOB_NAME` procedure to generate a name and then use the output in the `CREATE_JOB` procedure. The `GENERATE_JOB_NAME` procedure call generates a number from a sequence, which is the job name. You can prefix the number with a string. The job name will then be the string with the number from the sequence appended to it. See "GENERATE_JOB_NAME Function" on page 93-66 for more information.

Table 93–11 (Cont.) CREATE_JOB Procedure Parameters

Parameter	Description
job_type	This attribute specifies the type of job that you are creating. If it is not specified, an error is generated. The supported values are: ■ 'PLSQL_BLOCK' This specifies that the job is an anonymous PL/SQL block. Job or program arguments are not supported when the job or program type is PLSQL_BLOCK. In this case, the number of arguments must be 0. ■ 'STORED_PROCEDURE' This specifies that the job is a PL/SQL or Java stored procedure, or an external C subprogram. Only procedures, not functions with return values, are supported. ■ 'EXECUTABLE' This specifies that the job is a job external to the database. External jobs are anything that can be executed from the operating system's command line. Anydata arguments are not supported with a job or program type of EXECUTABLE. The job owner must have the CREATE EXTERNAL JOB system privilege before the job can be enabled or run. ■ 'CHAIN' This specifies that the job is a chain. Arguments are not supported for a chain, so number_of_arguments must be 0.
job_action	This attribute specifies the action of the job. The following actions are possible: For a PL/SQL block, the action is to execute PL/SQL code. These blocks must end with a semi-colon. For example, my_proc(); or BEGIN my_proc(); END; or DECLARE arg pls_integer := 10; BEGIN my_proc2(arg); END;. Note that the Scheduler wraps job_action in its own block and passes the following to PL/SQL for execution: DECLARE ... BEGIN job_action END; This is done to declare some internal Scheduler variables. You can include any Scheduler metadata attribute except event_message in your PL/SQL code. You use the attribute name as you use any other PL/SQL identifier, and the Scheduler assigns it a value. See Table 93–22 on page 93-46 for details on available metadata attributes. For a stored procedure, the action is the name of the stored procedure. You have to specify the schema if the procedure resides in another schema than the job. PL/SQL procedures with INOUT or OUT arguments are not supported as job_action when the job or program type is STORED_PROCEDURE. For an executable, the action is the name of the external executable, including the full path name and any command-line arguments. For a chain, the action is the name of a Scheduler chain object. You have to specify the schema of the chain if it resides in a different schema than the job. If job_action is not specified for an inline program, an error is generated when creating the job.

Table 93–11 (Cont.) CREATE_JOB Procedure Parameters

Parameter	Description
number_of_arguments	This attribute specifies the number of arguments that the job expects. The range is 0-255, with the default being 0.
program_name	The name of the program associated with this job. If the program is of type EXECUTABLE, the job owner must have the CREATE EXTERNAL JOB system privilege before the job can be enabled or run.
start_date	This attribute specifies the first date on which this job is scheduled to start. If start_date and repeat_interval are left null, then the job is scheduled to run as soon as the job is enabled. For repeating jobs that use a calendaring expression to specify the repeat interval, start_date is used as a reference date. The first time the job will be scheduled to run is the first match of the calendaring expression that is on or after the current date. The Scheduler cannot guarantee that a job will execute on an exact time because the system may be overloaded and thus resources unavailable.
event_condition	This is a conditional expression based on the columns of the event source queue table. The expression must have the syntax of an Advanced Queuing rule. Accordingly, you can include user data properties in the expression provided that the message payload is an object type, and that you prefix object attributes in the expression with tab.user_data. For more information on rules, see the DBMS_AQADM.ADD_SUBSCRIBER procedure.
queue_spec	This argument specifies the queue into which events that start this particular job will be enqueued (the **source queue**). If the source queue is a secure queue, the queue_spec argument is a string containing a pair of values of the form *queue_name*, *agent name*. For non-secure queues, only the queue name need be provided. If a fully qualified queue name is not provided, the queue is assumed to be in the job owner's schema. In the case of secure queues, the agent name provided should belong to a valid agent that is currently subscribed to the queue.
repeat_interval	This attribute specifies how often the job should repeat. You can specify the repeat interval by using calendaring or PL/SQL expressions. The expression specified is evaluated to determine the next time the job should run. If repeat_interval is not specified, the job will run only once at the specified start date. See "Calendaring Syntax" on page 93-4 for further information.
schedule_name	The name of the schedule, window, or window group associated with this job.
end_date	This attribute specifies the date after which the job will expire and will no longer be executed. When end_date is reached, the job is disabled. The STATE of the job will be set to COMPLETED, and the enabled flag will be set to FALSE. If no value for end_date is specified, the job will repeat forever unless max_runs or max_failures is set, in which case the job stops when either value is reached. The value for end_date must be after the value for start_date. If it is not, an error is generated when the job is enabled.
job_priority	This attribute designates the priority of a job relative to other jobs in the same job class only. If two jobs in the same class are scheduled to start at the same time, the one with the higher priority takes precedence. Acceptable values are 1 through 5, where 1 is the highest priority. Default value is 3.
comments	This attribute specifies a comment about the job. By default, this attribute is NULL.

Table 93–11 (Cont.) CREATE_JOB Procedure Parameters

Parameter	Description
enabled	This attribute specifies whether the job is created enabled or not. The possible settings are TRUE or FALSE. By default, this attribute is set to FALSE and, therefore, the job is created as disabled. A disabled job means that the metadata about the job has been captured and the job exists as a database object but the Scheduler will ignore it and the job coordinator will not pick the job for processing. In order for the job coordinator to process the job, the job has to be enabled. You can enable a job by setting this argument to TRUE or by using the ENABLE procedure.
auto_drop	This flag, if TRUE, causes a job to be automatically dropped after it has completed or has been disabled. A job is considered completed if: ▪ Its end date (or its schedule's end date) has passed ▪ It has run max_runs number of times. max_runs must be set with SET_ATTRIBUTE. ▪ It is not a repeating job and has run once A job is disabled when it has failed max_failures times. max_failures is also set with SET_ATTRIBUTE. If this flag is set to FALSE, the jobs are not dropped and their metadata is kept until the job is explicitly dropped with the DROP_JOB procedure. By default, jobs are created with auto_drop set to TRUE.

Usage Notes

Jobs are created disabled by default. You must explicitly enable them so that they will become active and scheduled. Before enabling a job, ensure that all program arguments, if any, are defined, either by defining default values in the program object or by supplying values with the job.

To create a job in your own schema, you need to have the CREATE JOB privilege. A user with the CREATE ANY JOB privilege can create a job in any schema. If the job being created will reside in another schema, the job name must be qualified with the schema name. For a job of type EXECUTABLE (or for a job that points to a program of type EXECUTABLE), the job owner must have the CREATE EXTERNAL JOB system privilege before the job can be enabled or run.

Associating a job with a particular class or program requires EXECUTE privileges for that class or program.

Not all possible job attributes can be set with CREATE_JOB. Some must be set after the job is created. For example, job arguments must be set with the SET_JOB_ARGUMENT_VALUE Procedures or the SET_JOB_ANYDATA_VALUE Procedures. Other job attributes, such as job_priority and max_runs, are set with the SET_ATTRIBUTE Procedure.

Note: The Scheduler runs event-based jobs for each occurrence of an event that matches the job's event condition. However, events that occur while the job is already running are ignored; the event gets consumed, but does not trigger another run of the job.

CREATE_JOB_CLASS Procedure

This procedure creates a job class. Job classes are created in the SYS schema.

Syntax

```
DBMS_SCHEDULER.CREATE_JOB_CLASS (
   job_class_name            IN VARCHAR2,
   resource_consumer_group   IN VARCHAR2 DEFAULT NULL,
   service                   IN VARCHAR2 DEFAULT NULL,
   logging_level             IN PLS_INTEGER
                                DEFAULT DBMS_SCHEDULER.LOGGING_RUNS,
   log_history               IN PLS_INTEGER DEFAULT NULL,
   comments                  IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 93–12 CREATE_JOB_CLASS Procedure Parameters

Parameter	Description
job_class_name	The name of the class being created. A schema other than SYS cannot be specified. This attribute specifies the name of the job class and uniquely identifies the job class. The name has to be unique in the SQL namespace. For example, a job class cannot have the same name as a table in a schema.
resource_consumer_group	This attribute specifies the resource consumer group this class is associated with. A resource consumer group is a set of synchronous or asynchronous sessions that are grouped together based on their processing needs. A job class has a many-to-one relationship with a resource consumer group. The resource consumer group that the job class associates with will determine the resources that will be allocated to the job class. If the resource consumer group that a job class is associated with is dropped, the job class will then be associated with the default resource consumer group. If no resource consumer group is specified, the job class is associated with the default resource consumer group. If the specified resource consumer group does not exist when creating the job class, an error occurs. If resource_consumer_group is specified, you cannot specify a service (it must be NULL). Also, if a service is specified, resource_consumer_group must be NULL.

Table 93–12 (Cont.) CREATE_JOB_CLASS Procedure Parameters

Parameter	Description
service	This attribute specifies the database service that the jobs in this class will have affinity to. In a RAC environment, this means that the jobs in this class will only run on those database instances that are assigned to the specific service. If a service is specified, resource_consumer_group must be NULL. Note that a service can be mapped to a resource consumer group, so you can also control resources allocated to jobs by specifying a service. See DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING for details. If no service is specified, the job class will belong to the default service, which means it will have no service affinity and any one of the database instances within the cluster might run the job. If the service that a job class belongs to is dropped, the job class will then belong to the default service. If the specified service does not exist when creating the job class, then an error occurs.
logging_ level	This attribute specifies how much information is logged. The three possible options are: ■ DBMS_SCHEDULER.LOGGING_OFF No logging will be performed for any jobs in this class. ■ DBMS_SCHEDULER.LOGGING_RUNS The Scheduler will write detailed information to the job log for all runs of each job in this class. This is the default. ■ DBMS_SCHEDULER.LOGGING_FULL In addition to recording every run of a job, the Scheduler will record all operations performed on all jobs in this class. In other words, every time a job is created, enabled, disabled, altered, and so on will be recorded in the log.
log_history	This enables you to control the amount of logging the Scheduler performs. To prevent the job log and the window log from growing indiscriminately, the Scheduler has an attribute that specifies how much history (in days) to keep. Once a day, the Scheduler will automatically purge all log entries from both the job log as well as the window log that are older than the specified history. The default is 30 days. You can change the default by using the SET_SCHEDULER_ATTRIBUTE procedure. For example, to change it to 90 days, issue the following statement: DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE ('log_history','90'); The range of valid values is 1 through 999.
comments	This attribute is for an optional comment about the job class. By default, this attribute is NULL.

Usage Notes

For users to create jobs that belong to a job class, the job owner must have EXECUTE privileges on the job class. Therefore, after the job class has been created, EXECUTE privileges must be granted on the job class so that users create jobs belonging to that class. You can also grant the EXECUTE privilege to a role.

Creating a job class requires the MANAGE SCHEDULER system privilege.

CREATE_PROGRAM Procedure

This procedure creates a program.

Syntax

```
DBMS_SCHEDULER.CREATE_PROGRAM (
   program_name             IN VARCHAR2,
   program_type             IN VARCHAR2,
   program_action           IN VARCHAR2,
   number_of_arguments      IN PLS_INTEGER DEFAULT 0,
   enabled                  IN BOOLEAN DEFAULT FALSE,
   comments                 IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 93–13 CREATE_PROGRAM Procedure Parameters

Parameter	Description
program_name	This attribute specifies a unique identifier for the program. The name has to be unique in the SQL namespace. For example, a program cannot have the same name as a table in a schema. If no name is specified, then an error occurs.
program_type	This attribute specifies the type of program you are creating. If it is not specified then you will get an error. There are three supported values for program_type: ■ 'PLSQL_BLOCK' This specifies that the program is a PL/SQL block. Job or program arguments are not supported when the job or program type is PLSQL_BLOCK. In this case, the number of arguments must be 0. ■ 'STORED_PROCEDURE' This specifies that the program is a PL/SQL or Java stored procedure, or an external C subprogram. Only procedures, not functions with return values, are supported. PL/SQL procedures with INOUT or OUT arguments are not supported. ■ 'EXECUTABLE' This specifies that the program is external to the database. External programs implies anything that can be executed from the operating system's command line. AnyData arguments are not supported with job or program type EXECUTABLE.

Table 93–13 (Cont.) CREATE_PROGRAM Procedure Parameters

Parameter	Description
program_action	This attribute specifies the action of the program. The following actions are possible: For a PL/SQL block, the action is to execute PL/SQL code. These blocks must end with a semi-colon. For example, `my_proc();` or `BEGIN my_proc(); END;` or `DECLARE arg pls_integer := 10; BEGIN my_proc2(arg); END;`. Note that the Scheduler wraps `job_action` in its own block and passes the following to PL/SQL for execution: `DECLARE ... BEGIN job_action END;` This is done to declare some internal Scheduler variables. You can include any Scheduler metadata attribute except `event_message` in your PL/SQL code. You use the attribute name as you use any other PL/SQL identifier, and the Scheduler assigns it a value. See Table 93–22 on page 93-46 for details on available metadata attributes. For a stored procedure, the action is the name of the stored procedure. You have to specify the schema if the procedure resides in another schema than the job. For an executable, the action is the name of the external executable, including the full path name and any command-line arguments. If `program_action` is not specified, an error is generated If it is an anonymous block, special Scheduler metadata may be accessed using the following variable names: `job_name`, `job_owner`, `job_start`, `window_start`, `window_end`. For more information on these, see the information regarding `define_metadata_argument`.
number_of_arguments	This attribute specifies the number of arguments the program takes. If this parameter is not specified then the default will be 0. A program can have a maximum of 255 arguments. If the `program_type` is `PLSQL_BLOCK`, this field is ignored.
enabled	This flag specifies whether the program should be created enabled or not. If the flag is set to `TRUE`, then validity checks will be made and the program will be created `ENABLED` should all the checks be successful. By default, this flag is set to `FALSE`, which means that the program is not created enabled. You can also call the `ENABLE` procedure to enable the program before it can be used.
comments	A comment about the program. By default, this attribute is `NULL`.

Usage Notes

To create a program in his own schema, a user needs the `CREATE JOB` privilege. A user with the `CREATE ANY JOB` privilege can create a program in any schema. A program is created in a disabled state by default (unless the enabled field is set to `TRUE`). It cannot be executed by a job until it is enabled.

For other users to use your programs, they must have `EXECUTE` privileges, therefore once a program has been created, you have to grant `EXECUTE` privileges on it.

CREATE_SCHEDULE Procedure

This procedure creates a schedule.

Syntax

```
DBMS_SCHEDULER.CREATE_SCHEDULE (
   schedule_name          IN VARCHAR2,
   start_date             IN TIMESTAMP WITH TIMEZONE DEFAULT NULL,
   repeat_interval        IN VARCHAR2,
   end_date               IN TIMESTAMP WITH TIMEZONE DEFAULT NULL,
   comments               IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 93–14 CREATE_SCHEDULE Procedure Parameters

Parameter	Description
schedule_name	This attribute specifies a unique identifier for the schedule. The name has to be unique in the SQL namespace. For example, a schedule cannot have the same name as a table in a schema. If no name is specified, then an error occurs.
start_date	This attribute specifies the first date on which this schedule becomes valid. For a repeating schedule, the value for start_date is a reference date. In this case, the start of the schedule is not the start_date. It depends on the repeat interval specified. start_date is used to determine the first instance of the schedule. If start_date is specified in the past and no value for repeat_interval is specified, the schedule is invalid. For a repeating job or window, start_date can be derived from the repeat_interval, if it is not specified.
repeat_interval	This attribute specifies how often the schedule should repeat. It is expressed using calendaring syntax. See "Calendaring Syntax" on page 93-4 for further information. PL/SQL expressions are not allowed as repeat intervals for named schedules.
end_date	The date after which jobs will not run and windows will not open. A non-repeating schedule that has no end_date will be valid forever. end_date has to be after the start_date. If this is not the case, then an error will be generated when the schedule is created.
comments	This attribute specifies an optional comment about the schedule. By default, this attribute is NULL.

Usage Notes

This procedure requires the CREATE JOB privilege to create a schedule in one's own schema or the CREATE ANY JOB privilege to create a schedule in someone else's schema by specifying schema.schedule_name. Once a schedule has been created, it can be used by other users. The schedule is created with access to PUBLIC. Therefore, there is no need to explicitly grant access to the schedule.

CREATE_WINDOW Procedures

This procedure creates a recurring time window and associates it with a resource plan. The window can then be used to schedule jobs, which run under the associated resource plan.

The procedure is overloaded.

Syntax

Creates a window using a named schedule object:

```
DBMS_SCHEDULER.CREATE_WINDOW (
   window_name             IN VARCHAR2,
   resource_plan           IN VARCHAR2,
   schedule_name           IN VARCHAR2,
   duration                IN INTERVAL DAY TO SECOND,
   window_priority         IN VARCHAR2                DEFAULT 'LOW',
   comments                IN VARCHAR2                DEFAULT NULL);
```

Creates a window using an inlined schedule:

```
DBMS_SCHEDULER.CREATE_WINDOW (
   window_name             IN VARCHAR2,
   resource_plan           IN VARCHAR2,
   start_date              IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
   repeat_interval         IN VARCHAR2,
   end_date                IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
   duration                IN INTERVAL DAY TO SECOND,
   window_priority         IN VARCHAR2                DEFAULT 'LOW',
   comments                IN VARCHAR2                DEFAULT NULL);
```

Parameters

Table 93–15 CREATE_WINDOW Procedure Parameters

Parameter	Description
window_name	This attribute uniquely identifies the window. The name has to be unique in the SQL namespace. All windows are in the SYS schema, so you can optionally preface the window name with 'SYS.'
resource_plan	This attribute specifies the resource plan that is automatically activated when the window opens. When the window closes, the system switches to the appropriate resource plan, which in most cases is the resource plan that was in effect before the window opened, but can also be the resource plan of yet another window. Only one resource plan can be associated with a window. It may be NULL or the empty string (""). When it is NULL, the resource plan that is in effect when the window opens stays in effect for the duration of the window. When it is the empty string, the resource manager is disabled for the duration of the window. If the window is open and the resource plan is dropped, then the resource allocation for the duration of the window is not affected.

Table 93–15 (Cont.) CREATE_WINDOW Procedure Parameters

Parameter	Description
start_date	This attribute specifies the first date on which this window is scheduled to open. If the value for start_date specified is in the past or is not specified, the window opens as soon as it is created. For repeating windows that use a calendaring expression to specify the repeat interval, the value for start_date is a reference date. The first time the window opens depends on the repeat interval specified and the value for start_date.
duration	This attribute specifies how long the window will be open for. For example, 'interval '5' hour' for five hours. There is no default value for this attribute. Therefore, if none is specified when creating the window, an error occurs. The duration is of type interval day to seconds and ranges from one minute to 99 days.
schedule_name	The name of the schedule associated with the window.
repeat_interval	This attribute specifies how often the window should repeat. It is expressed using the Scheduler's calendaring syntax. See "Calendaring Syntax" on page 93-4 for more information. A PL/SQL expression cannot be used to specify the repeat interval for a window. The expression specified is evaluated to determine the next time the window should open. If no repeat_interval is specified, the window will open only once at the specified start date.
end_date	This attribute specifies the date after which the window will no longer open. When the value for end_date is reached, the window is disabled. In the *_SCHEDULER_WINDOWS views, the enabled flag of the window will be set to FALSE. A non-repeating window that has no value for end_date opens only once for the duration of the window. For a repeating window, if no end_date is specified then the window will keep repeating forever. The end_date has to be after the start_date. If this is not the case, then an error is generated when the window is created.
window_priority	This attribute is only relevant when two windows overlap. Because only one window can be in effect at one time, the window priority will be used to determine which window will be opened. The two possible values for this attribute are 'HIGH' and 'LOW'. A high priority window has precedence over a low priority window, which implies that the low priority window does not open if it overlaps with a high priority window. By default, a window is created with a priority of 'LOW'.
comments	This attribute specifies an optional comment about the window. By default, this attribute is NULL.

Usage Notes

Creating a window requires the MANAGE SCHEDULER privilege. Windows always reside in the SYS schema.

Scheduler windows are the principal mechanism used to automatically switch resource plans according to a schedule. You can also manually activate a resource plan by using the ALTER SYSTEM SET RESOURCE_MANAGER_PLAN statement or the DBMS_RESOURCE_MANAGER.SWITCH_PLAN package procedure. Note that either of these manual methods can also disable resource plan switching by Scheduler windows. For more information, see *Oracle Database Administrator's Guide* and "SWITCH_PLAN Procedure" on page 85-29.

CREATE_WINDOW_GROUP Procedure

This procedure creates a new window group.

Syntax

```
DBMS_SCHEDULER.CREATE_WINDOW_GROUP (
   group_name            IN VARCHAR2,
   window_list           IN VARCHAR2 DEFAULT NULL,
   comments              IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 93–16 CREATE_WINDOW_GROUP Procedure Parameters

Parameter	Description
group_name	The name of the window group
window_list	A list of the windows assigned to the window group. If a window that does not exist is specified, an error is generated and the window group is not created. Windows can also be added using the ADD_WINDOW_GROUP_MEMBER procedure. A window group cannot be a member of another window group. Can be NULL.
comments	A comment about the window group

Usage Notes

Creating a window group requires the MANAGE SCHEDULER privilege. Window groups reside in the SYS schema. Window groups, like windows, are created with access to PUBLIC, therefore, no privileges are required to access window groups.

A window group cannot contain another window group.

DEFINE_ANYDATA_ARGUMENT Procedure

This procedure defines a name or default value for a program argument that is of a complex type and must be encapsulated within an ANYDATA object. A job that references the program can override the default value.

Syntax

```
DBMS_SCHEDULER.DEFINE_ANYDATA_ARGUMENT (
   program_name            IN VARCHAR2,
   argument_position       IN PLS_INTEGER,
   argument_name           IN VARCHAR2 DEFAULT NULL,
   argument_type           IN VARCHAR2,
   default_value           IN SYS.ANYDATA,
   out_argument            IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 93–17 DEFINE_ANYDATA_ARGUMENT Procedure Parameters

Parameter	Description
program_name	The name of the program to be altered. A program with this name must exist.
argument_position	The position of the argument as it is passed to the executable. Argument numbers go from one to the number_of_arguments specified for the program. This must be unique, so it will replace any argument already defined at this position.
argument_name	The name to assign to the argument. It is optional, but must be unique for the program if it is specified. If you assign a name, the name can then be used by other package procedures, including the SET_JOB_ANYDATA_VALUE Procedures.
argument_type	The data type of the argument being defined. This is not verified or used by the Scheduler. It is only used by the user of the program when deciding what value to assign to the argument.
default_value	The default value to be assigned to the argument encapsulated within an AnyData object. This is optional.
out_argument	This parameter is reserved for future use. It must be set to FALSE.

Usage Notes

All program arguments from 1 to the number_of_arguments value must be defined before a program can be enabled. If a default value for an argument is not defined with this procedure, a value must be defined in the job.

Defining a program argument requires that you be the owner of the program or have ALTER privileges on that program. You can also define a program argument if you have the CREATE ANY JOB privilege.

See Also:

- "DEFINE_PROGRAM_ARGUMENT Procedures" on page 93-48
- "SET_JOB_ANYDATA_VALUE Procedures" on page 93-88

DEFINE_CHAIN_EVENT_STEP Procedure

This procedure adds or replaces a chain step and associates it with an event schedule or an inline event. Once started in a running chain, this step will not complete until the specified event has occurred. Every step in a chain must be defined before the chain can be enabled and used. Defining a step gives it a name and specifies what happens during the step. If a step already exists with this name, the new step will replace the old one.

Syntax

```
DBMS_SCHEDULER.DEFINE_CHAIN_EVENT_STEP (
   chain_name              IN VARCHAR2,
   step_name               IN VARCHAR2,
   event_schedule_name     IN VARCHAR2,
   timeout                 IN INTERVAL DAY TO SECOND DEFAULT NULL);

DBMS_SCHEDULER.DEFINE_CHAIN_EVENT_STEP (
   chain_name              IN VARCHAR2,
   step_name               IN VARCHAR2,
   event_condition         IN VARCHAR2,
   queue_spec              IN VARCHAR2,
   timeout                 IN INTERVAL DAY TO SECOND DEFAULT NULL);
```

Parameters

Table 93–18 DEFINE_CHAIN_EVENT_STEP Procedure Parameters

Parameter	Description
chain_name	The name of the chain that the step is in
step_name	The name of the step
event_schedule_name	The name of the event schedule that the step waits for
timeout	This parameter is reserved for future use
event_condition	See the CREATE_EVENT_SCHEDULE Procedure
queue_spec	See the CREATE_EVENT_SCHEDULE Procedure

Usage Notes

Defining a chain step requires ALTER privileges on the chain either by being the owner of the chain, or by having the ALTER object privilege on the chain or by having the CREATE ANY JOB system privilege.

See Also: "DEFINE_CHAIN_STEP Procedure" on page 93-45

DEFINE_CHAIN_RULE Procedure

This procedure adds a new rule to an existing chain, specified as a condition-action pair. The condition is expressed using either SQL or the Scheduler chain condition syntax, and indicates the prerequisites for the action to occur. The action specifies what is to be done as a result of the condition being met.

An actual rule object is created to store the rule in the schema in which the chain resides. If a rule name is given, this name will be used for the rule object. If a rule name is given and a rule already exists with this name in the chain's schema, the existing rule will be altered. (A schema different than the chain's schema cannot be specified). If no rule name is given, one will be generated of the form `SCHED_RULE${N}`.

Syntax

```
DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
   chain_name              IN VARCHAR2,
   condition               IN VARCHAR2,
   action                  IN VARCHAR2,
   rule_name               IN VARCHAR2 DEFAULT NULL,
   comments                IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 93–19 DEFINE_CHAIN_RULE Procedure Parameters

Parameter	Description
`chain_name`	The name of the chain to alter
`condition`	A boolean expression expressed using either SQL or the Scheduler chain condition syntax. (Scheduler chain condition syntax is described below.) The expression must evaluate to `TRUE` for the action to be performed. If the condition is expressed with SQL, it must use the syntax of a `SELECT` statement `WHERE` clause. You can refer to chain step attributes by using the chain step name as a bind variable. The bind variable syntax is `:`*`step_name.attribute`*. (*`step_name`* refers to a typed object.) Possible attributes are: `completed`, `state`, `start_date`, `end_date`, `error_code`, and `duration`. Possible values for the `state` attribute include: `'NOT_STARTED'`, `'SCHEDULED'`, `'RUNNING'`, `'PAUSED'`, `'STALLED'`, `'SUCCEEDED'`, `'FAILED'`, and `'STOPPED'`. If a step is in the state `'SUCCEEDED'`, `'FAILED'`, or `'STOPPED'`, its `completed` attribute is set to `'TRUE'`, otherwise `completed` is `'FALSE'`. Every chain must have a rule that evaluates to `TRUE` to start the chain. For this purpose, you can use a rule that has `'TRUE'` as its condition if you are using Schedule chain condition syntax, or `'1=1'` as its condition if you are using SQL syntax.

Table 93–19 (Cont.) DEFINE_CHAIN_RULE Procedure Parameters

Parameter	Description
action	The action to be performed when the rule evaluates to TRUE. The action must consist of at least a keyword with an optional value and an optional delay clause. Possible actions include: ■ [AFTER *delay_interval*] START *step_1*[,*step_2* ...] ■ STOP *step_1*[,*step_2* ...] ■ END [{*end_value*|*step_name.error_code*}] At the beginning of the START action, a delay clause can be given which specifies a delay interval to wait before performing the action. *delay_interval* is a formatted datetime interval of the form HH:MM:SS.
rule_name	The name of the rule that will be created. If no rule_name is given, one will be generated of the form SCHED_RULE$_{N}.
comments	An optional comment describing the rule. This will be stored in the rule object created.

Chain Condition Syntax

The Scheduler chain condition syntax provides an easy way to construct a condition using the states and error codes of steps in the current chain. The following are the available constructs, all of which are boolean expressions:

```
TRUE
FALSE
stepname [NOT] SUCCEEDED
stepname [NOT] FAILED
stepname [NOT] STOPPED
stepname [NOT] COMPLETED
stepname ERROR_CODE IN (integer, integer, integer ...)
stepname ERROR_CODE NOT IN (integer, integer, integer ...)
stepname ERROR_CODE = integer
stepname ERROR_CODE != integer
stepname ERROR_CODE <> integer
stepname ERROR_CODE > integer
stepname ERROR_CODE >= integer
stepname ERROR_CODE < integer
stepname ERROR_CODE <= integer
```

The following boolean operators are available to create more complex conditions:

```
expression AND expression
expression OR expression
NOT (expression)
```

integer can be positive or negative. Parentheses may be used for clarity or to enforce ordering. You must use parentheses with the NOT operator.

Usage Notes

Defining a chain rule requires ALTER privileges on the chain (either by being the owner, or by having ALTER privileges on the chain or by having the CREATE ANY JOB system privilege).

You must define at least one rule that starts the chain and at least one that ends it. See the section "Adding Rules to a Chain" in *Oracle Database Administrator's Guide* for more information.

Examples

The following are examples of using rule conditions and rule actions.

Rule Conditions Using Scheduler Chain Condition Syntax

```
'step1 completed'
-- satisfied when step step1 has completed. (step1 completed is also TRUE when any
-- of the following are TRUE: step1 succeeded, step1 failed, step1 stopped.)

'step1 succeeded and step2 succeeded'
-- satisfied when steps step1 and step2 have both succeeded

'step1 error_code > 100'
-- satisfied when step step1 has failed with an error_code greater than 100

'step1 error_code IN (1, 3, 5, 7)'
-- satisfied when step step1 has failed with an error_code of 1, 3, 5, or 7
```

Rule Conditions Using SQL Syntax

```
':step1.completed = ''TRUE'' AND :step1.end_date >SYSDATE-1/24'
--satisfied when step step1 completed less than an hour ago

':step1.duration > interval ''5'' minute'
-- satisfied when step step1 has completed and took longer than 5 minutes to
complete
```

Rule Actions

```
'AFTER 01:00:00 START step1, step2'
--After an hour start steps step1 and step2

'STOP step1'
--Stop step step1

END step4.error_code'
--End the chain with the error code that step step4 finished with. If step4 has
not completed, the chain will be ended unsuccessfully with error code 27435.

'END' or 'END 0'
--End the chain successfully (with error_code 0)

'END 100'
--End the chain unsuccessfully with error code 100.
```

DEFINE_CHAIN_STEP Procedure

This procedure adds or replaces a chain step and associates it with a program or a nested chain. When the chain step is started, the specified program or chain is run. If a step already exists with the name supplied in the `chain_name` argument, the new step replaces the old one.

The chain owner must have `EXECUTE` privileges on the program or chain associated with the step. Only one program or chain can run during a step.

Syntax

```
DBMS_SCHEDULER.DEFINE_CHAIN_STEP (
   chain_name              IN VARCHAR2,
   step_name               IN VARCHAR2,
   program_name            IN VARCHAR2);
```

Parameters

Table 93–20 DEFINE_CHAIN_STEP Procedure Parameters

Parameter	Description
`chain_name`	The name of the chain to alter.
`step_name`	The name of the step being defined. If a step already exists with this name, the new step will replace the old one.
`program_name`	The name of a program or chain to run during this step. The chain owner must have `EXECUTE` privileges on this program or chain.

Usage Notes

Defining a chain step requires `ALTER` privileges on the chain (either by being the owner, or by having `ALTER` privileges on the chain or by having the `CREATE ANY JOB` system privilege).

> **See Also:** "DEFINE_CHAIN_EVENT_STEP Procedure" on page 93-41

DEFINE_METADATA_ARGUMENT Procedure

This procedure defines a special metadata argument for the program. The Scheduler can pass Scheduler metadata through this argument to your stored procedure or other executable. You cannot set values for jobs using this argument.

Syntax

```
DBMS_SCHEDULER.DEFINE_METADATA_ARGUMENT (
  program_name            IN VARCHAR2,
  metadata_attribute      IN VARCHAR2,
  argument_position       IN PLS_INTEGER,
  argument_name           IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 93–21 DEFINE_METADATA_ARGUMENT Procedure Parameters

Parameter	Description
program_name	The name of the program to be altered
metadata_attribute	The metadata to be passed. Valid metadata attributes are: 'job_name', 'job_subname', 'job_owner', 'job_start', 'window_start', 'window_end', and 'event_message'. Table Table 93–22 describes these attributes in detail.
argument_position	The position of the argument as it is passed to the executable. This cannot be greater than the number_of_arguments specified for the program. This must be unique, so it will replace any argument already defined at this position.
argument_name	The name to assign to the argument. It is optional, but must be unique for the program if it is specified. If you assign a name, the name can then be used by other package procedures.

Table 93–22 Metadata Attributes

Metadata Attribute	Data Type	Description
job_name	VARCHAR2	Name of the currently running job
job_subname	VARCHAR2	Subname of the currently running job. The name + subname form a unique identifier for a job that is running a chain step. NULL if the job is not part of a chain.
job_owner	VARCHAR2	Owner of the currently running job
job_start	TIMESTAMP WITH TIME ZONE	When the currently running job started
window_start	TIMESTAMP WITH TIME ZONE	If the job was started by a window, the time that the window opened
window_end	TIMESTAMP WITH TIME ZONE	If the job was started by a window, the time that the window is scheduled to close
event_message	(See Description)	For an event-based job, the message content of the event that started the job. The data type of this attribute depends on the queue used for the event. It has the same type as the USER_DATA column of the queue table.

Usage Notes

Defining a program argument requires that you be the owner of the program or have `ALTER` privileges on that program. You can also define a program argument if you have the `CREATE ANY JOB` privilege.

All metadata attributes except `event_message` can also be used in PL/SQL blocks that you enter into the `job_action` or `program_action` attributes of jobs or programs, respectively. You use the attribute name as you use any other PL/SQL identifier, and the Scheduler assigns it a value.

DEFINE_PROGRAM_ARGUMENT Procedures

This procedure defines a name or default value for a program argument. If no default value is defined for a program argument, the job that references the program must supply an argument value. (The job can also override a default value.)

This procedure is overloaded.

Syntax

Defines a program argument without a default value:

```
PROCEDURE define_program_argument(
   program_name            IN VARCHAR2,
   argument_position       IN PLS_INTEGER,
   argument_name           IN VARCHAR2 DEFAULT NULL,
   argument_type           IN VARCHAR2,
   out_argument            IN BOOLEAN DEFAULT FALSE);
```

Defines a program argument with a default value:

```
PROCEDURE define_program_argument(
   program_name            IN VARCHAR2,
   argument_position       IN PLS_INTEGER,
   argument_name           IN VARCHAR2 DEFAULT NULL,
   argument_type           IN VARCHAR2,
   default_value           IN VARCHAR2,
   out_argument            IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 93–23 DEFINE_PROGRAM_ARGUMENT Procedure Parameters

Parameter	Description
`program_name`	The name of the program to be altered. A program with this name must exist.
`argument_position`	The position of the argument as it is passed to the executable. Argument numbers go from one to the `number_of_arguments` specified for the program. This must be unique so it will replace any argument already defined at this position.
`argument_name`	The name to assign to the argument. It is optional, but must be unique for the program if it is specified. If you assign a name, the name can then be used by other package procedures, including the SET_JOB_ARGUMENT_VALUE Procedures.
`argument_type`	The data type of the argument being defined. This is not verified or used by the Scheduler. It is only used by the user of the program when deciding what value to assign to the argument.
`default_value`	The default value to be assigned to the argument if none is specified by the job.
`out_argument`	This parameter is reserved for future use. It must be set to `FALSE`.

Usage Notes

All program arguments from 1 to the `number_of_arguments` value must be defined before a program can be enabled. If a default value for an argument is not defined with this procedure, a value must be defined in the job.

Defining a program argument requires that you be the owner of the program or have ALTER privileges on that program. You can also define a program argument if you have the CREATE ANY JOB privilege.

See Also:

- "DEFINE_ANYDATA_ARGUMENT Procedure" on page 93-40
- "SET_JOB_ARGUMENT_VALUE Procedures" on page 93-89

DISABLE Procedure

This procedure disables a program, job, chain, window, or window group.

Syntax

```
DBMS_SCHEDULER.DISABLE (
   name              IN VARCHAR2,
   force             IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 93–24 DISABLE Procedure Parameters

Parameter	Description
name	The name of the object being disabled. Can be a comma-delimited list. If a job class name is specified, then all the jobs in the job class are disabled. The job class is not disabled. If a window group name is specified, then the window group will be disabled, but the windows that are members of the window group will not be disabled.
force	Whether to ignore dependencies. See the usage notes for more information.

Usage Notes

Disabling an object that is already disabled does not generate an error. Because the DISABLE procedure is used for several Scheduler objects, when disabling windows and window groups, they must be preceded by SYS.

The purpose of the force option is to point out dependencies. No dependent objects are altered.

To run DISABLE for a window or window group, you must have the MANAGE SCHEDULER privilege. Otherwise, you must be the owner of the object being disabled or have ALTER privileges on that object or have the CREATE ANY JOB privilege.

Jobs

Disabling a job means that, although the metadata of the job is there, it should not run and the job coordinator will not pick up these jobs for processing. When a job is disabled, its state in the job queue is changed to disabled.

If force is set to FALSE and the job is currently running, an error is returned.

If force is set to TRUE, the job is disabled, but the currently running instance is allowed to finish.

Programs

When a program is disabled, the status is changed to disabled. A disabled program implies that, although the metadata is still there, jobs that point to this program cannot run.

If force is set to FALSE, the program must be unreferenced by any job otherwise an error will occur.

If `force` is set to `TRUE`, those jobs that point to the program will not be disabled, however, they will fail at runtime because their program will not be valid.

Running jobs that point to the program are not affected by the `DISABLE` call, and are allowed to continue

Any argument that pertains to the program will not be affected when the program is disabled.

Windows

This means that the window will not open, however, the metadata of the window is still there, so it can be reenabled.

If `force` is set to `FALSE`, the window must not be open or referenced by any job otherwise an error will occur.

If `force` is set to `TRUE`, disabling a window that is open will succeed but the window will not be closed. It will prevent the window from opening in the future until it is re-enabled.

When the window is disabled, those jobs that have the window as their schedule will not be disabled.

Window Groups

When a window group is disabled, jobs, other than a running job, that has the window group as its schedule will not run even if the member windows open. However, if the job had one of the window group members as its schedule, it would still run.

The metadata of the window group is still there, so it can be reenabled. Note that the members of the window group will still open.

If `force` is set to `FALSE`, the window group must not have any members that are open or referenced by any job otherwise an error will occur.

If `force` is set to `TRUE`:

- The window group is disabled and the open window will be not closed or disabled. It will be allowed to continue to its end.
- The window group is disabled but those jobs that have the window group as their schedule will not be disabled.

Job Chains

When a chain is disabled, the metadata for the chain is still there, but jobs that point to it will not be able to be run. This allows changes to the chain to be made safely without the risk of having an incompletely specified chain run.

If `force` is set to `FALSE`, the chain must be unreferenced by any job, otherwise an error will occur.

If `force` is set to `TRUE`, those jobs that point to the chain will not be disabled, however, they will fail at runtime.

Running jobs that point to this chain are not affected by the `DISABLE` call and are allowed to complete.

DROP_CHAIN Procedure

This procedure drops an existing chain.

Syntax

```
DBMS_SCHEDULER.DROP_CHAIN (
   chain_name              IN VARCHAR2,
   force                   IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 93–25 DROP_CHAIN Procedure Parameters

Parameter	Description
chain_name	The name of a chain to drop. Can also be a comma-delimited list of chains.
force	If force is set to FALSE, the chain must be unreferenced by any job otherwise an error will occur. If force is set to TRUE, all jobs pointing to the chain are disabled before dropping the chain. Running jobs that point to this chain will be stopped before the chain is dropped.

Usage Notes

Dropping a chain requires alter privileges on the chain (either by being the owner, or by having ALTER privileges on the chain or by having the CREATE ANY JOB system privilege).

All steps associated with the chain are dropped. If no rule set was specified when the chain was created, then the automatically created rule set and evaluation context associated with the chain are also dropped, so the user needs to have the privileges required to do this. See the DBMS_RULE_ADM.DROP_RULE_SET and DBMS_RULE_ADM.DROP_EVALUATION_CONTEXT procedures for more information.

If force is FALSE, no jobs must be using this chain. If force is TRUE, any jobs that use this chain will be disabled before dropping the chain (and any of these jobs that are running will be stopped).

DROP_CHAIN_RULE Procedure

This procedure removes a rule from an existing chain. The rule object corresponding to this rule will also be dropped. The chain will not be disabled. If dropping this rule makes the chain invalid, the user should first disable the chain to ensure that it does not run.

Syntax

```
DBMS_SCHEDULER.DROP_CHAIN_RULE (
   chain_name              IN VARCHAR2,
   rule_name               IN VARCHAR2,
   force                   IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 93–26 DROP_CHAIN_RULE Procedure Parameters

Parameter	Description
chain_name	The name of the chain to alter
rule_name	The name of the rule to drop
force	If force is set to TRUE, the drop operation proceeds even if the chain is currently running. The running chain is not stopped or interrupted. If force is set to FALSE and the chain is running, an error is generated.

Usage Notes

Dropping a chain rule requires alter privileges on the chain (either by being the owner, or by having ALTER privileges on the chain or by having the CREATE ANY JOB system privilege).

Dropping a chain rule also drops the underlying rule database object so the user needs to have the privileges to drop this rule object. See the DBMS_RULE_ADM.DROP_RULE procedure for more information.

DROP_CHAIN_STEP Procedure

This procedure drops a chain step. If this chain step is still used in the chain rules, the chain will be disabled.

Syntax

```
DBMS_SCHEDULER.DROP_CHAIN_STEP (
   chain_name              IN VARCHAR2,
   step_name               IN VARCHAR2,
   force                   IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 93–27 DROP_CHAIN_STEP Procedure Parameters

Parameter	Description
chain_name	The name of the chain to alter
step_name	The name of the step being dropped. Can be a comma-separated list.
force	If force is set to TRUE, this succeeds even if this chain is currently running. The running chain will not be stopped or interrupted. If force is set to FALSE and this chain is currently running, an error is thrown.

Usage Notes

Dropping a chain step requires ALTER privileges on the chain (either by being the owner, or by having ALTER privileges on the chain or by having the CREATE ANY JOB system privilege).

DROP_JOB Procedure

This procedure drops a job or all jobs in a job class. It results in the job being removed from the job queue, its metadata being removed, and no longer being visible in the *_SCHEDULER_JOBS views. Therefore, no more runs of the job will be executed. Dropping a job also drops all argument values set for that job.

Syntax

```
DBMS_SCHEDULER.DROP_JOB (
   job_name                IN VARCHAR2,
   force                   IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 93–28 DROP_JOB Procedure Parameters

Parameter	Description
job_name	The name of a job or job class. Can be a comma-delimited list. For a job class, the SYS schema should be specified. If the name of a job class is specified, the jobs that belong to that job class are dropped, but the job class itself is not dropped.
force	If force is set to FALSE, and an instance of the job is running at the time of the call, the call results in an error. If force is set to TRUE, the Scheduler first attempts to stop the running job instance (by issuing the STOP_JOB call with the force flag set to false), and then drops the job.

Usage Notes

Dropping a job requires ALTER privileges on the job either by being the owner of the job, or by having the ALTER object privilege on the job or by having the CREATE ANY JOB system privilege.

DROP_JOB_CLASS Procedure

This procedure drops a job class. Dropping a job class means that all the metadata about the job class is removed from the database.

Syntax

```
DBMS_SCHEDULER.DROP_JOB_CLASS (
   job_class_name          IN VARCHAR2,
   force                   IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 93–29 DROP_JOB_CLASS Procedure Parameters

Parameter	Description
job_class_name	The name of the job class. Can be a comma-delimited list.
force	If force is set to FALSE, a class must be unreferenced by any jobs to be dropped otherwise an error will occur. If force is set to TRUE, jobs belonging to the class are disabled and their class is set to the default class. Only if this is successful will the class be dropped. Running jobs that belong to the job class are not affected.

Usage Notes

Dropping a job class requires the MANAGE SCHEDULER system privilege.

DROP_PROGRAM Procedure

This procedure drops a program. Any arguments that pertain to the program are also dropped when the program is dropped.

Syntax

```
DBMS_SCHEDULER.DROP_PROGRAM (
   program_name            IN VARCHAR2,
   force                   IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 93–30 DROP_PROGRAM Procedure Parameters

Parameter	Description
program_name	The name of the program to be dropped. Can be a comma-delimited list.
force	If force is set to FALSE, the program must be unreferenced by any job otherwise an error will occur. If force is set to TRUE, all jobs referencing the program are disabled before dropping the program. Running jobs that point to the program are not affected by the DROP_PROGRAM call, and are allowed to continue.

Usage Notes

Dropping a program requires that you be the owner of the program or have ALTER privileges on that program. You can also drop a program if you have the CREATE ANY JOB privilege.

DROP_PROGRAM_ARGUMENT Procedures

This procedure drops a program argument. An argument can be specified by either name (if one has been given) or position.

The procedure is overloaded.

Syntax

Drops a program argument by position:

```
DBMS_SCHEDULER.DROP_PROGRAM_ARGUMENT (
   program_name            IN VARCHAR2,
   argument_position       IN PLS_INTEGER);
```

Drops a program argument by name:

```
DBMS_SCHEDULER.DROP_PROGRAM_ARGUMENT (
   program_name            IN VARCHAR2,
   argument_name           IN VARCHAR2);
```

Parameters

Table 93–31 DROP_PROGRAM_ARGUMENT Procedure Parameters

Parameter	Description
program_name	The name of the program to be altered. A program with this name must exist.
argument_name	The name of the argument being dropped
argument_position	The position of the argument to be dropped

Usage Notes

Dropping a program argument requires that you be the owner of the program or have ALTER privileges on that program. You can also drop a program argument if you have the CREATE ANY JOB privilege.

DROP_SCHEDULE Procedure

This procedure drops a schedule.

Syntax

```
DBMS_SCHEDULER.DROP_SCHEDULE (
   schedule_name    IN VARCHAR2,
   force            IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 93–32 DROP_SCHEDULE Procedure Parameters

Parameter	Description
schedule_name	The name of the schedule. Can be a comma-delimited list.
force	If force is set to FALSE, the schedule must be unreferenced by any job or window otherwise an error will occur. If force is set to TRUE, any jobs or windows that use this schedule will be disabled before the schedule is dropped Running jobs and open windows that point to the schedule are not affected.

Usage Notes

You must be the owner of the schedule being dropped or have ALTER privileges for the schedule or the CREATE ANY JOB privilege.

DROP_WINDOW Procedure

This procedure drops a window. All metadata about the window is removed from the database. All references to the window are removed from window groups.

Syntax

```
DBMS_SCHEDULER.DROP_WINDOW (
   window_name             IN VARCHAR2,
   force                   IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 93–33 DROP_WINDOW Procedure Parameters

Parameter	Description
window_name	The name of the window. Can be a comma-delimited list.
force	If force is set to FALSE, the window must be not be open or referenced by any job otherwise an error will occur. If force is set to TRUE, the window will be dropped and those jobs that have the window as their schedule will be disabled. However, jobs that have a window group of which the dropped window was a member as their schedule will not be disabled. If the window is open then, the Scheduler attempts to first close the window and then drop it. When the window is closed, normal close window rules apply. Running jobs that have the window as their schedule will be allowed to continue, unless the stop_on_window_close flag was set to TRUE for the job. If this is the case, the job will be stopped when the window is dropped.

Usage Notes

Dropping a window requires the MANAGE SCHEDULER privilege.

DROP_WINDOW_GROUP Procedure

This procedure drops a window group but not the windows that are members of this window group.

Syntax

```
DBMS_SCHEDULER.DROP_WINDOW_GROUP (
   group_name              IN VARCHAR2
   force                   IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 93–34 DROP_WINDOW_GROUP Procedure Parameters

Parameter	Description
group_name	The name of the window group
force	If force is set to FALSE, the window group must be unreferenced by any job otherwise an error will occur. If force is set to TRUE, the window group will be dropped and those jobs that have the window group as their schedule will be disabled. Running jobs that have the window group as their schedule are allowed to continue, even if the stop_on_window_close flag was set to TRUE when for the job. If a member of the window group that is being dropped is open, the window group can still be dropped.

Usage Notes

If you want to drop all the windows that are members of this group but not the window group itself, you can use the DROP_WINDOW procedure and provide the name of the window group to the call.

To drop a window group, you must have the MANAGE SCHEDULER privilege.

ENABLE Procedure

This procedure enables a program, job, chain, window, or window group. When an object is enabled, the enabled flag is set to TRUE. By default, jobs, chains, and programs are created disabled and windows and window groups are created enabled.

Validity checks are performed before enabling an object. If the check fails, the object is not enabled, and an appropriate error is returned. This procedure does not return an error if the object was already enabled.

Syntax

```
DBMS_SCHEDULER.ENABLE (
   name              IN VARCHAR2);
```

Parameters

Table 93–35 ENABLE Procedure Parameters

Parameter	Description
name	The name of the Scheduler object being enabled. Can be a comma-delimited list of names. If a job class name is specified, then all the jobs in the job class are enabled. If a window group name is specified, then the window group will be enabled, but the windows that are members of the window group, will not be enabled.

Usage Notes

Because the ENABLE procedure is used for several Scheduler objects, when enabling windows or window groups, they must be preceded by SYS.

To run ENABLE for a window or window group, you must have the MANAGE SCHEDULER privilege. Otherwise, you must be the owner of the object being enabled or have ALTER privileges on that object or have the CREATE ANY JOB privilege. For a job of type EXECUTABLE (or for a job that points to a program of type EXECUTABLE), the job owner must have the CREATE EXTERNAL JOB system privilege before the job can be enabled or run.

EVALUATE_CALENDAR_STRING Procedure

You can define repeat intervals of jobs, windows or schedules using the Scheduler's calendaring syntax. This procedure evaluates the calendar expression and tells you what the next execution date of a job or window will be. This is very useful for testing the correct definition of the calendar string without having to actually schedule the job or window.

This procedure can also be used to get multiple steps of the repeat interval by passing the `next_run_date` returned by one invocation as the `return_date_after` argument of the next invocation of this procedure.

Syntax

```
DBMS_SCHEDULER.EVALUATE_CALENDAR_STRING (
   calendar_string    IN  VARCHAR2,
   start_date         IN  TIMESTAMP WITH TIME ZONE,
   return_date_after  IN  TIMESTAMP WITH TIME ZONE,
   next_run_date      OUT TIMESTAMP WITH TIME ZONE);
```

Parameters

Table 93–36 EVALUATE_CALENDAR_STRING Procedure Parameters

Parameter	Description
`calendar_string`	The calendar string to be evaluated. The string must be in the calendaring syntax described in "Operational Notes" on page 93-4.
`start_date`	The date after which the repeat interval becomes valid. It can also be used to fill in specific items that are missing from the calendar string. Can optionally be `NULL`.
`return_date_after`	With the `start_date` and the calendar string, the Scheduler has sufficient information to determine all valid execution dates. By setting this argument, the Scheduler knows which one of all possible matches to return. When a `NULL` value is passed for this argument, the Scheduler automatically fills in `systimestamp` as its value.
`next_run_date`	The first timestamp that matches the calendar string and start date that occurs after the value passed in for the `return_date_after` argument.

Examples

The following code fragment can be used to determine the next five dates a job will run given a specific calendar string.

```
SET SERVEROUTPUT ON;
ALTER SESSION set NLS_DATE_FORMAT = 'DD-MON-YYYY HH24:MI:SS';
Session altered.

DECLARE
start_date        TIMESTAMP;
return_date_after TIMESTAMP;
next_run_date     TIMESTAMP;
BEGIN
start_date :=
  to_timestamp_tz('01-JAN-2003 10:00:00','DD-MON-YYYY HH24:MI:SS');
return_date_after := start_date;
FOR i IN 1..5 LOOP
  DBMS_SCHEDULER.EVALUATE_CALENDAR_STRING(
```

```
    'FREQ=DAILY;BYHOUR=9;BYMINUTE=30;BYDAY=MON,TUE,WED,THU,FRI',
    start_date, return_date_after, next_run_date);
DBMS_OUTPUT.PUT_LINE('next_run_date: ' || next_run_date);
return_date_after := next_run_date;
END LOOP;
END;
/

next_run_date: 02-JAN-03 09.30.00.000000 AM
next_run_date: 03-JAN-03 09.30.00.000000 AM
next_run_date: 06-JAN-03 09.30.00.000000 AM
next_run_date: 07-JAN-03 09.30.00.000000 AM
next_run_date: 08-JAN-03 09.30.00.000000 AM

PL/SQL procedure successfully completed.
```

Usage Notes

No specific Scheduler privileges are required.

EVALUATE_RUNNING_CHAIN Procedure

This procedure forces reevaluation of the rules of a running chain to trigger any rules for which the conditions have been satisfied. The job passed as an argument must point to a chain and must be running. If the job is not running, an error is thrown. (RUN_JOB can be used to start the job.)

If any of the steps of the chain are themselves running chains, another EVALUATE_RUNNING_CHAIN is performed on each of the nested running chains.

Syntax

```
DBMS_SCHEDULER.EVALUATE_RUNNING_CHAIN (
   job_name              IN VARCHAR2);
```

Parameters

Table 93–37 EVALUATE_RUNNING_CHAIN Procedure Parameter

Parameter	Description
job_name	The name of the running job (pointing to a chain) to reevaluate the rules for

Usage Notes

Running EVALUATE_RUNNING_CHAIN on a job requires alter privileges on the job (either by being the owner, or by having ALTER privileges on the job or by having the CREATE ANY JOB system privilege).

Note: The Scheduler automatically evaluates a chain:

- At the start of the chain job
- When a chain step completes
- When an event occurs that is associated with one of the event steps of the chain

For most chains, this is sufficient. EVALUATE_RUNNING_CHAIN should be used only under the following circumstances:

- After manual intervention of a running chain with the ALTER_RUNNING_CHAIN procedure
- When chain rules use SQL syntax and the rule conditions contain elements that are not under the control of the Scheduler.

In these cases, EVALUATE_RUNNING_CHAIN may not be needed if you set the evaluation_interval attribute when you created the chain.

GENERATE_JOB_NAME Function

This function returns a unique name for a job. The name will be of the form {prefix}N where N is a number from a sequence. If no prefix is specified, the generated name will, by default, be JOB$_1, JOB$_2, JOB$_3, and so on. If 'SCOTT' is specified as the prefix, the name will be SCOTT1, SCOTT2, and so on.

Syntax

```
DBMS_SCHEDULER.GENERATE_JOB_NAME (
   prefix        IN VARCHAR2 DEFAULT 'JOB$_') RETURN VARCHAR2;
```

Parameters

Table 93–38 GENERATE_JOB_NAME Function Parameter

Parameter	Description
prefix	The prefix to use when generating the job name

Usage Notes

If the prefix is explicitly set to NULL, the name will be just the sequence number. In order to successfully use such numeric names, they must be surrounded by double quotes throughout the DBMS_SCHEDULER calls. A prefix cannot be longer than 18 characters and cannot end with a digit.

Note that, even though the GENERATE_JOB_NAME function will never return the same job name twice, there is a small chance that the returned name matches an already existing database object.

No specific Scheduler privileges are required to use this function.

GET_ATTRIBUTE Procedure

This procedure retrieves the value of an attribute of a Scheduler object. It is overloaded to output values of the following types: VARCHAR2, TIMESTAMP WITH TIMEZONE, BOOLEAN, PLS_INTEGER, and INTERVAL DAY TO SECOND.

Syntax

```
DBMS_SCHEDULER.GET_ATTRIBUTE (
   name           IN VARCHAR2,
   attribute      IN VARCHAR2,
   value          OUT [VARCHAR2, TIMESTAMP WITH TIMEZONE,
                      PLS_INTEGER, BOOLEAN, INTERVAL DAY TO SECOND);

DBMS_SCHEDULER.GET_ATTRIBUTE (
   name           IN VARCHAR2,
   attribute      IN VARCHAR2,
   value          OUT [VARCHAR2, TIMESTAMP WITH TIMEZONE,
                      PLS_INTEGER, BOOLEAN, INTERVAL DAY TO SECOND],
   value2         OUT VARCHAR2);
```

Parameters

Table 93–39 GET_ATTRIBUTE Procedure Parameters

Parameter	Description
name	The name of the object
attribute	The attribute being retrieved
value	The existing value of the attribute
value2	Most attributes have only one value associated with them, but some can have two. The value2 argument is for this optional second value.

Usage Notes

To run GET_ATTRIBUTE for a job class, you must have the MANAGE SCHEDULER privilege or have EXECUTE privileges on the class. For a schedule, window, or a window group, no privileges are necessary. Otherwise, you must be the owner of the object or have ALTER or EXECUTE privileges on that object or have the CREATE ANY JOB privilege.

GET_SCHEDULER_ATTRIBUTE Procedure

This procedure retrieves the value of a Scheduler attribute.

Syntax

```
DBMS_SCHEDULER.GET_SCHEDULER_ATTRIBUTE (
   attribute      IN VARCHAR2,
   value          OUT VARCHAR2);
```

Parameters

Table 93–40 GET_SCHEDULER_ATTRIBUTE Procedure Parameters

Parameter	Description
attribute	The name of the attribute
value	The existing value of the attribute

Usage Notes

To run GET_SCHEDULER_ATTRIBUTE, you must have the MANAGE SCHEDULER privilege.

Table 93–41 lists the Scheduler attributes that you can retrieve. For more detail on these attributes, see Table 93–61 on page 93-90 and the section "Configuring the Scheduler" in *Oracle Database Administrator's Guide*.

Table 93–41 Scheduler Attributes Retrievable with GET_SCHEDULER_ATTRIBUTE

Scheduler Attribute	Description
default_timezone	Default time zone used by the Scheduler for repeat intervals and windows
log_history	Retention period in days for job and window logs
max_job_slave_processes	Maximum number of job slave processes that the Scheduler can start. May be NULL.
current_open_window	Name of the currently open window
event_expiry_time	Time in seconds before an event generated by the Scheduler and enqueued onto the Scheduler event queue expires. May be NULL.

OPEN_WINDOW Procedure

This procedure opens a window independent of its schedule. This window will open and the resource plan associated with it, will take effect immediately for the duration specified or for the normal duration of the window if no duration is given. Only an enabled window can be manually opened.

Syntax

```
DBMS_SCHEDULER.OPEN_WINDOW (
   window_name             IN VARCHAR2,
   duration                IN INTERVAL DAY TO SECOND,
   force                   IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 93–42 OPEN_WINDOW Procedure Parameters

Parameter	Description
window_name	The name of the window
duration	The duration of the window. It is of type interval day to second. If it is NULL, then the window will be opened for the regular duration as specified in the window metadata.
force	If force is set to FALSE, opening an already open window, will generate an error. If force is set to TRUE: You can open a window that is already open. The window stays open for the duration specified in the call, from the time the OPEN_WINDOW command was issued. Consider an example to illustrate this. window1 was created with a duration of four hours. It has how been open for two hours. If at this point you reopen window1 using the OPEN_WINDOW call and do not specify a duration, then window1 will be open for another four hours because it was created with that duration. If you specified a duration of 30 minutes, the window will close in 30 minutes. The Scheduler automatically closes any window that is open at that time, even if it has a higher priority. For the duration of this manually opened window, the Scheduler does not open any other scheduled windows even if they have a higher priority.

Usage Notes

If there are jobs running when the window opens, the resources allocated to them might change due to the switch in resource plan.

Opening a window manually has no impact on regular scheduled runs of the window. The next open time of the window is not updated, and will be as determined by the regular scheduled opening.

When a window that was manually opened closes, the rules about overlapping windows are applied to determine which other window should be opened at that time if any at all.

If a window fails to switch resource plans because the designated resource plan no longer exists or because resource plan switching by windows is disabled (for example, by using the ALTER SYSTEM statement with the force option), the failure to switch resource plans is recorded in the window log.

Opening a window requires the MANAGE SCHEDULER privilege.

PURGE_LOG Procedure

By default, the Scheduler automatically purges all rows in the job log and window log that are older than 30 days. The PURGE_LOG procedure is used to purge additional rows from the job and window log.

Rows in the job log table pertaining to the steps of a chain are purged only when the entry for the main chain job is purged (either manually or automatically).

Syntax

```
DBMS_SCHEDULER.PURGE_LOG (
   log_history             IN PLS_INTEGER  DEFAULT 0,
   which_log               IN VARCHAR2     DEFAULT 'JOB_AND_WINDOW_LOG',
   job_name                IN VARCHAR2     DEFAULT NULL);
```

Parameters

Table 93–43 PURGE_LOG Procedure Parameters

Parameter	Description
log_history	This specifies how much history (in days) to keep. The valid range is 0 - 999. If set to 0, no history is kept.
which_log	This specifies which type of log. Valid values for which_log are job_log, window_log, and job_and_window_log.
job_name	This specifies which job-specific entries must be purged from the jog log. This can be a comma-delimited list of job names and job classes. Whenever job_name has a value other than NULL, the which_log argument implicitly includes the job log.

Usage Notes

This procedure requires the MANAGE SCHEDULER privilege.

Examples

The following will completely purge all rows from both the job log and the window log:

```
DBMS_SCHEDULER.PURGE_LOG();
```

The following will purge all rows from the window log that are older than 5 days:

```
DBMS_SCHEDULER.PURGE_LOG(5, 'window_log');
```

The following will purge all rows from the window log that are older than 1 day and all rows from the job log that are related to jobs in jobclass1 and that are older than 1 day:

```
DBMS_SCHEDULER.PURGE_LOG(1, 'job_and_window_log', 'sys.jobclass1');
```

REMOVE_EVENT_QUEUE_SUBSCRIBER Procedure

This procedure unsubscribes a user from the Scheduler event queue SYS.SCHEDULER$_EVENT_QUEUE.

Syntax

```
DBMS_SCHEDULER.REMOVE_EVENT_QUEUE_SUBSCRIBER (
   subscriber_name         IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 93–44 REMOVE_EVENT_QUEUE_SUBSCRIBER Procedure Parameters

Parameter	Description
subscriber_name	Name of the Oracle Streams Advanced Queuing (AQ) agent for which to remove the subscription. If NULL, the user name of the calling user is used.

Usage Notes

After the agent is unsubscribed, it is deleted. If the agent does not exist or is not currently subscribed to the Scheduler event queue, an error is raised.

REMOVE_WINDOW_GROUP_MEMBER Procedure

This procedure removes one or more windows from an existing window group.

Syntax

```
DBMS_SCHEDULER.REMOVE_WINDOW_GROUP_MEMBER (
   group_name              IN VARCHAR2,
   window_list             IN VARCHAR2);
```

Parameters

Table 93–45 REMOVE_WINDOW_GROUP_MEMBER Procedure Parameters

Parameter	Description
group_name	The name of the window group.
window_list	The name of the window or windows.

Usage Notes

If any of the windows specified is either invalid, does not exist, or is not a member of the given group, the call fails. Removing a window from a group requires the MANAGE SCHEDULER privilege.

Dropping an open window from a window group has no impact on running jobs that has the window as its schedule since the jobs would only be stopped when a window closes.

RESET_JOB_ARGUMENT_VALUE Procedures

This procedure resets (clears) the value previously set to an argument for a job.

RESET_JOB_ARGUMENT_VALUE is overloaded.

Syntax

Clears a previously set job argument value by argument position:

```
DBMS_SCHEDULER.RESET_JOB_ARGUMENT_VALUE (
   job_name                IN VARCHAR2,
   argument_position       IN PLS_INTEGER);
```

Clears a previously set job argument value by argument name:

```
DBMS_SCHEDULER.RESET_JOB_ARGUMENT_VALUE (
   job_name                IN VARCHAR2,
   argument_name           IN VARCHAR2);
```

Parameters

Table 93–46 RESET_JOB_ARGUMENT_VALUE Procedure Parameters

Parameter	Description
job_name	The name of the job being altered
argument_position	The position of the program argument being reset
argument_name	The name of the program argument being reset

Usage Notes

If the corresponding program argument has no default value, the job will be disabled. Resetting a program argument of a job belonging to another user requires ALTER privileges on that job. Arguments can be specified by position or by name.

RESET_JOB_ARGUMENT_VALUE requires that you be the owner of the job or have ALTER privileges on that job. You can also reset a job argument value if you have the CREATE ANY JOB privilege.

RUN_CHAIN Procedures

This procedure immediately runs a chain or part of a chain by creating a run-once job with the job name given. If no `job_name` is given, one will be generated of the form `RUN_CHAIN$_chainnameN`, where *`chainname`* is the first 8 characters of the chain name and *N* is an integer.

If a list of start steps is given, only those steps will be started when the chain begins running. Steps not in the list that would normally have started are skipped and paused (so that they or the steps after them do not run). If `start_steps` is `NULL`, then the chain will start normally—that is, an initial evaluation will be done to see which steps to start running).

If a list of initial step states is given, the newly created chain job sets every listed step to the state specified for that step before evaluating the chain rules to see which steps to start. (Steps in the list are not started.)

Syntax

Runs a chain, with a list of start steps.

```
DBMS_SCHEDULER.RUN_CHAIN (
   chain_name                IN VARCHAR2,
   start_steps               IN VARCHAR2,
   job_name                  IN VARCHAR2 DEFAULT NULL);
```

Runs a chain, with a list of initial step states.

```
DBMS_SCHEDULER.RUN_CHAIN (
   chain_name               IN VARCHAR2,
   step_state_list          IN SYS.SCHEDULER$_STEP_TYPE_LIST,
   job_name                 IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 93–47 RUN_CHAIN Procedure Parameters

Parameter	Description
`chain_name`	The name of the chain to run
`job_name`	The name of the job to create to run the chain
`start_steps`	Comma-separated list of the steps to start when the chain starts running

Table 93–47 (Cont.) RUN_CHAIN Procedure Parameters

Parameter	Description

step_state_list	List of chain steps with an initial state (SUCCEEDED or FAILED) to set for each.
	The list is provided as nested table type SYS.SCHEDULER$_STEP_TYPE_LIST, where each element is of type SYS.SCHEDULER$_STEP_TYPE.
	`CREATE TYPE sys.scheduler$_step_type as object (`
	`  step_name  varchar2(32),`
	`  step_type  varchar2(32));`
	
	`CREATE TYPE sys.scheduler$_step_type_list IS TABLE OF`
	`  sys.scheduler$_step_type;`
	
	Set the attributes of sys.scheduler$_step_type as follows:
	step_name The name of the step
	step_type 'SUCCEEDED' or 'FAILED *error_number*'
	where *error_number* is a positive or negative integer.

Usage Notes

Running a chain requires CREATE JOB if the job is being created in the user's schema, or CREATE ANY JOB otherwise. In addition, the owner of the job being created needs execute privileges on the chain (by being the owner of the chain, by having the EXECUTE privilege on the chain, or by having the EXECUTE ANY PROGRAM system privilege).

Examples

The following example illustrates how to start a chain in the middle by providing the initial state of some chain steps.

```
declare
  initial_step_states sys.scheduler$_step_type_list;
begin
  initial_step_states := sys.scheduler$_step_type_list(
    sys.scheduler$_step_type('step1', 'SUCCEEDED'),
    sys.scheduler$_step_type('step2', 'FAILED 27486'),
    sys.scheduler$_step_type('step3', 'SUCCEEDED'),
    sys.scheduler$_step_type('step5', 'SUCCEEDED'));
  dbms_scheduler.run_chain('my_chain', initial_step_states);
end;
/
```

RUN_JOB Procedure

This procedure runs a job immediately.

Syntax

```
DBMS_SCHEDULER.RUN_JOB (
   job_name                IN VARCHAR2,
   use_current_session     IN BOOLEAN DEFAULT TRUE);
```

Parameters

Table 93–48 RUN_JOB Procedure Parameters

Parameter	Description
job_name	The name of the job being run
use_current_session	This specifies whether the job run should occur in the same session as the one that the procedure was invoked from. When use_current_session is set to TRUE: ■ You can test a job and see any possible errors on the command line. ■ run_count, last_start_date, last_run_duration, and failure_count are not updated. ■ RUN_JOB can be run in parallel with a regularly scheduled job run. When use_current_session is set to FALSE: ■ You need to check the job log to find error information. ■ run_count, last_start_date, last_run_duration, and failure_count are updated. ■ RUN_JOB fails if a regularly scheduled job is running.

Usage Notes

The job does not have to be enabled. If the job is disabled, the following validity checks are performed before running it:

- The job points to a valid job class.
- The job owner has EXECUTE privileges on the job class.
- If a program or chain is referenced, the program/chain exists.
- If a program or chain is referenced, the job owner has privileges to execute the program/chain.
- All argument values have been set (or have defaults).
- The job owner has the CREATE EXTERNAL JOB privilege if this is an external job.

The job can be run in two different modes. One is in the current user session. In this case, the call to RUN_JOB will block until it has completed the job. Any errors that occur during the execution of the job will be returned as errors to the RUN_JOB procedure. The other option is to run the job immediately like a regular job. In this case, RUN_JOB returns immediately and the job will be picked up by the coordinator and passed on to a job slave for execution. The Scheduler views and logs must be queried for the outcome of the job.

Multiple user sessions can use RUN_JOB in their sessions simultaneously when use_current_session is set to TRUE.

When using RUN_JOB with jobs that point to chains, use_current_session must be FALSE.

RUN_JOB requires that you be the owner of the job or have ALTER privileges on that job. You can also run a job if you have the CREATE ANY JOB privilege.

SET_ATTRIBUTE Procedure

This procedure changes an attribute of an object. It is overloaded to accept values of the following types: VARCHAR2, TIMESTAMP WITH TIMEZONE, BOOLEAN, PLS_INTEGER, and INTERVAL DAY TO SECOND. To set an attribute to NULL, the SET_ATTRIBUTE_NULL procedure should be used. What attributes can be set depends on the object being altered. With the exception of the object name, all object attributes can be changed.

SET_ATTRIBUTE is overloaded.

Syntax

```
DBMS_SCHEDULER.SET_ATTRIBUTE (
   name           IN VARCHAR2,
   attribute      IN VARCHAR2,
   value          IN [VARCHAR2, TIMESTAMP WITH TIMEZONE,
                      PLS_INTEGER, BOOLEAN, INTERVAL DAY TO SECOND]);

DBMS_SCHEDULER.SET_ATTRIBUTE (
   name           IN VARCHAR2,
   attribute      IN VARCHAR2,
   value          IN [VARCHAR2, TIMESTAMP WITH TIMEZONE,
                      PLS_INTEGER, BOOLEAN, INTERVAL DAY TO SECOND],
   value2         IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 93–49 SET_ATTRIBUTE Procedure Parameters

Parameter	Description
name	The name of the object
attribute	See Table 93–50 through Table 93–57.
value	The new value being set for the attribute. This cannot be NULL. To set an attribute value to NULL, use the SET_ATTRIBUTE_NULL procedure.
value2	Most attributes have only one value associated with them, but some can have two. The value2 argument is for this optional second value.

Usage Notes

If an object is altered and it was in the enabled state, the Scheduler will first disable it, make the change and then re-enable it. If any errors are encountered during the enable process, the object is not re-enabled and an error is generated.

If an object is altered and it was in the disabled state, it will remain disabled after it is altered.

To run SET_ATTRIBUTE for a window, window group, or job class, you must have the MANAGE SCHEDULER privilege. Otherwise, you must be the owner of the object being altered or have ALTER privileges on that object or have the CREATE ANY JOB privilege.

Job

If there is a running instance of the job when the SET_ATTRIBUTE call is made, it is not affected by the call. The change is only seen in future runs of the job.

If any of the schedule attributes of a job are altered while the job is running, the time of the next job run will be scheduled using the new schedule attributes. Schedule attributes of a job include `schedule_name`, `start_date`, `end_date`, and `repeat_interval`.

If any of the program attributes of a job are altered while the job is running, the new program attributes will take effect the next time the job runs. Program attributes of a job include `program_name`, `job_action`, `job_type`, and `number_of_arguments`. This is also the case for job argument values that have been set.

Granting `ALTER` on a job will let a user alter all attributes of that job except its program attributes (`program_name`, `job_type`, `job_action`, `program_action`, and `number_of_arguments`) and will not allow a user to use a PL/SQL expression to specify the schedule for a job.

We recommend you not to alter a job that was automatically created for you by the database. Jobs that were created by the database have the column `SYSTEM` set to `TRUE` in job views.

Program

If any currently running jobs use the program that is altered, they will continue to run with the program definition prior to the alter. The job will run with the new program definition the next time the job executes.

Schedule

If a schedule is altered, the change will not affect running jobs and open windows that use this schedule. The change will only be in effect the next time the jobs runs or the window opens.

Job Class

With the exception of the default job class, all job classes can be altered. To alter a job class, you must have the `MANAGE SCHEDULER` privilege.

When a job class is altered, running jobs that belong to the class are not affected. The change only takes effect for jobs that have not started running yet.

Window

When a window is altered, it does not affect an active window. The changes only take effect the next time the window opens.

To change resource plans, you must first set the `RESOURCE_MANAGER_PLAN` initialization parameter in the `init.ora` file or issue an `ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = *my_plan*` statement before the window opens.

Job Attribute Values

Table 93–50 lists job attribute values.

> **Note:** See the `CREATE_JOB` procedure for more complete descriptions of the attributes in this table.

Table 93–50 Job Attribute Values

Name	Description
logging_level	This attribute specifies how much information is logged. The three possible options are: DBMS_SCHEDULER.LOGGING_OFF No logging will be performed for any jobs in this class. DBMS_SCHEDULER.LOGGING_RUNS The Scheduler will write detailed information to the job log for all runs of each job in this class. DBMS_SCHEDULER.LOGGING_FULL In addition to recording every run of a job, the Scheduler will record all operations performed on all jobs in this class. In other words, every time a job is created, enabled, disabled, altered, and so on will be recorded in the log.
restartable	This attribute specifies whether a job can be restarted in case of failure. By default, jobs are not restartable and this attribute is set to FALSE. Setting this to TRUE means that if a job fails while running, it will be restarted from the beginning point of the job. In the case of a chain job, if this attribute is TRUE, the chain is restarted from the beginning after an application failure. If this attribute is FALSE, or if there has been a database failure, the chain is restarted at the last running step. The restart_on_recovery attribute of that step then determines if the step is restarted or marked as stopped. (If marked as stopped, the chain evaluates rules and continues.) Note that setting this attribute to TRUE might lead to data inconsistencies in some situations, for example, if data is committed within a job. Retries on errors are not counted as regular runs. The run count or failure count is not incremented until the job succeeds or has failed all its six retries. The restartable attribute is used by the Scheduler to determine whether to retry the job not only on regular application errors, but after a database malfunction as well. The Scheduler will retry the job a maximum of six times. The first time, it will wait for one second and multiply this wait time with a factor of 10 each time thereafter. Both the run count and failure count are incremented by 1 if the job has failed all its six retries. If the job immediately succeeds, or it succeeds on one of its retries, run count is incremented by 1. The Scheduler will stop retrying a job when: ■ One of the retries succeeds. ■ All of its six retries have failed. ■ The next retry would occur after the next regularly scheduled run of the job. The Scheduler no longer retries the job if the next scheduled retry is past the next regularly scheduled run for repeating jobs.
max_failures	This attribute specifies the number of times a job can fail on consecutive scheduled runs before it is automatically disabled. Once a job is disabled, it is no longer executed and its STATE is set to BROKEN in the *_SCHEDULER_JOB views. max_failures can be an integer between 1 to 1,000,000. By default, it is set to NULL, which indicates that new instances of the job will be started regardless of how many previous instances have failed.

Table 93–50 (Cont.) Job Attribute Values

Name	Description
max_runs	This attribute specifies the maximum number of consecutive scheduled runs of the job. Once max_runs is reached, the job is disabled and its state is changed to COMPLETED. max_runs can be an integer between 1 and 1,000,000. By default, it is set to NULL, which means that it will repeat forever or until end_date or max_failures is reached.
max_run_duration	This attribute specifies the maximum amount of time that the job should be allowed to run. Its datatype is INTERVAL DAY TO SECOND. If this attribute is set to a non-zero and non-NULL value, and job duration exceeds this value, the Scheduler raises an event of type JOB_OVER_MAX_DUR. It is then up to your event handler to decide whether or not to allow the job to continue.
job_weight	This attribute is for expert users of parallel technology only. If your job will be using parallel technology, you can set the value of this attribute to the degree of parallelism of your SQL inside the job. job_weight has a range of 1-100, with 1 being the default
instance_stickiness	This attribute should only be used for a database running in RAC mode. By default, it is set to TRUE. If you set instance_stickiness to TRUE, jobs start running on the instance with the lightest load and the Scheduler thereafter attempts to run on the instance that it last ran on. If that instance is either down or so overloaded that it will not start new jobs for a significant period of time, another instance will run the job. If the interval between runs is large, instance_stickiness will be ignored an the job will be handled as if it were a non-sticky job. If instance_stickiness is set to FALSE, each instance of the job runs on the first instance available. For non-RAC environments, this attribute is not useful because there is only one instance.
stop_on_window_close	This attribute only applies if the schedule of a job is a window or a window group. Setting this attribute to TRUE implies that the job should be stopped once the associated window is closed. The job is stopped using the stop_job procedure with force set to FALSE. By default, stop_on_window_close is set to FALSE. Therefore, if you do not set this attribute, the job will be allowed to continue after the window closes. Note that, although the job is allowed to continue, its resource allocation will probably change because closing a window generally also implies a change in resource plans.
job_priority	This attribute specifies the priority of this job relative to other jobs in the same class as this job. If multiple jobs within a class are scheduled to be executed at the same time, the job priority determines the order in which jobs from that class are picked up for execution by the job coordinator. It can be a value from 1 through 5, with 1 being the first to be picked up for job execution. If no job priority is specified when creating a job, the default priority of 3 is assigned to it.

Table 93–50 (Cont.) Job Attribute Values

Name	Description
schedule_limit	In heavily loaded systems, jobs are not always started at their scheduled time. This attribute enables you to have the Scheduler not start a job at all if the delay in starting the job is larger than the interval specified. It can be a value of 1 minute to 99 days. For example, if a job was supposed to start at noon and the schedule limit is set to 60 minutes, the job will not be run if it has not started to run by 1:00 p.m. If schedule_limit is not specified, the job is executed at some later date as soon as there are resources available to run it. By default, this attribute is set to null, which indicates that the job can be run at any time after its scheduled time. A scheduled job run that is skipped because of this attribute does not count against the number of runs and failures of the job. An entry in the job log will be made to reflect the skipped run.
program_name	The name of a program object to use with this job. If this is set, job_action, job_type and number_of_arguments should be NULL.
job_action	The action that the job performs, depending on the job_type attribute. For example, if job_type is 'STORED_PROCEDURE', job_action contains the name of the stored procedure.
job_type	The type of this job. Can be any of: 'PLSQL_BLOCK', 'STORED_PROCEDURE', 'EXECUTABLE', and 'CHAIN'. If this is set, program_name must be NULL.
number_of_arguments	The number of arguments if the program is inlined. If this is set, program_name should be NULL.
schedule_name	The name of a schedule or window or window group to use as the schedule for this job. If this is set, end_date, start_date and repeat_interval should all be NULL.
repeat_interval	Either a PL/SQL function returning the next date on which to run, or calendaring syntax expression. If this is set, schedule_name should be NULL. See "Calendaring Syntax" on page 93-4 for more information.
start_date	The original date on which this job started or will be scheduled to start. If this is set, schedule_name should be NULL.
end_date	The date after which the job will no longer run. It will be dropped if auto_drop is set or disabled with the state changed to COMPLETED if it is. If this is set, schedule_name should be NULL.
job_class	The class this job is associated with.
comments	An optional comment.
auto_drop	Whether the job should be dropped after having completed.
event_spec	This attribute takes two values: the value argument should contain the event condition and the value2 argument should contain the queue specification. For details on what these arguments should contain, see the descriptions for the event_condition and queue_spec arguments in the CREATE_JOB procedure.

Table 93–50 (Cont.) Job Attribute Values

Name	Description
raise_events	This attribute tells the Scheduler at what stages of the job's execution events should be raised. It is a bit vector in which zero or more of the following bits can be set. Each bit has a package constant corresponding to it. ■ job_started CONSTANT PLS_INTEGER := 1 ■ job_succeeded CONSTANT PLS_INTEGER := 2 ■ job_failed CONSTANT PLS_INTEGER :=4 ■ job_broken CONSTANT PLS_INTEGER :=8 ■ job_completed CONSTANT PLS_INTEGER :=16 ■ job_stopped CONSTANT PLS_INTEGER :=32 ■ job_sch_lim_reached CONSTANT PLS_INTEGER :=64 ■ job_disabled CONSTANT PLS_INTEGER :=128 ■ job_chain_stalled CONSTANT PLS_INTEGER :=256 ■ job_all_events CONSTANT PLS_INTEGER := 511 ■ job_run_completed CONSTANT PLS_INTEGER := job_succeeded + job_failed + job_stopped Table Table 93–51 describes these event types in detail.

Table 93–51 Event Types Raised by the Scheduler

Event Type	Description
job_started	The job started
job_succeeded	The job completed successfully
job_failed	The job failed, either by throwing an error or by abnormally terminating
job_broken	The job has been disabled and has changed to the BROKEN state because it exceeded the number of failures defined by the max_failures job attribute
job_completed	The job completed because it reached its max_runs or end_date
job_stopped	The job was stopped by a call to STOP_JOB
job_sch_lim_reached	The job's schedule limit was reached. The job was not started because the delay in starting the job exceeded the value of the schedule_limit job attribute.
job_disabled	The job was disabled by the Scheduler or by a call to SET_ATTRIBUTE
job_chain_stalled	A job running a chain was put into the CHAIN_STALLED state. A running chain becomes stalled if there are no steps running or scheduled to run and the chain evaluation_interval is set to NULL. No progress will be made in the chain unless there is manual intervention.
job_all_events	Not an event, but a constant that provides an easy way for you to enable all events
job_run_completed	A job run either failed, succeeded, or was stopped

Program Attribute Values

Table 93–52 lists program attribute values.

Note: See the CREATE_PROGRAM procedure for more complete descriptions of the attributes in this table.

Table 93–52 Program Attribute Values

Name	Description
program_action	The action that the program performs, depending on the program_type attribute. For example, if program_type is 'STORED_PROCEDURE', program_action contains the name of the stored procedure.
program_type	The type of program. This must be one of the following supported program types: 'PLSQL_BLOCK', 'STORED_PROCEDURE', and 'EXECUTABLE'.
number_of_arguments	The number of arguments required by the stored procedure or other executable that the program invokes
comments	An optional comment. This can describe what the program does, or give usage details.

Job Class Values

Table 93–53 lists job class attribute values.

Note: See the CREATE_JOB_CLASS procedure for more complete descriptions of the attributes in this table.

Table 93–53 Job Class Attribute Values

Name	Description
resource_consumer_group	The resource consumer group a class is associated with. If resource_consumer_group is set, service must be NULL.
service	The database service that the jobs in the job class have affinity to. If service is set, resource_consumer_group must be NULL.
logging_level	This attribute specifies how much information is logged. The three possible options are: ■ DBMS_SCHEDULER.LOGGING_OFF No logging will be performed for any jobs in this class. ■ DBMS_SCHEDULER.LOGGING_RUNS The Scheduler will write detailed information to the job log for all runs of each job in this class. ■ DBMS_SCHEDULER.LOGGING_FULL In addition to recording every run of a job, the Scheduler will record all operations performed on all jobs in this class. In other words, every time a job is created, enabled, disabled, altered, and so on will be recorded in the log.

Table 93–53 (Cont.) Job Class Attribute Values

Name	Description
log_history	This enables you to control the amount of logging the Scheduler performs. To prevent the job log and the window log from growing indiscriminately, the Scheduler has an attribute that specifies how much history (in days) to keep. Once a day, the Scheduler will automatically purge all log entries from both the job log as well as the window log that are older than the specified history. The default is 30 days. You can change the default by using the SET_SCHEDULER_ATTRIBUTE procedure. For example, to change it to 90 days, issue the following statement: `DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE ('log_history','90');` The range of valid values is 1 through 999.
comments	An optional comment about the class.

Window Attribute Values

Table 93–54 lists window attribute values.

> **Note:** See the CREATE_WINDOW procedure for more complete descriptions of the attributes in this table.

Table 93–54 Window Attribute Values

Name	Description
resource_plan	The resource plan to be associated with a window. When the window opens, the system will switch to this resource plan. When the window closes, the original resource plan will be restored. If a resource plan has been made active with the force option, no resource plan switch will occur. Only one resource plan can be associated with a window. It may be NULL or the empty string (""). When it is NULL, the resource plan that is in effect when the window opens stays in effect for the duration of the window. When it is the empty string, the resource manager is disabled for the duration of the window.
window_priority	The priority of the window. Must be one of 'LOW' (default) or 'HIGH'.
duration	The duration of the window.
schedule_name	The name of a schedule to use with this window. If this is set, start_date, end_date, and repeat_interval must all be NULL.
repeat_interval	A string using the calendaring syntax. PL/SQL date functions are not allowed. If this is set, schedule_name must be NULL. See "Calendaring Syntax" on page 93-4 for more information.
start_date	The next date on which this window is scheduled to open. If this is set, schedule_name must be NULL.
end_date	The date after which the window will no longer open. If this is set, schedule_name must be NULL.
comments	An optional comment about the window.

Window Group Attribute Values

Table 93–55 lists window group attribute values.

Table 93–55 Window Group Attribute Values

Name	Description
comments	An optional comment about the window group.

Schedule Attribute Values

Table 93–56 lists schedule attribute values.

> **Note:** See the CREATE_SCHEDULE and CREATE_CALENDAR_SCHEDULE procedures for more complete descriptions of the attributes in this table.

Table 93–56 Schedule Attribute Values

Name	Description
repeat_interval	An expression using the calendaring syntax. See "Calendaring Syntax" on page 93-4 for more information.
comments	An optional comment.
end_date	The cutoff date after which the schedule will not specify any dates.
start_date	The start or reference date used by the calendaring syntax.
event_spec	This attribute takes two values: the value argument should contain the event condition and the value2 argument should contain the queue specification. For details on what these arguments should contain, see the descriptions for the event_condition and queue_spec arguments to the CREATE_JOB procedure.

Chain Attribute Values

Table 93–57 lists chain attribute values.

> **Note:** See the CREATE_CHAIN procedure for more complete descriptions of the attributes in this table.

Table 93–57 Chain Attribute Values

Name	Description
evaluation_interval	If this is not NULL, evaluation of the chain occurs not only at normal evaluation times (when the job starts, when a step completes, or when an event that is associated with an event step arrives), but also periodically at this interval. For most chains, the normal evaluation times are sufficient. Because evaluation of a large chain is CPU intensive, this attribute should be used only when chain rules use SQL syntax and the rule conditions contain elements that are not under the control of the Scheduler.
comments	An optional comment describing the purpose of the chain.

SET_ATTRIBUTE_NULL Procedure

This procedure sets an attribute of an object to NULL. What attributes can be set depends on the object being altered. If the object is enabled, it will be disabled before being altered and be reenabled afterward. If the object cannot be reenabled, an error is generated and the object will be left in a disabled state.

Syntax

```
DBMS_SCHEDULER.SET_ATTRIBUTE_NULL (
   name              IN VARCHAR2,
   attribute         IN VARCHAR2);
```

Parameters

Table 93–58 SET_ATTRIBUTE_NULL Procedure Parameters

Parameter	Description
name	The name of the object
attribute	The attribute being changed

Usage Notes

To run SET_ATTRIBUTE_NULL for a window, window group, or job class, you must have the MANAGE SCHEDULER privilege. Otherwise, you must be the owner of the object being altered or have ALTER privileges on that object or have the CREATE ANY JOB privilege.

SET_JOB_ANYDATA_VALUE Procedures

This procedure sets the value for an argument of the associated program for a job, encapsulated in an `AnyData` object. It overrides any default value set for the program argument. `NULL` is a valid assignment for a program argument. The argument can be specified by position or by name. You can specify by name only when:

- The job points to a saved program object
- The argument was assigned a name with the DEFINE_ANYDATA_ARGUMENT Procedure

No type checking of the argument is done at any time by the Scheduler.

`SET_JOB_ANYDATA_VALUE` is overloaded.

Syntax

Sets a program argument by its position.

```
DBMS_SCHEDULER.SET_JOB_ANYDATA_VALUE (
   job_name                IN VARCHAR2,
   argument_position       IN PLS_INTEGER,
   argument_value          IN SYS.ANYDATA);
```

Sets a program argument by its name.

```
DBMS_SCHEDULER.SET_JOB_ANYDATA_VALUE (
   job_name                IN VARCHAR2,
   argument_name           IN VARCHAR2,
   argument_value          IN SYS.ANYDATA);
```

Parameters

Table 93–59 SET_JOB_ANYDATA_VALUE Procedure Parameters

Parameter	Description
`job_name`	The name of the job to be altered
`argument_name`	The name of the program argument being set
`argument_position`	The position of the program argument being set
`argument_value`	The new value to be assigned to the program argument, encapsulated in an `AnyData` object

Usage Notes

`SET_JOB_ANYDATA_VALUE` requires that you be the owner of the job or have `ALTER` privileges on that job. You can also set a job argument value if you have the `CREATE ANY JOB` privilege.

See Also:

- "SET_JOB_ARGUMENT_VALUE Procedures" on page 93-89
- "DEFINE_ANYDATA_ARGUMENT Procedure" on page 93-40

SET_JOB_ARGUMENT_VALUE Procedures

This procedure sets the value of an argument of the associated program for a job. It overrides any default value set for the program argument. NULL is a valid assignment for a program argument. The argument can be specified by position or by name. You can specify by name only when:

- The job points to a saved program object
- The argument was assigned a name with the DEFINE_PROGRAM_ARGUMENT Procedures or the DEFINE_METADATA_ARGUMENT Procedure

No type checking of the argument is done at any time by the Scheduler.

SET_JOB_ARGUMENT_VALUE is overloaded.

Syntax

Sets an argument value by position:

```
DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE (
   job_name                IN VARCHAR2,
   argument_position       IN PLS_INTEGER,
   argument_value          IN VARCHAR2);
```

Sets an argument value by name:

```
DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE (
   job_name                IN VARCHAR2,
   argument_name           IN VARCHAR2,
   argument_value          IN VARCHAR2);
```

Parameters

Table 93–60 SET_JOB_ARGUMENT_VALUE Procedure Parameters

Parameter	Description
job_name	The name of the job to be altered
argument_name	The name of the program argument being set
argument_position	The position of the program argument being set
argument_value	The new value to be set for the program argument. To set a non-VARCHAR value, use the SET_JOB_ANYDATA_ARGUMENT_VALUE procedure.

Usage Notes

SET_JOB_ARGUMENT_VALUE requires that you be the owner of the job or have ALTER privileges on that job. You can also set a job argument value if you have the CREATE ANY JOB privilege.

See Also:

- "SET_JOB_ANYDATA_VALUE Procedures" on page 93-88
- "DEFINE_PROGRAM_ARGUMENT Procedures" on page 93-48

SET_SCHEDULER_ATTRIBUTE Procedure

This procedure sets the value of a Scheduler attribute. This takes effect immediately but the resulting changes may not be seen immediately. The attributes you can set are `default_timezone`, `max_job_slave_processes`, and `log_history`.

Syntax

```
DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE (
   attribute      IN VARCHAR2,
   value          IN VARCHAR2);
```

Parameters

Table 93–61 SET_SCHEDULER_ATTRIBUTE Procedure Parameters

Parameter	Description
`attribute`	The name of the Scheduler attribute. Possible values are: ■ `default_timezone`: It is very important that this attribute is set. Whenever a `repeat_interval` is specified without setting the `start_date`, the Scheduler needs to know which time zone it must apply to the repeat interval syntax. For example, if the repeat interval is specified as `"FREQ=DAILY;BYHOUR=22"` the job will repeat every day at 10pm, but 10pm in which time zone? If no `start_date` is specified the Scheduler will pick up the time zone from this `default_timezone` attribute. If you want your job or window to follow daylight savings adjustments, you must set this attribute to the proper region name. For instance, if your database resides in Paris, you would set this to `'Europe/Warsaw'`. Daylight saving adjustments will not be followed if you specify an absolute offset. For example, '-8:00' would be correct for only half of the year in San Francisco. If no value is specified for this attribute, the Scheduler uses the time zone of `systimestamp` when the job or window is enabled. This is always an absolute offset and will not follow daylight savings adjustments. ■ `log_history`: This enables you to control the amount of logging the Scheduler performs. ■ `max_job_slave_processes`: This enables you to set a maximum number of slave processes for a particular system configuration and load. The default value is `NULL`, and the valid range is 1-999. Even though the Scheduler automatically determines what the optimum number of slave processes is for a given system configuration and load, you still might want to set a fixed limit on the Scheduler. If this is the case, you can set this attribute. Although the number set by `max_job_slave_processes` is a real maximum, it does not mean the Scheduler will start the specified number of slaves. For example, even though this attribute is set to 10, the Scheduler might still determine that is should not start more than 3 slave processes. However, if it wants to start 15, but it is set to 10, it will not start more than 10. ■ `event_expiry_time`: The time in seconds before an event generated by the Scheduler (in other words, a message enqueued by the Scheduler into the Scheduler event queue) expires (in other words, is automatically purged from the queue). If `NULL`, Scheduler-generated events expire after 24 hours.
`value`	The new value of the attribute

Usage Notes

To run SET_SCHEDULER_ATTRIBUTE, you must have the MANAGE SCHEDULER privilege.

STOP_JOB Procedure

This procedure stops currently running jobs or all jobs in a job class. Any instance of the job will be stopped. After stopping the job, the state of a one-time job will be set to STOPPED whereas the state of a repeating job will be set to SCHEDULED or COMPLETED depending on whether the next run of the job is scheduled.

If a job pointing to a chain is stopped, all steps of the running chain that are running will be stopped.

Syntax

```
DBMS_SCHEDULER.STOP_JOB (
   job_name         IN VARCHAR2
   force            IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 93–62 STOP_JOB Procedure Parameters

Parameter	Description
job_name	The name of the job or job class. Can be a comma-delimited list. For a job class, the SYS schema should be specified. If the name of a job class is specified, the jobs that belong to that job class are stopped. The job class is not affected by this call.
force	If force is set to FALSE, the Scheduler tries to gracefully stop the job using an interrupt mechanism. This method gives control back to the slave process, which can update the status of the job in the job queue to stopped. If this fails, an error is returned. If force is set to TRUE, the Scheduler will immediately terminate the job slave. Oracle recommends that STOP_JOB with force set to TRUE be used only after a STOP_JOB with force set to FALSE has failed. Use of the force option requires the MANAGE SCHEDULER system privilege.

Usage Notes

STOP_JOB without the force option requires that you be the owner of the job or have ALTER privileges on that job. You can also stop a job if you have the CREATE ANY JOB or MANAGE SCHEDULER privilege.

STOP_JOB with the force option requires that you have the MANAGE SCHEDULER privilege.

94

DBMS_SERVER_ALERT

The DBMS_SERVER_ALERT package enables you to configure the Oracle Database server to issue an alert when a threshold for a specified server metric has been violated. You can configure both warning and critical thresholds for a large number of predefined metrics.

If a warning threshold is reached, the server generates a severity level 5 alert. If a critical threshold is reached, the server generates a severity level 1 alert.

The chapter contains the following topics:

Using DBMS_SERVER_ALERT

This section contains topics which relate to using the `DBMS_SERVER_ALERT` package. The following topics define constants used in package procedures.

- Object Types
- Relational Operators
- Supported Metrics

Object Types

You qualify the metric by an individual object for the following object types.

Table 94–1 Object Types Defined as Constants

Constant	Description
OBJECT_TYPE_SYSTEM	Metrics collected on the system level for each instance.
OBJECT_TYPE_FILE	Metrics collected on the file level. These are used for AVERAGE_FILE_READ_TIME and AVERAGE_FILE_WRITE_TIME metrics.
OBJECT_TYPE_SERVICE	Metrics collected on the service level. Currently ELAPSED_TIME_PER_CALL and CPU_TIME_PER_CALL are collected.
OBJECT_TYPE_TABLESPACE	Metrics collected on the tablespace level.
OBJECT_TYPE_EVENT_CLASS	Metrics collected on wait event class level. Currently supported metrics are AVG_USERS_WAITING and DB_TIME_WAITING.
OBJECT_TYPE_SESSION	Metrics collected on the session level. Currently only BLOCKED_USERS is collected. The threshold can only be set at the instance level, which means that no object name should be specified when setting the threshold for this type of metric.

Relational Operators

You can specify a relational comparison operator to determine whether or not a given metric's value violates the threshold setting. The server supports the following operators.

Table 94–2 Relational Operators Defined as Constants

Constant	Description
OPERATOR_CONTAINS	A metric value matching an entry in a list of threshold values is considered a violation.
OPERATOR_DO_NOT_CHECK	The metric value is not compared to the threshold value, and no alerts are generated. Use this operator to disable alerts for a metric.
OPERATOR_EQ	A metric value equal to the threshold value is considered a violation.
OPERATOR_GE	A metric value greater than or equal to the threshold value is considered a violation.
OPERATOR_GT	A metric value greater than the threshold value is considered a violation.
OPERATOR_LE	A metric value less than or equal to the threshold value is considered a violation.
OPERATOR_LT	A metric value less than the threshold value is considered a violation.
OPERATOR_NE	A metric value not equal to the threshold value is considered a violation.

Supported Metrics

The following metrics are supported. All internal metric names are supplied as package contants.

Table 94–3 List of Supported Metrics

Metric Name (Internal)	Metric Name (External)	Units
SQL_SRV_RESPONSE_TIME	Service Response (for each execution)	Seconds
BUFFER_CACHE_HIT	Buffer Cache Hit (%)	% of cache accesses
LIBRARY_CACHE_HIT	Library Cache Hit (%)	% of cache accesses
LIBRARY_CACHE_MISS	Library Cache Miss (%)	% of cache accesses
MEMORY_SORTS_PCT	Sorts in Memory (%)	% of sorts
REDO_ALLOCATION_HIT	Redo Log Allocation Hit	% of redo allocations
TRANSACTION_RATE	Number of Transactions (for each second)	Transactions for each Second
PHYSICAL_READS_SEC	Physical Reads (for each second)	Reads for each Second
PHYSICAL_READS_TXN	Physical Reads (for each transaction)	Reads for each Transaction
PHYSICAL_WRITES_SEC	Physical Writes (for each second)	Writes for each Second
PHYSICAL_WRITES_TXN	Physical Writes (for each transaction)	Writes for each Transaction
PHYSICAL_READS_DIR_SEC	Direct Physical Reads (for each second)	Reads for each Second
PHYSICAL_READS_DIR_TXN	Direct Physical Reads (for each transaction)	Reads for each Transaction
PHYSICAL_WRITES_DIR_SEC	Direct Physical Writes (for each second)	Writes for each Second
PHYSICAL_WRITES_DIR_TXN	Direct Physical Writes (for each transaction)	Writes for each Transaction
PHYSICAL_READS_LOB_SEC	Direct LOB Physical Reads (for each second)	Reads for each Second
PHYSICAL_READS_LOB_TXN	Direct LOB Physical Reads (for each transaction)	Reads for each Transaction
PHYSICAL_WRITES_LOB_SEC	Direct LOB Physical Writes (for each second)	Writes for each Second
PHYSICAL_WRITES_LOB_TXN	Direct LOB Physical Writes (for each transaction)	Writes for each Transaction
REDO_GENERATED_SEC	Redo Generated (for each second)	Redo Bytes for each Second
REDO_GENERATED_TXN	Redo Generated (for each transaction)	Redo Bytes for each Transaction
DATABASE_WAIT_TIME	Database Wait Time (%)	% of all database time
DATABASE_CPU_TIME	Database CPU Time (%)	% of all database time
LOGONS_SEC	Cumulative Logons (for each second)	Logons for each Second

Table 94–3 (Cont.) List of Supported Metrics

Metric Name (Internal)	Metric Name (External)	Units
LOGONS_TXN	Cumulative Logons (for each transaction)	Logons for each Transaction
LOGONS_CURRENT	Current Number of Logons	Number of Logons
OPEN_CURSORS_SEC	Cumulative Open Cursors (for each second)	Cursors for each Second
OPEN_CURSORS_TXN	Cumulative Open Cursors (for each transaction)	Cursors for each Transaction
OPEN_CURSORS_CURRENT	Current Number of Cursors	Number of Cursors
USER_COMMITS_SEC	User Commits (for each second)	Commits for each Second
USER_COMMITS_TXN	User Commits (for each transaction)	Commits for each Transaction
USER_ROLLBACKS_SEC	User Rollbacks (for each second)	Rollbacks for each Second
USER_ROLLBACKS_TXN	User Rollbacks (for each transaction)	Rollbacks for each Transaction
USER_CALLS_SEC	User Calls (for each second)	Calls for each Second
USER_CALLS_TXN	User Calls (for each transaction)	Calls for each Transaction
RECURSIVE_CALLS_SEC	Recursive Calls (for each second)	Calls for each Second
RECURSIVE_CALLS_TXN	Recursive Calls (for each transaction)	Calls for each Transaction
SESS_LOGICAL_READS_SEC	Session Logical Reads (for each second)	Reads for each Second
SESS_LOGICAL_READS_TXN	Session Logical Reads (for each transaction)	Reads for each Transaction
DBWR_CKPT_SEC	DBWR Checkpoints (for each second)	Checkpoints for each Second
LOG_SWITCH_SEC	Background Checkpoints (for each second)	Checkpoints for each Second
REDO_WRITES_SEC	Redo Writes (for each second)	Writes for each Second
REDO_WRITES_TXN	Redo Writes (for each transaction)	Writes for each Transaction
LONG_TABLE_SCANS_SEC	Scans on Long Tables (for each second)	Scans for each Second
LONG_TABLE_SCANS_TXN	Scans on Long Tables (for each transaction)	Scans for each Transaction
TOTAL_TABLE_SCANS_SEC	Total Table Scans (for each second)	Scans for each Second
TOTAL_TABLE_SCANS_TXN	Total Table Scans (for each transaction)	Scans for each Transaction
FULL_INDEX_SCANS_SEC	Fast Full Index Scans (for each second)	Scans for each Second
FULL_INDEX_SCANS_TXN	Fast Full Index Scans (for each transaction)	Scans for each Transaction
TOTAL_INDEX_SCANS_SEC	Total Index Scans (for each second)	Scans for each Second
TOTAL_INDEX_SCANS_TXN	Total Index Scans (for each transaction)	Scans for each Transaction
TOTAL_PARSES_SEC	Total Parses (for each second)	Parses for each Second

Table 94–3 (Cont.) List of Supported Metrics

Metric Name (Internal)	Metric Name (External)	Units
TOTAL_PARSES_TXN	Total Parses(for each transaction)	Parses for each Transaction
HARD_PARSES_SEC	Hard Parses(for each second)	Parses for each Second
HARD_PARSES_TXN	Hard Parses(for each transaction)	Parses for each Transaction
PARSE_FAILURES_SEC	Parse Failures (for each second)	Parses for each Second
PARSE_FAILURES_TXN	Parse Failures (for each transaction)	Parses for each Transaction
DISK_SORT_SEC	Sorts to Disk (for each second)	Sorts for each Second
DISK_SORT_TXN	Sorts to Disk (for each transaction)	Sorts for each Transaction
ROWS_PER_SORT	Rows Processed for each Sort	Rows for each Sort
EXECUTE_WITHOUT_PARSE	Executes Performed Without Parsing	% of all executes
SOFT_PARSE_PCT	Soft Parse (%)	% of all parses
CURSOR_CACHE_HIT	Cursor Cache Hit (%)	% of soft parses
USER_CALLS_PCT	User Calls (%)	% of all calls
TXN_COMMITTED_PCT	Transactions Committed (%)	% of all transactions
NETWORK_BYTES_SEC	Network Bytes, for each second	Bytes for each Second
RESPONSE_TXN	Response (for each transaction)	Seconds for each Transaction
DATA_DICT_HIT	Data Dictionary Hit (%)	% of dictionary accesses
DATA_DICT_MISS	Data Dictionary Miss (%)	% of dictionary accesses
SHARED_POOL_FREE_PCT	Shared Pool Free(%)	% of shared pool
AVERAGE_FILE_READ_TIME	Average File Read Time	Microseconds
AVERAGE_FILE_WRITE_TIME	Average File Write Time	Microseconds
DISK_IO	Disk I/O	Milliseconds
PROCESS_LIMIT_PCT	Process Limit Usage (%)	% of maximum value
SESSION_LIMIT_PCT	Session Limit Usage (%)	% of maximum value
USER_LIMIT_PCT	User Limit Usage (%)	% of maximum value
AVG_USERS_WAITING	Average Number of Users Waiting on a Class of Wait Events	Count of sessions
DB_TIME_WAITING	Percent of Database Time Spent Waiting on a Class of Wait Events	% of Database Time
APPL_DESGN_WAIT_SCT	Application Design Wait (by session count)	Count of sessions
APPL_DESGN_WAIT_TIME	Application Design Wait (by time)	Microseconds
PHYS_DESGN_WAIT_SCT	Physical Design Wait (by session count)	Count of sessions
PHYS_DESGN_WAIT_TIME	Physical Design Wait (by time)	Microseconds
CONTENTION_WAIT_SCT	Internal Contention Wait (by session count)	Count of sessions
CONTENTION_WAIT_TIME	Internal Contention Wait (by time)	Microseconds
PSERVICE_WAIT_SCT	Process Service Wait (by session count)	Count of sessions

Table 94–3 (Cont.) List of Supported Metrics

Metric Name (Internal)	Metric Name (External)	Units
PSERVICE_WAIT_TIME	Process Service Wait (by time)	Microseconds
NETWORK_MSG_WAIT_SCT	Network Message Wait (by session count)	Count of sessions
NETWORK_MSG_WAIT_TIME	Network Message Wait (by time)	Microseconds
DISK_IO_WAIT_SCT	Disk I/O Wait (by session count)	Count of sessions
OS_SERVICE_WAIT_SCT	Operating System Service Wait (by session count)	Count of sessions
OS_SERVICE_WAIT_TIME	Operating System Service Wait (by time)	Microseconds
DBR_IO_LIMIT_WAIT_SCT	Resource Mgr I/O Limit Wait (by session count)	Count of sessions
DBR_IO_LIMIT_WAIT_TIME	Resource Mgr I/O Limit Wait (by time)	Microseconds
DBR_CPU_LIMIT_WAIT_SCT	Resource Mgr CPU Limit Wait (by session count)	Count of sessions
DBR_CPU_LIMIT_WAIT_TIME	Resource Mgr CPU Limit Wait (by time)	Microseconds
DBR_USR_LIMIT_WAIT_SCT	Resource Mgr User Limit Wait (by session count)	Count of sessions
DBR_USR_LIMIT_WAIT_TIME	Resource Mgr User Limit Wait (by time)	Microseconds
OS_SCHED_CPU_WAIT_SCT	Operating System Scheduler CPU Wait (by session count)	Count of sessions
OS_SCHED_CPU__WAIT_TIME	Operating System Scheduler CPU Wait (by time)	Microseconds
CLUSTER_MSG_WAIT_SCT	Cluster Messaging Wait (by session count)	Count of sessions
CLUSTER_MSG_WAIT_TIME	Cluster Messaging Wait (by time)	Microseconds
OTHER_WAIT_SCT	Other Waits (by session count)	Count of sessions
OTHER_WAIT_TIME	Other Waits (by time)	Microseconds
ENQUEUE_TIMEOUTS_SEC	Enqueue Timeouts (for each second)	Timeouts for each Second
ENQUEUE_TIMEOUTS_TXN	Enqueue Timeouts (for each transaction)	Timeouts for each Transaction
ENQUEUE_WAITS_SEC	Enqueue Waits (for each second)	Waits for each Second
ENQUEUE_WAITS_TXN	Enqueue Waits (for each transaction)	Waits for each Transaction
ENQUEUE_DEADLOCKS_SEC	Enqueue Deadlocks (for each second)	Deadlocks for each Second
ENQUEUE_DEADLOCKS_TXN	Enqueue Deadlocks (for each transaction)	Deadlocks for each Transaction
ENQUEUE_REQUESTS_SEC	Enqueue Requests (for each second)	Requests for each Second
ENQUEUE_REQUESTS_TXN	Enqueue Requests (for each transaction)	Requests for each Transaction

Table 94–3 (Cont.) List of Supported Metrics

Metric Name (Internal)	Metric Name (External)	Units
DB_BLKGETS_SEC	DB Block Gets (for each second)	Gets for each Second
DB_BLKGETS_TXN	DB Block Gets (for each transaction)	Gets for each Transaction
CONSISTENT_GETS_SEC	Consistent Gets (for each second)	Gets for each Second
CONSISTENT_GETS_TXN	Consistent Gets (for each transaction)	Gets for each Transaction
DB_BLKCHANGES_SEC	DB Block Changes (for each second)	Changes for each Second
DB_BLKCHANGES_TXN	DB Block Changes (for each transaction)	Changes for each Transaction
CONSISTENT_CHANGES_SEC	Consistent Changes (for each second)	Changes for each Second
CONSISTENT_CHANGES_TXN	Consistent Changes (for each transaction)	Changes for each Transaction
SESSION_CPU_SEC	Database CPU (for each second)	Microseconds for each Second
SESSION_CPU_TXN	Database CPU (for each transaction)	Microseconds for each Transaction
CR_BLOCKS_CREATED_SEC	CR Blocks Created (for each second)	Blocks for each Second
CR_BLOCKS_CREATED_TXN	CR Blocks Created (for each transaction)	Blocks for each Transaction
CR_RECORDS_APPLIED_SEC	CR Undo Records Applied (for each second)	Records for each Second
CR_RECORDS_APPLIED_TXN	CR Undo Records Applied (for each transaction)	Records for each Transaction
RB_RECORDS_APPLIED_SEC	Rollback Undo Records Applied (for each second)	Records for each Second
RB_RECORDS_APPLIED_TXN	Rollback Undo Records Applied(for each transaction)	Records for each Transaction
LEAF_NODE_SPLITS_SEC	Leaf Node Splits (for each second)	Splits for each Second
LEAF_NODE_SPLITS_TXN	Leaf Node Splits (for each transaction)	Splits for each Transaction
BRANCH_NODE_SPLITS_SEC	Branch Node Splits (for each second)	Splits for each Second
BRANCH_NODE_SPLITS_TXN	Branch Node Splits (for each transaction)	Splits for each Transaction
GC_BLOCKS_CORRUPT	Global Cache Blocks Corrupt	Blocks
GC_BLOCKS_LOST	Global Cache Blocks Lost	Blocks
GC_AVG_CR_GET_TIME	Global Cache CR Request	Milliseconds
GC_AVG_CUR_GET_TIME	Global Cache Current Request	Milliseconds
PX_DOWNGRADED_SEC	Downgraded Parallel Operations (for each second)	Operations for each Second
PX_DOWNGRADED_25_SEC	Downgraded to 25% and more (for each second)	Operations for each Second
PX_DOWNGRADED_50_SEC	Downgraded to 50% and more (for each second)	Operations for each Second
PX_DOWNGRADED_75_SEC	Downgraded to 75% and more (for each second)	Operations for each Second

Table 94–3 (Cont.) List of Supported Metrics

Metric Name (Internal)	Metric Name (External)	Units
PX_DOWNGRADED_SER_SEC	Downgraded to serial (for each second)	Operations for each Second
BLOCKED_USERS	Number of Users blocked by some Session	Number of Users
PGA_CACHE_HIT	PGA Cache Hit (%)	% bytes processed in PGA
ELAPSED_TIME_PER_CALL	Elapsed time for each user call for each service	Microseconds for each call
CPU_TIME_PER_CALL	CPU time for each user call for each service	Microseconds for each call
TABLESPACE_PCT_FULL	Tablespace space usage	% full
TABLESPACE_BYT_FREE	Tablespace bytes space usage	Kilobytes free

Summary of DBMS_SERVER_ALERT Subprograms

Table 94–4 DBMS_SERVER_ALERT Package Subprograms

Subprogram	Description
EXPAND_MESSAGE Function on page 94-12	Expands alert messages
GET_THRESHOLD Procedure on page 94-13	Gets the current threshold settings for a specified metric
SET_THRESHOLD Procedure on page 94-14	Sets the warning and critical thresholds for a specified metric

EXPAND_MESSAGE Function

This function expands alert messages.

Syntax

```
DBMS_SERVER_ALERT.EXPAND_MESSAGE(
   user_language            IN   VARCHAR2,
   message_id               IN   NUMBER,
   argument_1               IN   VARCHAR2,
   argument_2               IN   VARCHAR2,
   argument_3               IN   VARCHAR2,
   argument_4               IN   VARCHAR2,
   argument_5               IN   VARCHAR2)
  RETURN VARCHAR2;
```

Parameters

Table 94–5 EXPAND_MESSAGE Procedure Parameters

Parameter	Description
user_language	The language of the current session.
message_id	Id of the alert message
argument_1	The first argument in the alert message.
argument_2	The second argument in the alert message.
argument_3	The third argument in the alert message.
argument_4	The fourth argument in the alert message.
argument_5	The fifth argument in the alert message.

GET_THRESHOLD Procedure

This procedure gets the current threshold settings for the specified metric.

Syntax

```
DBMS_SERVER_ALERT.GET_THRESHOLD(
   metrics_id               IN   BINARY_INTEGER,
   warning_operator         OUT  BINARY_INTEGER,
   warning_value            OUT  VARCHAR2,
   critical_operator        OUT  BINARY_INTEGER,
   critical_value           OUT  VARCHAR2,
   observation_period       OUT  BINARY_INTEGER,
   consecutive_occurrences  OUT  BINARY_INTEGER,
   instance_name            IN   VARCHAR2,
   object_type              IN   BINARY_INTEGER,
   object_name              IN   VARCHAR2);
```

Parameters

Table 94–6 GET_THRESHOLD Procedure Parameters

Parameter	Description
metrics_id	The internal name of the metric. See "Supported Metrics" on page 94-5.
warning_operator	The operator for the comparing the actual value with the warning threshold.
warning_value	The warning threshold value.
critical_operator	The operator for the comparing the actual value with the critical threshold.
critical_value	The critical threshold value.
observation_period	The period at which the metric values are computed and verified against the threshold setting.
consecutive_ occurrences	The number of observation periods the metric value should violate the threshold value before the alert is issued.
instance_name	The name of the instance for which the threshold is set. This is NULL for database-wide alerts.
object_type	Either OBJECT_TYPE_SYSTEM or OBJECT_TYPE_SERVICE.
object_name	The name of the object.

SET_THRESHOLD Procedure

This procedure sets the warning and critical thresholds for a specified metric.

Syntax

```
DBMS_SERVER_ALERT.SET_THRESHOLD(
   metrics_id               IN   BINARY_INTEGER,
   warning_operator         IN   BINARY_INTEGER,
   warning_value            IN   VARCHAR2,
   critical_operator        IN   BINARY_INTEGER,
   critical_value           IN   VARCHAR2,
   observation_period       IN   BINARY_INTEGER,
   consecutive_occurrences  IN   BINARY_INTEGER,
   instance_name            IN   VARCHAR2,
   object_type              IN   BINARY_INTEGER,
   object_name              IN   VARCHAR2);
```

Parameters

Table 94–7 SET_THRESHOLD Procedure Parameters

Parameter	Description
metrics_id	The internal name of the metric. See "Supported Metrics" on page 94-5.
warning_operator	The operator for the comparing the actual value with the warning threshold (such as OPERATOR_GE). See "Relational Operators" on page 94-4.
warning_value	The warning threshold value. This is NULL if no warning threshold is set. A list of values may be specified for OPERATOR_CONTAINS.
critical_operator	The operator for the comparing the actual value with the critical threshold. See "Relational Operators" on page 94-4.
critical_value	The critical threshold value. This is NULL if not set. A list of values may be specified for OPERATOR_CONTAINS.
observation_period	The period at which the metric values are computed and verified against the threshold setting. The valid range is 1 to 60 minutes.
consecutive_ occurrences	The number of observation periods the metric value should violate the threshold value before the alert is issued.
instance_name	The name of the instance for which the threshold is set. This is NULL for database-wide alerts.
object_type	See "Object Types" on page 94-3.
object_name	The name of the object. This is NULL for SYSTEM.

95

DBMS_SERVICE

The `DBMS_SERVICE` package lets you create, delete, activate and deactivate services for a single instance.

The chapter contains the following topics:

- Using DBMS_SERVICE
  - Overview
  - Security Model
  - Constants
  - Exceptions
- Summary of DBMS_SERVICE Subprograms

> **See Also:** *Oracle Database Oracle Clusterware and Oracle Real Application Clusters Administration and Deployment Guide* for administering services in Real Application Clusters.

Using DBMS_SERVICE

This section contains topics which relate to using the `DBMS_SERVICE` package.

Overview

DBMS_SERVICE supports the management of services in the RDBMS for the purposes of workload measurement, management, prioritization, and XA/and distributed transaction management.

Oracle Real Application Clusters (RAC) has a functionality to manage service names across instances. This package allows the creation, deletion, starting and stopping of services in both RAC and a single instance. Additionally it provides the ability to disconnect all sessions which connect to the instance with a service name when RAC removes that service name from the instance.

> **See Also:** For more information about Oracle Real Application Clusters, *Oracle Database Oracle Clusterware and Oracle Real Application Clusters Administration and Deployment Guide*.

Security Model

Privileges

The client using this package should have the `ALTER SYSTEM` execution privilege and the `V$SESSION` table read privilege.

Schemas

This package should be installed under `SYS` schema.

Roles

The `EXECUTE` privilege of the package is granted to the `DBA` role only.

Constants

The DBMS_SERVICE package uses the constants shown in following tables

- Constants used in calling arguments are described in Table 95–1, " Constants used in Calling Arguments"
- Constants used in connection balancing goal arguments are described inTable 95–2, " Constants used in Connection Balancing Goal Arguments"
- Constants used TAF failover attribute arguments are described inTable 95–3, " Constants used in TAF Failover Attribute Arguments"

Table 95–1 Constants used in Calling Arguments

Name	Type	Value	Description
GOAL_NONE	NUMBER	0	Disables Load Balancing Advisory
GOAL_SERVICE_TIME	NUMBER	1	Load Balancing Advisory is based on elapsed time for work done in the service plus available bandwidth to the service
GOAL_THROUGHPUT	NUMBER	2	Load Balancing Advisory is based on the rate that work is completed in the service plus available bandwidth to the service

Table 95–2 Constants used in Connection Balancing Goal Arguments

Name	Type	Value	Description
CLB_GOAL_SHORT	NUMBER	1	Connection load balancing uses Load Balancing Advisory, when Load Balancing Advisory is enabled (either goal_service_time or goal_throughput). When GOAL=NONE (no load balancing advisory), connection load balancing uses an abridged advice based on CPU utilization.
CLB_GOAL_LONG	NUMBER	2	Balances the number of connections per instance using session count per service. This setting is recommended for applications with long connections such as forms. This setting can be used with Load Balancing Advisory when the connection pool is sized to accommodate gravitation within the pool itself (without adding or removing connections). The latter is the most efficient design.

Table 95–3 Constants used in TAF Failover Attribute Arguments

Name	Type	Value	Description
FAILOVER_METHOD_NONE	VARCHAR2	0	Server side TAF is not enabled for this service
FAILOVER_METHOD_BASIC	VARCHAR2	1	Server side TAF method is BASIC. BASIC is the only value currently supported. This means that a new connection is established at failure time. It is not possible to pre-establish a backup connection. (which is to say, PRECONNECT is not supported)

Table 95–3 (Cont.) Constants used in TAF Failover Attribute Arguments

Name	Type	Value	Description
FAILOVER_TYPE_NONE	NUMBER		Server side TAF type is NONE
FAILOVER_TYPE_SESSION	NUMBER		Server side TAF failover type is SESSION. At failure time, if the failover type is SESSION, TAF will re-connect to a surviving node and re-establish a vanilla database session. Customizations (for example, ALTER SESSION) must be re-executed in a failover callback.
FAILOVER_TYPE_SELECT	NUMBER		Server side TAF failover type is SELECT
FAILOVER_RETRIES	NUMBER		Number of retries to use during a failover. Specifies the number of times that TAF should attempt the re-connect and re-authenticate pair. The value must be integral and greater than 0. The maximum value is UB4MAXVAL
FAILOVER_DELAY	NUMBER		Number of seconds delay before trying to failover. Specifies the delay (in seconds) that TAF will incur if the re-connect / re-authentication fails. The value must be integral and greater than 0. The maximum value is UB4MAXVAL.

Usage Notes

- If a TAF callback has been registered, then the failover retries and failover delay are ignored. If an error occurs, TAF will continue to re-attempt the connect and authentication as long as the callback returns a value of OCI_FO_RETRY. Any delay should be coded into the callback logic
- Server side TAF settings override client-side counterparts that might be configured in TNS connect descriptors. If TAF is not configured on the client side, then at a minimum, the failover type must be set to enable TAF. If the failover type is set on the server side, then the failover method will default to BASIC. Delay and retries are optional and may be specified independently.

Exceptions

The following table lists the exceptions raised by DBMS_SERVICE package.

Table 95–4 DBMS_SERVICE Exceptions

Exception	Error Code	Description
NULL_SERVICE_NAME	44301	The service name argument was found to be NULL
NULL_NETWORK_NAME	44302	The network name argument was found to be NULL
SERVICE_EXISTS	44303	This service name was already in existence
SERVICE_DOES_NOT_EXIST	44304	The specified service was not in existence
SERVICE_IN_USE	44305	The specified service was running
SERVICE_NAME_TOO_LONG	44306	The service name was too long
NETWORK_PREFIX_TOO_LONG	44307	The network name, excluding the domain, was too long
NOT_INITIALIZED	44308	The services layer was not yet initialized
GENERAL_FAILURE	44309	There was an unknown failure
MAX_SERVICES_EXCEEDED	44310	The maximum number of services has been reached
SERVICE_NOT_RUNNING	44311	The specified service was not running
DATABASE_CLOSED	44312	The database was closed
INVALID_INSTANCE	44313	The instance name argument was not valid
NETWORK_EXISTS	44314	The network name was already in existence
NULL_ATTRIBUTES	44315	All attributes specified were NULL
INVALID_ARGUMENT	44316	Invalid argument supplied
DATABASE_READONLY	44317	The database is open read-only
MAX_SN_LENGTH	44318	The total length of all running service network names exceeded the maximum allowable length

Summary of DBMS_SERVICE Subprograms

Table 95–5 DBMS_SERVICE Package Subprograms

Subprogram	Description
CREATE_SERVICE Procedure on page 95-9	Creates service
DELETE_SERVICE Procedure on page 95-10	Deletes service
DISCONNECT_SESSION Procedure on page 95-11	Disconnects service
MODIFY_SERVICE Procedure on page 95-12	Modifies service
START_SERVICE Procedure on page 95-13	Activates service
STOP_SERVICE Procedure on page 95-14	Stops service

CREATE_SERVICE Procedure

This procedure creates a service name in the data dictionary. Services are also created in the data dictionary implicitly when you set the service in the `service_name` parameter or by means of the `ALTER SYSTEM SET SERVICE_NAMES` command.

Syntax

```
DBMS_SERVICE.CREATE_SERVICE(
   service_name        IN VARCHAR2,
   network_name        IN VARCHAR2,
   goal                IN NUMBER DEFAULT NULL,
   dtp                 IN BOOLEAN DEFAULT NULL,
   aq_ha_notifications IN BOOLEAN DEFAULT NULL,
   failover_method     IN VARCHAR2 DEFAULT NULL,
   failover_type       IN VARCHAR2 DEFAULT NULL,
   failover_retries    IN NUMBER DEFAULT NULL,
   failover_delay      IN NUMBER DEFAULT NULL,
   clb_goal            IN NUMBER DEFAULT NULL);
```

Parameters

Table 95–6 CREATE_SERVICE Procedure Parameters

Parameter	Description
`service_name`	The name of the service limited to 64 characters in the Data Dictionary
`network_name`	The network name of the service as used in SQLNet connect descriptors for client connections. This is limited to the NET `service_names` character set (see *Oracle Database Net Services Reference*).
`goal`	The workload management goal directive for the service. Valid values: ■ `DBMS_SERVICE.GOAL_SERVICE_TIME` ■ `DBMS_SERVICE.GOAL_THROUGHPUT` ■ `DBMS_SERVICE.GOAL_NONE`
`dtp`	Declares the service to be for DTP or distributed transactions including XA transactions
`aq_ha_notifications`	Determines whether HA events are sent via AQ for this service
`failover_method`	The TAF failover method for the service
`failover_type`	The TAF failover type for the service
`failover_retries`	The TAF failover retries for the service
`failover_delay`	The TAF failover delay for the service
`clb_goal`	Method used for Connection Load Balancing (see Table 95–2, " Constants used in Connection Balancing Goal Arguments")

Examples

```
DBMS_SERVICE.CREATE_SERVICE('ernie.us.oracle.com','ernie.us.oracle.com');
```

DELETE_SERVICE Procedure

This procedure deletes a service from the data dictionary.

Syntax

```
DBMS_SERVICE.DELETE_SERVICE(
   service_name   IN VARCHAR2);
```

Parameters

Table 95–7 DELETE_SERVICE Procedure Parameters

Parameter	Description
service_name	The name of the service limited to 64 characters in the Data Dictionary

Examples

```
DBMS_SERVICE.DELETE_SERVICE('ernie.us.oracle.com');
```

DISCONNECT_SESSION Procedure

This procedure disconnects sessions with the named service at the current instance.

Syntax

```
DBMS_SERVICE.DISCONNECT_SESSION(
   service_name   IN VARCHAR2);
```

Parameters

Table 95–8 DISCONNECT_SESSION Procedure Parameters

Parameter	Description
service_name	The name of the service limited to 64 characters in the Data Dictionary

Usage Notes

- This procedure can be used in the context of a single instance as well as with Real Application Clusters.
- This subprogram does not return until all corresponding sessions are disconnected. Therefore, use the DBMS_JOB package or put the SQL session in background if the caller does not want to wait for all corresponding sessions disconnected.

Examples

This disconnects sessions with service_name 'ernie.us.oracle.com'.

```
DBMS_SERVICE.DISCONNECT_SESSION('ernie.us.oracle.com');
```

MODIFY_SERVICE Procedure

This procedure modifies an existing service.

Syntax

```
DBMS_SERVICE.MODIFY_SERVICE(
   service_name        IN VARCHAR2,
   goal                IN NUMBER DEFAULT NULL,
   dtp                 IN BOOLEAN DEFAULT NULL,
   aq_ha_notifications IN BOOLEAN DEFAULT NULL,
   failover_method     IN VARCHAR2 DEFAULT NULL,
   failover_type       IN VARCHAR2 DEFAULT NULL,
   failover_retries    IN NUMBER DEFAULT NULL,
   failover_delay      IN NUMBER DEFAULT NULL,
   clb_goal            IN NUMBER DEFAULT NULL);
```

Parameters

Table 95–9 MODIFY_SERVICE Procedure Parameters

Parameter	Description
service_name	The name of the service limited to 64 characters in the Data Dictionary
goal	The workload management goal directive for the service. Valid values: ■ DBMS_SERVICE.GOAL_SERVICE_TIME ■ DBMS_SERVICE.GOAL_THROUGHPUT ■ DBMS_SERVICE.GOAL_NONE
dtp	Declares the service to be for DTP or distributed transactions including XA transactions
aq_ha_notifications	Determines whether HA events are sent via AQ for this service
failover_method	The TAF failover method for the service
failover_type	The TAF failover type for the service
failover_retries	The TAF failover retries for the service
failover_delay	The TAF failover delay for the service
clb_goal	Method used for Connection Load Balancing (see Table 95–2, " Constants used in Connection Balancing Goal Arguments")

START_SERVICE Procedure

This procedure starts a service. This procedure alters the `service_name` IOP to contain this `service_name`. In RAC, implementing this option will act on the instance specified.

Syntax

```
DBMS_SERVICE.START_SERVICE(
   service_name  IN VARCHAR2,
   instance_name IN VARCHAR2);
```

Parameters

Table 95–10 START_SERVICE Procedure Parameters

Parameter	Description
`service_name`	The name of the service limited to 64 characters in the Data Dictionary
`instance_name`	The name of the instance where the service should be activated (optional). The instance on which to start the service. `NULL` results in starting of the service on the local instance. In single instance this can only be the current instance or `NULL`. Specify `DBMS_SERVICE.ALL_INSTANCES` to start the service on all configured instances.

Examples

```
DBMS_SERVICE.START_SERVICE('ernie.us.oracle.com');
```

STOP_SERVICE Procedure

This procedure stops a service, altering the `service_name` IOP to remove this `service_name`. In RAC this will call out to CRS to stop the service optionally on the instance specified.

Syntax

```
DBMS_SERVICE.STOP_SERVICE(
   service_name   IN VARCHAR2,
   instance_name  IN VARCHAR2);
```

Parameters

Table 95–11 STOP_SERVICE Procedure Parameters

Parameter	Description
`service_name`	The name of the service limited to 64 characters in the Data Dictionary
`instance_name`	The name of the instance where the service should be stopped (optional). The instance on which to stop the service. `NULL` results in stopping of the service locally. n single instance this can only be the current instance or `NULL`. The default in RAC and exclusive case is `NULL`. Specify `DBMS_SERVICE.ALL_INSTANCES` to stop the service on all configured instances.

Examples

```
DBMS_SERVICE.STOP_SERVICE('ernie.us.oracle.com');
```

96

DBMS_SESSION

This package provides access to SQL `ALTER SESSION` and `SET ROLE` statements, and other session information, from PL/SQL. You can use `DBMS_SESSION` to set preferences and security levels.

This chapter contains the following topics:

Using DBMS_SESSION

Security Model

This package runs with the privileges of the calling user, rather than the package owner SYS.

Operational Notes

You should not attempt to turn `close_cached_open_cursors` on or off.

Summary of DBMS_SESSION Subprograms

Table 96–1 DBMS_SESSION Package Subprograms

Subprogram	Description
CLEAR_ALL_CONTEXT Procedure on page 96-6	Clears all context information
CLEAR_CONTEXT Procedure on page 96-7	Clears the context
CLEAR_IDENTIFIER Procedure on page 96-8	Clears the identifier
CLOSE_DATABASE_LINK Procedure on page 96-9	Closes database link
FREE_UNUSED_USER_MEMORY Procedure on page 96-10	Lets you reclaim unused memory after performing operations requiring large amounts of memory
IS_ROLE_ENABLED Function on page 96-12	Determines if the named role is enabled for the session.
IS_SESSION_ALIVE Function on page 96-13	Determines if the specified session is active
LIST_CONTEXT Procedures on page 96-14	Returns a list of active namespace and context for the current session
SESSION _TRACE_DISABLE Procedure on page 96-19	Resets the session-level SQL trace for the session from which it was called.
SESSION _TRACE_ENABLE Procedure on page 96-20	Enables session-level SQL trace for the invoking session
RESET_PACKAGE Procedure on page 96-21	De-instantiates all packages in the session
SET_CONTEXT Procedure on page 96-23	Sets or resets the value of a context attribute
SET_IDENTIFIER on page 96-25	Sets the identifier
SET_NLS Procedure on page 96-26	Sets Globalization Support (NLS)
SET_ROLE Procedure on page 96-27	Sets role
SET_SQL_TRACE Procedure on page 96-28	Turns tracing on or off
SWITCH_CURRENT_ CONSUMER_GROUP Procedure on page 96-29	Facilitates changing the current resource consumer group of a user's current session
UNIQUE_SESSION_ID Function on page 96-31	Returns an identifier that is unique for all sessions currently connected to this database

CLEAR_ALL_CONTEXT Procedure

Syntax

```
DBMS_SESSION.CLEAR_ALL_CONTEXT
   namespace          VARCHAR2);
```

Parameters

Table 96–2 CLEAR_ALL_CONTEXT Procedure Parameters

Parameter	Description
namespace	The namespace where the application context information is to be cleared. Required.

Usage Notes

This procedure must be invoked directly or indirectly by the trusted package.

CLEAR_CONTEXT Procedure

Syntax

```
DBMS_SESSION.CLEAR_CONTEXT
   namespace         VARCHAR2,
   client_identifier VARCHAR2
   attribute         VARCHAR2);
```

Parameters

Table 96–3 CLEAR_CONTEXT Procedure Parameters

Parameter	Description
namespace	The namespace in which the application context is to be cleared. Required. For a session-local context, namespace must be specified. If namespace is defined as Session Local Context, then client_identifier is optional since it is only associated with a globally accessed context. For a globally accessed context, namespace must be specified. NULL is a valid value for client_identifier because a session with no identifier set can see a context that looks like the (namespace, attribute, value, username, null) set using SET_CONTEXT.
client_ identifier	Applies to a global context and is optional for other types of contexts; 64-byte maximum.
attribute	The specific attribute in the namespace to be cleared. Optional. the default is NULL. If you specify attribute as NULL, then (namespace, attribute, value) for that namespace are cleared from the session. If attribute is not specified, then all context information that has the namespace and client_ identifier arguments is cleared.

Usage Notes

This procedure must be invoked directly or indirectly by the trusted package.

CLEAR_IDENTIFIER Procedure

This procedure removes the `set_client_id` in the session.

Syntax

```
DBMS_SESSION.CLEAR_IDENTIFIER;
```

Usage Notes

This procedure is executable by public.

CLOSE_DATABASE_LINK Procedure

This procedure closes an open database link. It is equivalent to the following SQL statement:

```
ALTER SESSION CLOSE DATABASE LINK <name>
```

Syntax

```
DBMS_SESSION.CLOSE_DATABASE_LINK (
   dblink VARCHAR2);
```

Parameters

Table 96–4 CLOSE_DATABASE_LINK Procedure Parameters

Parameter	Description
dblink	Name of the database link to close.

FREE_UNUSED_USER_MEMORY Procedure

This procedure reclaims unused memory after performing operations requiring large amounts of memory (more than 100K).

Examples of operations that use large amounts of memory include:

- Large sorting where entire `sort_area_size` is used and `sort_area_size` is hundreds of KB.
- Compiling large PL/SQL packages, procedures, or functions.
- Storing hundreds of KB of data within PL/SQL indexed tables.

You can monitor user memory by tracking the statistics "session UGA memory" and "session PGA memory" in the `v$sesstat` or `v$statname` fixed views. Monitoring these statistics also shows how much memory this procedure has freed.

> **Note:** This procedure should only be used in cases where memory is at a premium. It should be used infrequently and judiciously.

Syntax

```
DBMS_SESSION.FREE_UNUSED_USER_MEMORY;
```

Return Values

The behavior of this procedure depends upon the configuration of the server operating on behalf of the client:

- **Dedicated server**: This returns unused PGA memory and session memory to the operating system. Session memory is allocated from the PGA in this configuration.
- **Shared server**: This returns unused session memory to the `shared_pool`. Session memory is allocated from the `shared_pool` in this configuration.

Usage Notes

In order to free memory using this procedure, the memory must not be in use.

After an operation allocates memory, only the same type of operation can reuse the allocated memory. For example, after memory is allocated for sort, even if the sort is complete and the memory is no longer in use, only another sort can reuse the sort-allocated memory. For both sort and compilation, after the operation is complete, the memory is no longer in use, and the user can call this procedure to free the unused memory.

An indexed table implicitly allocates memory to store values assigned to the indexed table's elements. Thus, the more elements in an indexed table, the more memory the RDBMS allocates to the indexed table. As long as there are elements within the indexed table, the memory associated with an indexed table is in use.

The scope of indexed tables determines how long their memory is in use. Indexed tables declared globally are indexed tables declared in packages or package bodies. They allocate memory from session memory. For an indexed table declared globally, the memory remains in use for the lifetime of a user's login (lifetime of a user's session), and is freed after the user disconnects from ORACLE.

Indexed tables declared locally are indexed tables declared within functions, procedures, or anonymous blocks. These indexed tables allocate memory from PGA

memory. For an indexed table declared locally, the memory remains in use for as long as the user is still running the procedure, function, or anonymous block in which the indexed table is declared.After the procedure, function, or anonymous block is finished running, the memory is then available for other locally declared indexed tables to use (in other words, the memory is no longer in use).

Assigning an uninitialized, "empty" indexed table to an existing index table is a method to explicitly re-initialize the indexed table and the memory associated with the indexed table. After this operation, the memory associated with the indexed table is no longer in use, making it available to be freed by calling this procedure. This method is particularly useful on indexed tables declared globally which can grow during the lifetime of a user's session, as long as the user no longer needs the contents of the indexed table.

The memory rules associated with an indexed table's scope still apply; this method and this procedure, however, allow users to intervene and to explicitly free the memory associated with an indexed table.

Examples

The following PL/SQL illustrates the method and the use of procedure `FREE_UNUSED_USER_MEMORY`.

```
CREATE PACKAGE foobar
   type number_idx_tbl is table of number indexed by binary_integer;

   store1_table  number_idx_tbl;     --  PL/SQL indexed table
   store2_table  number_idx_tbl;     --  PL/SQL indexed table
   store3_table  number_idx_tbl;     --  PL/SQL indexed table
   ...
END;            --  end of foobar

DECLARE
   ...
   empty_table   number_idx_tbl;     --  uninitialized ("empty") version
BEGIN
   FOR i in 1..1000000 loop
     store1_table(i) := i;           --  load data
   END LOOP;
   ...
   store1_table := empty_table;      --  "truncate" the indexed table
   ...
   -
   dbms_session.free_unused_user_memory;  -- give memory back to system

   store1_table(1) := 100;           --  index tables still declared;
   store2_table(2) := 200;           --  but truncated.
   ...
END;
```

IS_ROLE_ENABLED Function

This function determines if the named role is enabled for this session.

Syntax

```
DBMS_SESSION.IS_ROLE_ENABLED (
   rolename    VARCHAR2)
  RETURN BOOLEAN;
```

Parameters

Table 96–5 IS_ROLE_ENABLED Function Parameters

Parameter	Description
rolename	Name of the role.

Return Values

Table 96–6 IS_ROLE_ENABLED Function Return Values

Return	Description
is_role_enabled	TRUE or FALSE, depending on whether the role is enabled.

IS_SESSION_ALIVE Function

This function determines if the specified session is active.

Syntax

```
DBMS_SESSION.IS_SESSION_ALIVE (
   uniqueid VARCHAR2)
  RETURN BOOLEAN;
```

Parameters

Table 96–7 IS_SESSION_ALIVE Function Parameters

Parameter	Description
uniqueid	Unique ID of the session: This is the same one as returned by UNIQUE_SESSION_ID.

Return Values

Table 96–8 IS_SESSION_ALIVE Function Return Values

Return	Description
is_session_alive	TRUE or FALSE, depending on whether the session is active.

LIST_CONTEXT Procedures

This procedure returns a list of active namespaces and contexts for the current session.

Syntax

```
TYPE AppCtxRecTyp IS RECORD (
   namespace VARCHAR2(30),
   attribute VARCHAR2(30),
   value     VARCHAR2(256));

TYPE AppCtxTabTyp IS TABLE OF AppCtxRecTyp INDEX BY BINARY_INTEGER;

DBMS_SESSION.LIST_CONTEXT (
   list OUT AppCtxTabTyp,
   size OUT NUMBER);
```

Parameters

Table 96–9 LIST_CONTEXT Procedure Parameters

Parameter	Description
`list`	Buffer to store a list of application context set in the current session.

Return Values

Table 96–10 LIST_CONTEXT Procedure Return Values

Return	Description
`list`	A list of (namespace, attribute, values) set in current session.
`size`	Returns the number of entries in the buffer returned.

Usage Notes

The context information in the list appears as a series of <namespace> <attribute> <value>. Because `list` is a table type variable, its size is dynamically adjusted to the size of returned list.

MODIFY_PACKAGE_STATE Procedure

This procedure can be used to perform various actions (as specified by the `action_flags` parameter) on the session state of all PL/SQL program units active in the session. This takes effect after the PL/SQL call that made the current invocation finishes running. The procedure uses the `DBMS_SESSION` constants listed in Table 96–12.

Syntax

```
DBMS_SESSION.MODIFY_PACKAGE_STATE(
   action_flags IN PLS_INTEGER);
```

Parameters

Table 96–11 MODIFY_PACKAGE_STATE Procedure Parameters

Parameter	Description
`action_flags`	Bit flags that determine the action taken on PL/SQL program units: `DBMS_SESSION.FREE_ALL_RESOURCES` (or `1`)—frees all memory associated with each of the previously run PL/SQL programs from the session. Clears the current values of any package globals and closes cached cursors. On subsequent use, the PL/SQL program units are reinstantiated and package globals are reinitialized. Invoking `MODIFY_PACKAGE_STATE` with the `DBMS_SESSION.FREE_ALL_RESOURCES` parameter provides functionality identical to the `DBMS_SESSION.RESET_PACKAGE()` interface. `DBMS_SESSION.REINITIALIZE` (or `2`)—reinitializes packages without actually being freed and recreated from scratch. Instead the package memory is reused. In terms of program semantics, the `DBMS_SESSION.REINITIALIZE` flag is similar to the `DBMS_SESSION.FREE_ALL_RESOURCES` flag in that both have the effect of reinitializing all packages. However, `DBMS_SESSION.REINITIALIZE` should exhibit better performance than the `DBMS_SESSION.FREE_ALL_RESOURCES` option because: ▪ Packages are reinitialized without actually being freed and recreated from scratch. Instead the package memory gets reused. ▪ Any open cursors are closed, semantically speaking. However, the cursor resource is not actually freed. It is simply returned to the PL/SQL cursor cache. The cursor cache is not flushed. Hence, cursors corresponding to frequently accessed static SQL in PL/SQL remains cached in the PL/SQL cursor cache and the application does not incur the overhead of opening, parsing, and closing a new cursor for those statements on subsequent use. ▪ The session memory for PL/SQL modules without global state (such as types, stored-procedures) will not be freed and recreated.

Usage Notes

See the parameter descriptions in Table 96–13 for the differences between the flags and why DBMS_SESSION.REINITIALIZE exhibits better performance than DBMS_SESSION.FREE_ALL_RESOURCES.

Table 96–12 Action_flags Constants for MODIFY_PACKAGE_STATE

Constant	Description
FREE_ALL_RESOURCES	PLS_INTEGER:= 1
REINITIALIZE	PLS_INTEGER:= 2

- Reinitialization refers to the process of resetting all package variables to their initial values and running the initialization block (if any) in the package bodies. Consider the package:

```
package P is
  n number;
  m number := P2.foo;
  d date := SYSDATE;
  cursor c is select * from emp;
  procedure bar;
end P;
/
package body P is
  v   varchar2(20) := 'hello';
  procedure bar is
  begin
    ...
  end;
  procedure init_pkg is
  begin
    ....
  end;
begin
  -- initialization block
  init_pkg;
  ...
  ...
end P;
/
```

For the package P, reinitialization involves:

- Setting P.n to NULL
- Invoking function P2.foo and setting P.m to the value returned from P2.foo
- Setting P.d to the return value of SYSDATE built-in
- Closing cursor P.c if it was previously opened
- Setting P.v to 'hello'
- Running the initialization block in the package body
- The reinitialization for a package is done only if the package is actually referenced subsequently. Furthermore, the packages are reinitialized in the order in which they are referenced subsequently.

- When using FREE_ALL_RESOURCES or REINITIALIZE, make sure that resetting package variable values does not affect the application.
- Because DBMS_SESSION.REINITIALIZE does not actually cause all the package state to be freed, in some situations, the application could use significantly more session memory than if the FREE_ALL_RESOURCES flag or the RESET_PACKAGE procedure had been used. For instance, after performing DBMS_SESSION.MODIFY_PACKAGE_STATE(DBMS_SESSION.REINITIALIZE), if the application does not refer to many of the packages that were previously referenced, then the session memory for those packages will remain until the end of the session (or until DBMS_SESSION.RESET_PACKAGE is called).
- Because the client-side PL/SQL code cannot reference remote package variables or constants, you must explicitly use the values of the constants. For example, DBMS_SESSION.MODIFY_PACKAGE_STATE(DBMS_SESSION.REINITIALIZE) does not compile on the client because it uses the constant DBMS_SESSION.REINITIALIZE.

  Instead, use DBMS_SESSION.MODIFY_PACKAGE_STATE(2) on the client, because the argument is explicitly provided.

Examples

This example illustrates the use of DBMS_SESSION.MODIFY_PACKAGE_STATE. Consider a package P with some global state (a cursor c and a number cnt). When the package is first initialized, the package variable cnt is 0 and the cursor c is CLOSED. Then, in the session, change the value of cnt to 111 and also execute an OPEN operation on the cursor. If you call print_status to display the state of the package, you see that cnt is 111 and that the cursor is OPEN. Next, call DBMS_SESSION.MODIFY_PACKAGE_STATE. If you print the status of the package P again using print_status, you see that cnt is 0 again and the cursor is CLOSED. If the call to DBMS_SESSION.MODIFY_PACKAGE_STATE had not been made, then the second print_status would have printed 111 and OPEN.

```
create or replace package P is
  cnt    number := 0;
  cursor c is select * from emp;
  procedure print_status;
end P;
/
show errors;

create or replace package body P is
  procedure print_status is
  begin
    dbms_output.put_line('P.cnt = ' || cnt);
    if c%ISOPEN then
      dbms_output.put_line('P.c is OPEN');
    else
      dbms_output.put_line('P.c is CLOSED');
    end if;
  end;
end P;
/
show errors;

SQL> set serveroutput on;
SQL> begin
  2   P.cnt := 111;
  3   open p.c;
```

```
  4   P.print_status;
  5  end;
  6  /
P.cnt = 111
P.c is OPEN

PL/SQL procedure successfully completed.

SQL> begin
  2   dbms_session.modify_package_state(dbms_session.reinitialize);
  3  end;
  4  /

PL/SQL procedure successfully completed.

SQL> set serveroutput on;
SQL>
SQL> begin
  2   P.print_status;
  3  end;
  4  /
P.cnt = 0
P.c is CLOSED

PL/SQL procedure successfully completed.
```

SESSION _TRACE_DISABLE Procedure

This procedure resets the session-level SQL trace for the session from which it was called. Client ID and service/module/action traces are not affected.

Syntax

```
DBMS_SESSION.SESSION_TRACE_ENABLE();
```

SESSION _TRACE_ENABLE Procedure

This procedure enables session-level SQL trace for the invoking session. Invoking this procedure results in SQL tracing of every SQL statement issued by the session.

Syntax

```
DBMS_SESSION.SESSION_TRACE_ENABLE(
   waits  IN   BOOLEAN DEFAULT TRUE,
   binds  IN   BOOLEAN DEFAULT FALSE);
```

Parameters

Table 96–13 SESSION_TRACE_ENABLE Procedure Parameters

Parameter	Description
`waits`	Specifies if wait information is to be traced
`binds`	Specifies if bind information is to be traced

RESET_PACKAGE Procedure

This procedure de-instantiates all packages in this session. It frees the package state.

> **Note:** See "SESSION _TRACE_ENABLE Procedure" on page 96-20. The MODIFY_PACKAGE_STATE interface, introduced in Oracle9i, provides an equivalent of the RESET_PACKAGE capability. It is an efficient, lighter-weight variant for reinitializing the state of all PL/SQL packages in the session.

Memory used for caching the execution state is associated with all PL/SQL functions, procedures, and packages that were run in a session.

For packages, this collection of memory holds the current values of package variables and controls the cache of cursors opened by the respective PL/SQL programs. A call to `RESET_PACKAGE` frees the memory associated with each of the previously run PL/SQL programs from the session, and, consequently, clears the current values of any package globals and closes any cached cursors.

`RESET_PACKAGE` can also be used to reliably restart a failed program in a session. If a program containing package variables fails, then it is hard to determine which variables need to be reinitialized. `RESET_PACKAGE` guarantees that all package variables are reset to their initial values.

Syntax

```
DBMS_SESSION.RESET_PACKAGE;
```

Usage Notes

Because the amount of memory consumed by all executed PL/SQL can become large, you might use `RESET_PACKAGE` to trim down the session memory footprint at certain points in your database application. However, make sure that resetting package variable values will not affect the application. Also, remember that later execution of programs that have lost their cached memory and cursors will perform slower, because they need to re-create the freed memory and cursors.

`RESET_PACKAGE` does not free the memory, cursors, and package variables immediately when called.

> **Note:** `RESET_PACKAGE` only frees the memory, cursors, and package variables after the PL/SQL call that made the invocation finishes running.

For example, PL/SQL procedure `P1` calls PL/SQL procedure `P2`, and `P2` calls `RESET_PACKAGE`. The `RESET_PACKAGE` effects do not occur until procedure `P1` finishes execution (the PL/SQL call ends).

Examples

This SQL*Plus script runs a large program with many PL/SQL program units that may or may not use global variables, but it doesn't need them beyond this execution:

```
EXECUTE large_plsql_program1;
```

To free up PL/SQL cached session memory:

```
EXECUTE DBMS_SESSION.RESET_PACKAGE;
```

To run another large program:

```
EXECUTE large_plsql_program2;
```

SET_CONTEXT Procedure

This procedure sets the context, of which there are four types: session local, globally initialized, externally initialized, and globally accessed.

Of its five parameters, only the first three are required; the final two parameters are optional, used only in globally accessed contexts. Further parameter information appears in the parameter table and the usage notes.

Syntax

```
DBMS_SESSION.SET_CONTEXT (
   namespace VARCHAR2,
   attribute VARCHAR2,
   value     VARCHAR2,
   username  VARCHAR2,
   client_id VARCHAR2 );
```

Parameters

Table 96–14 SET_CONTEXT Procedure Parameters

Parameter	Description
namespace	The namespace of the application context to be set, limited to 30 bytes.
attribute	The attribute of the application context to be set, limited to 30 bytes.
value	The value of the application context to be set, limited to 4 kilobytes.
username	The database username attribute of the application context. Default: NULL
client_id	The application-specific client_id attribute of the application context (64-byte maximum). Default: NULL

Usage Notes

Note the following:

- For 8i compatibility, only the first three parameters are used.
- The first three parameters are required for all types of context.
- The username parameter must be a valid SQL identifier
- The client_id parameter must be a string of at most 64 bytes. It is case-sensitive and must match the argument provided for set_identifier.
- If the namespace parameter is a global context namespace, then the username parameter is matched against the current database user name in the session, and the client_id parameter will be matched against the current client_id in the session. If these parameters are not set, NULL is assumed, enabling any user to see the context values.
- This procedure must be invoked directly or indirectly by the trusted package
- The caller of SET_CONTEXT must be in the calling stack of a procedure that has been associated to the context namespace through a CREATE CONTEXT statement. The checking of the calling stack does not cross a DBMS boundary.

- No limit applies to the number of attributes that can be set in a namespace. An attribute retains its value during the user's session unless it is reset by the user.

SET_IDENTIFIER

This procedure sets the client ID in the session.

Syntax

```
DBMS_SESSION.SET_IDENTIFIER (
   client_id VARCHAR2);
```

Parameters

Table 96–15 SET_IDENTIFIER Procedure Parameters

Parameter	Description
client_id	The application-specific identifier of the current database session.

Usage Notes

Note the following:

- SET_IDENTIFIER initializes the current session with a client identifier to identify the associated global application context
- client_id is case sensitive; it must match the client_id parameter in the set_context
- This procedure is executable by public

SET_NLS Procedure

This procedure sets up your Globalization Support (NLS). It is equivalent to the following SQL statement:

```
ALTER SESSION SET <nls_parameter> = <value>
```

Syntax

```
DBMS_SESSION.SET_NLS (
   param VARCHAR2,
   value VARCHAR2);
```

Parameters

Table 96–16 SET_NLS Procedure Parameters

Parameter	Description
`param`	Globalization Support parameter. The parameter name must begin with 'NLS'.
`value`	Parameter value. If the parameter is a text literal, then it needs embedded single-quotes. For example, "`set_nls('nls_date_format','"DD-MON-YY"')`".

SET_ROLE Procedure

This procedure enables and disables roles. It is equivalent to the `SET ROLE` SQL statement.

Syntax

```
DBMS_SESSION.SET_ROLE (
   role_cmd VARCHAR2);
```

Parameters

Table 96–17 SET_ROLE Procedure Parameters

Parameter	Description
`role_cmd`	This text is appended to "set role" and then run as SQL.

SET_SQL_TRACE Procedure

This procedure turns tracing on or off. It is equivalent to the following SQL statement:

```
ALTER SESSION SET SQL_TRACE ...
```

Syntax

```
DBMS_SESSION.SET_SQL_TRACE (
   sql_trace boolean);
```

Parameters

Table 96–18 SET_SQL_TRACE Procedure Parameters

Parameter	Description
sql_trace	TRUE turns tracing on, FALSE turns tracing off.

SWITCH_CURRENT_CONSUMER_GROUP Procedure

This procedure changes the current resource consumer group of a user's current session.

This lets you switch to a consumer group if you have the switch privilege for that particular group. If the caller is another procedure, then this enables the user to switch to a consumer group for which the owner of that procedure has switch privilege.

Syntax

```
DBMS_SESSION.switch_current_consumer_group (
   new_consumer_group     IN  VARCHAR2,
   old_consumer_group     OUT VARCHAR2,
   initial_group_on_error IN  BOOLEAN);
```

Parameters

Table 96–19 SWITCH_CURRENT_CONSUMER_GROUP Procedure Parameters

Parameter	Description
new_consumer_group	Name of consumer group to which you want to switch.
old_consumer_group	Name of the consumer group from which you just switched out.
initial_group_on_error	If TRUE, then sets the current consumer group of the caller to his/her initial consumer group in the event of an error.

Return Values

This procedure outputs the old consumer group of the user in the parameter old_consumer_group.

Note: You can switch back to the old consumer group later using the value returned in old_consumer_group.

Exceptions

Table 96–20 SWITCH_CURRENT_CONSUMER_GROUP Procedure Exceptions

Exception	Description
29368	Non-existent consumer group.
1031	Insufficient privileges.
29396	Cannot switch to OTHER_GROUPS consumer group.

Usage Notes

The owner of a procedure must have privileges on the group from which a user was switched (old_consumer_group) in order to switch them back. There is one exception: The procedure can always switch the user back to his/her initial consumer group (skipping the privilege check).

By setting initial_group_on_error to TRUE, SWITCH_CURRENT_CONSUMER_GROUP puts the current session into the default group, if it can't put it into the group designated by new_consumer_group. The error associated with the attempt to move

a session into new_consumer_group is raised, even though the current consumer group has been changed to the initial consumer group.

Examples

```
CREATE OR REPLACE PROCEDURE high_priority_task is
   old_group varchar2(30);
   prev_group varchar2(30);
   curr_user varchar2(30);
BEGIN
  -- switch invoker to privileged consumer group. If we fail to do so, an
  -- error will be thrown, but the consumer group will not change
  -- because 'initial_group_on_error' is set to FALSE

  dbms_session.switch_current_consumer_group('tkrogrp1', old_group, FALSE);
  -- set up exception handler (in the event of an error, we do not want to
  -- return to caller while leaving the session still in the privileged
  -- group)

  BEGIN
    -- perform some operations while under privileged group

  EXCEPTION
    WHEN OTHERS THEN
     -- It is possible that the procedure owner does not have privileges
     -- on old_group. 'initial_group_on_error' is set to TRUE to make sure
     -- that the user is moved out of the privileged group in such a
     -- situation

     dbms_session.switch_current_consumer_group(old_group,prev_group,TRUE);
     RAISE;
    END;

  -- we've succeeded. Now switch to old_group, or if cannot do so, switch
  -- to caller's initial consumer group

  dbms_session.switch_current_consumer_group(old_group,prev_group,TRUE);
END high_priority_task;
/
```

UNIQUE_SESSION_ID Function

This function returns an identifier that is unique for all sessions currently connected to this database. Multiple calls to this function during the same session always return the same result.

Syntax

```
DBMS_SESSION.UNIQUE_SESSION_ID
  RETURN VARCHAR2;
```

Pragmas

```
pragma restrict_references(unique_session_id,WNDS,RNDS,WNPS);
```

Return Values

Table 96–21 UNIQUE_SESSION_ID Function Return Values

Return	Description
`unique_session_id`	Returns up to 24 bytes.

97

DBMS_SHARED_POOL

The DBMS_SHARED_POOL package provides access to the shared pool, which is the shared memory area where cursors and PL/SQL objects are stored. DBMS_SHARED_POOL enables you to display the sizes of objects in the shared pool, and mark them for keeping or unkeeping in order to reduce memory fragmentation.

This chapter contains the following topics:

Using DBMS_SHARED_POOL

Overview

The procedures provided here may be useful when loading large PL/SQL objects. When large PL/SQL objects are loaded, users response time is affected because of the large number of smaller objects that need to be aged out from the shared pool to make room (due to memory fragmentation). In some cases, there may be insufficient memory to load the large objects.

DBMS_SHARED_POOL is also useful for frequently executed triggers. You may want to keep compiled triggers on frequently used tables in the shared pool.

Additionally, DBMS_SHARED_POOL supports sequences. Sequence numbers are lost when a sequence is aged out of the shared pool. DBMS_SHARED_POOL is useful for keeping sequences in the shared pool and thus preventing the loss of sequence numbers.

Operational Notes

To create DBMS_SHARED_POOL, run the DBMSPOOL.SQL script. The PRVTPOOL.PLB script is automatically executed after DBMSPOOL.SQL runs. These scripts are *not* run by CATPROC.SQL.

Summary of DBMS_SHARED_POOL Subprograms

Table 97–1 DBMS_SHARED_POOL Package Subprograms

Subprogram	Description
ABORTED_REQUEST_ THRESHOLD Procedure on page 97-6	Sets the aborted request threshold for the shared pool
KEEP Procedure on page 97-7	Keeps an object in the shared pool
SIZES Procedure on page 97-8	Shows objects in the shared pool that are larger than the specified size
UNKEEP Procedure on page 97-9	Unkeeps the named object

ABORTED_REQUEST_THRESHOLD Procedure

This procedure sets the aborted request threshold for the shared pool.

Syntax

```
DBMS_SHARED_POOL.ABORTED_REQUEST_THRESHOLD (
   threshold_size NUMBER);
```

Parameters

Table 97–2 ABORTED_REQUEST_THRESHOLD Procedure Parameters

Parameter	Description
`threshold_size`	Size, in bytes, of a request which does not try to free unpinned (not "unkeep-ed") memory within the shared pool. The range of `threshold_size` is 5000 to ~2 GB inclusive.

Exceptions

An exception is raised if the threshold is not in the valid range.

Usage Notes

Usually, if a request cannot be satisfied on the free list, then the RDBMS tries to reclaim memory by freeing objects from the LRU list and checking periodically to see if the request can be fulfilled. After finishing this step, the RDBMS has performed a near equivalent of an '`ALTER SYSTEM FLUSH SHARED_POOL`'.

Because this impacts all users on the system, this procedure "localizes" the impact to the process failing to find a piece of shared pool memory of size greater than `thresh_hold` size. This user gets the 'out of memory' error without attempting to search the LRU list.

KEEP Procedure

This procedure keeps an object in the shared pool. Once an object has been kept in the shared pool, it is not subject to aging out of the pool. This may be useful for frequently used large objects. When large objects are brought into the shared pool, several objects may need to be aged out to create a contiguous area large enough.

Syntax

```
DBMS_SHARED_POOL.KEEP (
   name VARCHAR2,
   flag CHAR      DEFAULT 'P');
```

Parameters

Table 97–3 KEEP Procedure Parameters

Parameter	Description
name	Name of the object to keep. The value for this identifier is the concatenation of the address and hash_value columns from the v$sqlarea view. This is displayed by the SIZES procedure. Currently, TABLE and VIEW objects may not be kept.
flag	(Optional) If this is not specified, then the package assumes that the first parameter is the name of a package/procedure/function and resolves the name. Set to 'P' or 'p' to fully specify that the input is the name of a package/procedure/function. Set to 'T' or 't' to specify that the input is the name of a type. Set to 'R' or 'r' to specify that the input is the name of a trigger. Set to 'Q' or 'q' to specify that the input is the name of a sequence. In case the first argument is a cursor address and hash-value, the parameter should be set to any character except 'P' or 'p' or 'Q' or 'q' or 'R' or 'r' or 'T' or 't'.

Exceptions

An exception is raised if the named object cannot be found.

Usage Notes

There are two kinds of objects:

- PL/SQL objects, triggers, sequences, and types which are specified by name
- SQL cursor objects which are specified by a two-part number (indicating a location in the shared pool).

For example:

```
DBMS_SHARED_POOL.KEEP('scott.hispackage')
```

This keeps package HISPACKAGE, owned by SCOTT. The names for PL/SQL objects follow SQL rules for naming objects (for example, delimited identifiers and multibyte names are allowed). A cursor can be kept by DBMS_SHARED_POOL.KEEP('0034CDFF, 20348871'). The complete hexadecimal address must be in the first 8 characters.

SIZES Procedure

This procedure shows objects in the `shared_pool` that are larger than the specified size. The name of the object is also given, which can be used as an argument to either the `KEEP` or `UNKEEP` calls.

Syntax

```
DBMS_SHARED_POOL.SIZES (
   minsize NUMBER);
```

Parameters

Table 97–4 SIZES Procedure Parameters

Parameter	Description
`minsize`	Size, in kilobytes, over which an object must be occupying in the shared pool, in order for it to be displayed.

Usage Notes

Issue the SQLDBA or SQLPLUS `'SET SERVEROUTPUT ON SIZE XXXXX'` command prior to using this procedure so that the results are displayed.

UNKEEP Procedure

This procedure unkeeps the named object.

Syntax

```
DBMS_SHARED_POOL.UNKEEP (
   name VARCHAR2,
   flag CHAR     DEFAULT 'P');
```

Caution: This procedure may not be supported in the future if automatic mechanisms are implemented to make this unnecessary.

Parameters

Table 97–5 UNKEEP Procedure Parameters

Parameter	Description
`name`	Name of the object to unkeep. See description of the `name` object for the `KEEP` procedure.
`flag`	See description of the flag parameter for the `KEEP` procedure.

Exceptions

An exception is raised if the named object cannot be found.

98

DBMS_SPACE

The DBMS_SPACE package enables you to analyze segment growth and space requirements.

This chapter contains the following topics:

Using DBMS_SPACE

- Security Model

Security Model

This package runs with SYS privileges. The execution privilege is granted to PUBLIC. Subprograms in this package run under the caller security. The user must have ANALYZE privilege on the object.

Data Structures

The DBMS_SPACE package defines a RECORD type and a TABLE type.

RECORD Types

ASA_RECO_ROW Record Type

TABLE Types

ASA_RECO_ROW_TB Table Type

ASA_RECO_ROW Record Type

This type contains the column type of individual columns returned by the ASA_RECOMMENDATIONS Function.

Syntax

```
TYPE asa_reco_row IS RECORD (
   tablespace_name       VARCHAR2(30),
   segment_owner         VARCHAR2(30),
   segment_name          VARCHAR2(30),
   segment_type          VARCHAR2(18),
   partition_name        VARCHAR2(30),
   allocated_space       NUMBER,
   used_space            NUMBER,
   reclaimable_space     NUMBER,
   chain_rowexcess       NUMBER,
   recommendations       VARCHAR2(1000),
   c1                    VARCHAR2(1000),
   c2                    VARCHAR2(1000),
   c3                    VARCHAR2(1000),
   task_id               NUMBER,
   mesg_id               NUMBER);
```

Attributes

Table 98–1 ASA_RECO_ROW Attributes

Field	Description
tablespace_name	Name of the tablespace containing the object
segment_owner	Name of the schema
segment_name	Name of the object
segment_type	Type of the segment 'TABLE','INDEX' and so on
partition_name	Name of the partition
allocated_space	Space allocated to the segment
used_space	Space actually used by the segment
reclaimable_space	Reclaimable free space in the segment
chain_rowexcess	Percentage of excess chain row pieces that can be eliminated
recommendations	Recommendation or finding for this segment
c1	Command associated with the recommendation
c2	Command associated with the recommendation
c3	Command associated with the recommendation
task_id	Advisor Task that processed this segment
mesg_id	Message ID corresponding to the recommendation

ASA_RECO_ROW_TB Table Type

Syntax

```
TYPE asa_reco_row_tb IS TABLE OF asa_reco_row;
```

Summary of DBMS_SPACE Subprograms

Table 98–2 DBMS_SPACE Package Subprograms

Subprogram	Description
ASA_RECOMMENDATIONS Function on page 98-8	Returns recommendations/findings of segment advisor run automatically by the system or manually invoked by the user
CREATE_INDEX_COST Procedure on page 98-9	Determines the cost of creating an index on an existing table
CREATE_TABLE_COST Procedures on page 98-10	Determines the size of the table given various attributes
FREE_BLOCKS Procedure on page 98-12	Returns information about free blocks in an object (table, index, or cluster)
OBJECT_DEPENDENT_ SEGMENTS Function on page 98-14	Returns the list of segments that are associated with the object
OBJECT_GROWTH_TREND Function on page 98-16	A table function where each row describes the space usage of the object at a specific point in time
SPACE_USAGE Procedure on page 98-18	Returns information about free blocks in an auto segment space managed segment
UNUSED_SPACE Procedure on page 98-20	Returns information about unused space in an object (table, index, or cluster)

ASA_RECOMMENDATIONS Function

This function returns returns recommendations using the stored results of the auto segment advisor. This function returns results from the latest run on any given object.

Syntax

```
DBMS_SPACE.ASA_RECOMMENDATIONS (
   all_runs        IN    VARCHAR2 DEFAULT := TRUE,
   show_manual     IN    VARCHAR2 DEFAULT := TRUE,
   show_findings   IN    VARCHAR2 DEFAULT := FALSE)
 RETURN ASA_RECO_ROW_TB PIPELINED;
```

Parameters

Table 98–3 CREATE_INDEX_COST Procedure Parameters

Parameter	Description
`all_runs`	If `TRUE`, returns recommendations/findings for all runs of auto segment advisor. If `FALSE`, returns the results of the `LATEST` run only. `LATEST` does not make sense for manual invocation of segment advisor. This is applicable only for auto advisor.
`show_manual`	If `TRUE`, we show the results of manual invocations only. The auto advisor results are excluded. If `FALSE`, results of manual invocation of segment advisor are not returned.
`show_findings`	Show only the findings instead of the recommendations

CREATE_INDEX_COST Procedure

This procedure determines the cost of creating an index on an existing table. The input is the DDL statement that will be used to create the index. The procedure will output the storage required to create the index.

Syntax

```
DBMS_SPACE.CREATE_INDEX_COST (
   ddl             IN    VARCHAR2,
   used_bytes      OUT   NUMBER,
   alloc_bytes     OUT   NUMBER,
   plan_table      IN    VARCHAR2 DEFAULT NULL);
```

Pragmas

```
pragma restrict_references(create_index_cost,WNDS);
```

Parameters

Table 98–4 CREATE_INDEX_COST Procedure Parameters

Parameter	Description
ddl	The create index DDL statement
used_bytes	The number of bytes representing the actual index data
alloc_bytes	Size of the index when created in the tablespace
plan_table	Which plan table to use, default NULL

Usage Notes

- The table on which the index is created must already exist.
- The computation of the index size depends on statistics gathered on the segment.
- It is imperative that the table must have been analyzed recently.
- In the absence of correct statistics, the results may be inaccurate, although the procedure will not raise any errors.

CREATE_TABLE_COST Procedures

This procedure is used in capacity planning to determine the size of the table given various attributes. The size of the object can vary widely based on the tablespace storage attributes, tablespace block size, and so on. There are two overloads of this procedure.

- The first version takes the column information of the table as argument and outputs the table size.
- The second version takes the average row size of the table as argument and outputs the table size.

This procedure can be used on tablespace of dictionary managed and locally managed extent management as well as manual and auto segment space management.

Syntax

```
DBMS_SPACE.CREATE_TABLE_COST (
   tablespace_name    IN VARCHAR2,
   avg_row_size       IN NUMBER,
   row_count          IN NUMBER,
   pct_free           IN NUMBER,
   used_bytes         OUT NUMBER,
   alloc_bytes        OUT NUMBER);

DBMS_SPACE.CREATE_TABLE_COST (
   tablespace_name    IN VARCHAR2,
   colinfos           IN CREATE_TABLE_COST_COLUMNS,
   row_count          IN NUMBER,
   pct_free           IN NUMBER,
   used_bytes         OUT NUMBER,
   alloc_bytes        OUT NUMBER);

CREATE TYPE create_table_cost_colinfo IS OBJECT (
   COL_TYPE   VARCHAR(200),
   COL_SIZE   NUMBER);
```

Parameters

Table 98–5 CREATE_TABLE_COST Procedure Parameters

Parameter	Description
tablespace_name	The tablespace in which the object will be created. The default is SYSTEM tablespace.
avg_row_size	The anticipated average row size in the table
colinfos	The description of the columns
row_count	The anticipated number of rows in the table
pct_free	The percentage of free space in each block for future expansion of existing rows due to updates
used_bytes	The space used by user data
alloc_bytes	The size of the object taking into account the tablespace extent characteristics

Usage Notes

- The `used_bytes` represent the actual bytes used by the data. This includes the overhead due to the block metadata, pctfree etc.
- The `alloc_bytes` represent the size of the table when it is created in the tablespace. This takes into account, the size of the extents in the tablespace and tablespace extent management properties.

Examples

```
-- review the parameters
SELECT argument_name, data_type, type_owner, type_name
FROM all_arguments
WHERE object_name = 'CREATE_TABLE_COST'
AND overload = 2

-- examine the input parameter type
SELECT text
FROM dba_source
WHERE name = 'CREATE_TABLE_COST_COLUMNS';

-- drill down further into the input parameter type
SELECT text
FROM dba_source
WHERE name = 'create_table_cost_colinfo';

set serveroutput on

DECLARE
 ub NUMBER;
 ab NUMBER;
 cl sys.create_table_cost_columns;
BEGIN
  cl := sys.create_table_cost_columns( sys.create_table_cost_colinfo('NUMBER',10),
        sys.create_table_cost_colinfo('VARCHAR2',30),
        sys.create_table_cost_colinfo('VARCHAR2',30),
        sys.create_table_cost_colinfo('DATE',NULL));

  DBMS_SPACE.CREATE_TABLE_COST('SYSTEM',cl,100000,0,ub,ab);

  DBMS_OUTPUT.PUT_LINE('Used Bytes: ' || TO_CHAR(ub));
  DBMS_OUTPUT.PUT_LINE('Alloc Bytes: ' || TO_CHAR(ab));
END;
/
```

FREE_BLOCKS Procedure

This procedure returns information about free blocks in an object (table, index, or cluster). See SPACE_USAGE Procedure for returning free block information in an auto segment space managed segment.

Syntax

```
DBMS_SPACE.FREE_BLOCKS (
   segment_owner     IN  VARCHAR2,
   segment_name      IN  VARCHAR2,
   segment_type      IN  VARCHAR2,
   freelist_group_id IN  NUMBER,
   free_blks         OUT NUMBER,
   scan_limit        IN  NUMBER DEFAULT NULL,
   partition_name    IN  VARCHAR2 DEFAULT NULL);
```

Pragmas

```
pragma restrict_references(free_blocks,WNDS);
```

Parameters

Table 98–6 FREE_BLOCKS Procedure Parameters

Parameter	Description
segment_owner	Schema name of the segment to be analyzed
segment_name	Segment name of the segment to be analyzed
segment_type	Type of the segment to be analyzed (TABLE, INDEX, or CLUSTER): ■ TABLE ■ TABLE PARTITION ■ TABLE SUBPARTITION ■ INDEX ■ INDEX PARTITION ■ INDEX SUBPARTITION ■ CLUSTER ■ LOB ■ LOB PARTITION ■ LOB SUBPARTITION
freelist_group_id	Freelist group (instance) whose free list size is to be computed
free_blks	Returns count of free blocks for the specified group
scan_limit	Maximum number of free list blocks to read (optional). Use a scan limit of X you are interested only in the question, "Do I have X blocks on the free list?"
partition_name	Partition name of the segment to be analyzed. This is only used for partitioned tables; the name of subpartition should be used when partitioning is composite.

Examples

The following uses the CLUS cluster in SCOTT schema with 4 freelist groups. It returns the number of blocks in freelist group 3 in CLUS.

```
DBMS_SPACE.FREE_BLOCKS('SCOTT', 'CLUS', 'CLUSTER', 3, :free_blocks);
```

Note: An error is raised if scan_limit is not a positive number.

OBJECT_DEPENDENT_SEGMENTS Function

This table function, given an object, returns the list of segments that are associated with the object.

Syntax

```
DBMS_SPACE.OBJECT_DEPENDENT_SEGMENTS(
   objowner     IN     VARCHAR2,
   objname      IN     VARCHAR2,
   partname     IN     VARCHAR2,
   objtype      IN     NUMBER)
  RETURN dependent_segments_table PIPELINED;
```

Parameters

Table 98–7 OBJECT_DEPENDENT_SEGMENTS Function Parameters

Parameter	Description
objowner	The schema containing the object
objname	The name of the object
partname	The name of the partition
objtype	Type of the object: ■ OBJECT_TYPE_TABLE constant positive := 1; ■ OBJECT_TYPE_NESTED_TABLE constant positive := 2; ■ OBJECT_TYPE_INDEX constant positive := 3; ■ OBJECT_TYPE_CLUSTER constant positive := 4; ■ OBJECT_TYPE_TABLE_PARTITION constant positive := 7; ■ OBJECT_TYPE_INDEX_PARTITION constant positive := 8; ■ OBJECT_TYPE_TABLE_SUBPARTITION constant positive := 9; ■ OBJECT_TYPE_INDEX_SUBPARTITION constant positive := 10; ■ OBJECT_TYPE_MV constant positive := 13; ■ OBJECT_TYPE_MVLOG constant positive := 14;

Return Values

The content of one row of a dependent_segments_table:

```
TYPE object_dependent_segment IS RECORD (
   segment_owner      VARCHAR2(100),
   segment_name       VARCHAR2(100),
   segment_type       VARCHAR2(100),
   tablespace_name    VARCHAR2(100),
   partition_name     VARCHAR2(100),
```

```
  lob_column_name    VARCHAR2(100));
```

Table 98–8 OBJECT_DEPENDENT_SEGMENT Type Parameters

Parameter	Description
segment_owner	The schema containing the segment
segment_name	The name of the segment
segmemnt_type	The type of the segment, such as table, index or LOB
tablespace_name	The name of the tablespace
partition_name	The name of the partition, if any
lob_column_name	The name of the LOB column, if any

OBJECT_GROWTH_TREND Function

This is a table function. The output will be in the form of one or more rows where each row describes the space usage of the object at a specific point in time. Either the space usage totals will be retrieved from Automatic Workload Repository Facilities (AWRF), or the current space usage will be computed and combined with space usage deltas retrieved from AWRF.

Syntax

```
DBMS_SPACE.OBJECT_GROWTH_TREND (
   object_owner           IN    VARCHAR2,
   object_name            IN    VARCHAR2,
   object_type            IN    VARCHAR2,
   partition_name         IN    VARCHAR2 DEFAULT NULL,
   start_time             IN    TIMESTAMP DEFAULT NULL,
   end_time               IN    TIMESTAMP DEFAULT NULL,
   interval               IN    DSINTERVAL_UNCONSTRAINED DEFAULT NULL,
   skip_interpolated      IN    VARCHAR2 DEFAULT 'FALSE',
   timeout_seconds        IN    NUMBER DEFAULT NULL,
   single_datapoint_flag  IN    VARCHAR2 DEFAULT 'TRUE')
 RETURN object_growth_trend_table PIPELINED;
```

Parameters

Table 98–9 OBJECT_GROWTH_TREND Function Parameters

Parameter	Description
`object_owner`	The schema containing the object
`object_name`	The name of the object
`object_type`	The type of the object
`partition_name`	The name of the partition
`start_time`	Statistics generated after this time will be used in generating the growth trend
`end_time`	Statistics generated until this time will be used in generating the growth trend
`interval`	The interval at which to sample
`skip_interpolated`	Whether interpolation of missing values should be skipped
`timeout_seconds`	The time-out value for the function in seconds
`single_data_point_flag`	Whether in the absence of statistics the segment should be sampled

Return Values

The `object_growth_trend_row` and `object_growth_trend_table` are used by the `OBJECT_GROWTH_TREND` table function to describe its output.

```
TYPE object_growth_trend_row IS RECORD(
   timepoint      TIMESTAMP,
   space_usage    NUMBER,
   space_alloc    NUMBER,
   quality        VARCHAR(20));
```

Table 98–10 OBJECT_GROWTH_TREND_ROW Type Parameters

Parameter	Description
timepoint	The time at which the statistic was recorded
space_usage	The space used by data
space_alloc	The size of the segment including overhead and unused space
quality	The quality of result: "GOOD", "INTERPOLATED", "PROJECTION"

```
TYPE object_growth_trend_table IS TABLE OF object_growth_trend_row;
```

SPACE_USAGE Procedure

This procedure shows the space usage of data blocks under the segment High Water Mark. You can calculate usage for LOBS, LOB PARTITIONS and LOB SUBPARTITIONS. This procedure can only be used on tablespaces that are created with auto segment space management.The bitmap blocks, segment header, and extent map blocks are not accounted for by this procedure.

Syntax

```
DBMS_SPACE.SPACE_USAGE(
   segment_owner      IN  VARCHAR2,
   segment_name       IN  VARCHAR2,
   segment_type       IN  VARCHAR2,
   unformatted_blocks OUT NUMBER,
   unformatted_bytes  OUT NUMBER,
   fs1_blocks         OUT NUMBER,
   fs1_bytes          OUT NUMBER,
   fs2_blocks         OUT NUMBER,
   fs2_bytes          OUT NUMBER,
   fs3_blocks         OUT NUMBER,
   fs3_bytes          OUT NUMBER,
   fs4_blocks         OUT NUMBER,
   fs4_bytes          OUT NUMBER,
   full_blocks        OUT NUMBER,
   full_bytes         OUT NUMBER,
   partition_name     IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 98–11 SPACE_USAGE Procedure Parameters

Parameter	Description
segment_owner	Schema name of the segment to be analyzed
segment_name	Name of the segment to be analyzed
partition_name	Partition name of the segment to be analyzed
segment_type	Type of the segment to be analyzed (TABLE, INDEX, or CLUSTER)
unformatted_blocks	Total number of blocks that are unformatted
unformatted bytes	Total number of bytes that are unformatted
fs1_blocks	Number of blocks that has at least 0 to 25% free space
fs1_bytes	Number of bytes that has at least 0 to 25% free space
fs2_blocks	Number of blocks that has at least 25 to 50% free space
fs2_bytes	Number of bytes that has at least 25 to 50% free space
fs3_blocks	Number of blocks that has at least 50 to 75% free space
fs3_bytes	Number of bytes that has at least 50 to 75% free space
fs4_blocks	Number of blocks that has at least 75 to 100% free space
fs4_bytes	Number of bytes that has at least 75 to 100% free space
ful1_blocks	Total number of blocks that are full in the segment
full_bytes	Total number of bytes that are full in the segment

Examples

```
variable unf number;
variable unfb number;
variable fs1 number;
variable fs1b number;
variable fs2 number;
variable fs2b number;
variable fs3 number;
variable fs3b number;
variable fs4 number;
variable fs4b number;
variable full number;
variable fullb number;

begin
dbms_space.space_usage('U1','T',
                       'TABLE',
                       :unf, :unfb,
                       :fs1, :fs1b,
                       :fs2, :fs2b,
                       :fs3, :fs3b,
                       :fs4, :fs4b,
                       :full, :fullb);
end;
/
print unf ;
print unfb ;
print fs4 ;
print fs4b;
print fs3 ;
print fs3b;
print fs2 ;
print fs2b;
print fs1 ;
print fs1b;
print full;
print fullb;
```

UNUSED_SPACE Procedure

This procedure returns information about unused space in an object (table, index, or cluster).

Syntax

```
DBMS_SPACE.UNUSED_SPACE (
   segment_owner              IN  VARCHAR2,
   segment_name               IN  VARCHAR2,
   segment_type               IN  VARCHAR2,
   total_blocks               OUT NUMBER,
   total_bytes                OUT NUMBER,
   unused_blocks              OUT NUMBER,
   unused_bytes               OUT NUMBER,
   last_used_extent_file_id   OUT NUMBER,
   last_used_extent_block_id  OUT NUMBER,
   last_used_block            OUT NUMBER,
   partition_name             IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 98–12 UNUSED_SPACE Procedure Parameters

Parameter	Description
segment_owner	Schema name of the segment to be analyzed.
segment_name	Segment name of the segment to be analyzed.
segment_type	Type of the segment to be analyzed (TABLE, INDEX, or CLUSTER): ■ TABLE ■ TABLE PARTITION ■ TABLE SUBPARTITION ■ INDEX ■ INDEX PARTITION ■ INDEX SUBPARTITION ■ CLUSTER ■ LOB ■ LOB PARTITION ■ LOB SUBPARTITION
total_blocks	Returns total number of blocks in the segment.
total_bytes	Returns total number of blocks in the segment, in bytes.
unused_blocks	Returns number of blocks which are not used.
unused_bytes	Returns, in bytes, number of blocks which are not used.
last_used_extent_file_id	Returns the file ID of the last extent which contains data.
last_used_extent_block_id	Returns the starting block ID of the last extent which contains data.
last_used_block	Returns the last block within this extent which contains data.

Table 98–12 (Cont.) UNUSED_SPACE Procedure Parameters

Parameter	Description
partition_name	Partition name of the segment to be analyzed. This is only used for partitioned tables; the name of subpartition should be used when partitioning is compose.

Examples

The following declares the necessary bind variables and executes.

```
DBMS_SPACE.UNUSED_SPACE('SCOTT', 'EMP', 'TABLE', :total_blocks,
   :total_bytes,:unused_blocks, :unused_bytes, :lastextf,
   :last_extb, :lastusedblock);
```

99

DBMS_SPACE_ADMIN

The DBMS_SPACE_ADMIN package provides functionality for locally managed tablespaces.

> **See Also:** *Oracle Database Administrator's Guide* for an example and description of using DBMS_SPACE_ADMIN.

This chapter contains the following topics:

Using DBMS_SPACE_ADMIN

This section contains topics which relate to using the `DBMS_SPACE_ADMIN` package.

Security Model

This package runs with SYS privileges; therefore, any user who has privilege to execute the package can manipulate the bitmaps.

Constants

Table 99–1 DBMS_SPACE_ADMIN Constants

Constant	Type	Value	Description
SEGMENT_VERIFY_EXTENTS	POSITIVE	1	Verifies that the space owned by segment is appropriately reflected in the bitmap as used
SEGMENT_VERIFY_EXTENTS_GLOBAL	POSITIVE	2	Verifies that the space owned by segment is appropriately reflected in the bitmap as used and that no other segment claims any of this space to be used by it
SEGMENT_MARK_CORRUPT	POSITIVE	3	Marks a temporary segment as corrupt whereby facilitating its elimination from the dictionary (without space reclamation).
SEGMENT_MARK_VALID	POSITIVE	4	Marks a corrupt temporary segment as valid. It is useful when the corruption in the segment extent map or elsewhere has been resolved and the segment can be dropped normally.
SEGMENT_DUMP_EXTENT_MAP	POSITIVE	5	Dumps the extent map for a given segment
TABLESPACE_VERIFY_BITMAP	POSITIVE	6	Verifies the bitmap of the tablespace with extent maps of the segments in that tablespace to make sure everything is consistent
TABLESPACE_EXTENT_MAKE_FREE	POSITIVE	7	Marks the DBA range (extent) as free in the bitmaps
TABLESPACE_EXTENT_MAKE_USED	POSITIVE	8	Marks the DBA range (extent) as used in the bitmaps
SEGMENT_VERIFY_BASIC	POSITIVE	9	Performs the basic metadata checks
SEGMENT_VERIFY_DEEP	POSITIVE	10	Performs deep verification
SEGMENT_VERIFY_SPECIFIC	POSITIVE	11	Performs a specific check for the segment
HWM_CHECK	POSITIVE	12	Checks HWM
BMB_CHECK	POSITIVE	13	Checks integrity among L1, L2 and L3 BMBs
SEG_DICT_CHECK	POSITIVE	14	Checks consistency of segment header with corresponding SEG entry
EXTENT_TS_BITMAP_CHECK	POSITIVE	15	Checks whether the tablespace bitmaps corresponding to the extent map are marked used
DB_BACKPOINTER_CHECK	POSITIVE	16	Checks whether the L1 BMBs, L2 BMBs, L3 BMBs and data blocks point to the same parent segment
EXTENT_SEGMENT_BITMAP_CHECK	POSITIVE	17	Checks whether the bitmap blocks are consistent with the extent map

Table 99–1 (Cont.) DBMS_SPACE_ADMIN Constants

Constant	Type	Value	Description
BITMAPS_CHECK	POSITIVE	18	Checks from the datablocks that the bitmap states representing the blocks are consistent
TS_VERIFY_BITMAPS	POSITIVE	19	Checks whether the tablespace bitmaps are consistent with the extents belonging to that tablespace
TS_VERIFY_DEEP	POSITIVE	20	Performs TS_VERIFY_BITMAPS and TS_VERIFY_SEGMENTS with DEEP option
TS_VERIFY_SEGMENTS	POSITIVE	21	Performs ASSM_SEGMENT_ VERIFY on all segments in the tablespace. will take either the basic or the deep option
SEGMENTS_DUMP_ BITMAP_SUMMARY	POSITIVE	27	Dumps only bitmap block summaries

Operational Notes

Before migrating the SYSTEM tablespace, the following conditions must be met. These conditions are enforced by the TABLESPACE_MIGRATE_TO_LOCAL procedure, except for the cold backup.

- The database must have a default temporary tablespace that is not SYSTEM.
- Dictionary-managed tablespaces cannot have any rollback segments.
- A locally managed tablespace must have at least one online rollback segment. If you are using automatic undo management, an undo tablespace must be online.
- All tablespaces—except the tablespace containing the rollback segment or the undo tablespace—must be read-only.
- You must have a cold backup of the database.
- The system must be in restricted mode.

Summary of DBMS_SPACE_ADMIN Subprograms

Table 99–2 DBMS_SPACE_ADMIN Package Subprograms

Subprogram	Description
ASSM_SEGMENT_VERIFY Procedure on page 99-8	Verifies segments created in ASSM tablespaces
ASSM_TABLESPACE_VERIFY Procedure on page 99-9	Verifies ASSM tablespaces
SEGMENT_CORRUPT Procedure on page 99-10	Marks the segment corrupt or valid so that appropriate error recovery can be done
SEGMENT_DROP_CORRUPT Procedure on page 99-11	Drops a segment currently marked corrupt (without reclaiming space)
SEGMENT_DUMP Procedure on page 99-12	Dumps the segment header and extent maps of a given segment
SEGMENT_VERIFY Procedure on page 99-13	Verifies the consistency of the extent map of the segment
TABLESPACE_FIX_BITMAPS Procedure on page 99-14	Marks the appropriate DBA range (extent) as free or used in bitmap
TABLESPACE_FIX_SEGMENT_STATES Procedure on page 99-15	Fixes the state of the segments in a tablespace in which migration was aborted
TABLESPACE_MIGRATE_FROM_LOCAL Procedure on page 99-16	Migrates a locally-managed tablespace to dictionary-managed tablespace
TABLESPACE_MIGRATE_TO_LOCAL Procedure on page 99-17	Migrates a tablespace from dictionary managed format to locally managed format
TABLESPACE_REBUILD_BITMAPS Procedure on page 99-18	Rebuilds the appropriate bitmaps
TABLESPACE_REBUILD_QUOTAS Procedure on page 99-19	Rebuilds quotas for given tablespace
TABLESPACE_RELOCATE_BITMAPS Procedure on page 99-20	Relocates the bitmaps to the destination specified
TABLESPACE_VERIFY Procedure on page 99-21	Verifies that the bitmaps and extent maps for the segments in the tablespace are in sync

ASSM_SEGMENT_VERIFY Procedure

Given a segment definition, the procedure verifies the basic consistency of the space metadata blocks as well as consistency between space metadata and segment data blocks. This procedure verifies segments created in ASSM (Automatic Segment Space Management) tablespaces.

There is however a difference between basic verification and deep verification:

- Basic verification involves consistency checks of space metadata, such as integrity among level 1, level 2, level 3 bitmap blocks, consistency of segment extent map and level 1 bitmap ranges.
- Deep verification involves consistency checks between datablocks and space metadata blocks such as whether the datablocks point correctly to the parent level 1 bitmap blocks, whether the freeness states in the datablocks are consistent with the freeness states of bits in level 1 bitmap blocks corresponding to the datablocks

Syntax

```
DBMS_SPACE_ADMIN.ASSM_SEGMENT_VERIFY (
   segment_owner   IN VARCHAR2,
   segment_name    IN VARCHAR2,
   segment_type    IN VARCHAR2,
   partition_name  IN VARCHAR2,
   verify_option   IN POSITIVE  DEFAULT SEGMENT_VERIFY_BASIC,
   attrib          IN POSITIVE  DEFAULT NULL);
```

Parameters

Table 99–3 ASSM_SEGMENT_VERIFY Procedure Parameters

Parameter	Description
segment_owner	The schema that owns the segment
segment_name	The name of the segment to be verified
segment_type	The segment namespace is one of TABLE, TABLE PARTITION, TABLE SUBPARTITION, INDEX, INDEX PARTITION, INDEX SUBPARTITION, LOB, LOB PARTITION, LOB SUBPARTITION, CLUSTER
partition_name	Name of the partition or subpartition
verify_option	■ SEGMENT_VERIFY_DEEP: Perform deep verification ■ SEGMENT_VERIFY_BASIC: Perform the basic metadata checks (Default)
attrib	Used when option SEGMENT_VERIFY_SPECIFIC

Usage Notes

- Using this procedure requires SYSDBA privileges to execute.
- You can determine the relative file # and header block # (header_relative_file and header_block parameters) by querying DBA_SEGMENTS.
- This procedure outputs a dump file named sid_ora_*process_ID*.trc to the location specified in the USER_DUMP_DEST initialization parameter.

ASSM_TABLESPACE_VERIFY Procedure

This procedures verifies all the segments created in an ASSM tablespace. The verification per segment is either basic consistency checks of the space metadata blocks as well as consistency checks between space metadata and segment data blocks.

Syntax

```
DBMS_SPACE_ADMIN.ASSM_TABLESPACE_VERIFY (
   tablespace_name   IN VARCHAR2,
   ts_option         IN POSITIVE,
   segment_option    IN POSITIVE DEFAULT NULL);
```

Parameters

Table 99–4 ASSM_TABLESPACE_VERIFY Procedure Parameters

Parameter	Description
tablespace_name	Name of the tablespace to verify, tablespace should be ASSM
ts_option	■ TABLESPACE_VERIFY_BITMAPS: The bitmaps are verified against the extents. This will detect bits that are marked used or free wrongly and will also detect multiple allocation of extents. The file metadata will be validated against file$ and control file. This is the default option (1). ■ TABLESPACE_VERIFY_DEEP: This option is used to verify the file bitmaps as well perform checks on all the segments (2). ■ TABLESPACE_VERIFY_SEGMENTS: This option is used to invoke SEGMENT_VERIFY on all the segments in the tablespace. Optionally the user can write a script that queries all the segments in the tablespace and invoke SEGMENT_VERIFY (3).
segment_option	This is same as OPTION specified for SEGMENT_VERIFY procedure. When the TABLESPACE_VERIFY_SEGMENTS or TABLESPACE_VERIFY_DEEP is chosen, the SEGMENT_OPTION can be specified optionally. The only valid segment options applicable are SEGMENT_VERIFY_DEEP and SEGMENT_VERIFY_BASIC.

Usage Notes

Using this procedure requires SYSDBA privileges to execute.

This procedure outputs a dump file named sid_ora_*process_ID*.trc to the location specified in the USER_DUMP_DEST initialization parameter.

SEGMENT_CORRUPT Procedure

This procedure marks the segment corrupt or valid so that appropriate error recovery can be done. It cannot be used on the SYSTEM tablespace.

Syntax

```
DBMS_SPACE_ADMIN.SEGMENT_CORRUPT (
   tablespace_name         IN    VARCHAR2,
   header_relative_file    IN    POSITIVE,
   header_block            IN    POSITIVE,
   corrupt_option          IN    POSITIVE  DEFAULT SEGMENT_MARK_CORRUPT);
```

Parameters

Table 99–5 SEGMENT_CORRUPT Procedure Parameters

Parameter	Description
tablespace_name	Name of tablespace in which segment resides.
header_relative_file	Relative file number of segment header.
header_block	Block number of segment header.
corrupt_option	SEGMENT_MARK_CORRUPT (default) or SEGMENT_MARK_VALID.

Examples

The following example marks the segment as corrupt:

```
EXECUTE DBMS_SPACE_ADMIN.SEGMENT_CORRUPT('USERS', 4, 33, 3);
```

Alternately, the next example marks a corrupt segment valid:

```
EXECUTE DBMS_SPACE_ADMIN.SEGMENT_CORRUPT('USERS', 4, 33, 4);
```

SEGMENT_DROP_CORRUPT Procedure

This procedure drops a segment currently marked corrupt (without reclaiming space). For this to work, the segment should have been marked *temporary*. To mark a corrupt segment as temporary, issue a `DROP` command on the segment.

Syntax

```
DBMS_SPACE_ADMIN.SEGMENT_DROP_CORRUPT (
   tablespace_name         IN    VARCHAR2,
   header_relative_file    IN    POSITIVE,
   header_block            IN    POSITIVE);
```

Parameters

Table 99–6 SEGMENT_DROP_CORRUPT Procedure Parameters

Parameter	Description
`tablespace_name`	Name of tablespace in which segment resides.
`header_relative_file`	Relative file number of segment header.
`header_block`	Block number of segment header.

Usage Notes

The procedure cannot be used on the `SYSTEM` tablespace.

The space for the segment is not released, and it must be fixed by using the TABLESPACE_FIX_BITMAPS Procedure or the TABLESPACE_REBUILD_BITMAPS Procedure.

Examples

```
EXECUTE DBMS_SPACE_ADMIN.SEGMENT_DROP_CORRUPT('USERS', 4, 33);
```

SEGMENT_DUMP Procedure

This procedure dumps the segment header and bitmap blocks of a specific segment to the location specified in the USER_DUMP_DEST initialization parameter.

Syntax

```
DBMS_SPACE_ADMIN.SEGMENT_DUMP (
   tablespace_name         IN    VARCHAR2,
   header_relative_file    IN    POSITIVE,
   header_block            IN    POSITIVE,
   dump_option             IN    POSITIVE  DEFAULT SEGMENT_DUMP_EXTENT_MAP);
```

Parameters

Table 99–7 SEGMENT_DUMP Procedure Parameters

Parameter	Description
tablespace_name	Name of tablespace in which segment resides.
header_relative_file	Relative file number of segment header.
header_block	Block number of segment header.
dump_option	SEGMENT_DUMP_EXTENT_MAP.
	SEGMENT_DUMP_BITMAP_SUMMARY.

Usage Notes

- You can produce a slightly abbreviated dump, which includes the segment header and bitmap block summaries, without percent-free states of each block if you pass SEGMENT_DUMP_BITMAP_SUMMARY as the dump_option parameter.
- You can determine the relative file # and header block # (header_relative_file and header_block parameters) by querying DBA_SEGMENTS.

Examples

```
EXECUTE DBMS_SPACE_ADMIN.SEGMENT_DUMP('USERS', 4, 33);
```

SEGMENT_VERIFY Procedure

This procedure checks the consistency of the segment extent map with the tablespace file bitmaps.

Syntax

```
DBMS_SPACE_ADMIN.SEGMENT_VERIFY (
   tablespace_name         IN    VARCHAR2,
   header_relative_file    IN    POSITIVE,
   header_block            IN    POSITIVE,
   verify_option           IN    POSITIVE  DEFAULT SEGMENT_VERIFY_EXTENTS);
```

Parameters

Table 99–8 SEGMENT_VERIFY Procedure Parameters

Parameters	Description
tablespace_name	Name of tablespace in which segment resides.
header_relative_file	Relative file number of segment header.
header_block	Block number of segment header.
verify_option	What kind of check to do: SEGMENT_VERIFY_EXTENTS or SEGMENT_VERIFY_EXTENTS_GLOBAL.

Usage Notes

Anomalies are output as dba-range, bitmap-block, bitmap-block-range, anomaly-information, in the trace file for all dba-ranges found to have incorrect space representation. The kinds of problems which would be reported are free space not considered free, used space considered free, and the same space considered used by multiple segments.

Examples

The following example verifies that the segment with segment header at relative file number 4, block number 33, has its extent maps and bitmaps in sync.

```
EXECUTE DBMS_SPACE_ADMIN.SEGMENT_VERIFY('USERS', 4, 33, 1);
```

> **Note:** All DBMS_SPACE_ADMIN package examples use the tablespace USERS which contains SCOTT.EMP.

TABLESPACE_FIX_BITMAPS Procedure

This procedure marks the appropriate DBA range (extent) as free or used in bitmap. It cannot be used on the SYSTEM tablespace.

Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_FIX_BITMAPS (
   tablespace_name         IN    VARCHAR2,
   dbarange_relative_file  IN    POSITIVE,
   dbarange_begin_block    IN    POSITIVE,
   dbarange_end_block      IN    POSITIVE,
   fix_option              IN    POSITIVE);
```

Parameters

Table 99–9 TABLESPACE_FIX_BITMAPS Procedure Parameters

Parameter	Description
tablespace_name	Name of tablespace.
dbarange_relative_file	Relative file number of DBA range (extent).
dbarange_begin_block	Block number of beginning of extent.
dbarange_end_block	Block number (inclusive) of end of extent.
fix_option	TABLESPACE_EXTENT_MAKE_FREE or TABLESPACE_EXTENT_MAKE_USED.

Examples

The following example marks bits for 51 blocks for relative file number 4, beginning at block number 33 and ending at 83, as USED in bitmaps.

```
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_FIX_BITMAPS('USERS', 4, 33, 83, 7);
```

Alternately, specifying an option of 8 marks the bits FREE in bitmaps. The BEGIN and END blocks should be in extent boundary and should be extent multiple. Otherwise, an error is raised.

TABLESPACE_FIX_SEGMENT_STATES Procedure

Use this procedure to fix the state of the segments in a tablespace in which migration was aborted. During tablespace migration to or from local, the segments are put in a transient state. If migration is aborted, the segment states are corrected by SMON when event 10906 is set. Database with segments in such a transient state cannot be downgraded. The procedure can be used to fix the state of such segments.

Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_FIX_SEGMENT_STATES (
   tablespace_name     IN    VARCHAR);
```

Parameters

Table 99–10 TABLESPACE_FIX_SEGMENT_STATES Procedure Parameters

Parameter Name	Purpose
tablespace_name	Name of the tablespace whose segments need to be fixed.

Usage Notes

The tablespace must be kept online and read/write when this procedure is called.

Examples

```
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_FIX_SEGMENT_STATES('TS1')
```

TABLESPACE_MIGRATE_FROM_LOCAL Procedure

This procedure migrates a locally-managed tablespace to a dictionary-managed tablespace. You cannot use this procedure for SYSTEM tablespace.

Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_FROM_LOCAL (
   tablespace_name         IN    VARCHAR2);
```

Parameter

Table 99–11 TABLESPACE_MIGRATE_FROM_LOCAL Procedure Parameter

Parameter	Description
tablespace_name	Name of tablespace.

Usage Notes

The tablespace must be kept online and read/write during migration. Migration of temporary tablespaces and migration of SYSTEM tablespaces are not supported.

Examples

```
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_FROM_LOCAL('USERS');
```

TABLESPACE_MIGRATE_TO_LOCAL Procedure

Use this procedure to migrate the tablespace from a dictionary-managed format to a locally managed format. Tablespaces migrated to locally managed format are user managed.

Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_TO_LOCAL (
   tablespace_name     IN     VARCHAR,
   unit_size           IN     POSITIVE DEFAULT NULL,
   rfno                IN     INTGER DEFAULT NULL);
```

Parameters

Table 99–12 TABLESPACE_MIGRATE_TO_LOCAL Procedure Parameters

Parameter Name	Purpose
`tablespace_name`	Name of the tablespace to be migrated.
`unit_size`	Unit size (which is the size of the smallest possible chunk of space that can be allocated) in the tablespace.
`rfno`	Relative File Number of the file where the bitmap blocks should be placed (optional).

Usage Notes

Before you migrate the `SYSTEM` tablespace, you should migrate any dictionary-managed tablespaces that you may want to use in read/write mode to locally managed. After the `SYSTEM` tablespace is migrated, you cannot change dictionary-managed tablespaces to read/write.

See Also: *Oracle Database Administrator's Guide*

The tablespace must be kept online and read/write during migration. Note that temporary tablespaces cannot be migrated.

Allocation Unit may be specified optionally. The default is calculated by the system based on the highest common divisor of all extents (used or free) for the tablespace. This number is further trimmed based on the `MINIMUM EXTENT` for the tablespace (5 if `MINIMUM EXTENT`T is not specified). Thus, the calculated value will not be larger than the `MINIMUM EXTENT` for the tablespace. The last free extent in every file will be ignored for GCD calculation. If you specify the unit size, it has to be a factor of the UNIT size calculated by the system, otherwise an error message is returned.

The Relative File Number parameter is used to place the bitmaps in a desired file. If space is not found in the file, an error is issued. The data file specified should be part of the tablespace being migrated. If the dataflow is not specified then the system will choose a dataflow in which to place the initial bitmap blocks. If space is not found for the initial bitmaps, an error will be raised.

Examples

To migrate a tablespace 'TS1' with minimum extent size 1m, use

```
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_TO_LOCAL('TS1', 512, 2);
```

The bitmaps will be placed in file with relative file number 2.

TABLESPACE_REBUILD_BITMAPS Procedure

This procedure rebuilds the appropriate bitmaps. If no bitmap block DBA is specified, then it rebuilds all bitmaps for the given tablespace.

The procedure cannot be used on the SYSTEM tablespace.

Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_REBUILD_BITMAPS (
   tablespace_name         IN    VARCHAR2,
   bitmap_relative_file    IN    POSITIVE   DEFAULT NULL,
   bitmap_block            IN    POSITIVE   DEFAULT NULL);
```

Parameters

Table 99–13 TABLESPACE_REBUILD_BITMAPS Procedure Parameters

Parameter	Description
tablespace_name	Name of tablespace.
bitmap_relative_file	Relative file number of bitmap block to rebuild.
bitmap_block	Block number of bitmap block to rebuild.

Usage Notes

Note: Only full rebuild is supported.

Examples

The following example rebuilds bitmaps for all the files in the USERS tablespace.

```
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_REBUILD_BITMAPS('USERS');
```

TABLESPACE_REBUILD_QUOTAS Procedure

This procedure rebuilds quotas for the given tablespace.

Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_REBUILD_QUOTAS (
   tablespace_name         IN    VARCHAR2);
```

Parameters

Table 99–14 TABLESPACE_REBUILD_QUOTAS Procedure Parameters

Parameter	Description
tablespace_name	Name of tablespace.

Examples

```
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_REBUILD_QUOTAS('USERS');
```

TABLESPACE_RELOCATE_BITMAPS Procedure

Use this procedure to relocate the bitmaps to the destination specified.

Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_RELOCATE_BITMAPS (
   tablespace_name    IN      VARCHAR2,
   filno              IN      POSITIVE,
   blkno              IN      POSITIVE);
```

Parameters

Table 99–15 TABLESPACE_RELOCATE_BITMAPS Procedure Parameters

Parameter Name	Purpose
tablespace_name	Name of Tablespace.
filno	Relative File Number of the destination file.
blkno	Block Number of the destination dba.

Usage Notes

Migration of a tablespace from dictionary managed to locally managed format could result in the creation of SPACE HEADER segment that contains the bitmap blocks. The SPACE HEADER segment is treated as user data. If the user wishes to explicitly resize a file at or below the space header segment, an error is issued. Use the TABLESPACE_RELOCATE_BITMAPS command to move the control information to a different destination and then resize the file.

This procedure cannot be used on the SYSTEM tablespace.

The tablespace must be kept online and read/write during relocation of bitmaps. This can be done only on migrated locally managed tablespaces.

Examples

```
EXECUTE  DBMS_SPACE_ADMIN.TABLESPACE_RELOCATE_BITMAPS('TS1', 3, 4);
```

Moves the bitmaps to file 3, block 4.

> **Note:** The source and the destination addresses should not overlap. The destination block number is rounded down to the unit boundary. If there is user data in that location an error is raised.

TABLESPACE_VERIFY Procedure

This procedure verifies that the bitmaps and extent maps for the segments in the tablespace are in sync.

Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_VERIFY (
   tablespace_name         IN    VARCHAR2,
   verify_option           IN    POSITIVE DEFAULT TABLESPACE_VERIFY_BITMAP);
```

Parameters

Table 99–16 TABLESPACE_VERIFY Procedure Parameters

Parameter	Description
tablespace_name	Name of tablespace.
verify_option	TABLESPACE_VERIFY_BITMAP.

Examples

```
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_VERIFY('USERS');
```

100

DBMS_SQL

The DBMS_SQL package provides an interface to use dynamic SQL to parse any data manipulation language (DML) or data definition language (DDL) statement using PL/SQL. For example, you can enter a DROP TABLE statement from within a stored procedure by using the PARSE procedure supplied with the DBMS_SQL package.

See Also:
- For more information on native dynamic SQL, see *Oracle Database PL/SQL User's Guide and Reference.*
- For a comparison of DBMS_SQL and native dynamic SQL, see *Oracle Database Application Developer's Guide - Fundamentals.*

This chapter contains the following topics:

Using DBMS_SQL

Overview

Oracle lets you to write stored procedures and anonymous PL/SQL blocks that use dynamic SQL. Dynamic SQL statements are not embedded in your source program; rather, they are stored in character strings that are input to, or built by, the program at runtime. This enables you to create more general-purpose procedures. For example, dynamic SQL lets you create a procedure that operates on a table whose name is not known until runtime.

Native Dynamic SQL is an alternative to `DBMS_SQL` that lets you place dynamic SQL statements directly into PL/SQL blocks. In most situations, Native Dynamic SQL is easier to use and performs better than `DBMS_SQL`. However, Native Dynamic SQL itself has certain limitations:

- There is no support for so-called Method 4 (for dynamic SQL statements with an unknown number of inputs or outputs)
- There is no support for SQL statements larger than 32K bytes

Also, there are some tasks that can only be performed using `DBMS_SQL`.

The ability to use dynamic SQL from within stored procedures generally follows the model of the Oracle Call Interface (OCI).

See Also: *Oracle Call Interface Programmer's Guide*

PL/SQL differs somewhat from other common programming languages, such as C. For example, addresses (also called pointers) are not user-visible in PL/SQL. As a result, there are some differences between the Oracle Call Interface and the `DBMS_SQL` package. These differences include the following:

- The OCI uses bind by address, while the `DBMS_SQL` package uses bind by value.
- With `DBMS_SQL` you must call `VARIABLE_VALUE` to retrieve the value of an `OUT` parameter for an anonymous block, and you must call `COLUMN_VALUE` after fetching rows to actually retrieve the values of the columns in the rows into your program.
- The current release of the `DBMS_SQL` package does not provide `CANCEL` cursor procedures.
- Indicator variables are not required, because `NULL`s are fully supported as values of a PL/SQL variable.

A sample usage of the `DBMS_SQL` package follows. For users of the Oracle Call Interfaces, this code should seem fairly straightforward.

Security Model

DBMS_SQL is compiled with AUTHID CURRENT_USER.

Any DBMS_SQL subprograms called from an anonymous PL/SQL block are run using the privileges of the current user.

> **See Also:** For more information about invoking subprograms using either Invoker or Definer Rights, see *Oracle Database PL/SQL User's Guide and Reference*

Constants

```
v6 constant INTEGER     := 0;
native constant INTEGER := 1;
v7 constant INTEGER     := 2;
```

Types

General Types

- DESC_REC, DESC_TAB
- VARCHAR2A, DESC_REC2
- VARCHAR2_TABLE

Bulk SQL Types

- BFILE_TABLE
- BINARY_DOUBLE_TABLE
- BLOB_TABLE
- CLOB_TABLE
- DATE_TABLE
- INTERVAL_DAY_TO_SECOND_TABLE
- INTERVAL_YEAR_TO_MONTH_TABLE
- NUMBER_TABLE
- TIME_TABLE
- TIME_WITH_TIME_ZONE_TABLE
- TIMESTAMP_TABLE
- TIMESTAMP_WITH_LTZ_TABLE
- UROWID_TABLE
- VARCHAR2_TABLE

BFILE_TABLE

```
TYPE bfile_table IS TABLE OF BFILE INDEX BY BINARY_INTEGER;
```

BINARY_DOUBLE_TABLE

```
TYPE binary_double_table IS TABLE OF BINARY_DOUBLE  INDEX BY BINARY_INTEGER;
```

BINARY_FLOAT_TABLE

```
TYPE binary_float_table IS TABLE OF BINARY_FLOAT INDEX BY BINARY_INTEGER;
```

BLOB_TABLE

```
TYPE blob_table IS TABLE OF BLOB INDEX BY BINARY_INTEGER;
```

CLOB_TABLE

```
TYPE clob_table IS TABLE OF CLOB INDEX BY BINARY_INTEGER;
```

DATE_TABLE

```
type date_table IS TABLE OF DATE INDEX BY BINARY_INTEGER;
```

INTERVAL_DAY_TO_SECOND_TABLE

```
 TYPE interval_day_to_second_Table IS TABLE OF
```

```
    dsinterval_unconstrained INDEX BY binary_integer;
```

INTERVAL_YEAR_TO_MONTH_TABLE

```
TYPE interval_year_to_month_table IS TABLE OF yminterval_unconstrained INDEX BY
BINARY_INTEGER;
```

DESC_REC, DESC_TAB

```
TYPE desc_rec IS RECORD (
      col_type            BINARY_INTEGER := 0,
      col_max_len         BINARY_INTEGER := 0,
      col_name            VARCHAR2(32)   := '',
      col_name_len        BINARY_INTEGER := 0,
      col_schema_name     VARCHAR2(32)   := '',
      col_schema_name_len BINARY_INTEGER := 0,
      col_precision       BINARY_INTEGER := 0,
      col_scale           BINARY_INTEGER := 0,
      col_charsetid       BINARY_INTEGER := 0,
      col_charsetform     BINARY_INTEGER := 0,
      col_null_ok         BOOLEAN        := TRUE);
TYPE desc_tab IS TABLE OF desc_rec INDEX BY BINARY_INTEGER;
```

NUMBER_TABLE

```
TYPE number_table IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;
```

TIME_TABLE

```
TYPE time_table IS TABLE OF time_unconstrained INDEX BY BINARY_INTEGER;
```

TIME_WITH_TIME_ZONE_TABLE

```
TYPE time_with_time_zone_table IS TABLE OF TIME_TZ_UNCONSTRAINED INDEX
BY BINARY_INTEGER;
```

TIMESTAMP_TABLE

```
TYPE timestamp_table IS TABLE OF timestamp_unconstrained INDEX BY BINARY_INTEGER;
```

TIMESTAMP_WITH_LTZ_TABLE

```
TYPE timestamp_with_ltz_table IS TABLE OF
    TIMESTAMP_LTZ_UNCONSTRAINED INDEX BY binary_integer;
```

UROWID_TABLE

```
TYPE urowid_table IS TABLE OF UROWID INDEX BY BINARY_INTEGER;
```

VARCHAR2_TABLE

```
TYPE varchar2_table IS TABLE OF VARCHAR2(2000) INDEX BY BINARY_INTEGER;
```

VARCHAR2A, DESC_REC2

```
TYPE varchar2a IS TABLE OF VARCHAR2(32767) INDEX BY BINARY_INTEGER;
  TYPE desc_rec2 IS RECORD (
        col_type            binary_integer := 0,
        col_max_len         binary_integer := 0,
        col_name            varchar2(32767) := '',
        col_name_len        binary_integer := 0,
        col_schema_name     varchar2(32)   := '',
```

```
        col_schema_name_len binary_integer := 0,
        col_precision       binary_integer := 0,
        col_scale           binary_integer := 0,
        col_charsetid       binary_integer := 0,
        col_charsetform     binary_integer := 0,
        col_null_ok         boolean        := TRUE);
  TYPE desc_tab2 IS TABLE OF desc_rec2 INDEX BY BINARY_INTEGER;
```

VARCHAR2S

```
TYPE varchar2s IS TABLE OF VARCHAR2(256) INDEX BY BINARY_INTEGER;
```

Exceptions

```
inconsistent_type EXCEPTION;
  pragma exception_init(inconsistent_type, -6562);
```

This exception is raised by the COLUMN_VALUE Procedure or the VARIABLE_VALUE Procedures when the type of the given `OUT` parameter (for where to put the requested value) is different from the type of the value.

Operational Notes

Execution Flow

1. OPEN_CURSOR
2. PARSE
3. BIND_VARIABLE or BIND_ARRAY
4. DEFINE_COLUMN, DEFINE_COLUMN_LONG, or DEFINE_ARRAY
5. EXECUTE
6. FETCH_ROWS or EXECUTE_AND_FETCH
7. VARIABLE_VALUE, COLUMN_VALUE, or COLUMN_VALUE_LONG
8. CLOSE_CURSOR

OPEN_CURSOR

To process a SQL statement, you must have an open cursor. When you call the OPEN_CURSOR Function function, you receive a cursor `ID` number for the data structure representing a valid cursor maintained by Oracle. These cursors are distinct from cursors defined at the precompiler, OCI, or PL/SQL level, and are used only by the `DBMS_SQL` package.

PARSE

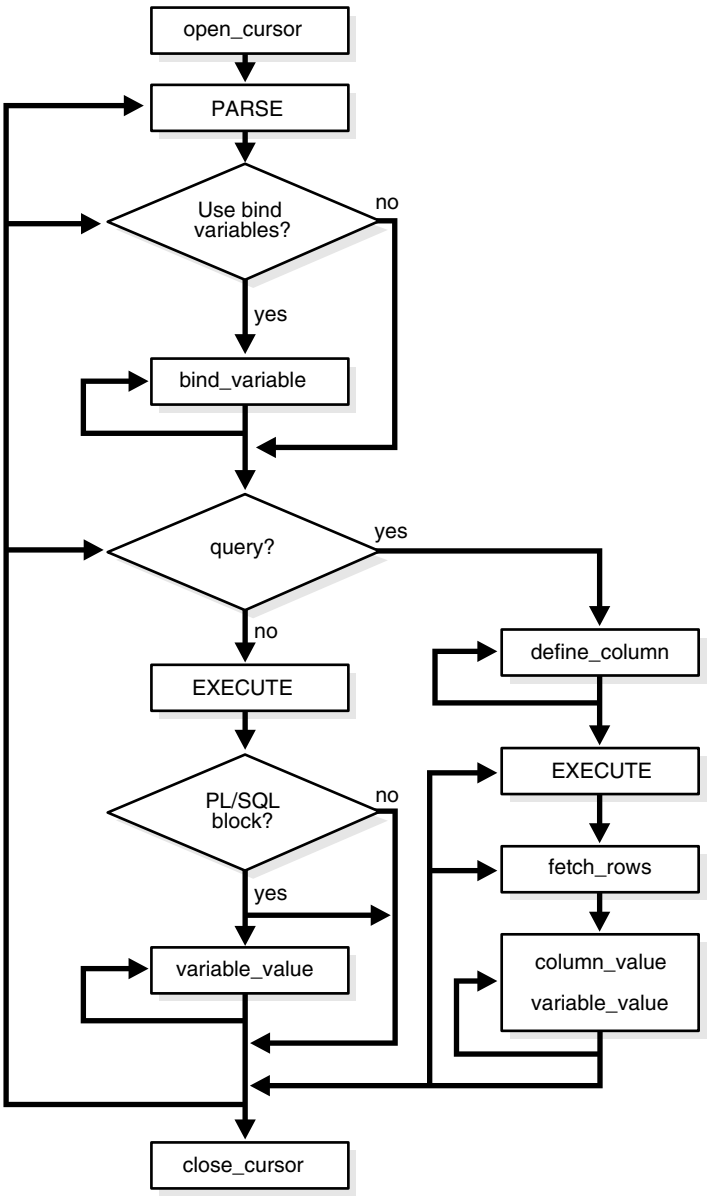
Every SQL statement must be parsed by calling the PARSE Procedure. Parsing the statement checks the statement's syntax and associates it with the cursor in your program.

You can parse any DML or DDL statement. DDL statements are run on the parse, which performs the implied commit.

> **Note:** When parsing a DDL statement to drop a package or a procedure, a deadlock can occur if you're still using a procedure in the package. After a call to a procedure, that procedure is considered to be in use until execution has returned to the user side. Any such deadlock timeouts after five minutes.

The execution flow of `DBMS_SQL` is shown in Figure 100–1.

Figure 100–1 DBMS_SQL Execution Flow



BIND_VARIABLE or BIND_ARRAY

Many DML statements require that data in your program be input to Oracle. When you define a SQL statement that contains input data to be supplied at runtime, you must use placeholders in the SQL statement to mark where data must be supplied.

For each placeholder in the SQL statement, you must call one of the bind procedures, the BIND_ARRAY Procedures on page 100-25 or the BIND_VARIABLE Procedures on page 100-28, to supply the value of a variable in your program (or the values of an array) to the placeholder. When the SQL statement is subsequently run, Oracle uses the data that your program has placed in the output and input, or bind, variables.

`DBMS_SQL` can run a DML statement multiple times — each time with a different bind variable. The `BIND_ARRAY` procedure lets you bind a collection of scalars, each value of which is used as an input variable once for each `EXECUTE`. This is similar to the array interface supported by the OCI.

DEFINE_COLUMN, DEFINE_COLUMN_LONG, or DEFINE_ARRAY

The columns of the row being selected in a SELECT statement are identified by their relative positions as they appear in the select list, from left to right. For a query, you must call one of the define procedures (DEFINE_COLUMN, DEFINE_COLUMN_LONG, or DEFINE_ARRAY) to specify the variables that are to receive the SELECT values, much the way an INTO clause does for a static query.

Use the DEFINE_COLUMN_LONG procedure to define LONG columns, in the same way that DEFINE_COLUMN is used to define non-LONG columns. You must call DEFINE_COLUMN_LONG before using the COLUMN_VALUE_LONG procedure to fetch from the LONG column.

Use the DEFINE_ARRAY procedure to define a PL/SQL collection into which you want to fetch rows in a single SELECT statement. DEFINE_ARRAY provides an interface to fetch multiple rows at one fetch. You must call DEFINE_ARRAY before using the COLUMN_VALUE procedure to fetch the rows.

EXECUTE

Call the EXECUTE function to run your SQL statement.

FETCH_ROWS or EXECUTE_AND_FETCH

The FETCH_ROWS function retrieves the rows that satisfy the query. Each successive fetch retrieves another set of rows, until the fetch is unable to retrieve anymore rows. Instead of calling EXECUTE and then FETCH_ROWS, you may find it more efficient to call EXECUTE_AND_FETCH if you are calling EXECUTE for a single execution.

VARIABLE_VALUE, COLUMN_VALUE, or COLUMN_VALUE_LONG

For queries, call COLUMN_VALUE to determine the value of a column retrieved by the FETCH_ROWS call. For anonymous blocks containing calls to PL/SQL procedures or DML statements with `returning` clause, call VARIABLE_VALUE to retrieve the values assigned to the output variables when statements were run.

To fetch just part of a LONG database column (which can be up to two gigabytes in size), use the COLUMN_VALUE_LONG procedure. You can specify the offset (in bytes) into the column value, and the number of bytes to fetch.

CLOSE_CURSOR

When you no longer need a cursor for a session, close the cursor by calling CLOSE_CURSOR. If you are using an Oracle Open Gateway, then you may need to close cursors at other times as well. Consult your *Oracle Open Gateway* documentation for additional information.

If you neglect to close a cursor, then the memory used by that cursor remains allocated even though it is no longer needed.

Processing Queries

If you are using dynamic SQL to process a query, then you must perform the following steps:

1. Specify the variables that are to receive the values returned by the SELECT statement by calling the DEFINE_COLUMN Procedure, the DEFINE_COLUMN_LONG Procedure, or the DEFINE_ARRAY Procedure.
2. Run your SELECT statement by calling the EXECUTE Function.
3. Call the FETCH_ROWS Function (or EXECUTE_AND_FETCH) to retrieve the rows that satisfied your query.

4. Call COLUMN_VALUE Procedure or COLUMN_VALUE_LONG Procedure to determine the value of a column retrieved by the FETCH_ROWS Function for your query. If you used anonymous blocks containing calls to PL/SQL procedures, then you must call the VARIABLE_VALUE Procedures to retrieve the values assigned to the output variables of these procedures.

Processing Updates, Inserts, and Deletes

If you are using dynamic SQL to process an `INSERT`, `UPDATE`, or `DELETE`, then you must perform the following steps:

1. You must first run your `INSERT`, `UPDATE`, or `DELETE` statement by calling the EXECUTE Function.
2. If statements have the `returning` clause, then you must call the VARIABLE_VALUE Procedures to retrieve the values assigned to the output variables.

Locating Errors

There are additional functions in the DBMS_SQL package for obtaining information about the last referenced cursor in the session. The values returned by these functions are only meaningful immediately after a SQL statement is run. In addition, some error-locating functions are only meaningful after certain DBMS_SQL calls. For example, you call the LAST_ERROR_POSITION Function immediately after a PARSE.

Examples

This section provides example procedures that make use of the DBMS_SQL package.

Example 1

This example does not require the use of dynamic SQL because the text of the statement is known at compile time., but it illustrate the basic concept underlying the package.

The DEMO procedure deletes all of the employees from the EMP table whose salaries are greater than the salary that you specify when you run DEMO.

```
CREATE OR REPLACE PROCEDURE demo(salary IN NUMBER) AS
    cursor_name INTEGER;
    rows_processed INTEGER;
BEGIN
    cursor_name := dbms_sql.open_cursor;
    DBMS_SQL.PARSE(cursor_name, 'DELETE FROM emp WHERE sal > :x',
                   DBMS_SQL.NATIVE);
    DBMS_SQL.BIND_VARIABLE(cursor_name, ':x', salary);
    rows_processed := DBMS_SQL.EXECUTE(cursor_name);
    DBMS_SQL.CLOSE_CURSOR(cursor_name);
EXCEPTION
WHEN OTHERS THEN
    DBMS_SQL.CLOSE_CURSOR(cursor_name);
END;
```

Example 2

The following sample procedure is passed a SQL statement, which it then parses and runs:

```
CREATE OR REPLACE PROCEDURE exec(STRING IN varchar2) AS
    cursor_name INTEGER;
    ret INTEGER;
BEGIN
   cursor_name := DBMS_SQL.OPEN_CURSOR;
```

DDL statements are run by the parse call, which performs the implied commit.

```
   DBMS_SQL.PARSE(cursor_name, string, DBMS_SQL.NATIVE);
   ret := DBMS_SQL.EXECUTE(cursor_name);
   DBMS_SQL.CLOSE_CURSOR(cursor_name);
END;
```

Creating such a procedure enables you to perform the following operations:

- The SQL statement can be dynamically generated at runtime by the calling program.
- The SQL statement can be a DDL statement or a DML without binds.

For example, after creating this procedure, you could make the following call:

```
exec('create table acct(c1 integer)');
```

You could even call this procedure remotely, as shown in the following example. This lets you perform remote DDL.

```
exec@hq.com('CREATE TABLE acct(c1 INTEGER)');
```

Example 3

The following sample procedure is passed the names of a source and a destination table, and copies the rows from the source table to the destination table. This sample procedure assumes that both the source and destination tables have the following columns:

```
id        of type NUMBER
name      of type VARCHAR2(30)
birthdate of type DATE
```

This procedure does not specifically require the use of dynamic SQL; however, it illustrates the concepts of this package.

```
CREATE OR REPLACE PROCEDURE copy (
     source      IN VARCHAR2,
     destination IN VARCHAR2) IS
     id_var             NUMBER;
     name_var           VARCHAR2(30);
     birthdate_var      DATE;
     source_cursor      INTEGER;
     destination_cursor INTEGER;
     ignore             INTEGER;
  BEGIN

  -- Prepare a cursor to select from the source table:
     source_cursor := dbms_sql.open_cursor;
     DBMS_SQL.PARSE(source_cursor,
         'SELECT id, name, birthdate FROM ' || source,
          DBMS_SQL.NATIVE);
     DBMS_SQL.DEFINE_COLUMN(source_cursor, 1, id_var);
     DBMS_SQL.DEFINE_COLUMN(source_cursor, 2, name_var, 30);
     DBMS_SQL.DEFINE_COLUMN(source_cursor, 3, birthdate_var);
     ignore := DBMS_SQL.EXECUTE(source_cursor);

  -- Prepare a cursor to insert into the destination table:
     destination_cursor := DBMS_SQL.OPEN_CURSOR;
     DBMS_SQL.PARSE(destination_cursor,
                  'INSERT INTO ' || destination ||
                  ' VALUES (:id_bind, :name_bind, :birthdate_bind)',
                   DBMS_SQL.NATIVE);

  -- Fetch a row from the source table and insert it into the destination table:
     LOOP
       IF DBMS_SQL.FETCH_ROWS(source_cursor)>0 THEN
         -- get column values of the row
         DBMS_SQL.COLUMN_VALUE(source_cursor, 1, id_var);
         DBMS_SQL.COLUMN_VALUE(source_cursor, 2, name_var);
         DBMS_SQL.COLUMN_VALUE(source_cursor, 3, birthdate_var);

  -- Bind the row into the cursor that inserts into the destination table. You
  -- could alter this example to require the use of dynamic SQL by inserting an
  -- if condition before the bind.
       DBMS_SQL.BIND_VARIABLE(destination_cursor, ':id_bind', id_var);
       DBMS_SQL.BIND_VARIABLE(destination_cursor, ':name_bind', name_var);
       DBMS_SQL.BIND_VARIABLE(destination_cursor, ':birthdate_bind',
birthdate_var);
       ignore := DBMS_SQL.EXECUTE(destination_cursor);
     ELSE

  -- No more rows to copy:
```

```
        EXIT;
      END IF;
    END LOOP;

  -- Commit and close all cursors:
     COMMIT;
     DBMS_SQL.CLOSE_CURSOR(source_cursor);
     DBMS_SQL.CLOSE_CURSOR(destination_cursor);
   EXCEPTION
     WHEN OTHERS THEN
       IF DBMS_SQL.IS_OPEN(source_cursor) THEN
         DBMS_SQL.CLOSE_CURSOR(source_cursor);
       END IF;
       IF DBMS_SQL.IS_OPEN(destination_cursor) THEN
         DBMS_SQL.CLOSE_CURSOR(destination_cursor);
       END IF;
       RAISE;
  END;
/
```

Examples 3, 4, and 5: Bulk DML

This series of examples shows how to use bulk array binds (table items) in the SQL DML statements `DELETE`, `INSERT`, and `UPDATE`.

In a `DELETE` statement, for example, you could bind in an array in the `WHERE` clause and have the statement be run for each element in the array:

```
DECLARE
  stmt VARCHAR2(200);
  dept_no_array DBMS_SQL.NUMBER_TABLE;
  c NUMBER;
  dummy NUMBER;
begin
  dept_no_array(1) := 10; dept_no_array(2) := 20;
  dept_no_array(3) := 30; dept_no_array(4) := 40;
  dept_no_array(5) := 30; dept_no_array(6) := 40;
  stmt := 'delete from emp where deptno = :dept_array';
  c := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(c, stmt, DBMS_SQL.NATIVE);
  DBMS_SQL.BIND_ARRAY(c, ':dept_array', dept_no_array, 1, 4);
  dummy := DBMS_SQL.EXECUTE(c);
  DBMS_SQL.CLOSE_CURSOR(c);

  EXCEPTION WHEN OTHERS THEN
    IF DBMS_SQL.IS_OPEN(c) THEN
      DBMS_SQL.CLOSE_CURSOR(c);
    END IF;
    RAISE;
END;
/
```

In the preceding example, only elements 1 through 4 are used as specified by the `BIND_ARRAY` call. Each element of the array potentially deletes a large number of employees from the database.

Here is an example of a bulk `INSERT` statement:

```
DECLARE
  stmt VARCHAR2(200);
  empno_array DBMS_SQL.NUMBER_TABLE;
  empname_array DBMS_SQL.VARCHAR2_TABLE;
```

```
  c NUMBER;
  dummy NUMBER;
BEGIN
  FOR i in 0..9 LOOP
    empno_array(i) := 1000 + i;
    empname_array(I) := get_name(i);
  END LOOP;
  stmt := 'INSERT INTO emp VALUES(:num_array, :name_array)';
  c := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(c, stmt, DBMS_SQL.NATIVE);
  DBMS_SQL.BIND_ARRAY(c, ':num_array', empno_array);
  DBMS_SQL.BIND_ARRAY(c, ':name_array', empname_array);
  dummy := DBMS_SQL.EXECUTE(c);
  DBMS_SQL.CLOSE_CURSOR(c);

  EXCEPTION WHEN OTHERS THEN
    IF DBMS_SQL.IS_OPEN(c) THEN
      DBMS_SQL.CLOSE_CURSOR(c);
    END IF;
    RAISE;
END;
/
```

When the execute takes place, all 10 of the employees are inserted into the table.

Finally, here is an example of an bulk `UPDATE` statement.

```
Declare
  stmt VARCHAR2(200);
  emp_no_array DBMS_SQL.NUMBER_TABLE;
  emp_addr_array DBMS_SQL.VARCHAR2_TABLE;
  c NUMBER;
  dummy NUMBER;
BEGIN
  for i in 0..9 loop
    emp_no_array(i) := 1000 + i;
    emp_addr_array(I) := get_new_addr(i);
  END LOOP;
  stmt := 'update emp set ename = :name_array
    WHERE empno = :num_array';
  c := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(c, stmt, DBMS_SQL.NATIVE);
  DBMS_SQL.BIND_ARRAY(c, ':num_array', empno_array);
  DBMS_SQL.BIND_ARRAY(c, ':name_array', empname_array);
  dummy := DBMS_SQL.EXECUTE(c);
  DBMS_SQL.CLOSE_CURSOR(c);

  EXCEPTION WHEN OTHERS THEN
    IF DBMS_SQL.IS_OPEN(c) THEN
      DBMS_SQL.CLOSE_CURSOR(c);
    END IF;
    RAISE;
END;
/
```

When the EXECUTE Function call happens, the addresses of all employees are updated at once. The two collections are always stepped in unison. If the `WHERE` clause returns more than one row, then all those employees get the address the `addr_array` happens to be pointing to at that time.

Examples 6 and 7: Defining an Array

The following examples show how to use the DEFINE_ARRAY procedure:

```
declare
  c       NUMBER;
  d       NUMBER;
  n_tab   DBMS_SQL.NUMBER_TABLE;
  indx    NUMBER := -10;
BEGIN
  c := DBMS_SQL.OPEN_CURSOR;
  dBMS_SQL.PARSE(c, 'select n from t order by 1', DBMS_SQL.NATIVE);

  DBMS_SQL.DEFINE_ARRAY(c, 1, n_tab, 10, indx);

  d := DBMS_SQL.EXECUTE(c);
  loop
    d := DBMS_SQL.FETCH_ROWS(c);

    DBMS_SQL.COLUMN_VALUE(c, 1, n_tab);

    EXIT WHEN d != 10;
  END LOOP;

  DBMS_SQL.CLOSE_CURSOR(c);

  EXCEPTION WHEN OTHERS THEN
    IF DBMS_SQL.IS_OPEN(c) THEN
      DBMS_SQL.CLOSE_CURSOR(c);
    END IF;
    RAISE;
END;
/
```

Each time the preceding example does a FETCH_ROWS Function call, it fetches 10 rows that are kept in DBMS_SQL buffers. When the COLUMN_VALUE Procedure call is run, those rows move into the PL/SQL table specified (in this case n_tab), at positions -10 to -1, as specified in the DEFINE statements. When the second batch is fetched in the loop, the rows go to positions 0 to 9; and so on.

A current index into each array is maintained automatically. This index is initialized to "indx" at EXECUTE and keeps getting updated every time a COLUMN_VALUE call is made. If you reexecute at any point, then the current index for each DEFINE is re-initialized to "indx".

In this way the entire result of the query is fetched into the table. When FETCH_ROWS cannot fetch 10 rows, it returns the number of rows actually fetched (if no rows could be fetched, then it returns zero) and exits the loop.

Here is another example of using the DEFINE_ARRAY procedure:

Consider a table MULTI_TAB defined as:

```
CREATE TABLE multi_tab (num NUMBER,
                        dat1 DATE,
                        var VARCHAR2(24),
                        dat2 DATE)
```

To select everything from this table and move it into four PL/SQL tables, you could use the following simple program:

```
declare
  c       NUMBER;
```

```
  d       NUMBER;
  n_tab  DBMS_SQL.NUMBER_TABLE;
  d_tab1 DBMS_SQL.DATE_TABLE;
  v_tab  DBMS_SQL.VARCHAR2_TABLE;
  d_tab2 DBMS_SQL.DATE_TABLE;
  indx NUMBER := 10;
BEGIN

  c := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(c, 'select * from multi_tab order by 1', DBMS_SQL.NATIVE);

  DBMS_SQL.DEFINE_ARRAY(c, 1, n_tab,  5, indx);
  DBMS_SQL.DEFINE_ARRAY(c, 2, d_tab1, 5, indx);
  DBMS_SQL.DEFINE_ARRAY(c, 3, v_tab,  5, indx);
  DBMS_SQL.DEFINE_ARRAY(c, 4, d_tab2, 5, indx);

  d := DBMS_SQL.EXECUTE(c);

  loop
    d := DBMS_SQL.FETCH_ROWS(c);

    DBMS_SQL.COLUMN_VALUE(c, 1, n_tab);
    DBMS_SQL.COLUMN_VALUE(c, 2, d_tab1);
    DBMS_SQL.COLUMN_VALUE(c, 3, v_tab);
    DBMS_SQL.COLUMN_VALUE(c, 4, d_tab2);

    EXIT WHEN d != 5;
  END LOOP;

  DBMS_SQL.CLOSE_CURSOR(c);

/*
```

The four tables can be used for anything. One usage might be to use `BIND_ARRAY` to move the rows to another table by using a query such as '`INSERT` into `SOME_T` values (:a, :b, :c, :d);

```
*/

EXCEPTION WHEN OTHERS THEN
    IF DBMS_SQL.IS_OPEN(c) THEN
      DBMS_SQL.CLOSE_CURSOR(c);
    END IF;
    RAISE;
END;
/
```

Example 8: Describe Columns

This can be used as a substitute to the SQL*Plus `DESCRIBE` call by using a `SELECT *` query on the table that you want to describe.

```
DECLARE
  c           NUMBER;
  d           NUMBER;
  col_cnt     INTEGER;
  f           BOOLEAN;
  rec_tab     DBMS_SQL.DESC_TAB;
  col_num    NUMBER;
  PROCEDURE print_rec(rec in DBMS_SQL.DESC_REC) IS
  BEGIN
```

```
    DBMS_OUTPUT.NEW_LINE;
    DBMS_OUTPUT.PUT_LINE('col_type            =    '
                         || rec.col_type);
    DBMS_OUTPUT.PUT_LINE('col_maxlen          =    '
                         || rec.col_max_len);
    DBMS_OUTPUT.PUT_LINE('col_name            =    '
                         || rec.col_name);
    DBMS_OUTPUT.PUT_LINE('col_name_len        =    '
                         || rec.col_name_len);
    DBMS_OUTPUT.PUT_LINE('col_schema_name     =    '
                         || rec.col_schema_name);
    DBMS_OUTPUT.PUT_LINE('col_schema_name_len =    '
                         || rec.col_schema_name_len);
    DBMS_OUTPUT.PUT_LINE('col_precision       =    '
                         || rec.col_precision);
    DBMS_OUTPUT.PUT_LINE('col_scale           =    '
                         || rec.col_scale);
    DBMS_OUTPUT.PUT('col_null_ok         =    ');
    IF (rec.col_null_ok) THEN
      DBMS_OUTPUT.PUT_LINE('true');
    ELSE
      DBMS_OUTPUT.PUT_LINE('false');
    END IF;
  END;
BEGIN
  c := DBMS_SQL.OPEN_CURSOR;

  DBMS_SQL.PARSE(c, 'SELECT * FROM scott.bonus', DBMS_SQL.NATIVE);

  d := DBMS_SQL.EXECUTE(c);

  DBMS_SQL.DESCRIBE_COLUMNS(c, col_cnt, rec_tab);

/*
 * Following loop could simply be for j in 1..col_cnt loop.
 * Here we are simply illustrating some of the PL/SQL table
 * features.
 */
  col_num := rec_tab.first;
  IF (col_num IS NOT NULL) THEN
    LOOP
      print_rec(rec_tab(col_num));
      col_num := rec_tab.next(col_num);
      EXIT WHEN (col_num IS NULL);
    END LOOP;
  END IF;

  DBMS_SQL.CLOSE_CURSOR(c);
END;
/
```

Example 9: RETURNING clause

The `RETURNING` clause was added to DML statements in an earlier Oracle database release. With this clause, `INSERT`, `UPDATE`, and `DELETE` statements can return values of expressions. These values are returned in bind variables.

`DBMS_SQL.BIND_VARIABLE` is used to bind these outbinds if a single row is inserted, updated, or deleted. If multiple rows are inserted, updated, or deleted, then `DBMS_SQL.BIND_ARRAY` is used. `DBMS_SQL.VARIABLE_VALUE` must be called to get the values in these bind variables.

Note: This is similar to DBMS_SQL.VARIABLE_VALUE, which must be called after running a PL/SQL block with an out-bind inside DBMS_SQL.

i) Single row insert

```
CREATE OR REPLACE PROCEDURE single_Row_insert
     (c1 NUMBER, c2 NUMBER, r OUT NUMBER) is
c NUMBER;
n NUMBER;
begin
  c := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(c, 'INSERT INTO tab VALUES (:bnd1, :bnd2) ' ||
                    'RETURNING c1*c2 INTO :bnd3', DBMS_SQL.NATIVE);
DBMS_SQL.BIND_VARIABLE(c, 'bnd1', c1);
  DBMS_SQL.BIND_VARIABLE(c, 'bnd2', c2);
  DBMS_SQL.BIND_VARIABLE(c, 'bnd3', r);
  n := DBMS_SQL.EXECUTE(c);
  DBMS_SQL.VARIABLE_VALUE(c, 'bnd3', r); -- get value of outbind variable
  DBMS_SQL.CLOSE_CURSOR(c);
END;
/
```

ii) Single row update

```
CREATE OR REPLACE PROCEDURE single_Row_update
     (c1 NUMBER, c2 NUMBER, r out NUMBER) IS
c NUMBER;
n NUMBER;
BEGIN
  c := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(c, 'UPDATE tab SET c1 = :bnd1, c2 = :bnd2 ' ||
                    'WHERE rownum < 2' ||
                    'RETURNING c1*c2 INTO :bnd3', DBMS_SQL.NATIVE);
  DBMS_SQL.BIND_VARIABLE(c, 'bnd1', c1);
  DBMS_SQL.BIND_VARIABLE(c, 'bnd2', c2);
  DBMS_SQL.BIND_VARIABLE(c, 'bnd3', r);
  n := DBMS_SQL.EXECUTE(c);
  DBMS_SQL.VARIABLE_VALUE(c, 'bnd3', r);-- get value of outbind variable
  DBMS_SQL.CLOSE_CURSOR(c);
END;
/
```

iii) Single row delete

```
CREATE OR REPLACE PROCEDURE single_Row_Delete
     (c1 NUMBER, c2 NUMBER, r OUT NUMBER) is
c NUMBER;
n number;
BEGIN
  c := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(c, 'delete from tab ' ||
                    'where rownum < 2 ' ||
                   'returning c1*c2 into :bnd3', DBMS_SQL.NATIVE);
  DBMS_SQL.BIND_VARIABLE(c, 'bnd1', c1);
  DBMS_SQL.BIND_VARIABLE(c, 'bnd2', c2);
  DBMS_SQL.BIND_VARIABLE(c, 'bnd3', r);
  n := DBMS_SQL.EXECUTE(c);
  DBMS_SQL.VARIABLE_VALUE(c, 'bnd3', r);-- get value of outbind variable
```

```
  DBMS_SQL.CLOSE_CURSOR(c);
END;
/
```

iv) Multiple row insert

```
CREATE OR REPLACE PROCEDURE multi_Row_insert
     (c1 DBMS_SQL.NUMBER_TABLE, c2 DBMS_SQL.NUMBER_TABLE,
      r OUT DBMS_SQL.NUMBER_TABLE) is
c NUMBER;
n NUMBER;
BEGIN
  c := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(c, 'insert into tab VALUES (:bnd1, :bnd2) ' ||
                    'RETURNING c1*c2 INTO :bnd3', DBMS_SQL.NATIVE);
  DBMS_SQL.BIND_ARRAY(c, 'bnd1', c1);
  DBMS_SQL.BIND_ARRAY(c, 'bnd2', c2);
  DBMS_SQL.BIND_ARRAY(c, 'bnd3', r);
  n := DBMS_SQL.EXECUTE(c);
  DBMS_SQL.VARIABLE_VALUE(c, 'bnd3', r);-- get value of outbind variable
  DBMS_SQL.CLOSE_CURSOR(c);
END;
/
```

v) Multiple row Update.

```
CREATE OR REPLACE PROCEDURE multi_Row_update
     (c1 NUMBER, c2 NUMBER, r OUT DBMS_SQL.NUMBER_TABLE) IS
c NUMBER;
n NUMBER;
BEGIN
  c := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(c, 'UPDATE tab SET c1 = :bnd1 WHERE c2 = :bnd2 ' ||
                    'RETURNING c1*c2 INTO :bnd3', DBMS_SQL.NATIVE);
  DBMS_SQL.BIND_VARIABLE(c, 'bnd1', c1);
  DBMS_SQL.BIND_VARIABLE(c, 'bnd2', c2);
  DBMS_SQL.BIND_ARRAY(c, 'bnd3', r);
  n := DBMS_SQL.EXECUTE(c);
  DBMS_SQL.VARIABLE_VALUE(c, 'bnd3', r);-- get value of outbind variable
  DBMS_SQL.CLOSE_CURSOR(c);
END;
/
```

Note: bnd1 and bnd2 can be array as well. The value of the expression for all the rows updated will be in bnd3. There is no way of differentiating which rows got updated of each value of bnd1 and bnd2.

vi) Multiple row delete

```
CREATE OR REPLACE PROCEDURE multi_row_delete
     (c1 DBMS_SQL.NUMBER_TABLE,
      r OUT DBMS_SQL.NUMBER_TABLE) is
c NUMBER;
n NUMBER;
BEGIN
  c := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(c, 'DELETE FROM tab WHERE c1 = :bnd1' ||
                    'RETURNING c1*c2 INTO :bnd2', DBMS_SQL.NATIVE);
```

```
  DBMS_SQL.BIND_ARRAY(c, 'bnd1', c1);
  DBMS_SQL.BIND_ARRAY(c, 'bnd2', r);
  n := DBMS_SQL.EXECUTE(c);
  DBMS_SQL.VARIABLE_VALUE(c, 'bnd2', r);-- get value of outbind variable
  DBMS_SQL.CLOSE_CURSOR(c);
END;
/
```

vii) Out-bind in bulk PL/SQL

```
CREATE OR REPLACE PROCEDURE foo (n NUMBER, square OUT NUMBER) IS
BEGIN square := n * n; END;/

CREATE OR REPLACE PROCEDURE bulk_plsql
   (n DBMS_SQL.NUMBER_TABLE, square OUT DBMS_SQL.NUMBER_TABLE) IS
c NUMBER;
r NUMBER;
BEGIN
  c := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(c, 'BEGIN foo(:bnd1, :bnd2); END;', DBMS_SQL.NATIVE);
  DBMS_SQL.BIND_ARRAY(c, 'bnd1', n);
  DBMS_SQL.BIND_ARRAY(c, 'bnd2', square);
  r := DBMS_SQL.EXECUTE(c);
  DBMS_SQL.VARIABLE_VALUE(c, 'bnd2', square);
END;
/
```

Note: `DBMS_SQL.BIND_ARRAY` of `number_Table` internally binds a number. The number of times statement is run depends on the number of elements in an inbind array.

Summary of DBMS_SQL Subprograms

Table 100–1 DBMS_SQL Package Subprograms

Subprogram	Description
BIND_ARRAY Procedures on page 100-25	Binds a given value to a given collection
BIND_VARIABLE Procedures on page 100-28	Binds a given value to a given variable
CLOSE_CURSOR Procedure on page 100-32	Closes given cursor and frees memory
COLUMN_VALUE Procedure on page 100-33	Returns value of the cursor element for a given position in a cursor
COLUMN_VALUE_LONG Procedure on page 100-36	Returns a selected part of a `LONG` column, that has been defined using `DEFINE_COLUMN_LONG`
DEFINE_ARRAY Procedure on page 100-37	Defines a collection to be selected from the given cursor, used only with `SELECT` statements
DEFINE_COLUMN Procedure on page 100-39	Defines a column to be selected from the given cursor, used only with `SELECT` statements
DEFINE_COLUMN_LONG Procedure on page 100-41	Defines a `LONG` column to be selected from the given cursor, used only with `SELECT` statements
DESCRIBE_COLUMNS Procedure on page 100-42	Describes the columns for a cursor opened and parsed through `DBMS_SQL`
DESCRIBE_COLUMNS2 Procedure on page 100-43	Describes describes the specified column, an alternative to DESCRIBE_COLUMNS Procedure
EXECUTE Function on page 100-44	Executes a given cursor
EXECUTE_AND_FETCH Function on page 100-45	Executes a given cursor and fetch rows
FETCH_ROWS Function on page 100-46	Fetches a row from a given cursor
IS_OPEN Function on page 100-47	Returns `TRUE` if given cursor is open
LAST_ERROR_POSITION Function on page 100-48	Returns byte offset in the SQL statement text where the error occurred
LAST_ROW_COUNT Function on page 100-49	Returns cumulative count of the number of rows fetched
LAST_ROW_ID Function on page 100-50	Returns `ROWID` of last row processed
LAST_SQL_FUNCTION_CODE Function on page 100-51	Returns SQL function code for statement
OPEN_CURSOR Function on page 100-52	Returns cursor ID number of new cursor
PARSE Procedure on page 100-53	Parses given statement
VARIABLE_VALUE Procedures on page 100-55	Returns value of named variable for given cursor

BIND_ARRAY Procedures

This procedure binds a given value or set of values to a given variable in a cursor, based on the name of the variable in the statement.

Syntax

```
DBMS_SQL.BIND_ARRAY (
   c                  IN INTEGER,
   name               IN VARCHAR2,
   <table_variable>   IN <datatype>
 [,index1             IN INTEGER,
   index2             IN INTEGER)] );
```

Where the <table_variable> and its corresponding <datatype> can be any one of the following matching pairs:

```
<clob_tab>     Clob_Table
<bflt_tab>     Binary_Float_Table
<bdbl_tab>     Binary_Double_Table
<blob_tab>     Blob_Table
<bfile_tab>    Bfile_Table
<date_tab>     Date_Table
<num_tab>      Number_Table
<urowid_tab>   Urowid_Table
<vchr2_tab>    Varchar2_Table
```

Notice that the BIND_ARRAY procedure is overloaded to accept different datatypes.

Parameters

Table 100–2 BIND_ARRAY Procedure Parameters

Parameter	Description
c	ID number of the cursor to which you want to bind a value.
name	Name of the collection in the statement.
table_variable	Local variable that has been declared as <datatype>.
index1	Index for the table element that marks the lower bound of the range.
index2	Index for the table element that marks the upper bound of the range.

Usage Notes

The length of the bind variable name should be <=30 bytes.

For binding a range, the table must contain the elements that specify the range — tab(index1) and tab(index2) — but the range does not have to be dense. Index1 must be less than or equal to index2. All elements between tab(index1) and tab(index2) are used in the bind.

If you do not specify indexes in the bind call, and two different binds in a statement specify tables that contain a different number of elements, then the number of elements actually used is the minimum number between all tables. This is also the case if you specify indexes — the minimum range is selected between the two indexes for all tables.

Not all bind variables in a query have to be array binds. Some can be regular binds and the same value are used for each element of the collections in expression evaluations (and so forth).

> **See Also:** "Examples 3, 4, and 5: Bulk DML" on page 100-16 for examples of how to bind collections.

Bulk Array Binds

Bulk selects, inserts, updates, and deletes can enhance the performance of applications by bundling many calls into one. The DBMS_SQL package lets you work on collections of data using the PL/SQL table type.

Table items are unbounded homogeneous collections. In persistent storage, they are like other relational tables and have no intrinsic ordering. But when a table item is brought into the workspace (either by querying or by navigational access of persistent data), or when it is created as the value of a PL/SQL variable or parameter, its elements are given subscripts that can be used with array-style syntax to get and set the values of elements.

The subscripts of these elements need not be dense, and can be any number including negative numbers. For example, a table item can contain elements at locations -10, 2, and 7 only.

When a table item is moved from transient workspace to persistent storage, the subscripts are not stored; the table item is unordered in persistent storage.

At bind time the table is copied out from the PL/SQL buffers into local DBMS_SQL buffers (the same as for all scalar types) and then the table is manipulated from the local DBMS_SQL buffers. Therefore, if you change the table after the bind call, then that change does not affect the way the execute acts.

Types for Scalar and LOB Collections

You can declare a local variable as one of the following table-item types, which are defined as public types in DBMS_SQL.

```
TYPE binary_double_table
                    IS TABLE OF BINARY_DOUBLE  INDEX BY BINARY_INTEGER;
TYPE binary_float_table
                    IS TABLE OF BINARY_FLOAT   INDEX BY BINARY_INTEGER;
TYPE bfile_table    IS TABLE OF BFILE          INDEX BY BINARY_INTEGER;
TYPE blob_table     IS TABLE OF BLOB           INDEX BY BINARY_INTEGER;
TYPE clob_table     IS TABLE OF CLOB           INDEX BY BINARY_INTEGER;
TYPE date_table     IS TABLE OF DATE           INDEX BY BINARY_INTEGER;
TYPE interval_day_to_second_Table
                    IS TABLE OF dsinterval_unconstrained
                                               INDEX BY BINARY_INTEGER;
TYPE interval_year_to_MONTH_Table
                    IS TABLE OF yminterval_unconstrained
                                               INDEX BY BINARY_INTEGER;
TYPE number_table   IS TABLE OF NUMBER         INDEX BY BINARY_INTEGER;
TYPE time_table     IS TABLE OF time_unconstrained
                                               INDEX BY BINARY_INTEGER;
TYPE time_with_time_zone_table
                    IS TABLE OF time_tz_unconstrained
                                               INDEX BY BINARY_INTEGER;
TYPE timestamp_table
                    IS TABLE OF timestamp_unconstrained
                                               INDEX BY BINARY_INTEGER;
TYPE timestamp_with_ltz_Table
```

```
                  IS TABLE OF timestamp_ltz_unconstrained
                                                INDEX BY BINARY_INTEGER;
TYPE timestamp_with_time_zone_Table
                  IS TABLE OF timestamp_tz_unconstrained
                                                INDEX BY BINARY_INTEGER;
TYPE urowid_table   IS TABLE OF UROWID         INDEX BY BINARY_INTEGER;
TYPE varchar2_table IS TABLE OF VARCHAR2(2000) INDEX BY BINARY_INTEGER;

<tm_tab>   Time_Table
<ttz_tab>  Time_With_Time_Zone_Table
<tms_tab>  Timestamp_Table
<tstz_tab> Timestamp_With_ltz_Table;
<tstz_tab> Timestamp_With_Time_Zone_Table
<ids_tab>  Interval_Day_To_Second_Table
<iym_tab>  Interval_Year_To_Month_Table
```

BIND_VARIABLE Procedures

This procedures binds a given value or set of values to a given variable in a cursor, based on the name of the variable in the statement.

Syntax

```
DBMS_SQL.BIND_VARIABLE (
   c              IN INTEGER,
   name           IN VARCHAR2,
   value          IN <datatype>)
```

Where <datatype> can be any one of the following types:

```
BINARY_DOUBLE
BINARY_FLOAT
BFILE
BLOB
CLOB CHARACTER SET ANY_CS
DATE
DSINTERVAL_UNCONSTRAINED
NUMBER
TIME_UNCONSTRAINED
TIME_TZ_UNCONSTRAINED
TIMESTAMP_LTZ_UNCONSTRAINED
TIMESTAMP_TZ_UNCONSTRAINED
TIMESTAMP_UNCONSTRAINED
UROWID
VARCHAR2 CHARACTER SET ANY_CS
YMINTERVAL_UNCONSTRAINED
```

Notice that `BIND_VARIABLE` is overloaded to accept different datatypes.

The following syntax is also supported for `BIND_VARIABLE`. The square brackets [] indicate an optional parameter for the `BIND_VARIABLE` function.

```
DBMS_SQL.BIND_VARIABLE (
   c              IN INTEGER,
   name           IN VARCHAR2,
   value          IN VARCHAR2 CHARACTER SET ANY_CS [,out_value_size IN INTEGER]);
```

To bind `CHAR`, `RAW`, and `ROWID` data, you can use the following variations on the syntax:

```
DBMS_SQL.BIND_VARIABLE_CHAR (
   c              IN INTEGER,
   name           IN VARCHAR2,
   value          IN CHAR CHARACTER SET ANY_CS [,out_value_size IN INTEGER]);

DBMS_SQL.BIND_VARIABLE_RAW (
   c              IN INTEGER,
   name           IN VARCHAR2,
   value          IN RAW [,out_value_size IN INTEGER]);

DBMS_SQL.BIND_VARIABLE_ROWID (
   c              IN INTEGER,
   name           IN VARCHAR2,
   value          IN ROWID);
```

See Also: *Oracle Database Application Developer's Guide - Large Objects*

Pragmas

```
pragma restrict_references(bind_variable,WNDS);
```

Parameters

Table 100–3 BIND_VARIABLE Procedure Parameters

Parameter	Description
c	ID number of the cursor to which you want to bind a value.
name	Name of the variable in the statement.
value	Value that you want to bind to the variable in the cursor. For IN and IN/OUT variables, the value has the same type as the type of the value being passed in for this parameter.
out_value_size	Maximum expected OUT value size, in bytes, for the VARCHAR2, RAW, CHAR OUT or IN/OUT variable. If no size is given, then the length of the current value is used. This parameter must be specified if the value parameter is not initialized.

Usage Notes

If the variable is an IN or IN/OUT variable or an IN collection, then the given bind value must be valid for the variable or array type. Bind values for OUT variables are ignored.

The bind variables or collections of a SQL statement are identified by their names. When binding a value to a bind variable or bind array, the string identifying it in the statement must contain a leading colon, as shown in the following example:

```
SELECT emp_name FROM emp WHERE SAL > :X;
```

For this example, the corresponding bind call would look similar to

```
BIND_VARIABLE(cursor_name, ':X', 3500);

or
BIND_VARIABLE (cursor_name, 'X', 3500);
```

The length of the bind variable name should be <=30 bytes.

For binding a range, the table must contain the elements that specify the range — tab(index1) and tab(index2) — but the range does not have to be dense. Index1 must be less than or equal to index2. All elements between tab(index1) and tab(index2) are used in the bind.

If you do not specify indexes in the bind call, and two different binds in a statement specify tables that contain a different number of elements, then the number of elements actually used is the minimum number between all tables. This is also the case if you specify indexes — the minimum range is selected between the two indexes for all tables.

Not all bind variables in a query have to be array binds. Some can be regular binds and the same value are used for each element of the collections in expression evaluations (and so forth).

> **See Also:** "Examples 3, 4, and 5: Bulk DML" on page 100-16 for examples of how to bind collections.

Bulk Array Binds

Bulk selects, inserts, updates, and deletes can enhance the performance of applications by bundling many calls into one. The DBMS_SQL package lets you work on collections of data using the PL/SQL table type.

Table items are unbounded homogeneous collections. In persistent storage, they are like other relational tables and have no intrinsic ordering. But when a table item is brought into the workspace (either by querying or by navigational access of persistent data), or when it is created as the value of a PL/SQL variable or parameter, its elements are given subscripts that can be used with array-style syntax to get and set the values of elements.

The subscripts of these elements need not be dense, and can be any number including negative numbers. For example, a table item can contain elements at locations -10, 2, and 7 only.

When a table item is moved from transient workspace to persistent storage, the subscripts are not stored; the table item is unordered in persistent storage.

At bind time the table is copied out from the PL/SQL buffers into local DBMS_SQL buffers (the same as for all scalar types) and then the table is manipulated from the local DBMS_SQL buffers. Therefore, if you change the table after the bind call, then that change does not affect the way the execute acts.

Types for Scalar and LOB Collections

You can declare a local variable as one of the following table-item types, which are defined as public types in DBMS_SQL.

```
TYPE binary_double_table
                    IS TABLE OF BINARY_DOUBLE  INDEX BY BINARY_INTEGER;
TYPE binary_float_table
                    IS TABLE OF BINARY_FLOAT   INDEX BY BINARY_INTEGER;
TYPE bfile_table    IS TABLE OF BFILE          INDEX BY BINARY_INTEGER;
TYPE blob_table     IS TABLE OF BLOB           INDEX BY BINARY_INTEGER;
TYPE clob_table     IS TABLE OF CLOB           INDEX BY BINARY_INTEGER;
TYPE date_table     IS TABLE OF DATE           INDEX BY BINARY_INTEGER;
TYPE interval_day_to_second_Table
                    IS TABLE OF dsinterval_unconstrained
                                               INDEX BY BINARY_INTEGER;
TYPE interval_year_to_MONTH_Table
                    IS TABLE OF yminterval_unconstrained
                                               INDEX BY BINARY_INTEGER;
TYPE number_table   IS TABLE OF NUMBER         INDEX BY BINARY_INTEGER;
TYPE time_table     IS TABLE OF time_unconstrained
                                               INDEX BY BINARY_INTEGER;
TYPE time_with_time_zone_table
                    IS TABLE OF time_tz_unconstrained
                                               INDEX BY BINARY_INTEGER;
TYPE timestamp_table
                    IS TABLE OF timestamp_unconstrained
                                               INDEX BY BINARY_INTEGER;
TYPE timestamp_with_ltz_Table
                    IS TABLE OF timestamp_ltz_unconstrained
                                               INDEX BY BINARY_INTEGER;
TYPE timestamp_with_time_zone_Table
                    IS TABLE OF timestamp_tz_unconstrained
```

```
                                                  INDEX BY BINARY_INTEGER;
TYPE urowid_table   IS TABLE OF UROWID         INDEX BY BINARY_INTEGER;
TYPE varchar2_table IS TABLE OF VARCHAR2(2000) INDEX BY BINARY_INTEGER;

<tm_tab>   Time_Table
<ttz_tab>  Time_With_Time_Zone_Table
<tms_tab>  Timestamp_Table
<tstz_tab> Timestamp_With_ltz_Table;
<tstz_tab> Timestamp_With_Time_Zone_Table
<ids_tab>  Interval_Day_To_Second_Table
<iym_tab>  Interval_Year_To_Month_Table
```

CLOSE_CURSOR Procedure

This procedure closes a given cursor.

Syntax

```
DBMS_SQL.CLOSE_CURSOR (
   c    IN OUT INTEGER);
```

Pragmas

```
pragma restrict_references(close_cursor,RNDS,WNDS);
```

Parameters

Table 100–4 CLOSE_CURSOR Procedure Parameters

Parameter	Mode	Description
c	IN	ID number of the cursor that you want to close.
c	OUT	Cursor is set to null. After you call CLOSE_CURSOR, the memory allocated to the cursor is released and you can no longer fetch from that cursor.

COLUMN_VALUE Procedure

This procedure returns the value of the cursor element for a given position in a given cursor. This procedure is used to access the data fetched by calling FETCH_ROWS.

Syntax

```
DBMS_SQL.COLUMN_VALUE (
   c                IN  INTEGER,
   position         IN  INTEGER,
   value            OUT <datatype>
 [,column_error     OUT NUMBER]
 [,actual_length    OUT INTEGER]);
```

Where <datatype> can be any one of the following types:

```
BINARY_DOUBLE
BINARY_FLOAT
BFILE
BLOB
CLOB CHARACTER SET ANY_CS
DATE
DSINTERVAL_UNCONSTRAINED
NUMBER
TIME_TZ_UNCONSTRAINED
TIME_UNCONSTRAINED
TIMESTAMP_LTZ_UNCONSTRAINED
TIMESTAMP_TZ_UNCONSTRAINED
TIMESTAMP_UNCONSTRAINED
UROWID
VARCHAR2 CHARACTER SET ANY_CS
YMINTERVAL_UNCONSTRAINED

<tm_tab>   Time_Table
<ttz_tab>  Time_With_Time_Zone_Table
<tms_tab>  Timestamp_Table
<tstz_tab> Timestamp_With_ltz_Table;
<tstz_tab> Timestamp_With_Time_Zone_Table
<ids_tab>  Interval_Day_To_Second_Table
<iym_tab>  Interval_Year_To_Month_Table
```

Note: The square brackets [ ] indicate optional parameters.

See Also: *Oracle Database Application Developer's Guide - Large Objects*

Pragmas

```
pragma restrict_references(column_value,RNDS,WNDS);
```

The following syntax is also supported for the COLUMN_VALUE procedure:

```
DBMS_SQL.COLUMN_VALUE(
   c                IN  INTEGER,
   position         IN  INTEGER,
   <table_variable>  IN  <datatype>);
```

Where the <table_variable> and its corresponding <datatype> can be any one of these matching pairs:

```
<bdbl_tab>     Binary_Double_Table
<bflt_tab>     Binary_Float_Table
<bfile_tab>    Bfile_Table
<blob_tab>     Blob_Table
<clob_tab>     Clob_Table
<date_tab>     Date_Table
<ids_tab>      Interval_Day_To_Second_Table
<iym_tab>      Interval_Year_To_Month_Table
<num_tab>      Number_Table
<tm_tab>       Time_Table
<ttz_tab>      Time_With_Time_Zone_Table
<tms_tab>      Timestamp_Table
<tstz_tab>     Timestamp_With_ltz_Table;
<tstz_tab>     Timestamp_With_Time_Zone_Table
<urowid_tab>   Urowid_Table
<vchr2_tab>    Varchar2_Table
```

For columns containing CHAR, RAW, and ROWID data, you can use the following variations on the syntax:

```
DBMS_SQL.COLUMN_VALUE_CHAR (
   c               IN  INTEGER,
   position        IN  INTEGER,
   value           OUT CHAR CHARACTER SET ANY_CS
 [,column_error    OUT NUMBER]
 [,actual_length   OUT INTEGER]);

DBMS_SQL.COLUMN_VALUE_RAW (
   c               IN  INTEGER,
   position        IN  INTEGER,
   value           OUT RAW
 [,column_error    OUT NUMBER]
 [,actual_length   OUT INTEGER]);

DBMS_SQL.COLUMN_VALUE_ROWID (
   c               IN  INTEGER,
   position        IN  INTEGER,
   value           OUT ROWID
 [,column_error    OUT NUMBER]
 [,actual_length   OUT INTEGER]);
```

Parameters

Table 100–5 COLUMN_VALUE Procedure Parameters

Parameter	Description
c	ID number of the cursor from which you are fetching the values.
position	Relative position of the column in the cursor. The first column in a statement has position 1.
value	Returns the value at the specified column and row. If the row number specified is greater than the total number of rows fetched, then you receive an error message. Oracle raises exception ORA-06562, inconsistent_type, if the type of this output parameter differs from the actual type of the value, as defined by the call to DEFINE_COLUMN.

Table 100–5 (Cont.) COLUMN_VALUE Procedure Parameters

Parameter	Description
table_ variable	Local variable that has been declared <datatype>.
column_error	Returns any error code for the specified column value.
actual_length	The actual length, before any truncation, of the value in the specified column.

Exceptions

inconsistent_type (ORA-06562) is raised if the type of the given OUT parameter value is different from the actual type of the value. This type was the given type when the column was defined by calling procedure DEFINE_COLUMN.

COLUMN_VALUE_LONG Procedure

This procedure gets part of the value of a long column.

Syntax

```
DBMS_SQL.COLUMN_VALUE_LONG (
   c            IN  INTEGER,
   position     IN  INTEGER,
   length       IN  INTEGER,
   offset       IN  INTEGER,
   value        OUT VARCHAR2,
   value_length OUT INTEGER);
```

Pragmas

```
pragma restrict_references(column_value_long,RNDS,WNDS);
```

Parameters

Table 100–6 COLUMN_VALUE_LONG Procedure Parameters

Parameter	Description
c	Cursor ID number of the cursor from which to get the value.
position	Position of the column of which to get the value.
length	Number of bytes of the long value to fetch.
offset	Offset into the long field for start of fetch.
value	Value of the column as a VARCHAR2.
value_length	Number of bytes actually returned in value.

DEFINE_ARRAY Procedure

This procedure defines the collection for column into which you want to fetch rows (with a FETCH_ROWS call). This procedure lets you do batch fetching of rows from a single SELECT statement. A single fetch call brings over a number of rows into the PL/SQL aggregate object.

When you fetch the rows, they are copied into DBMS_SQL buffers until you run a COLUMN_VALUE call, at which time the rows are copied into the table that was passed as an argument to the COLUMN_VALUE call.

Scalar and LOB Types for Collections

You can declare a local variable as one of the following table-item types, and then fetch any number of rows into it using DBMS_SQL. (These are the same types as you can specify for the BIND_ARRAY procedure.)

```
TYPE binary_double_table
                    IS TABLE OF BINARY_DOUBLE  INDEX BY BINARY_INTEGER;
TYPE binary_float_table
                    IS TABLE OF BINARY_FLOAT   INDEX BY BINARY_INTEGER;
TYPE bfile_table    IS TABLE OF BFILE          INDEX BY BINARY_INTEGER;
TYPE blob_table     IS TABLE OF BLOB           INDEX BY BINARY_INTEGER;
TYPE clob_table     IS TABLE OF CLOB           INDEX BY BINARY_INTEGER;
TYPE date_table     IS TABLE OF DATE           INDEX BY BINARY_INTEGER;
TYPE interval_day_to_second_Table
                    IS TABLE OF dsinterval_unconstrained
                                               INDEX BY BINARY_INTEGER;
TYPE interval_year_to_MONTH_Table
                    IS TABLE OF yminterval_unconstrained
                                               INDEX BY BINARY_INTEGER;
TYPE number_table   IS TABLE OF NUMBER         INDEX BY BINARY_INTEGER;
TYPE time_table     IS TABLE OF time_unconstrained
                                               INDEX BY BINARY_INTEGER;
TYPE time_with_time_zone_table
                    IS TABLE OF time_tz_unconstrained
                                               INDEX BY BINARY_INTEGER;
TYPE timestamp_table
                    IS TABLE OF timestamp_unconstrained
                                               INDEX BY BINARY_INTEGER;
TYPE timestamp_with_ltz_Table
                    IS TABLE OF timestamp_ltz_unconstrained
                                               INDEX BY BINARY_INTEGER;
TYPE timestamp_with_time_zone_Table
                    IS TABLE OF timestamp_tz_unconstrained
                                               INDEX BY BINARY_INTEGER;
TYPE urowid_table   IS TABLE OF UROWID         INDEX BY BINARY_INTEGER;
TYPE varchar2_table IS TABLE OF VARCHAR2(2000) INDEX BY BINARY_INTEGER;

<tm_tab>   Time_Table
<ttz_tab>  Time_With_Time_Zone_Table
<tms_tab>  Timestamp_Table
<tstz_tab> Timestamp_With_ltz_Table;
<tstz_tab> Timestamp_With_Time_Zone_Table
<ids_tab>  Interval_Day_To_Second_Table
<iym_tab>  Interval_Year_To_Month_Table
```

Syntax

```
DBMS_SQL.DEFINE_ARRAY (
   c           IN INTEGER,
   position    IN INTEGER,
   <table_variable>    IN <datatype>
   cnt         IN INTEGER,
   lower_bnd   IN INTEGER);
```

Where <table_variable> and its corresponding <datatype> can be any one of the following matching pairs:

```
<clob_tab>     Clob_Table
<bflt_tab>     Binary_Float_Table
<bdbl_tab>     Binary_Double_Table
<blob_tab>     Blob_Table
<bfile_tab>    Bfile_Table
<date_tab>     Date_Table
<num_tab>      Number_Table
<urowid_tab>   Urowid_Table
<vchr2_tab>    Varchar2_Table
```

Notice that DEFINE_ARRAY is overloaded to accept different datatypes.

Pragmas

```
pragma restrict_references(define_array,RNDS,WNDS);
```

The subsequent FETCH_ROWS call fetch "count" rows. When the COLUMN_VALUE call is made, these rows are placed in positions indx, indx+1, indx+2, and so on. While there are still rows coming, the user keeps issuing FETCH_ROWS/COLUMN_VALUE calls. The rows keep accumulating in the table specified as an argument in the COLUMN_VALUE call.

Parameters

Table 100–7 DEFINE_ARRAY Procedure Parameters

Parameter	Description
c	ID number of the cursor to which you want to bind an array.
position	Relative position of the column in the array being defined. The first column in a statement has position 1.
table_variable	Local variable that has been declared as <datatype>.
cnt	Number of rows that must be fetched.
lower_bnd	Results are copied into the collection, starting at this lower bound index.

The count (cnt) must be an integer greater than zero; otherwise an exception is raised. The indx can be positive, negative, or zero. A query on which a DEFINE_ARRAY call was issued cannot contain array binds.

> **See Also:** "Examples 6 and 7: Defining an Array" on page 100-18 for examples of how to define collections.

DEFINE_COLUMN Procedure

This procedure defines a column to be selected from the given cursor. This procedure is only used with SELECT cursors.

The column being defined is identified by its relative position in the SELECT list of the statement in the given cursor. The type of the COLUMN value determines the type of the column being defined.

Syntax

```
DBMS_SQL.DEFINE_COLUMN (
   c              IN INTEGER,
   position       IN INTEGER,
   column         IN <datatype>)
```

Where <datatype> can be any one of the following types:

```
BINARY_DOUBLE
BINARY_FLOAT
BFILE
BLOB
CLOB CHARACTER SET ANY_CS
DATE
DSINTERVAL_UNCONSTRAINED
NUMBER
TIME_UNCONSTRAINED
TIME_TZ_UNCONSTRAINED
TIMESTAMP_LTZ_UNCONSTRAINED
TIMESTAMP_TZ_UNCONSTRAINED
TIMESTAMP_UNCONSTRAINED
UROWID
VARCHAR2 CHARACTER SET ANY_CS
YMINTERVAL_UNCONSTRAINED
```

Notice that DEFINE_COLUMN is overloaded to accept different datatypes.

> **See Also:** *Oracle Database Application Developer's Guide - Large Objects*

Pragmas

```
pragma restrict_references(define_column,RNDS,WNDS);
```

The following syntax is also supported for the DEFINE_COLUMN procedure:

```
DBMS_SQL.DEFINE_COLUMN (
   c              IN INTEGER,
   position       IN INTEGER,
   column         IN VARCHAR2 CHARACTER SET ANY_CS,
   column_size    IN INTEGER),
   urowid         IN INTEGER;
```

To define columns with CHAR, RAW, and ROWID data, you can use the following variations on the procedure syntax:

```
DBMS_SQL.DEFINE_COLUMN_CHAR (
   c              IN INTEGER,
   position       IN INTEGER,
   column         IN CHAR CHARACTER SET ANY_CS,
```

```
   column_size    IN INTEGER);

DBMS_SQL.DEFINE_COLUMN_RAW (
   c              IN INTEGER,
   position       IN INTEGER,
   column         IN RAW,
   column_size    IN INTEGER);

DBMS_SQL.DEFINE_COLUMN_ROWID (
   c              IN INTEGER,
   position       IN INTEGER,
   column         IN ROWID);
```

Parameters

Table 100–8 DEFINE_COLUMN Procedure Parameters

Parameter	Description
`c`	ID number of the cursor for the row being defined to be selected.
`position`	Relative position of the column in the row being defined. The first column in a statement has position 1.
`column`	Value of the column being defined. The type of this value determines the type for the column being defined.
`column_size`	Maximum expected size of the column value, in bytes, for columns of type `VARCHAR2`, `CHAR`, and `RAW`.

DEFINE_COLUMN_LONG Procedure

This procedure defines a LONG column for a SELECT cursor. The column being defined is identified by its relative position in the SELECT list of the statement for the given cursor. The type of the COLUMN value determines the type of the column being defined.

Syntax

```
DBMS_SQL.DEFINE_COLUMN_LONG (
   c              IN INTEGER,
   position       IN INTEGER);
```

Parameters

Table 100–9 DEFINE_COLUMN_LONG Procedure Parameters

Parameter	Description
c	ID number of the cursor for the row being defined to be selected.
position	Relative position of the column in the row being defined. The first column in a statement has position 1.

DESCRIBE_COLUMNS Procedure

This procedure describes the columns for a cursor opened and parsed through DBMS_SQL.

Syntax

```
DBMS_SQL.DESCRIBE_COLUMNS (
   c              IN  INTEGER,
   col_cnt        OUT INTEGER,
   desc_t         OUT DESC_TAB);
```

Parameters

Table 100–10 DESCRIBE_COLUMNS Procedure Parameters

Parameter	Description
c	ID number of the cursor for the columns being described.
col_cnt	Number of columns in the select list of the query.
desc_t	Table of DESC_REC, each DESC_REC describing a column in the query.

See Also: "Example 8: Describe Columns" on page 100-19 illustrates how to use DESCRIBE_COLUMNS.

DESCRIBE_COLUMNS2 Procedure

This function describes the specified column. This is an alternative to DESCRIBE_COLUMNS Procedure.

Syntax

```
DBMS_SQL.DESCRIBE_COLUMNS2 (
   c              IN  INTEGER,
   col_cnt        OUT INTEGER,
   desc_tab2      OUT DESC_TAB);
```

Pragmas

```
PRAGMA RESTRICT_REFERENCES(describe_columns2,WNDS);
```

Parameters

Table 100–11 DESCRIBE_COLUMNS2 Procedure Parameters

Parameter	Description
c	ID number of the cursor for the columns being described.
col_cnt	Number of columns in the select list of the query.
desc_tab2	The describe table to fill in with the description of each of the columns of the query. This table is indexed from one to the number of elements in the select list of the query.

EXECUTE Function

This function executes a given cursor. This function accepts the `ID` number of the cursor and returns the number of rows processed. The return value is only valid for `INSERT`, `UPDATE`, and `DELETE` statements; for other types of statements, including DDL, the return value is undefined and should be ignored.

Syntax

```
DBMS_SQL.EXECUTE (
   c   IN INTEGER)
  RETURN INTEGER;
```

Parameters

Table 100–12 EXECUTE Function Parameters

Parameter	Description
`c`	Cursor ID number of the cursor to execute.

EXECUTE_AND_FETCH Function

This function executes the given cursor and fetches rows. This function provides the same functionality as calling EXECUTE and then calling FETCH_ROWS. Calling EXECUTE_AND_FETCH instead, however, may reduce the number of network round-trips when used against a remote database.

The EXECUTE_AND_FETCH function returns the number of rows actually fetched.

Syntax

```
DBMS_SQL.EXECUTE_AND_FETCH (
   c              IN INTEGER,
   exact          IN BOOLEAN DEFAULT FALSE)
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(execute_and_fetch,WNDS);
```

Parameters

Table 100–13 EXECUTE_AND_FETCH Function Parameters

Parameter	Description
c	ID number of the cursor to execute and fetch.
exact	Set to TRUE to raise an exception if the number of rows actually matching the query differs from one. Note: Oracle does not support the exact fetch TRUE option with LONG columns. Even if an exception is raised, the rows are still fetched and available.

FETCH_ROWS Function

This function fetches a row from a given cursor. You can call FETCH_ROWS repeatedly as long as there are rows remaining to be fetched. These rows are retrieved into a buffer, and must be read by calling COLUMN_VALUE, for each column, after each call to FETCH_ROWS.

The FETCH_ROWS function accepts the ID number of the cursor to fetch, and returns the number of rows actually fetched.

Syntax

```
DBMS_SQL.FETCH_ROWS (
   c              IN INTEGER)
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(fetch_rows,WNDS);
```

Parameters

Table 100–14 FETCH_ROWS Function Parameters

Parameter	Description
c	ID number.

IS_OPEN Function

This function checks to see if the given cursor is currently open.

Syntax

```
DBMS_SQL.IS_OPEN (
   c              IN INTEGER)
  RETURN BOOLEAN;
```

Pragmas

```
pragma restrict_references(is_open,RNDS,WNDS);
```

Parameters

Table 100–15 IS_OPEN Function Parameters

Parameter	Description
c	Cursor ID number of the cursor to check.

Return Values

Table 100–16 IS_OPEN Function Return Values

Return Value	Description
TRUE	Given cursor is currently open.
FALSE	Given cursor is currently not open.

LAST_ERROR_POSITION Function

This function returns the byte offset in the SQL statement text where the error occurred. The first character in the SQL statement is at position 0.

Syntax

```
DBMS_SQL.LAST_ERROR_POSITION
   RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(last_error_position,RNDS,WNDS);
```

Usage Notes

Call this function after a `PARSE` call, before any other `DBMS_SQL` procedures or functions are called.

LAST_ROW_COUNT Function

This function returns the cumulative count of the number of rows fetched.

Syntax

```
DBMS_SQL.LAST_ROW_COUNT
   RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(last_row_count,RNDS,WNDS);
```

Usage Notes

Call this function after a FETCH_ROWS or an EXECUTE_AND_FETCH call. If called after an EXECUTE call, then the value returned is zero.

LAST_ROW_ID Function

This function returns the ROWID of the last row processed.

Syntax

```
DBMS_SQL.LAST_ROW_ID
   RETURN ROWID;
```

Pragmas

```
pragma restrict_references(last_row_id,RNDS,WNDS);
```

Usage Notes

Call this function after a FETCH_ROWS or an EXECUTE_AND_FETCH call.

LAST_SQL_FUNCTION_CODE Function

This function returns the SQL function code for the statement. These codes are listed in the *Oracle Call Interface Programmer's Guide.*

Syntax

```
DBMS_SQL.LAST_SQL_FUNCTION_CODE
   RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(last_sql_function_code,RNDS,WNDS);
```

Usage Notes

You should call this function immediately after the SQL statement is run; otherwise, the return value is undefined.

OPEN_CURSOR Function

This procedure opens a new cursor. When you no longer need this cursor, you must close it explicitly by calling `CLOSE_CURSOR`.

You can use cursors to run the same SQL statement repeatedly or to run a new SQL statement. When a cursor is reused, the contents of the corresponding cursor data area are reset when the new SQL statement is parsed. It is never necessary to close and reopen a cursor before reusing it.

Syntax

```
DBMS_SQL.OPEN_CURSOR
   RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(open_cursor,RNDS,WNDS);
```

Return Values

This function returns the cursor ID number of the new cursor.

PARSE Procedure

This procedure parses the given statement in the given cursor. All statements are parsed immediately. In addition, DDL statements are run immediately when parsed.

There are two versions of the PARSE procedure: one uses a VARCHAR2 statement as an argument, and the other uses a VARCHAR2S (table of VARCHAR2) as an argument.

Syntax

```
DBMS_SQL.PARSE (
   c                 IN   INTEGER,
   statement         IN   VARCHAR2,
   language_flag     IN   INTEGER);

DBMS_SQL.PARSE (
   c                 IN   INTEGER,
   statement         IN   VARCHAR2A,
   lb                IN   INTEGER,
   ub                IN   INTEGER,
   lfflg             IN   BOOLEAN,
   language_flag     IN   INTEGER);
```

The PARSE procedure also supports the following syntax for large SQL statements:

```
DBMS_SQL.PARSE (
   c                 IN   INTEGER,
   statement         IN   VARCHAR2S,
   lb                IN   INTEGER,
   ub                IN   INTEGER,
   lfflg             IN   BOOLEAN,
   language_flag     IN   INTEGER);
```

> **Note:** The procedure concatenates elements of a PL/SQL table statement and parses the resulting string. You can use this procedure to parse a statement that is longer than the limit for a single VARCHAR2 variable by splitting up the statement.

Parameters

Table 100–17 PARSE Procedure Parameters

Parameter	Description
c	ID number of the cursor in which to parse the statement.
statement	SQL statement to be parsed. Unlike PL/SQL statements, your SQL statement should not include a final semicolon. For example: DBMS_SQL.PARSE(cursor1, 'BEGIN proc; END;', 2); DBMS_SQL.PARSE(cursor1, 'INSERT INTO tab VALUES(1)', 2);
lb	Lower bound for elements in the statement.
ub	Upper bound for elements in the statement.

Table 100–17 (Cont.) PARSE Procedure Parameters

Parameter	Description
lfflg	If TRUE, then insert a linefeed after each element on concatenation.
language_flag	Determines how Oracle handles the SQL statement. The following options are recognized: ■ V6 (or 0) specifies version 6 behavior. ■ NATIVE (or 1) specifies normal behavior for the database to which the program is connected. ■ V7 (or 2) specifies Oracle database version 7 behavior.

Usage Notes

Note: Using DBMS_SQL to dynamically run DDL statements can result in the program hanging. For example, a call to a procedure in a package results in the package being locked until the execution returns to the user side. Any operation that results in a conflicting lock, such as dynamically trying to drop the package before the first lock is released, results in a hang.

The size limit for parsing SQL statements with the preceding syntax is 32KB.

Note: Because client-side code cannot reference remote package variables or constants, you must explicitly use the values of the constants.

For example, the following code does *not* compile on the client:

```
DBMS_SQL.PARSE(cur_hdl, stmt_str, DBMS_SQL.V7); -- uses constant DBMS_SQL.V7
```

The following code works on the client, because the argument is explicitly provided:

```
DBMS_SQL.PARSE(cur_hdl, stmt_str, 2); -- compiles on the client
```

Examples

To parse SQL statements larger than 32 KB, DBMS_SQL makes use of PL/SQL tables to pass a table of strings to the PARSE procedure. These strings are concatenated and then passed on to the Oracle server.

You can declare a local variable as the VARCHAR2S table-item type, and then use the PARSE procedure to parse a large SQL statement as VARCHAR2S.

The definition of the VARCHAR2S datatype is:

```
TYPE varchar2s IS TABLE OF VARCHAR2(256) INDEX BY BINARY_INTEGER;
```

Exceptions

If you create a type/procedure/function/package using DBMS_SQL that has compilation warnings, an ORA-24344 exception is raised, and the procedure is still created.

VARIABLE_VALUE Procedures

This procedure returns the value of the named variable for a given cursor. It is used to return the values of bind variables inside PL/SQL blocks or DML statements with `returning` clause.

Syntax

```
DBMS_SQL.VARIABLE_VALUE (
   c               IN  INTEGER,
   name            IN  VARCHAR2,
   value           OUT <datatype>);
```

Where <datatype> can be any one of the following types:

```
BINARY_DOUBLE
BINARY_FLOAT
BFILE
BLOB
CLOB CHARACTER SET ANY_CS
DATE
DSINTERVAL_UNCONSTRAINED
NUMBER
TIME_TZ_UNCONSTRAINED
TIME_UNCONSTRAINED
TIMESTAMP_LTZ_UNCONSTRAINED
TIMESTAMP_TZ_UNCONSTRAINED
TIMESTAMP_UNCONSTRAINED
UROWID
VARCHAR2 CHARACTER SET ANY_CS
YMINTERVAL_UNCONSTRAINED
```

The following syntax is also supported for the `VARIABLE_VALUE` procedure:

```
DBMS_SQL.VARIABLE_VALUE (
   c                 IN  INTEGER,
   name              IN  VARCHAR2,
   <table_variable>  IN  <datatype>);
```

Where the `<table_variable>` and its corresponding <datatype> can be any one of these matching pairs:

```
<bdbl_tab>     Binary_Double_Table
<bflt_tab>     Binary_Float_Table
<bfile_tab>    Bfile_Table
<blob_tab>     Blob_Table
<clob_tab>     Clob_Table
<date_tab>     Date_Table
<ids_tab>      Interval_Day_To_Second_Table
<iym_tab>      Interval_Year_To_Month_Table
<num_tab>      Number_Table
<tm_tab>       Time_Table
<ttz_tab>      Time_With_Time_Zone_Table
<tms_tab>      Timestamp_Table
<tstz_tab>     Timestamp_With_ltz_Table;
<tstz_tab>     Timestamp_With_Time_Zone_Table
<urowid_tab>   Urowid_Table
<vchr2_tab>    Varchar2_Table
```

For variables containing CHAR, RAW, and ROWID data, you can use the following variations on the syntax:

```
DBMS_SQL.VARIABLE_VALUE_CHAR (
   c               IN  INTEGER,
   name            IN  VARCHAR2,
   value           OUT CHAR CHARACTER SET ANY_CS);

DBMS_SQL.VARIABLE_VALUE_RAW (
   c               IN  INTEGER,
   name            IN  VARCHAR2,
   value           OUT RAW);

DBMS_SQL.VARIABLE_VALUE_ROWID (
   c               IN  INTEGER,
   name            IN  VARCHAR2,
   value           OUT ROWID);
```

Pragmas

```
pragma restrict_references(variable_value,RNDS,WNDS);
```

Parameters

Table 100–18 VARIABLE_VALUE Procedure Parameters

Parameter	Description
c	ID number of the cursor from which to get the values.
name	Name of the variable for which you are retrieving the value.
value	Returns the value of the variable for the specified position. Oracle raises exception ORA-06562, inconsistent_type, if the type of this output parameter differs from the actual type of the value, as defined by the call to BIND_VARIABLE.
position	Relative position of the column in the cursor. The first column in a statement has position 1.

101

DBMS_SQLTUNE

The DBMS_SQLTUNE package provides the interface to tune SQL statements.

The chapter contains the following topics:

Using DBMS_SQLTUNE

Overview

The DBMS_SQLTUNE package provides three interrelated areas of functionality:

- SQL Tuning Advisor Subprograms
- SQL Profile Subprograms
- SQL Tuning Set Subprograms

SQL Tuning Advisor

The SQL Tuning Advisor is one of a suite of Advisors, a set of expert systems that identifies and helps resolve database performance problems. Specifically, the SQL Tuning Advisor automates the tuning process of problematic SQL statements. That is, it takes one or more SQL statements as input and gives precise advice on how to tune the statements. The advice is provided is in the form of precise SQL actions for tuning the SQL along with their expected performance benefit.

The group of SQL Tuning Advisor Subprograms provide a task-oriented interface that lets you access the Advisor. You can call the following subprograms in the order given to use some of the SQL Tuning Advisor's features:

1. You use the CREATE_TUNING_TASK Functions to create a tuning task for tuning a single statement or a group of SQL statements.
2. The EXECUTE_TUNING_TASK Procedure executes a previously created tuning task.
3. The REPORT_TUNING_TASK Function displays the results of a tuning task.
4. You use the SCRIPT_TUNING_TASK Function to create a SQL*PLUS script which can then be executed to implement a set of Advisor recommendations

SQL Profile Subprograms

The SQL Tuning Advisor may recommend the creation of a SQL Profile to improve the performance of a statement. SQL Profiles consist of auxiliary statistics specific to the statement. The query optimizer makes estimates about cardinality, selectivity, and cost that can sometimes be off by a significant amount, resulting in poor execution plans. The SQL Profile addresses this problem by collecting additional information using sampling and partial execution techniques to adjust these estimates.

The group of SQL Profile Subprograms provides a mechanism for delivering statistics to the optimizer that targets one particular SQL statement, and helps the optimizer make good decisions for that statement by giving it the most accurate statistical information possible. For example:

- You can use the ACCEPT_SQL_PROFILE Procedure and Function to accept a SQL Profile recommended by the SQL Tuning Advisor.
- You can alter the STATUS, NAME, DESCRIPTION, and CATEGORY attributes of an existing SQL Profile with the ALTER_SQL_PROFILE Procedure.
- You can drop a SQL Profile with the DROP_SQL_PROFILE Procedure.

SQL Tuning Sets

The SQL Tuning Advisor input can be a single SQL statement or a set of statements. When tuning multiple statements in one advisor task, you give the input in the form of a SQL Tuning Set (STS). A SQL Tuning Set is a database object that stores SQL statements along with their execution context in a system-provided schema. SQL

Tuning Sets provide an infrastructure for dealing with SQL workloads and simplify tuning of a large number of SQL statements.

SQL Tuning Sets store SQL statements along with

- The execution context, such as the parsing schema name and bind values
- Execution statistics such as average elapsed time and execution count
- Execution plans - which are the sequence of operations Oracle performs to run SQL statements
- Row source statistics such as the number of rows processed for each operation executed within the plan

SQL Tuning Sets can be created by filtering or ranking SQL statements from several sources:

- The cursor cache using the SELECT_CURSOR_CACHE Function
- Top SQL statements from the Automatic Workload Repository using the SELECT_WORKLOAD_REPOSITORY Functions
- Other SQL Tuning Sets using the SELECT_SQLSET Function
- A user-defined workload

The complete group of SQL Tuning Set Subprograms facilitates this functionality. As examples:

- You use the CREATE_SQLSET Procedure and Function to creates a SQL tuning set object in the database
- The LOAD_SQLSET Procedure populates the SQL tuning set with a set of selected SQL
- The CAPTURE_CURSOR_CACHE_SQLSET Procedure collects SQL statements from the cursor cache over a specified time interval, attempting to build a realistic picture of system workload.

Import/Export SQL Tuning Sets and SQL Profiles

You use `DBMS_SQLTUNE` subprograms to move SQL Profiles and SQL Tuning Sets from one system to another using a common programmatic model. In both cases, you create a staging table on the source system and populate that staging table with the relevant data. You then move that staging table to the destination system following the method of your choice (such as datapump, import/export, or database link), where it is used to reconstitute the objects in their original form. These steps are implemented by means of subprograms included in this package:

1. Call the CREATE_STGTAB_SQLPROF Procedure or the CREATE_STGTAB_SQLSET Procedure to create the staging table on the source system.
2. Call the PACK_STGTAB_SQLPROF Procedure or PACK_STGTAB_SQLSET Procedure to populate the staging table with information from the source system.
3. Once you have moved the staging table to the destination system, you call the UNPACK_STGTAB_SQLPROF Procedure or the UNPACK_STGTAB_SQLSET Procedure to recreate the object on the new system.

> **See Also:** Oracle Database Performance Tuning Guide for more information about programmatic flow.

Security Model

This package is available to PUBLIC and performs its own security checking:

- As SQL tuning advisor relies on the advisor framework, so all tuning task interfaces (XXX_TUNING_TASK) require privilege ADVISOR.
- SQL Tuning Set subprograms (XXX_SQLSET) require either the ADMINISTER SQL TUNING SET or the ADMINISTER ANY SQL TUNING SET privilege. Users having the ADMINISTER SQL TUNING SET privilege can only create and modify a SQL tuning set they own, while the ADMINISTER ANY SQL TUNING SET privilege allows them to operate upon all SQL tuning sets, even those owned by other users. For example, using the CREATE_SQLSET Procedure and Function you can create a SQL tuning set to be owned by another user. In this case, the user need not necessarily have the ADMINISTER SQL TUNING SET privilege to operate upon her tuning set.
- Three different privileges are needed to invoke subprograms concerned with SQL Profiles:
  - CREATE ANY SQL PROFILE is needed to call the ACCEPT_SQL_PROFILE Procedure and Function
  - ALTER ANY SQL PROFILE is needed to call the ALTER_SQL_PROFILE Procedure
  - DROP ANY SQL PROFILE is needed to call the DROP_SQL_PROFILE Procedure

Data Structures

The DBMS_SQLTUNE package defines the following OBJECT type

Object Types

- SQLSET_ROW Object Type

SQLSET_ROW Object Type

The SQLSET_ROW object models the content of a SQL Tuning Set for the user. Logically, a SQL Tuning Set is a collection of SQLSET_ROWs where each SQLSET_ROW contains a single SQL statement along with its execution context, statistics, binds and plan. The SELECT_XXX subprograms each model a data source as a collection of SQLSET_ROWs, unique by (sql_id, plan_hash_value). Similarly, the LOAD_SQLSET procedure takes as input a cursor whose row type is SQLSET_ROW, treating each SQLSET_ROW in isolation according to the policies requested by the user.

Several subprograms in the DBMS_SQLTUNE package accept basic filters on the content of a SQL tuning set or data source. These filters are expressed in terms of the attributes within the SQLSET_ROW as defined.

Syntax

```
CREATE TYPE sqlset_row AS object (
  sql_id                   VARCHAR(13),
  force_matching_signature NUMBER,
  sql_text                 CLOB,
  object_list              sql_objects,
  bind_data                RAW(2000),
  parsing_schema_name      VARCHAR2(30),
  module                   VARCHAR2(48),
  action                   VARCHAR2(32),
  elapsed_time             NUMBER,
  cpu_time                 NUMBER,
  buffer_gets              NUMBER,
  disk_reads               NUMBER,
  direct_writes            NUMBER,
  rows_processed           NUMBER,
  fetches                  NUMBER,
  executions               NUMBER,
  end_of_fetch_count       NUMBER,
  optimizer_cost           NUMBER,
  optimizer_env            RAW(1000),
  priority                 NUMBER,
  command_type             NUMBER,
  first_load_time          VARCHAR2(19),
  stat_period              NUMBER,
  active_stat_period       NUMBER,
  other                    CLOB,
  plan_hash_value          NUMBER,
  sql_plan                 sql_plan_table_type,
  bind_list                sql_binds)
```

Attributes

Table 101–1 SQLSET_ROW Attributes

Attribute	Description
sql_id	Unique SQL ID
forcing_matching_signature	Signature with literals, case, and whitespace removed
sql_text	Full text for the statement
object_list	Currently not implemented

Table 101–1 (Cont.) SQLSET_ROW Attributes

Attribute	Description
bind_data	Bind data as captured for this SQL. Note that you cannot stipulate an argument for this parameter and also for bind_list - they are mutually exclusive.
parsing_schema	Schema where the SQL is parsed
module	Last application module for the SQL
action	Last application action for the SQL
elapsed_time	Sum total elapsed time for this SQL statement
cpu_time	Sum total CPU time for this SQL statement
buffer_gets	Sum total number of buffer gets
disk_reads	Sum total number of disk reads
direct_writes	Sum total number of direct writes
rows_processed	Sum total number of rows processed by this SQL
fetches	Sum total number of fetches
executions	Total executions of this SQL
end_of_fetch_count	Number of times the statement was fully executed with all of its rows fetched
optimizer_cost	Optimizer cost for this SQL
optimizer_env	Optimizer environment for this SQL statement
priority	User-defined priority (1,2,3)
command_type	Statement type, such as INSERT or SELECT.
first_load_time	Load time of parent cursor
stat_period	Period of time (seconds) when the statistics of this SQL statement were collected
active_stat_period	Effective period of time (in seconds) during which the SQL statement was active
other	Other column for user defined attributes
plan_hash_value	Plan hash value of the plan
sql_plan	Explain plan
bind_list	List of user specified binds for SQL This is used for user-specified workloads. Note that you cannot stipulate an argument for this parameter and also for bind_data - they are mutually exclusive.

Subprogram Groups

DBMS_SQLTUNE subprograms are grouped by function:

- SQL Tuning Advisor Subprograms
- SQL Profile Subprograms
- SQL Tuning Set Subprograms

SQL Tuning Advisor Subprograms

This subprogram group provides an interface to manage SQL tuning tasks.

Table 101–2 SQL Tuning Task Subprograms

Subprogram	Description
CANCEL_TUNING_TASK Procedure on page 101-22	Cancels the currently executing tuning task
CREATE_TUNING_TASK Functions on page 101-28	Creates a tuning of a single statement or SQL tuning set
DROP_TUNING_TASK Procedure on page 101-35	Drops a SQL tuning task
EXECUTE_TUNING_TASK Procedure on page 101-36	Executes a previously created tuning task
INTERRUPT_TUNING_TASK Procedure on page 101-37	Interrupts the currently executing tuning task
REPORT_TUNING_TASK Function on page 101-48	Displays the results of a tuning task
RESET_TUNING_TASK Procedure on page 101-49	Resets the currently executing tuning task to its initial state
RESUME_TUNING_TASK Procedure on page 101-50	Resumes a previously interrupted task that was created to tune a SQL tuning set.
SCRIPT_TUNING_TASK Function on page 101-51	Creates a SQL*PLUS script which can then be executed to implement a set of Advisor recommendations

The Summary of DBMS_SQLTUNE Subprograms contains a complete listing of all subprograms in the package.

SQL Profile Subprograms

This subprogram group provides an interface to manage SQL Profiles.

Table 101–3 SQL Profile Subprograms

Subprogram	Description
ACCEPT_SQL_PROFILE Procedure and Function on page 101-16	Creates a SQL Profile for the specified tuning task
ALTER_SQL_PROFILE Procedure on page 101-20	Alters specific attributes of an existing SQL Profile object
CREATE_STGTAB_SQLPROF Procedure on page 101-26	Creates the staging table used for copying SQL profiles from one system to another.
DROP_SQL_PROFILE Procedure on page 101-33	Drops the named SQL Profile from the database
PACK_STGTAB_SQLPROF Procedure on page 101-42	Moves profile data out of the `SYS` schema into the staging table
REMAP_STGTAB_SQLPROF Procedure on page 101-45	Changes the profile data values kept in the staging table prior to performing an unpack operation
SQLTEXT_TO_SIGNATURE Function on page 101-61	Returns a SQL text's signature
UNPACK_STGTAB_SQLPROF Procedure on page 101-62	Uses the profile data stored in the staging table to create profiles on this system

The Summary of DBMS_SQLTUNE Subprograms contains a complete listing of all subprograms in the package.

SQL Tuning Set Subprograms

This subprogram group provides an interface to manage SQL tuning sets.

Table 101–4 SQL Tuning Set Subprograms

Subprogram	Description
ADD_SQLSET_REFERENCE Function on page 101-19	Adds a new reference to an existing SQL tuning set to indicate its use by a client
CAPTURE_CURSOR_CACHE_SQLSET Procedure on page 101-23	Over a specified time interval incrementally captures a workload from the cursor cache into a SQL tuning set
CREATE_SQLSET Procedure and Function on page 101-25	Creates a SQL tuning set object in the database
CREATE_STGTAB_SQLSET Procedure on page 101-27	Creates a staging table through which SQL Tuning Sets are imported and exported
DELETE_SQLSET Procedure on page 101-32	Deletes a set of SQL statements from a SQL tuning set
DROP_SQLSET Procedure on page 101-34	Drops a SQL tuning set if it is not active
LOAD_SQLSET Procedure on page 101-38	Populates the SQL tuning set with a set of selected SQL
PACK_STGTAB_SQLSET Procedure on page 101-43	Copies tuning sets out of the `SYS` schema into the staging table
REMOVE_SQLSET_REFERENCE Procedure on page 101-47	Deactivates a SQL tuning set to indicate it is no longer used by the client
SELECT_CURSOR_CACHE Function on page 101-53	Collects SQL statements from the cursor cache
SELECT_SQLSET Function on page 101-57	Collects SQL statements from an existing SQL tuning set
SELECT_WORKLOAD_REPOSITORY Functions on page 101-59	Collects SQL statements from the workload repository
UNPACK_STGTAB_SQLSET Procedure on page 101-63	Copies one or more SQL tuning sets from the staging table
UPDATE_SQLSET Procedures on page 101-65	Updates whether selected string fields for a SQL statement in a SQL tuning set or the set numerical attributes of a SQL in a SQL tuning set

The Summary of DBMS_SQLTUNE Subprograms contains a complete listing of all subprograms in the package.

Summary of DBMS_SQLTUNE Subprograms

Table 101–5 DBMS_SQLTUNE Package Subprograms

Subprogram	Description	Group
ACCEPT_SQL_PROFILE Procedure and Function on page 101-16	Create a SQL Profile for the specified tuning task	SQL Profile Subprograms on page 101-11
ADD_SQLSET_REFERENCE Function on page 101-19	Adds a new reference to an existing SQL tuning set to indicate its use by a client	SQL Tuning Set Subprograms on page 101-12
ALTER_SQL_PROFILE Procedure on page 101-20	Alters specific attributes of an existing SQL Profile object	SQL Profile Subprograms on page 101-11
CANCEL_TUNING_TASK Procedure on page 101-22	Cancels the currently executing tuning task	SQL Tuning Advisor Subprograms on page 101-10
CAPTURE_CURSOR_CACHE_SQLSET Procedure on page 101-23	Over a specified time interval incrementally captures a workload from the cursor cache into a SQL tuning set	SQL Tuning Set Subprograms on page 101-12
CREATE_SQLSET Procedure and Function on page 101-25	Creates a SQL tuning set object in the database	SQL Tuning Set Subprograms on page 101-12
CREATE_STGTAB_SQLPROF Procedure on page 101-26	Creates the staging table used for copying SQL profiles from one system to another.	SQL Profile Subprograms on page 101-11
CREATE_STGTAB_SQLSET Procedure on page 101-27	Creates a staging table through which SQL Tuning Sets are imported and exported	SQL Tuning Set Subprograms on page 101-12
CREATE_TUNING_TASK Functions on page 101-28	Creates a tuning of a single statement or SQL tuning set	SQL Tuning Advisor Subprograms on page 101-10
DELETE_SQLSET Procedure on page 101-32	Deletes a set of SQL statements from a SQL tuning set	SQL Tuning Set Subprograms on page 101-12
DROP_SQL_PROFILE Procedure on page 101-33	Drops the named SQL Profile from the database	SQL Profile Subprograms on page 101-11
DROP_SQLSET Procedure on page 101-34	Drops a SQL tuning set if it is not active	SQL Tuning Set Subprograms on page 101-12
DROP_TUNING_TASK Procedure on page 101-35	Drops a SQL tuning task	SQL Tuning Advisor Subprograms on page 101-10
EXECUTE_TUNING_TASK Procedure on page 101-36	Executes a previously created tuning task	SQL Tuning Advisor Subprograms on page 101-10

Table 101–5 (Cont.) DBMS_SQLTUNE Package Subprograms

Subprogram	Description	Group
INTERRUPT_TUNING_TASK Procedure on page 101-37	Interrupts the currently executing tuning task	SQL Tuning Advisor Subprograms on page 101-10
LOAD_SQLSET Procedure on page 101-38	Populates the SQL tuning set with a set of selected SQL	SQL Tuning Set Subprograms on page 101-12
PACK_STGTAB_SQLPROF Procedure on page 101-42	Moves profile data out of the SYS schema into the staging table	SQL Profile Subprograms on page 101-11
PACK_STGTAB_SQLSET Procedure on page 101-43	Moves tuning sets out of the SYS schema into the staging table	SQL Tuning Set Subprograms on page 101-12
REMAP_STGTAB_SQLPROF Procedure on page 101-45	Changes the profile data values kept in the staging table prior to performing an unpack operation	SQL Profile Subprograms on page 101-11
REMAP_STGTAB_SQLSET Procedure on page 101-46	Changes the tuning set names and owners in the staging table so that they can be unpacked with different values than they had on the host system	SQL Tuning Set Subprograms on page 101-12
REMOVE_SQLSET_ REFERENCE Procedure on page 101-47	Deactivates a SQL tuning set to indicate it is no longer used by the client	SQL Tuning Set Subprograms on page 101-12
REPORT_TUNING_TASK Function on page 101-48	Displays the results of a tuning task	SQL Tuning Set Subprograms on page 101-12
RESET_TUNING_TASK Procedure on page 101-49	Resets the currently executing tuning task to its initial state	SQL Tuning Advisor Subprograms on page 101-10
RESUME_TUNING_TASK Procedure on page 101-50	Resumes a previously interrupted task that was created to tune a SQL tuning set.	SQL Tuning Advisor Subprograms on page 101-10
SCRIPT_TUNING_TASK Function on page 101-51	Creates a SQL*PLUS script which can then be executed to implement a set of Advisor recommendations	SQL Tuning Advisor Subprograms on page 101-10
SELECT_CURSOR_CACHE Function on page 101-53	Collects SQL statements from the cursor cache	SQL Tuning Set Subprograms on page 101-12
SELECT_SQLSET Function on page 101-57	Collects SQL statements from an existing SQL tuning set	SQL Tuning Set Subprograms on page 101-12
SELECT_WORKLOAD_ REPOSITORY Functions on page 101-59	Collects SQL statements from the workload repository	SQL Tuning Set Subprograms on page 101-12
SQLTEXT_TO_SIGNATURE Function on page 101-61	Returns a SQL text's signature	SQL Profile Subprograms on page 101-11

Table 101–5 (Cont.) DBMS_SQLTUNE Package Subprograms

Subprogram	Description	Group
UNPACK_STGTAB_SQLPROF Procedure on page 101-62	Uses the profile data stored in the staging table to create profiles on this system	SQL Profile Subprograms on page 101-11
UNPACK_STGTAB_SQLSET Procedure on page 101-63	Moves one or more SQL tuning sets from the staging table	SQL Tuning Set Subprograms on page 101-12
UPDATE_SQLSET Procedures on page 101-65	Updates selected fields for a SQL statement in a SQL tuning set	SQL Tuning Set Subprograms on page 101-12

ACCEPT_SQL_PROFILE Procedure and Function

This procedure creates a SQL Profile recommended by the SQL Tuning Advisor. The SQL text is normalized for matching purposes though it is stored in the data dictionary in de-normalized form for readability. SQL text is provided through a reference to the SQL Tuning task. If the referenced SQL statement doesn't exist, an error is reported.

> **See Also:** SQL Profile Subprograms on page 101-11 for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.ACCEPT_SQL_PROFILE (
   task_name    IN  VARCHAR2,
   object_id    IN  NUMBER   := NULL,
   name         IN  VARCHAR2 := NULL,
   description  IN  VARCHAR2 := NULL,
   category     IN  VARCHAR2 := NULL);
   task_owner   IN VARCHAR2  := NULL,
   replace      IN BOOLEAN   := FALSE,
   force_match  IN BOOLEAN   := FALSE);

DBMS_SQLTUNE.ACCEPT_SQL_PROFILE (
   task_name    IN  VARCHAR2,
   object_id    IN  NUMBER   := NULL,
   name         IN  VARCHAR2 := NULL,
   description  IN  VARCHAR2 := NULL,
   category     IN  VARCHAR2 := NULL;
   task_owner   IN VARCHAR2  := NULL,
   replace      IN BOOLEAN   := FALSE,
   force_match  IN BOOLEAN   := FALSE)
 RETURN VARCHAR2;
```

Parameters

Table 101–6 ACCEPT_SQL_PROFILE Procedure and Function Parameters

Parameter	Description
task_name	The (mandatory) name of the SQL tuning task
object_id	The identifier of the advisor framework object representing the SQL statement associated with the tuning task
name	The name of the SQL Profile. It cannot contain double quotation marks. The name is case sensitive. If not specified, the system will generate a unique name for the SQL Profile.
description	A user specified string describing the purpose of the SQL Profile. The description is truncated if longer than 256 characters. The maximum size is 500 characters.
category	This is the category name which must match the value of the SQLTUNE_CATEGORY parameter in a session for the session to use this SQL Profile. It defaults to the value "DEFAULT". This is also the default of the SQLTUNE_CATEGORY parameter. The category must be a valid Oracle identifier. The category name specified is always converted to upper case. The combination of the normalized SQL text and category name creates a unique key for a SQL Profile. An ACCEPT_SQL_PROFILE will fail if this combination is duplicated.

Table 101–6 (Cont.) ACCEPT_SQL_PROFILE Procedure and Function Parameters

Parameter	Description
task_owner	Owner of the tuning task. This is an optional parameter that has to be specified to accept a SQL Profile associated to a tuning task owned by another user. The current user is the default value.
replace	If the profile already exists, it will be replaced if this argument is TRUE. It is an error to pass a name that is already being used for another signature/category pair, even with replace set to TRUE.
force_match	If TRUE this causes SQL Profiles to target all SQL statements which have the same text after normalizing all literal values into bind variables. (Note that if a combination of literal values and bind values is used in a SQL statement, no bind transformation occurs.) This is analogous to the matching algorithm used by the FORCE option of the cursor_sharing parameter. If FALSE, literals are not transformed. This is analogous to the matching algorithm used by the EXACT option of the cursor_sharing parameter.

Return Values

The name of the SQL profile.

Usage Notes

The CREATE ANY SQL PROFILE privilege is required.

Examples

You use both the procedure and the function versions of the subprogram in the same way except you must specify a return value to invoke the function. Here we give examples of the procedure only.

In this example, you tune a single SQL statement form the workload repository and you create the SQL profile recommended by SQL tuning advisor.

```
variable stmt_task VARCHAR2(64);
variable sts_task VARCHAR2(64);

-- create a tuning task tune the statement
EXEC :stmt_task := DBMS_SQLTUNE.CREATE_TUNING_TASK(
   begin_snap  => 1, -
   end_snap    => 2, -
   sql_id      => 'ay1m3ssvtrh24');

-- execute the resulting task
EXEC DBMS_SQLTUNE.EXECUTE_TUNING_TASK(:stmt_task);

EXEC DBMS_SQLTUNE.ACCEPT_SQL_PROFILE(:stmt_task);
```

Note that you do not have to specify the ID (that is, object_id) for the advisor framework object created by SQL tuning advisor to represent the tuned SQL statement.

You might also want to accept the recommended SQL profile in a different category, (for example, TEST), so that it will not be used by default.

```
EXEC DBMS_SQLTUNE.ACCEPT_SQL_PROFILE (
   task_name  =>  :stmt_task, -
   category   =>  'TEST');
```

You can use command `ALTER SESSION SET SQLTUNE_CATEGORY = 'TEST'` to see how this profile behaves.

The following call creates a SQL profile that targets any SQL statement with the same `force_matching_signature` as the tuned statement.

```
EXEC DBMS_SQLTUNE.ACCEPT_SQL_PROFILE (task_name   =>  :stmt_task, -
                                      force_match =>  TRUE);
```

In the following example, you tune a SQL tuning set, and you create a SQL profile for only one of the SQL statements in the SQL tuning set. The SQL statement is represented by an advisor framework object with ID equal to '5'. Please notice that you must pass an object id to the `ACCEPT_SQL_PROFILE` procedure because there are potentially many SQL profiles for the tuning task. This object id is given along with the report.

```
EXEC :sts_task := DBMS_SQLTUNE.CREATE_TUNING_TASK ( -
  sqlset_name   => 'my_workload',  -
  rank1         => 'ELAPSED_TIME', -
  time_limit    => 3600,           -
  description   => 'my workload ordered by elapsed time');

-- execute the resulting task
EXEC DBMS_SQLTUNE.EXECUTE_TUNING_TASK(:sts_task);

 -- create the profile for the sql statement corresponding to object_id = 5.
EXEC DBMS_SQLTUNE.ACCEPT_SQL_PROFILE (
   task_name  =>  :sts_task, -
   object_id  =>  5);
```

ADD_SQLSET_REFERENCE Function

This procedure adds a new reference to an existing SQL tuning set to indicate its use by a client.

> **See Also:** SQL Tuning Set Subprograms on page 101-12 for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.ADD_SQLSET_REFERENCE (
   sqlset_name  IN  VARCHAR2,
   description  IN  VARCHAR2 := NULL)
 RETURN NUMBER;
```

Parameters

Table 101–7 ADD_SQLSET_REFERENCE Function Parameters

Parameter	Description
sqlset_name	The SQL tuning set name
description	The description of the usage of SQL tuning set. The description is truncated if longer than 256 characters.

Return Values

The identifier of the added reference.

Examples

You can add reference to a SQL tuning set. This prevents the tuning set from being modified while it is being used. References are automatically added when you invoke SQL tuning advisor on the SQL tuning set, so you should use this function for custom purposes only.

The function returns a reference ID that is used to remove it later. You use the REMOVE_SQLSET_REFERENCE Procedure to delete references to a SQL tuning set.

```
variable rid number;
EXEC :rid := DBMS_SQLTUNE.ADD_SQLSET_REFERENCE( -
                                sqlset_name => 'my_workload', -
                                description => 'my sts reference');
```

You can use the views USER/DBA_SQLSET_REFERENCES to find all references on a given SQL tuning set.

ALTER_SQL_PROFILE Procedure

This procedure alters specific attributes of an existing SQL Profile object. The following attributes can be altered (using these attribute names):

- "STATUS" can be set to "ENABLED" or "DISABLED"
- "NAME" can be reset to a valid name which must be a valid Oracle identifier and must be unique.
- "DESCRIPTION" can be set to any string of size no more than 500 characters
- "CATEGORY" can be reset to a valid category name which must be a valid Oracle identifier and must be unique when combined with normalized SQL text)

See Also: SQL Profile Subprograms on page 101-11 for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.ALTER_SQL_PROFILE (
   name                 IN  VARCHAR2,
   attribute_name       IN  VARCHAR2,
   value                IN  VARCHAR2);
```

Parameters

Table 101–8 ALTER_SQL_PROFILE Procedure Parameters

Parameter	Description
name	The (mandatory) name of the existing SQL Profile to alter
attribute_name	The (mandatory) attribute name to alter (case insensitive) using valid attribute names
value	The (mandatory) new value of the attribute using valid attribute values

Usage Notes

Requires the "ALTER ANY SQL PROFILE" privilege.

Examples

```
-- Disable a profile, so it will be not be used by any sessions.
EXEC DBMS_SQLTUNE.ALTER_SQL_PROFILE (name             =>  :pname,   -
                                     attribute_name   =>  'STATUS', -
                                     value            =>  'DISABLED');

-- Enable it back:
EXEC DBMS_SQLTUNE.ALTER_SQL_PROFILE ( name            =>   :pname,   -
                                      attribute_name  =>   'STATUS', -
                                      value           =>   'ENABLED');

-- Change the category of the profile so it will be used only by sessions
-- with category set to TEST.
-- Use ALTER SESSION SET SQLTUNE_CATEGORY = 'TEST' to see how this profile
-- behaves.
EXEC DBMS_SQLTUNE.ALTER_SQL_PROFILE ( name            =>  :pname,   -
                                      attribute_name  =>  'CATEGORY', -
                                      value           =>  'TEST');
```

```
-- Change it back:
EXEC DBMS_SQLTUNE.ALTER_SQL_PROFILE ( name            =>  :pname,   -
                                      attribute_name  =>  'CATEGORY', -
                                      value           =>  'DEFAULT');
```

CANCEL_TUNING_TASK Procedure

This procedure cancels the currently executing tuning task. All intermediate result data is deleted.

> **See Also:** SQL Tuning Advisor Subprograms on page 101-10 for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.CANCEL_TUNING_TASK(
 task_name          IN VARCHAR2);
```

Parameters

Table 101–9 CANCEL_TUNING_TASK Procedure Parameters

Parameter	Description
task_name	The name of the task to cancel

Examples

You cancel a task when you need to stop it executing and do not require to view any already-completed results.

```
EXEC DBMS_SQLTUNE.CANCEL_TUNING_TASK(:my_task);
```

CAPTURE_CURSOR_CACHE_SQLSET Procedure

Over a specified time interval this procedure incrementally captures a workload from the cursor cache into a SQL tuning set. The procedure captures a workload from the cursor cache into a SQL tuning set, polling the cache multiple times over a time period and updating the workload data stored there. It can execute over as long a period as required to capture an entire system workload.

> **See Also:** SQL Tuning Set Subprograms on page 101-12 for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.CAPTURE_CURSOR_CACHE_SQLSET (
    sqlset_name         IN VARCHAR2,
    time_limit          IN POSITIVE := 1800,
    repeat_interval     IN POSITIVE := 300,
    capture_option      IN VARCHAR2 := 'MERGE',
    capture_mode        IN NUMBER   := MODE_REPLACE_OLD_STATS,
    basic_filter        IN VARCHAR2 := NULL,
    sqlset_owner        IN VARCHAR2 := NULL);
```

Parameters

Table 101–10 CAPTURE_CURSOR_CACHE_SQLSET Procedure Parameters

Parameter	Description
sqlset_name	The SQL tuning set name
time_limit	The total amount of time, in seconds, to execute
repeat_interval	The amount of time, in seconds, to pause between sampling
capture_option	During capture, either insert new statements, update existing ones, or both. 'INSERT', 'UPDATE', or 'MERGE' just like load_option in load_sqlset
capture_mode	capture mode (UPDATE and MERGE capture options).Possible values: ■ MODE_REPLACE_OLD_STATS - Replace statistics when the number of executions seen is greater than that stored in the SQL tuning set ■ MODE_ACCUMULATE_STATS - Add new values to current values for SQL we already store. Note that this mode detects if a statement has been aged out, so the final value for a statistics will be the sum of the statistics of all cursors that statement existed under.
basic_filter	Filter to apply to cursor cache on each sampling (see select_xxx subprograms)
sqlset_owner	The owner of the SQL tuning set or NULL for current schema owner

Examples

In this example capture takes place over a 30-second period, polling the cache once every five seconds. This will capture all statements run during that period but not before or after. If the same statement appears a second time, the process replaces the stored statement with the new occurence.

Note that in production systems the time limit and repeat interval would be set much higher. You should tune the `time_limit` and `repeat_interval` parameters based on the workload time and cursor cache turnover properties of your system.

```
EXEC DBMS_SQLTUNE.CAPTURE_CURSOR_CACHE_SQLSET( -
                                      sqlset_name     => 'my_workload', -
                                      time_limit      =>  30, -
                                      repeat_interval =>  5);
```

In the following call you accumulate execution statistics as you go. This option produces an accurate picture of the cumulative activity of each cursor, even across age-outs, but it is more expensive than the previous example.

```
EXEC DBMS_SQLTUNE.CAPTURE_CURSOR_CACHE_SQLSET( -
                        sqlset_name     => 'my_workload', -
                        time_limit      => 30, -
                        repeat_interval => 5, -
                        capture_mode    => dbms_sqltune.MODE_ACCUMULATE_STATS);
```

This call performs a very inexpensive capture where you only insert new statements and do not update their statistics once they have been inserted into the SQL tuning set

```
EXEC DBMS_SQLTUNE.CAPTURE_CURSOR_CACHE_SQLSET( -
                                    sqlset_name     => 'my_workload', -
                                    time_limit      => 30, -
                                    repeat_interval => 5, -
                                    capture_option  => 'INSERT');
```

CREATE_SQLSET Procedure and Function

The procedure creates a SQL tuning set object in the database.

The function causes the system t o generate a name for the SQL Tuning Set.

> **See Also:** SQL Tuning Set Subprograms on page 101-12 for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.CREATE_SQLSET (
   sqlset_name  IN  VARCHAR2,
   description  IN  VARCHAR2 := NULL
   sqlset_owner IN  VARCHAR2 := NULL);

DBMS_SQLTUNE.CREATE_SQLSET (
   sqlset_name  IN  VARCHAR2 := NULL,
   description  IN  VARCHAR2 := NULL,
   sqlset_owner IN  VARCHAR2 := NULL)
 RETURN VARCHAR2;
```

Parameters

Table 101–11 CREATE_SQLSET Procedure Parameters

Parameter	Description
`sqlset_name`	The SQL tuning set name
`description`	The description of the SQL tuning set
`sqlset_owner`	The owner of the SQL tuning set, or `NULL` for the current schema owner

Examples

```
EXEC DBMS_SQLTUNE.CREATE_SQLSET(-
  sqlset_name => 'my_workload', -
  description => 'complete application workload');
```

CREATE_STGTAB_SQLPROF Procedure

This procedure creates the staging table used for copying SQL profiles from one system to another.

> **See Also:** SQL Profile Subprograms on page 101-11 for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.CREATE_STGTAB_SQLPROF (
   table_name            IN VARCHAR2,
   schema_name           IN VARCHAR2 := NULL,
   tablespace_name       IN VARCHAR2 := NULL);
```

Parameters

Table 101–12 CREATE_STGTAB_SQLPROF Procedure Parameters

Parameter	Description
`table_name`	The name of the table to create (case-sensitive). Required.
`schema_name`	The schema to create the table in, or `NULL` for current schema (case-sensitive)
`tablespace_name`	The tablespace to store the staging table within, or `NULL` for current user's default tablespace (case-sensitive)

Usage Notes

- Call this procedure once before issuing a call to the PACK_STGTAB_SQLPROF Procedure.
- This procedure can be called multiple times if you would like to have different SQL profiles in different staging tables.
- Note that this is a DDL operation, so it does not occur within a transaction.

Examples

Create a staging table to store profile data that can be moved to another system.

```
EXEC DBMS_SQLTUNE.CREATE_STGTAB_SQLPROF (table_name  => 'PROFILE_STGTAB');
```

CREATE_STGTAB_SQLSET Procedure

This procedure creates a staging table through which SQL Tuning Sets are imported and exported

> **See Also:** SQL Tuning Set Subprograms on page 101-12 for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.CREATE_STGTAB_SQLSET (
   table_name           IN VARCHAR2,
   schema_name          IN VARCHAR2 := NULL,
   tablespace_name      IN VARCHAR2 := NULL);
```

Parameters

Table 101–13 CREATE_STGTAB_SQLSET Procedure Parameters

Parameter	Description
table_name	The name of the table to create (case-sensitive)
schema_name	The schema in which to create the table in, or NULL for current schema (case-sensitive)
tablespace_name	The tablespace in which to store the staging table, or NULL for current user's default tablespace (case-sensitive)

Usage Notes

- Call this procedure once before issuing a call to the PACK_STGTAB_SQLSET Procedure.
- This procedure can be called multiple times if you would like to have different tuning sets in different staging tables.
- Note that this is a DDL operation, so it does not occur within a transaction.
- Users issuing the call must have permission to CREATE TABLE in the schema provided and the relevant tablespace.
- Please note that the staging table contains nested table columns and indexes, so it should not be renamed.

Examples

Create a staging table for packing and eventually exporting a SQL tuning sets

```
EXEC DBMS_SQLTUNE.CREATE_STGTAB_SQLSET(table_name => 'STGTAB_SQLSET');
```

CREATE_TUNING_TASK Functions

You can use different forms of this function to:

- Create a tuning task for a single statement given its text.
- Create a tuning task for a single statement from the Cursor Cache given its identifier.
- Create a tuning task for a single statement from the workload repository given a range of snapshot identifiers.
- Create a tuning task for a SQL tuning set.

In all cases, the function mainly creates an advisor task and sets its parameters.

> **See Also:** SQL Tuning Advisor Subprograms on page 101-10 for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.CREATE_TUNING_TASK(
  sql_text          IN CLOB,
  bind_list         IN sql_binds := NULL,
  user_name         IN VARCHAR2  := NULL,
  scope             IN VARCHAR2  := SCOPE_COMPREHENSIVE,
  time_limit        IN NUMBER    := TIME_LIMIT_DEFAULT,
  task_name         IN VARCHAR2  := NULL,
  description       IN VARCHAR2  := NULL)
RETURN VARCHAR2;

DBMS_SQLTUNE.CREATE_TUNING_TASK(
  sql_id            IN VARCHAR2,
  plan_hash_value   IN NUMBER   := NULL,
  scope             IN VARCHAR2 := SCOPE_COMPREHENSIVE,
  time_limit        IN NUMBER   := TIME_LIMIT_DEFAULT,
  task_name         IN VARCHAR2 := NULL,
  description       IN VARCHAR2 := NULL)
RETURN VARCHAR2;

DBMS_SQLTUNE.CREATE_TUNING_TASK(
  begin_snap       IN NUMBER,
  end_snap         IN NUMBER,
  sql_id           IN VARCHAR2,
  plan_hash_value IN NUMBER   := NULL,
  scope            IN VARCHAR2 := SCOPE_COMPREHENSIVE,
  time_limit       IN NUMBER   := TIME_LIMIT_DEFAULT,
  task_name        IN VARCHAR2 := NULL,
  description      IN VARCHAR2 := NULL)
RETURN VARCHAR2;

DBMS_SQLTUNE.CREATE_TUNING_TASK(
  sqlset_name        IN VARCHAR2,
  basic_filter       IN VARCHAR2 :=  NULL,
  object_filter      IN VARCHAR2 :=  NULL,
  rank1              IN VARCHAR2 :=  NULL,
  rank2              IN VARCHAR2 :=  NULL,
  rank3              IN VARCHAR2 :=  NULL,
  result_percentage IN NUMBER   :=  NULL,
  result_limit       IN NUMBER   :=  NULL,
  scope              IN VARCHAR2 :=  SCOPE_COMPREHENSIVE,
```

```
  time_limit        IN NUMBER   :=  TIME_LIMIT_DEFAULT,
  task_name         IN VARCHAR2 :=  NULL,
  description       IN VARCHAR2 :=  NULL
  plan_filter       IN VARCHAR2 :=  'MAX_ELAPSED_TIME',
  sqlset_owner      IN VARCHAR2 :=  NULL)
RETURN VARCHAR2;
```

Parameters

Table 101–14 CREATE_TUNING_TASK Function Parameters

Parameter	Description
sql_text	The text of a SQL statement
begin_snap	Begin snapshot identifier
end_snap	End snapshot identifier
sql_id	The identifier of a SQL statement
bind_list	An ordered list of bind values in ANYDATA type
plan_hash_value	The hash value of the SQL execution plan
sqlset_name	The SQL tuning set name
basic_filter	The SQL predicate to filter the SQL from the SQL tuning set
object_filter	The object filter
rank(i)	An order-by clause on the selected SQL
result_percentage	A percentage on the sum of a ranking measure
result_limit	The top L(imit) SQL from the (filtered/ranked) SQL
user_name	The username for whom the statement is to be tuned
scope	Tuning scope (limited/comprehensive)
time_limit	The maximum duration in seconds for the tuning session
task_name	An optional tuning task name
description	A task of the SQL tuning session to a maximum of 256 characters

Table 101–14 (Cont.) CREATE_TUNING_TASK Function Parameters

Parameter	Description
plan_filter	Plan filter. It is applicable in case there are multiple plans (plan_hash_value) associated with the same statement. This filter allows for selecting one plan (plan_hash_value) only. Possible values are: ■ LAST_GENERATED: plan with the most recent timestamp ■ FIRST_GENERATED: plan with the earliest timestamp, the opposite to LAST_GENERATED ■ LAST_LOADED: plan with the most recent first_load_time statistics information ■ FIRST_LOADED: plan with the earliest first_load_time statistics information, the opposite to LAST_LOADED ■ MAX_ELAPSED_TIME: plan with the maximum elapsed time ■ MAX_BUFFER_GETS: plan with the maximum buffer gets ■ MAX_DISK_READS: plan with the maximum disk reads ■ MAX_DIRECT_WRITES: plan with the maximum direct writes ■ MAX_OPTIMIZER_COST: plan with the maximum optimizer cost
sqlset_owner	The owner of the SQL tuning set, or NULL for the current schema owner

Return Values

A SQL tuning task name.

Examples

```
variable stmt_task VARCHAR2(64);
variable sts_task  VARCHAR2(64);

-- Sql text format
EXEC :stmt_task := DBMS_SQLTUNE.CREATE_TUNING_TASK( -
  sql_text => 'select quantity_sold from sales s, times t where s.time_id =
t.time_id and s.time_id = TO_DATE(''24-NOV-00'')');

-- Sql id format (cursor cache)
EXEC :stmt_task := DBMS_SQLTUNE.CREATE_TUNING_TASK(sql_id => 'ay1m3ssvtrh24');

-- tune in limited scope
EXEC :stmt_task := DBMS_SQLTUNE.CREATE_TUNING_TASK(sql_id => 'ay1m3ssvtrh24', -
   scope => 'LIMITED');

-- only give 10 minutes for tuning statement
EXEC :stmt_task := DBMS_SQLTUNE.CREATE_TUNING_TASK(sql_id => 'ay1m3ssvtrh24', -
   time_limit => 600);

-- Workload repository format
exec :stmt_task := DBMS_SQLTUNE.CREATE_TUNING_TASK(begin_snap => 1, -
   end_snap => 2, sql_id => 'ay1m3ssvtrh24');

-- Sql tuning set format (first we need to load an STS, then tune it)
```

```
-- Tune our statements in order by buffer gets, time limit of one hour
-- the default ranking measure is elapsed time.
EXEC :sts_task := DBMS_SQLTUNE.CREATE_TUNING_TASK( -
  sqlset_name  => 'my_workload', -
  rank1        => 'BUFFER_GETS', -
  time_limit   => 3600, -
  description  => 'tune my workload ordered by buffer gets');
```

DELETE_SQLSET Procedure

This procedure deletes a set of SQL statements from a SQL tuning set.

> **See Also:** SQL Tuning Set Subprograms on page 101-12 for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.DELETE_SQLSET (
   sqlset_name   IN  VARCHAR2,
   basic_filter  IN  VARCHAR2 := NULL,
   sqlset_owner  IN  VARCHAR2 := NULL);
```

Parameters

Table 101–15 DELETE_SQLSET Procedure Parameters

Parameter	Description
sqlset_name	The SQL tuning set name
basic_filter	SQL predicate to filter the SQL from the SQL tuning set. This basic filter is used as a where clause on the SQL tuning set content to select a desired subset of SQL from the Tuning Set.
sqlset_owner	The owner of the SQL tuning set, or NULL for current schema owner

Examples

```
-- Delete all statements in a sql tuning set.
EXEC DBMS_SQLTUNE.DELETE_SQLSET(sqlset_name   => 'my_workload');

-- Delete all statements in a sql tuning set which ran for less than a second
EXEC DBMS_SQLTUNE.DELETE_SQLSET(sqlset_name   => 'my_workload', -
                                basic_filter  => 'elapsed_time < 1000000');
```

DROP_SQL_PROFILE Procedure

This procedure drops the named SQL Profile from the database.

> **See Also:** SQL Profile Subprograms on page 101-11 for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.DROP_SQL_PROFILE (
   name          IN  VARCHAR2,
   ignore        IN  BOOLEAN  := FALSE);
```

Parameters

Table 101–16 DROP_SQL_PROFILE Procedure Parameters

Parameter	Description
`name`	The (mandatory) name of SQL Profile to be dropped. The name is case sensitive.
`ignore`	Ignores errors due to object not existing

Usage Notes

Requires the "`DROP ANY SQL PROFILE`" privilege.

Examples

```
-- Drop the profile:
EXEC DBMS_SQLTUNE.DROP_SQL_PROFILE(:pname);
```

DROP_SQLSET Procedure

This procedure drops a SQL tuning set if it is not active.

> **See Also:** SQL Tuning Set Subprograms on page 101-12 for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.DROP_SQLSET (
   sqlset_name   IN  VARCHAR2,
   sqlset_owner  IN  VARCHAR2 := NULL);
```

Parameters

Table 101–17 DROP_SQLSET Procedure Parameters

Parameter	Description
`sqlset_name`	The SQL tuning set name
`sqlset_owner`	The owner of the SQL tuning set, or `NULL` for current schema owner

Usage Notes

You cannot drop a SQL tuning set when it is referenced by one or more clients (for example, SQL tuning advisor).

Examples

```
-- Drop the sqlset.
EXEC DBMS_SQLTUNE.DROP_SQLSET ('my_workload');
```

DROP_TUNING_TASK Procedure

This procedure drops a SQL tuning task.The task and all its result data are deleted.

> **See Also:** SQL Tuning Advisor Subprograms on page 101-10 for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.DROP_TUNING_TASK(
 task_name          IN VARCHAR2);
```

Parameters

Table 101–18 DROP_TUNING_TASK Procedure Parameters

Parameter	Description
task_name	The name of the tuning task to drop

EXECUTE_TUNING_TASK Procedure

This procedures executes a previously created tuning task.

> **See Also:** SQL Tuning Advisor Subprograms on page 101-10 for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.EXECUTE_TUNING_TASK(
 task_name          IN VARCHAR2);
```

Parameters

Table 101–19 EXECUTE_TUNING_TASK Procedure Parameters

Parameter	Description
task_name	The name of the tuning task to execute

Examples

```
EXEC DBMS_SQLTUNE.EXECUTE_TUNING_TASK(:stmt_task);
```

INTERRUPT_TUNING_TASK Procedure

This procedure interrupts the currently executing tuning task. The task will end its operations as it would at normal exit so that the user will be able access the intermediate results.

> **See Also:** SQL Tuning Advisor Subprograms on page 101-10 for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.INTERRUPT_TUNING_TASK(
 task_name          IN VARCHAR2);
```

Parameters

Table 101–20 INTERRUPT_TUNING_TASK Procedure Parameters

Parameter	Description
task_name	The name of the tuning task to interrupt

Examples

```
EXEC DBMS_SQLTUNE.INTERRUPT_TUNING_TASK(:my_task);
```

LOAD_SQLSET Procedure

This procedure populates the SQL tuning set with a set of selected SQL. You can call the procedure multiple times to add new SQL statements or replace attributes of existing statements.

> **See Also:** SQL Tuning Set Subprograms on page 101-12 for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.LOAD_SQLSET (
   sqlset_name       IN  VARCHAR2,
   populate_cursor   IN  sqlset_cursor,
   load_option       IN VARCHAR2 := 'INSERT',
   update_option     IN VARCHAR2 := 'REPLACE',
   update_condition  IN VARCHAR2 :=  NULL,
   update_attributes IN VARCHAR2 :=  NULL,
   ignore_null       IN BOOLEAN  :=  TRUE,
   commit_rows       IN POSITIVE :=  NULL,
   sqlset_owner      IN VARCHAR2 := NULL);
```

Parameters

Table 101–21 LOAD_SQLSET Procedure Parameters

Parameter	Description
`sqlset_name`	The SQL tuning set name to populate
`populate_cursor`	The cursor reference from which to populate
`load_option`	Specifies how the statements will be loaded into the SQL tuning set. The possible values are: ■ `INSERT` (default) - add only new statements ■ `UPDATE` - update existing the SQL statements and ignores any new statements ■ `MERGE` - this is a combination of the two other options. This option inserts new statements and updates the information of the existing ones.
`update_option`	Specifies how the existing statements will be updated. This parameter is considered only if load_option is specified with '`UPDATE`'/'`MERGE`' as an option. The possible values are: ■ `REPLACE` (default) - update the statement using the new statistics, bind list, object list, and so on. ■ `ACCUMULATE` - when possible combine attributes (for example, statistics like `elapsed_time`, and so on) otherwise just replace the old values (for example, module, action, and so on) by the new provided ones. The SQL statement attributes that can be accumulated are: `elapsed_time`, `buffer_gets`, `direct_writes`, `disk_reads`, `row_processed`, `fetches`, `executions`, `end_of_fetch_count`, `stat_period` and `active_stat_period`.

Table 101–21 (Cont.) LOAD_SQLSET Procedure Parameters

Parameter	Description
update_condition	Specifies a where clause to execute the update operation. The update is performed only if the specified condition is true. The condition can refer to either the data source or destination. The condition must use the following prefixes to refer to attributes from the source or the destination: ■ OLD - to refer to statement attributes from the SQL tuning set (destination) ■ NEW - to refer to statements attributes from the input statements (source)
update_attributes	Specifies the list of a SQL statement attributes to update during a merge or update operation.The possible values are: ■ NULL (default) - the content of the input cursor except the execution context. On other terms, it is equivalent to ALL without execution context like module, action, and so on. ■ BASIC - statistics and binds only ■ TYPICAL - BASIC + SQL plans (without row source statistics) and without object reference list ■ ALL - all attributes including the execution context attributes like module, action, and so on. ■ List of comma separated attribute names to update - EXECUTION_CONTEXT, EXECUTION_STATISTICS, BIND_LIST, OBJECT_LIST, SQL_PLAN, SQL_PLAN_STATISTICS (similar to SQL_PLAN + row source statistics)
ignore_null	If TRUE do not update an attribute if the new value is NULL. That is, do not override with NULL values unless intentional.
commit_rows	If a value is provided, the load will commit after each set of that many statements is inserted. If NULL is provided, the load will commit only once, at the end of the operation.
sqlset_owner	The owner of the SQL tuning set, or the current schema owner or NULL for current owner

Exceptions

- This procedure returns an error when sqlset_name is invalid, or a corresponding SQL tuning set does not exist, or the populate_cursor is incorrect and cannot be executed.
- Exceptions are also raised when invalid filters are provided. Filters can be invalid either because they don't parse (for example, they refer to attributes not in sqlset_row), or because they violate the user's privileges.

Usage Notes

Rows in the input populate_cursor must be of type SQLSET_ROW.

Examples

In this example, you create and populate a SQL tuning set with all cursor cache statements with an elapsed time of 5 seconds or more excluding statements that belong to SYS schema (to simulate an application user workload). You select all attributes of the SQL statements and load them in the tuning set using the default mode, which will only load new statements, since the SQL tuning set is empty.

```
-- create the tuning set
EXEC DBMS_SQLTUNE.CREATE_SQLSET('my_workload');
-- populate the tuning set from the cursor cache
DECLARE
 cur DBMS_SQLTUNE.SQLSET_CURSOR;
BEGIN
 OPEN cur FOR
   SELECT VALUE(P)
     FROM table(
       DBMS_SQLTUNE.SELECT_CURSOR_CACHE(
         'parsing_schema_name <> ''SYS'' AND elapsed_time > 5000000',
          NULL, NULL, NULL, NULL, 1, NULL,
         'ALL')) P;

DBMS_SQLTUNE.LOAD_SQLSET(sqlset_name => 'my_workload',
                         populate_cursor => cur);

END;
/
```

Suppose now you wish to augment this information with what is stored in the workload repository (AWR). You populate the tuning set with `'ACCUMULATE'` as your `update_option` because it is assumed the cursors currently in the cache had aged out since the snapshot was taken.

You omit the `elapsed_time` filter because it is assumed that any statement captured in AWR is important, but still you throw away the `SYS`-parsed cursors to avoid recursive SQL.

```
DECLARE
  cur DBMS_SQLTUNE.SQLSET_CURSOR;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
      FROM table(
        DBMS_SQLTUNE.SELECT_WORKLOAD_REPOSITORY(1,2,
                                                'parsing_schema_name <> ''SYS''',
                                                NULL, NULL,NULL,NULL,
                                                1,
                                                NULL,
                                                'ALL')) P;

  DBMS_SQLTUNE.LOAD_SQLSET(sqlset_name => 'my_workload',
                           populate_cursor => cur,
                           Using DBMS_SQLTUNE
                           load_option => 'MERGE',
                           update_option => 'ACCUMULATE');
END;
```

The following example is a simple load that only inserts new statements from the workload repository, skipping existing ones (in the SQL tuning set). Note that `'INSERT'` is the default value for the `load_option` argument of the `LOAD_SQLSET` procedure.

```
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
  SELECT VALUE(P)
  FROM table(DBMS_SQLTUNE.SELECT_WORKLOAD_REPOSITORY(1,2)) P;
```

```
  DBMS_SQLTUNE.LOAD_SQLSET(sqlset_name => 'my_workload',
populate_cursor => cur);
END;
/
```

The next example demonstrates a load with `UPDATE` option. This updates statements that already exist in the SQL tuning set but does not add new ones. By default, old statistics are replaced by their new values.

```
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
    FROM table(DBMS_SQLTUNE.SELECT_CURSOR_CACHE) P;

  DBMS_SQLTUNE.LOAD_SQLSET(sqlset_name => 'my_workload',
                           populate_cursor => cur,
                           load_option => 'UPDATE');
END;
/
```

PACK_STGTAB_SQLPROF Procedure

This procedure copies profile data from the SYS. schema into the staging table.

> **See Also:** SQL Profile Subprograms on page 101-11 for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.PACK_STGTAB_SQLPROF (
   profile_name          IN VARCHAR2 := '%',
   profile_category      IN VARCHAR2 := 'DEFAULT',
   staging_table_name    IN VARCHAR2,
   staging_schema_owner  IN VARCHAR2 := NULL);
```

Parameters

Table 101–22 PACK_STGTAB_SQLPROF Procedure Parameters

Parameter	Description
profile_name	The name of the profile to pack (% wildcards acceptable, case-sensitive)
profile_category	The category to pack profiles from (% wildcards acceptable, case-sensitive)
staging_table_name	The name of the table to use (case-sensitive). Required.
staging_schema_owner	The schema where the table resides, or NULL for current schema (case-sensitive)

Usage Notes

- This procedures requires SELECT privilege on dba_sql_profiles and INSERT privilege on staging table.
- Note that this function issues a COMMIT after packing each SQL profile, so if an error is raised mid-execution, clear the staging table by deleting its rows.

Examples

Put only those profiles in the DEFAULT category into the staging table. This corresponds to all profiles that will be used by default on this system.

```
EXEC DBMS_SQLTUNE.PACK_STGTAB_SQLPROF (staging_table_name => 'PROFILE_STGTAB');
```

This is another example where you put all profiles into the staging table. Note this will even move profiles that are not currently being used by default but are in other categories, such as for testing purposes.

```
EXEC DBMS_SQLTUNE.PACK_STGTAB_SQLPROF (profile_category => '%', -
                                       staging_table_name => 'PROFILE_STGTAB');
```

PACK_STGTAB_SQLSET Procedure

This procedure copies one or more SQL tuning sets from their location in the SYS schema to a staging table created by the CREATE_STGTAB_SQLSET Procedure.

> **See Also:** SQL Tuning Set Subprograms on page 101-12 for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.PACK_STGTAB_SQLSET (
   sqlset_name          IN VARCHAR2,
   sqlset_owner         IN VARCHAR2 := NULL,
   staging_table_name   IN VARCHAR2,
   staging_schema_owner IN VARCHAR2 := NULL);
```

Parameters

Table 101–23 PACK_STGTAB_SQLSET Procedure Parameters

Parameter	Description
sqlset_name	The name of the SQL Tuning Set to pack (% wildcards acceptable, case-sensitive)
sqlset_owner	The category from which to pack SQL Tuning Sets (% wildcards acceptable, case-sensitive)
staging_table_name	The name of the table to use (case-sensitive)
staging_schema_owner	The schema where the table resides, or NULL for current schema (case-sensitive)

Usage Notes

- This procedure can be called several times to move more than one SQL tuning set. Users can then move the populated staging table to another system using any method, such as database link or datapump. Users can then call the UNPACK_STGTAB_SQLSET Procedure create the SQL tuning set on the other system.
- Note that this function issues a COMMIT after packing each SQL tuning set, so if an error is raised mid-execution, clear the staging table by deleting its rows.

Examples

Put all SQL tuning sets on the system in the staging table (to create a staging table, see the CREATE_STGTAB_SQLSET Procedure).

```
EXEC DBMS_SQLTUNE.PACK_STGTAB_SQLSET(sqlset_name        => '%', -
                                     sqlset_owner       => '%', -
                                     staging_table_name => 'STGTAB_SQLSET');
```

Put only those SQL tuning sets owned by the current user in the staging table.

```
EXEC DBMS_SQLTUNE.PACK_STGTAB_SQLSET(sqlset_name        => '%', -
                                     staging_table_name => 'STGTAB_SQLSET');
```

Pack a specific SQL tuning set.

```
EXEC DBMS_SQLTUNE.PACK_STGTAB_SQLSET(sqlset_name        => 'my_workload', -
                                     staging_table_name => 'STGTAB_SQLSET');
```

Pack a second SQL tuning set.

```
EXEC DBMS_SQLTUNE.PACK_STGTAB_SQLSET(sqlset_name        => 'workload_subset', -
                                     staging_table_name => 'STGTAB_SQLSET');
```

REMAP_STGTAB_SQLPROF Procedure

This procedure allows DBAs to change the profile data values kept in the staging table prior to performing an unpack operation. The procedure can be used to change the category of a profile.It can be used to change the name of a profile if one already exists on the system with the same name.

> **See Also:** SQL Profile Subprograms on page 101-11 for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.REMAP_STGTAB_SQLPROF (
  old_profile_name      IN VARCHAR2,
  new_profile_name      IN VARCHAR2 := NULL,
  new_profile_category  IN VARCHAR2 := NULL,
  staging_table_name    IN VARCHAR2,
  staging_schema_owner  IN VARCHAR2 := NULL);
```

Parameters

Table 101–24 REMAP_STGTAB_SQLPROF Procedure Parameters

Parameter	Description
old_profile_name	The name of the profile to target for a remap operation (case-sensitive)
new_profile_name	The new name of the profile, or NULL to remain the same (case-sensitive)
new_profile_category	The new category for the profile, or NULL to remain the same (case-sensitive)
staging_table_name	The name of the table on which to perform the remap operation (case-sensitive). Required.
staging_schema_owner	The schema where the table resides, or NULL for current schema (case-sensitive)

Usage Notes

Using this procedure requires the UPDATE privilege on the staging table.

Examples

Change the name of a profile before we unpack, to avoid conflicts

```
EXEC DBMS_SQLTUNE.REMAP_STGTAB_SQLPROF(old_profile_name    => :pname,          -
                                       new_profile_name    => 'IMP' || :pname, -
                                       staging_table_name  => 'PROFILE_STGTAB');
```

Change the SQL profile in the staging table to be 'TEST' category before we import it. This way users can test the profile on the new system before it is active.

```
EXEC DBMS_SQLTUNE.REMAP_STGTAB_SQLPROF(old_profile_name     => :pname,       -
                                       new_profile_category => 'TEST',       -
                                       staging_table_name   => 'PROFILE_STGTAB');
```

REMAP_STGTAB_SQLSET Procedure

This procedure changes the tuning set names and owners in the staging table so that they can be unpacked with different values than they had on the host system.

> **See Also:** SQL Profile Subprograms on page 101-11 for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.REMAP_STGTAB_SQLSET (
   old_sqlset_name        IN VARCHAR2,
   old_sqlset_owner       IN VARCHAR2 := NULL,
   new_sqlset_name        IN VARCHAR2 := NULL,
   new_sqlset_owner       IN VARCHAR2 := NULL,
   staging_table_name     IN VARCHAR2,
   taging_schema_owner   IN VARCHAR2 := NULL);
```

Parameters

Table 101–25 REMAP_STGTAB_SQLSET Procedure Parameters

Parameter	Description
`old_sqlset_name`	The name of the tuning set to target for a remap operation. Wildcards are not supported.
`old_sqlset_owner`	The new name of the tuning set owner to target for a remap operation. `NULL` for current schema owner
`new_sqlset_name`	The new name for the tuning set, or `NULL` to keep the same tuning set name.
`new_sqlset_owner`	The new owner for the tuning set, or `NULL` to remain the same owner name.
`staging_table_name`	The name of the table on which to perform the remap operation (case-sensitive)
`staging_schema_owner`	The name of staging table owner, or `NULL` for current schema owner (case-sensitive)

Usage Notes

You can call this procedure multiple times to remap more than one tuning set name or owner. Note that this procedure only handles one tuning set per call.

Examples

```
-- Change the name of an STS in the staging table before we unpack it.
EXEC DBMS_SQLTUNE.REMAP_STGTAB_SQLSET(old_sqlset_name    => 'my_workload', -
                                      old_sqlset_owner   => 'SH', -
                                      new_sqlset_name    => 'imp_workload', -
                                      staging_table_name => 'STGTAB_SQLSET');

-- Change the owner of an STS in the staging table before we unpack it.
EXEC DBMS_SQLTUNE.REMAP_STGTAB_SQLSET(old_sqlset_name    => 'imp_workload', -
                                      old_sqlset_owner   => 'SH', -
                                      new_sqlset_owner   => 'SYS', -
                                      staging_table_name => 'STGTAB_SQLSET');
```

REMOVE_SQLSET_REFERENCE Procedure

This procedure deactivates a SQL tuning set to indicate it is no longer used by the client.

> **See Also:** SQL Tuning Set Subprograms on page 101-12 for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.REMOVE_SQLSET_REFERENCE (
   sqlset_name   IN  VARCHAR2,
   reference_id  IN  NUMBER);
```

Parameters

Table 101–26 REMOVE_SQLSET_REFERENCE Procedure Parameters

Parameter	Description
sqlset_name	The SQL tuning set name
reference_id	The identifier of the reference to remove

Examples

You can remove references on a given SQL tuning set when you finish using it and want to make it writable again.

```
EXEC DBMS_SQLTUNE.REMOVE_SQLSET_REFERENCE( -
                                sqlset_name   => 'my_workload', -
                                reference_id  => :rid);
```

Use views USER/DBA_SQLSET_REFERENCES to find all references on a given SQL tuning set.

REPORT_TUNING_TASK Function

This procedure displays the results of a tuning task.

> **See Also:** SQL Tuning Advisor Subprograms on page 101-10 for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.REPORT_TUNING_TASK(
   task_name     IN  VARCHAR2,
   type          IN  VARCHAR2   := TYPE_TEXT,
   level         IN  VARCHAR2   := LEVEL_TYPICAL,
   section       IN  VARCHAR2   := SECTION_ALL,
   object_id     IN  NUMBER     := NULL,
   result_limit  IN  NUMBER     := NULL)
RETURN CLOB;
```

Parameters

Table 101–27 REPORT_TUNING_TASK Function Parameters

Parameter	Description
task_name	The name of the tuning task to report
type	The type of the report to produce. Possible values are TEXT, HTML and XML.
level	The format of the recommendations. Possible values are TYPICAL, BASIC and ALL.
section	The particular section in the report. Possible values are FINDING, PLAN, INFORMATION, ERROR and ALL.
object_id	The identifier of the advisor framework object that represents a given statement in the SQL tuning set
result_limit	The number of statements in a SQL tuning set for which a report is generated

Return Values

A CLOB containing the desired report. This means that you have to set the any SQL*Plus 'LONG' and 'LONGCHUNKSIZE' variables so that the report will print in entirety.

Examples

```
-- Get the whole report for the single statement case.
SELECT DBMS_SQLTUNE.REPORT_TUNING_TASK(:stmt_task) from dual;

-- Show me the summary for the sts case.
SELECT DBMS_SQLTUNE.REPORT_TUNING_TASK(:sts_task, 'TEXT', 'TYPICAL', 'SUMMARY')
FROM DUAL;

-- Show me the findings for the statement I'm interested in.
SELECT DBMS_SQLTUNE.REPORT_TUNING_TASK(:sts_task, 'TEXT', 'TYPICAL', 'FINDINGS',
5) from dual;
```

RESET_TUNING_TASK Procedure

This procedure is called on a tuning task that is not currently executing to prepare it for re-execution.

> **See Also:** SQL Tuning Advisor Subprograms on page 101-10 for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.RESET_TUNING_TASK(
 task_name          IN VARCHAR2);
```

Parameters

Table 101–28 RESET_TUNING_TASK Procedure Parameters

Parameter	Description
task_name	The name of the tuning task to reset

Examples

```
-- reset and re-execute a task
EXEC DBMS_SQLTUNE.RESET_TUNING_TASK(:sts_task);

-- re-execute the task
EXEC DBMS_SQLTUNE.EXECUTE_TUNING_TASK(:sts_task);
```

RESUME_TUNING_TASK Procedure

This procedure resumes a previously interrupted task that was created to tune a SQL tuning set.

> **See Also:** SQL Tuning Advisor Subprograms on page 101-10 for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.RESUME_TUNING_TASK(
 task_name         IN VARCHAR2,
 basic_filter      IN VARCHAR2 := NULL);
```

Parameters

Table 101–29 RESUME_TUNING_TASK Procedure Parameters

Parameter	Description
`task_name`	The name of the tuning task to resume
`basic_filter`	A SQL predicate to filter the SQL from the sql tuning set. Note that this filter will be applied in conjunction with the basic filter (i.e., parameter `basic_filter`) when calling CREATE_TUNING_TASK Functions.

Usage Notes

Resuming a single SQL tuning task (a task that was created to tune a single SQL statement as compared to a SQL Tuning Set) is not supported.

Examples

```
-- Interrupt the task
EXEC DBMS_SQLTUNE.INTERRUPT_TUNING_TASK(:conc_task);

-- Once a task is interrupted, we can elect to reset it, resume it, or check
-- out its results and then decide.  For this example we will just resume.

EXEC DBMS_SQLTUNE.RESUME_TUNING_TASK(:conc_task);
```

SCRIPT_TUNING_TASK Function

This function creates a SQL*PLUS script which can then be executed to implement a set of Advisor recommendations.

> **See Also:** SQL Tuning Advisor Subprograms on page 101-10 for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.SCRIPT_TUNING_TASK(
  task_name          IN VARCHAR2,
  rec_type           IN VARCHAR2,
  object_id          IN NUMBER,
  result_limit       IN NUMNBER,
  owner_name         IN VARCHAR2)
 RETURN CLOB;
```

Parameters

Table 101–30 SCRIPT_TUNING_TASK Procedure Parameters

Parameter	Description
task_name	The name of the tuning task for which to apply a script
rec_type	Filter the script by types of recommendations to include. Any subset of the following separated by commas: or 'ALL: ''PROFILES'' ''STATISTICS'' ''INDEXES'. For example, a script with profiles and statistics: 'PROFILES,STATISTICS'
object_id	Optionally filters by a single object ID
result_limit	Optionally shows commands for only top N SQL (ordered by object_id and ignored if an object_id is also specified)
owner_name	Owner of the relevant tuning task. Defaults to the current schema owner

Return Values

This function returns a CLOB containing the PL/SQL calls to be executed to implement the subset of recommendations dictated by the arguments.

Usage Notes

- Once the script is returned, it should then by checked by the DBA and executed.
- Wrap with a call to DBMS_ADVISOR.CREATE_FILE to put it into a file.

Examples

```
SET LINESIZE 140

-- Get a script for all actions recommended by the task.
SELECT DBMS_SQLTUNE.SCRIPT_TUNING_TASK(:stmt_task) FROM DUAL;

-- Get a script of just the sql profiles we should create.
SELECT DBMS_SQLTUNE.SCRIPT_TUNING_TASK(:stmt_task, 'PROFILES') FROM DUAL;

-- get a script of just stale / missing stats
```

```
SELECT DBMS_SQLTUNE.SCRIPT_TUNING_TASK(:stmt_task, 'STATISTICS') FROM DUAL;

-- Get a script with recommendations about just one SQL statement when we have
-- tuned an entire STS.
SELECT DBMS_SQLTUNE.SCRIPT_TUNING_TASK(:sts_task, 'ALL', 5) FROM DUAL;
```

SELECT_CURSOR_CACHE Function

This function collects SQL statements from the SQL Cursor Cache.

> **See Also:** SQL Tuning Set Subprograms on page 101-12 for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.SELECT_CURSOR_CACHE (
  basic_filter        IN   VARCHAR2 := NULL,
  object_filter       IN   VARCHAR2 := NULL,
  ranking_measure1    IN   VARCHAR2 := NULL,
  ranking_measure2    IN   VARCHAR2 := NULL,
  ranking_measure3    IN   VARCHAR2 := NULL,
  result_percentage   IN   NUMBER   := 1,
  result_limit        IN   NUMBER   := NULL,
  attribute_list      IN   VARCHAR2 := NULL)
 RETURN sys.sqlset PIPELINED;
```

Parameters

Table 101–31 SELECT_CURSOR_CACHE Procedure Parameters

Parameter	Description
`sqlset_name`	The SQL tuning set name
`basic_filter`	The SQL predicate to filter the SQL from the cursor cache defined on attributes of the `SQLSET_ROW`
`object_filter`	Specifies the objects that should exist in the object list of selected SQL from the cursor cache
`ranking_measure(n)`	An order-by clause on the selected SQL
`result_percentage`	A filter which picks the top N% according to the ranking measure given. Note that this applies only if one ranking measure is given.
`result_limit`	The top L(imit) SQL from the (filtered) source ranked by the ranking measure
`attribute_list`	List of SQL statement attributes to return in the result. The possible values are: ▪ `BASIC` (default) -all attributes (such as execution statistics and binds) are returned except the plans The execution context is always part of the result. ▪ `TYPICAL` - `BASIC` + SQL plan (without row source statistics) and without object reference list ▪ `ALL` - return all attributes ▪ Comma separated list of attribute names this allows to return only a subset of SQL attributes: `EXECUTION_STATISTICS`, `BIND_LIST`, `OBJECT_LIST`, `SQL_PLAN`,`SQL_PLAN_STATISTICS`: similar to `SQL_PLAN` + row source statistics

Return Values

This function returns a one `SQLSET_ROW` per `SQL_ID` or `PLAN_HASH_VALUE` pair found in each data source.

Usage Notes

Users need privileges on the cursor cache views.

Examples

```
-- Get sql ids and sql text for statements with 500 buffer gets.
SELECT sql_id, sql_text
FROM table(DBMS_SQLTUNE.SELECT_CURSOR_CACHE('buffer_gets > 500'))
ORDER BY sql_id;

-- Get all the information we have about a particular statement.
SELECT *
FROM table(DBMS_SQLTUNE.SELECT_CURSOR_CACHE('sql_id = ''4rm4183czbs7j'''));

-- Notice that some statements can have multiple plans.  The output of the
-- SELECT_XXX table functions is unique by (sql_id, plan_hash_value).  This is
-- because a data source can store multiple plans per sql statement.
SELECT sql_id, plan_hash_value
FROM table(dbms_sqltune.select_cursor_cache('sql_id = ''ay1m3ssvtrh24'''))
ORDER BY sql_id, plan_hash_value;

-- PL/SQL examples: load_sqlset will be called after opening a cursor, along the
-- lines given below

-- Select all statements in the cursor cache.
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT value(P)
    FROM table(DBMS_SQLTUNE.SELECT_CURSOR_CACHE) P;

  -- Process each statement (or pass cursor to load_sqlset).

  CLOSE cur;
END;
/


-- Look for statements not parsed by SYS.
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur for
    SELECT VALUE(P)
    FROM table(
     DBMS_SQLTUNE.SELECT_CURSOR_CACHE('parsing_schema_name <> ''SYS''')) P;

  -- Process each statement (or pass cursor to load_sqlset).

  CLOSE cur;
end;
/


-- All statements from a particular module/action.
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
```

```
    SELECT VALUE(P)
    FROM table(
      DBMS_SQLTUNE.SELECT_CURSOR_CACHE(
         'module = ''MY_APPLICATION'' and action = ''MY_ACTION''')) P;

  -- Process each statement (or pass cursor to load_sqlset)

  CLOSE cur;
END;
/


-- all statements that ran for at least five seconds
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
    FROM table(DBMS_SQLTUNE.SELECT_CURSOR_CACHE('elapsed_time > 5000000')) P;

  -- Process each statement (or pass cursor to load_sqlset)

  CLOSE cur;
end;
/


-- select all statements that pass a simple buffer_gets threshold and
-- are coming from an APPS user
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
    FROM table(
      DBMS_SQLTUNE.SELECT_CURSOR_CACHE(
        'buffer_gets > 100 and parsing_schema_name = ''APPS'''))P;

  -- Process each statement (or pass cursor to load_sqlset)

  CLOSE cur;
end;
/


-- select all statements exceeding 5 seconds in elapsed time, but also
-- select the plans (by default we only select execution stats and binds
-- for performance reasons - in this case the SQL_PLAN attribute of sqlset_row
-- is NULL)
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
    FROM table(dbms_sqltune.select_cursor_cache(
      'elapsed_time > 5000000', NULL, NULL, NULL, NULL, 1, NULL,
      'EXECUTION_STATISTICS, SQL_BINDS, SQL_PLAN')) P;

  -- Process each statement (or pass cursor to load_sqlset)
```

```
  CLOSE cur;
END;
/


-- Select the top 100 statements in the cursor cache ordering by elapsed_time.
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
    FROM table(DBMS_SQLTUNE.SELECT_CURSOR_CACHE(NULL,
                                                NULL,
                                                'ELAPSED_TIME', NULL, NULL,
                                                1,
                                                100)) P;

  -- Process each statement (or pass cursor to load_sqlset)

  CLOSE cur;
end;
/


-- Select the set of statements which cumulatively account for 90% of the
-- buffer gets in the cursor cache.  This means that the buffer gets of all
-- of these statements added up is approximately 90% of the sum of all
-- statements currently in the cache.
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
    FROM table(DBMS_SQLTUNE.SELECT_CURSOR_CACHE(NULL,
                                                NULL,
                                                'BUFFER_GETS', NULL, NULL,
                                                .9)) P;

  -- Process each statement (or pass cursor to load_sqlset).

  CLOSE cur;
END;
/
```

SELECT_SQLSET Function

This function reads SQLSET contents.

> **See Also:** SQL Tuning Set Subprograms on page 101-12 for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.SELECT_SQLSET (
  sqlset_name         IN   VARCHAR2,
  basic_filter        IN   VARCHAR2 := NULL,
  object_filter       IN   VARCHAR2 := NULL,
  ranking_measure1    IN   VARCHAR2 := NULL,
  ranking_measure2    IN   VARCHAR2 := NULL,
  ranking_measure3    IN   VARCHAR2 := NULL,
  result_percentage   IN   NUMBER   := 1,
  result_limit        IN   NUMBER   := NULL)
  attribute_list      IN   VARCHAR2 := NULL,
  plan_filter         IN   VARCHAR2 := NULL,
  sqlset_owner        IN   VARCHAR2 := NULL)
 RETURN sys.sqlset PIPELINED;
```

Parameters

Table 101–32 SELECT_SQLSET Procedure Parameters

Parameter	Description
sqlset_name	The SQL tuning set name
basic_filter	The SQL predicate to filter the SQL from the SQL Tuning Set defined on attributes of the SQLSET_ROW
object_filter	Specifies the objects that should exist in the object list of selected SQL from the cursor cache
ranking_measure(n)	An order-by clause on the selected SQL
result_percentage	A filter which picks the top N% according to the ranking measure given. Note that this applies only if one ranking measure is given.
result_limit	The top L(imit) SQL from the (filtered) source ranked by the ranking measure
attribute_list	List of SQL statement attributes to return in the result. The possible values are: ■ BASIC (default) -all attributes (such as execution statistics and binds) are returned except the plans The execution context is always part of the result. ■ TYPICAL - BASIC + SQL plan (without row source statistics) and without object reference list ■ ALL - return all attributes ■ Comma separated list of attribute names this allows to return only a subset of SQL attributes: EXECUTION_STATISTICS, BIND_LIST, OBJECT_LIST, SQL_PLAN,SQL_PLAN_STATISTICS (similar to SQL_PLAN + row source statistics)
plan_filter	The plan filter

Table 101–32 (Cont.) SELECT_SQLSET Procedure Parameters

Parameter	Description
sqlset_owner	The owner of the SQL tuning set, or NULL for the current schema owner

Return Values

This function returns a one SQLSET_ROW per SQL_ID or PLAN_HASH_VALUE pair found in each data source.

Examples

```
-- select from a sql tuning set
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE (P)
    FROM table(dbms_sqltune.select_sqlset('my_workload')) P;

  -- Process each statement (or pass cursor to load_sqlset)

  CLOSE cur;
END;
/
```

SELECT_WORKLOAD_REPOSITORY Functions

This function collects SQL statements from the workload repository. The overloaded forms let you:

- Collect SQL statements from all snapshots between `begin_snap` and `end_snap`.
- Collect SQL statements from a workload repository baseline.

> **See Also:** SQL Tuning Set Subprograms on page 101-12 for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.SELECT_WORKLAOD_REPOSITORY (
  begin_snap        IN NUMBER,
  end_snap          IN NUMBER,
  basic_filter      IN VARCHAR2 := NULL,
  object_filter     IN VARCHAR2 := NULL,
  ranking_measure1  IN VARCHAR2 := NULL,
  ranking_measure2  IN VARCHAR2 := NULL,
  ranking_measure3  IN VARCHAR2 := NULL,
  result_percentage IN NUMBER   := 1,
  result_limit      IN NUMBER   := NULL
  attribute_list    IN   VARCHAR2 := NULL)
 RETURN sys.sqlset PIPELINED;

DBMS_SQLTUNE.SELECT_WORKLAOD REPOSITORY (
  baseline_name     IN VARCHAR2,
  basic_filter      IN VARCHAR2 := NULL,
  object_filter     IN VARCHAR2 := NULL,
  ranking_measure1  IN VARCHAR2 := NULL,
  ranking_measure2  IN VARCHAR2 := NULL,
  ranking_measure3  IN VARCHAR2 := NULL,
  result_percentage IN NUMBER   := 1,
  result_limit      IN NUMBER   := NULL)
  attribute_list    IN   VARCHAR2 := NULL)
 RETURN sys.sqlset PIPELINED;
```

Parameters

Table 101–33 SELECT_WORKLOAD_REPOSITORY Function Parameters

Parameter	Description
`begin_snap`	Begin snapshot
`end_snap`	End snapshot
`baseline_name`	The name of the baseline period
`basic_filter`	The SQL predicate to filter the SQL from the workload repository defined on attributes of the `SQLSET_ROW`
`object_filter`	Specifies the objects that should exist in the object list of selected SQL from the SWRF
`ranking_measure(n)`	An order-by clause on the selected SQL
`result_percentage`	A filter which picks the top N% according to the ranking measure given. Note that this applies only if one ranking measure is given.

Table 101–33 (Cont.) SELECT_WORKLOAD_REPOSITORY Function Parameters

Parameter	Description
result_limit	The top L(imit) SQL from the (filtered) source ranked by the ranking measure
attribute_list	List of SQL statement attributes to return in the result. The possible values are: ■ BASIC (default) -all attributes (such as execution statistics and binds) are returned except the plans The execution context is always part of the result. ■ TYPICAL - BASIC + SQL plan (without row source statistics) and without object reference list ■ ALL - return all attributes ■ Comma separated list of attribute names allowing return of only a subset of SQL attributes: EXECUTION_STATISTICS, BIND_LIST, OBJECT_LIST, SQL_PLAN,SQL_PLAN_STATISTICS (similar to SQL_PLAN + row source statistics)

Return Values

This function returns a one SQLSET_ROW per SQL_ID or PLAN_HASH_VALUE pair found in each data source.

Examples

```
-- select statements from snapshots 1-2
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE (P)
    FROM table(dbms_sqltune.select_workload_repository(1,2)) P;

  -- Process each statement (or pass cursor to load_sqlset)

  CLOSE cur;
END;
/
```

SQLTEXT_TO_SIGNATURE Function

This function returns a SQL text's signature. The signature can be used to identify SQL text in `dba_sql_profiles`.

> **See Also:** SQL Profile Subprograms on page 101-11 for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.SQLTEXT_TO_SIGNATURE (
  sql_text    IN CLOB,
  force_match IN BOOLEAN  := FALSE)
 RETURN NUMBER;
```

Parameters

Table 101–34 SQLTEXT_TO_SIGNATURE Function Parameters

Parameter	Description
`sql_text`	SQL text whose signature is required. Required.
`force_match`	If `TRUE`, this returns a signature that supports SQL matching with literal values transformed into bind variables. If `FALSE`, returns the signature based on the text with literals not transformed

Return Values

This function returns the signature of the specified SQL text.

UNPACK_STGTAB_SQLPROF Procedure

This procedure copies profile data stored in the staging table to create profiles on the system.

> **See Also:** SQL Profile Subprograms on page 101-11 for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.UNPACK_STGTAB_SQLPROF (
   profile_name          IN VARCHAR2 := '%',
   profile_category      IN VARCHAR2 := 'DEFAULT',
   replace               IN BOOLEAN,
   staging_table_name    IN VARCHAR2,
   staging_schema_owner  IN VARCHAR2 := NULL);
```

Parameters

Table 101–35 UNPACK_STGTAB_SQLPROF Procedure Parameters

Parameter	Description
`profile_name`	The name of the profile to unpack (% wildcards acceptable, case-sensitive)
`profile_category`	The category from which to unpack profiles (% wildcards acceptable, case-sensitive)
`replace`	The option to replace profiles if they already exist. Note that profiles cannot be replaced if one in the staging table has the same name as an active profile in a different SQL statement. If `FALSE`, this function raises errors if you try to create a profile that already exists
`staging_table_name`	The name of the table on which to perform the remap operation (case-sensitive). Required.
`staging_schema_owner`	The schema where the table resides, or `NULL` for current schema (case-sensitive)

Usage Notes

Using this procedure requires the `CREATE ANY SQL PROFILE` privilege and the `SELECT` privilege on staging table.

Examples

```
-- Unpack all profiles stored in a staging table
EXEC DBMS_SQLTUNE.UNPACK_STGTAB_SQLPROF(replace           => FALSE, -
                                        staging_table_name => 'PROFILE_STGTAB');

-- If there is a failure during the unpack operation, users can find the profile
-- we failed on and perform a remap_stgtab_sqlprof operation targeting it.  Then
-- they can resume the unpack operation by setting replace to TRUE so that
-- the profiles that were already created will just be replaced
EXEC DBMS_SQLTUNE.UNPACK_STGTAB_SQLPROF(replace           => TRUE, -
                                        staging_table_name => 'PROFILE_STGTAB');
```

UNPACK_STGTAB_SQLSET Procedure

This procedure copies one or more SQL tuning sets from their location in the staging table into the SQL tuning sets schema, making them proper SQL tuning sets.

> **See Also:** SQL Tuning Set Subprograms on page 101-12 for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.UNPACK_STGTAB_SQLSET (
   sqlset_name          IN VARCHAR2 := '%',
   sqlset_owner         IN VARCHAR2 := NULL,
   replace              IN BOOLEAN,
   staging_table_name   IN VARCHAR2,
   staging_schema_owner IN VARCHAR2 := NULL);
```

Parameters

Table 101–36 UNPACK_STGTAB_SQLSET Procedure Parameters

Parameter	Description
sqlset_name	The name of the tuning set to unpack (not NULL). Wildcard characters ('%') are supported to unpack multiple tuning sets in a single call. For example, just specify '%' to unpack all tuning sets from the staging table.
sqlset_owner	The name of tuning set owner, or NULL for current schema owner. Wildcards supported.
replace	Replaces tuning set if they already exist.If FALSE, raises errors if you try to create a tuning set that already exists
staging_table_name	The name of the staging table, moved after a call to the PACK_STGTAB_SQLSET Procedure (case-sensitive)
staging_schema_owner	The name of staging table owner, or NULL for current schema owner (case-sensitive)

Usage Notes

- Users can drop the staging table after this procedure completes successfully.
- The unpack procedure commits after successfully loading each SQL tuning set. If it fails with one tuning set, no part of that tuning set will have been unpacked, but those which the subprogram had already apprehended will continue to exist.
- When failures occur due to SQL tuning set name or owner conflicts, users should use the REMAP_STGTAB_SQLSET Procedure to patch the staging table, and then call this procedure again to unpack those tuning sets that remain.

Examples

```
 -- unpack all STS in the staging table
EXEC DBMS_SQLTUNE.UNPACK_STGTAB_SQLSET(sqlset_name         => '%', -
                                       sqlset_owner        => '%', -
                                       replace             => FALSE, -
                                       staging_table_name  => 'STGTAB_SQLSET');

-- errors can arise during STS unpack when a STS in the staging table has the
-- same name/owner as STS on the system.  In this case, users should call
```

```
-- remap_stgtab_sqlset to patch the staging table and with which to call unpack
-- Replace set to TRUE.
EXEC DBMS_SQLTUNE.UNPACK_STGTAB_SQLSET(sqlset_name         => '%', -
                                       sqlset_owner        => '%', -
                                       replace             => TRUE, -
                                       staging_table_name  => 'STGTAB_SQLSET');
```

UPDATE_SQLSET Procedures

This procedure updates selected fields for SQL statement in a SQL tuning set.

> **See Also:** SQL Tuning Set Subprograms on page 101-12 for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.UPDATE_SQLSET (
   sqlset_name      IN  VARCHAR2,
   sql_id           IN  VARCHAR2,
   attribute_name   IN  VARCHAR2,
   attribute_value  IN  VARCHAR2 := NULL);

DBMS_SQLTUNE.UPDATE_SQLSET (
   sqlset_name      IN  VARCHAR2,
   sql_id           IN  VARCHAR2,
   attribute_name   IN  VARCHAR2,
   attribute_value IN NUMBER := NULL);
```

Parameters

Table 101–37 UPDATE_SQLSET Function Parameters

Parameter	Description
sqlset_name	The SQL tuning set name
sql_id	The identifier of the statement to update
attribute_name	The name of the attribute to modify
attribute_value	The new value of the attribute

102

DBMS_STAT_FUNCS

The DBMS_STAT_FUNCS package provides statistical functions.

This chapter contains the following topic:

- Summary of DBMS_STAT_FUNCS Subprograms

Summary of DBMS_STAT_FUNCS Subprograms

Table 102–1 DBMS_STAT_FUNCS Package Subprograms

Subprogram	Description
EXPONENTIAL_DIST_FIT Procedure on page 102-3	Tests how well a sample of values fits an exponential distribution
NORMAL_DIST_FIT Procedure on page 102-4	Tests how well a sample of values fits a normal distribution
POISSON_DIST_FIT Procedure on page 102-5	Tests how well a sample of values fits a Poisson distribution
SUMMARY Procedure on page 102-6	Summarizes a numerical column of a table
UNIFORM_DIST_FIT Procedure on page 102-7	Tests how well a sample of values fits a uniform distribution
WEIBULL_DIST_FIT Procedure on page 102-8	Tests how well a sample of values fits a Weibull distribution

EXPONENTIAL_DIST_FIT Procedure

This procedure tests how well a sample of values fits an exponential distribution.

Syntax

```
DBMS_STAT_FUNCS.EXPONENTIAL_DIST_FIT (
   ownername    IN    VARCHAR2,
   tablename    IN    VARCHAR2,
   columnname   IN    VARCHAR2,
   test_type    IN    VARCHAR2 DEFAULT 'KOLMOGOROV_SMIRNOV',
   lambda       IN    NUMBER,
   mu           IN    NUMBER,
   sig          OUT   NUMBER);
```

Parameters

Table 102–2 EXPONENTIAL_DIST_FIT Procedure Parameters

Parameter	Description
ownername	The schema where the table resides.
tablename	The table where the column resides.
columnname	The column of the table against which to run the test.
test_type	The type of test to use: 'CHI_SQUARED','KOLMOGOROV_SMIRNOV' or 'ANDERSON_DARLING'.
lambda	The scale parameter.
mu	The location parameter.
sig	The goodness of fit value, based on test type. A small value indicates a significant difference between the sample and the exponential distribution. A number close to 1 indicates a close match.

NORMAL_DIST_FIT Procedure

This procedure tests how well a sample of values fits a normal distribution.

Syntax

```
DBMS_STAT_FUNCS.NORMAL_DIST_FIT (
   ownername    IN    VARCHAR2,
   tablename    IN    VARCHAR2,
   columnname   IN    VARCHAR2,
   test_type    IN    VARCHAR2 DEFAULT 'SHAPIRO_WILKS',
   mean         IN    NUMBER,
   stdev        IN    NUMBER,
   sig          OUT   NUMBER);
```

Parameters

Table 102–3 NORMAL_DIST_FIT Procedure Parameters

Parameter	Description
ownername	The schema where the table resides.
tablename	The table where the column resides.
columnname	The column of the table against which to run the test.
test_type	The type of test to use: 'CHI_SQUARED', 'KOLMOGOROV_SMIRNOV', 'ANDERSON_DARLING' or 'SHAPIRO_WILKS'.
mean	The mean of the distribution against which to compare.
stdev	The standard deviation of the distribution against which to compare.
sig	The goodness of fit value, based on test type. A small value indicates a significant difference between the sample and the normal distribution. A number close to 1 indicates a close match.

POISSON_DIST_FIT Procedure

This procedure tests how well a sample of values fits a Poisson distribution.

Syntax

```
DBMS_STAT_FUNCS.POISSON_DIST_FIT (
   ownername    IN    VARCHAR2,
   tablename    IN    VARCHAR2,
   columnname   IN    VARCHAR2,
   test_type    IN    VARCHAR2 DEFAULT 'KOLMOGOROV_SMIRNOV',
   lambda       IN    NUMBER,
   sig          OUT   NUMBER);
```

Parameters

Table 102–4 POISSON_DIST_FIT Procedure Parameters

Parameter	Description
ownername	The schema where the table resides.
tablename	The table where the column resides.
columnname	The column of the table against which to run the test.
test_type	The type of test to use: 'KOLMOGOROV_SMIRNOV' or 'ANDERSON_DARLING'.
lambda	The lambda parameter is the shape parameter.
sig	The goodness of fit value, based on test type. A small value indicates a significant difference between the sample and the Poisson distribution. A number close to 1 indicates a close match.

SUMMARY Procedure

This procedure summarizes the numerical column specified in the `columnname` of `tablename`. The summary is returned as a Summary Type. Note that most of the output of `SUMMARY` can be obtained with currently available SQL.

Syntax

```
DBMS_STAT_FUNCS.SUMMARY (
   ownername    IN    VARCHAR2,
   tablename    IN    VARCHAR2,
   columnname   IN    VARCHAR2,
   sigma_value  IN    NUMBER DEFAULT 3,
   s           OUT    SummaryType);
```

Parameters

Table 102–5 SUMMARY Procedure Parameters

Parameter	Description
`ownername`	The schema where the table resides.
`tablename`	The table where the column resides.
`columnname`	The column of the table to be summarized.
`sigma_value`	The number of sigmas for the set of extreme values, defaults to 3.
`s`	The Record containing summary information about given column.

Definition of SummaryType

```
TYPE n_arr IS VARRAY(5) of NUMBER;
TYPE num_table IS TABLE of NUMBER;
TYPE summaryType IS RECORD (
   count             NUMBER,
   min               NUMBER,
   max               NUMBER,
   range             NUMBER,
   mean              NUMBER,
   cmode             num_table,
   variance          NUMBER,
   stddev            NUMBER,
   quantile_5        NUMBER,
   quantile_25       NUMBER,
   median            NUMBER,
   quantile_75       NUMBER,
   quantile_95       NUMBER,
   plus_x_sigma      NUMBER,
   minus_x_sigma     NUMBER,
   extreme_values    num_table,
   top_5_values      n_arr,
   bottom_5_values   n_arr);
```

UNIFORM_DIST_FIT Procedure

This procedure tests well a sample of values fits a uniform distribution.

Syntax

```
DBMS_STAT_FUNCS.UNIFORM_DIST_FIT (
   ownername    IN    VARCHAR2,
   tablename    IN    VARCHAR2,
   columnname   IN    VARCHAR2,
   var_type     IN    VARCHAR2 DEFAULT 'CONTINUOUS',
   test_type    IN    VARCHAR2 DEFAULT 'KOLMOGOROV_SMIRNOV',
   paramA       IN    NUMBER,
   paramB       IN    NUMBER,
   sig          OUT   NUMBER);
```

Parameters

Table 102–6 UNIFORM_DIST_FIT Procedure Parameters

Parameter	Description
ownername	The schema where the table resides.
tablename	The table where the column resides.
columnname	The column of the table against which to run the test.
var_type	The type of distribution: 'CONTINUOUS' (the default) or 'DISCRETE'
test_type	The type of test to use: 'CHI_SQUARED','KOLMOGOROV_SMIRNOV' or 'ANDERSON_DARLING'.
paramA	Parameter A estimated from the sample (the location parameter).
paramB	Parameter B estimated from the sample (the scale parameter).
sig	The goodness of fit value, based on test type. A small value indicates a significant difference between the sample and the uniform distribution. A number close to 1 indicates a close match.

WEIBULL_DIST_FIT Procedure

This procedure tests how well a sample of values fits a Weibull distribution.

Syntax

```
DBMS_STAT_FUNCS.WEIBULL_DIST_FIT (
   ownername    IN    VARCHAR2,
   tablename    IN    VARCHAR2,
   columnname   IN    VARCHAR2,
   test_type    IN    VARCHAR2 DEFAULT 'KOLMOGOROV_SMIRNOV',
   alpha        IN    NUMBER,
   mu           IN    NUMBER,
   beta         IN    NUMBER,
   sig          OUT   NUMBER);
```

Parameters

Table 102–7 WEIBULL_DIST_FIT Procedure Parameters

Parameter	Description
ownername	The schema where the table resides.
tablename	The table where the column resides.
columnname	The column of the table against which to run the test.
test_type	The type of test to use: 'CHI_SQUARED', 'KOLMOGOROV_SMIRNOV' or 'ANDERSON_DARLING'.
alpha	The scale parameter.
mu	The location parameter.
beta	The slope/shape parameter.
sig	The goodness of fit value, based on test type. A small value indicates a significant difference between the sample and the Weibull distribution. A number close to 1 indicates a close match.

103

DBMS_STATS

With the `DBMS_STATS` package you can view and modify optimizer statistics gathered for database objects.

> **See Also:** *Oracle Database Performance Tuning Guide*

This chapter contains the following topics:

Using DBMS_STATS

This section contains topics which relate to using the `DBMS_STATS` package.

Overview

The Oracle RDBMS allows you to collect statistics of many different kinds as an aid to to improving performance. This package is concerned with optimizer statistics only. Given that Oracle sets automatic statistics collection of this kind on by default, this package is intended for only specialized cases.

The statistics of interest to be viewed or modified can reside in the dictionary or in a table created in the user's schema for this purpose. You can also collect and manage user-defined statistics for tables and domain indexes using this package.

For example, if the `DELETE_COLUMN_STATS` procedure is invoked on a column for which an association is defined, user-defined statistics for that column are deleted in addition to deletion of the standard statistics.

Only statistics stored in the dictionary have an impact on the cost-based optimizer. You can also use `DBMS_STATS` to gather statistics in parallel

> **See Also:** *Oracle Database Performance Tuning Guide* for more information about "Managing Optimizer Statistics".

Types

Types for the minimum and maximum values and histogram endpoints include:

```
TYPE numarray  IS VARRAY(256) OF NUMBER;
TYPE datearray IS VARRAY(256) OF DATE;
TYPE chararray IS VARRAY(256) OF VARCHAR2(4000);
TYPE rawarray  IS VARRAY(256) OF RAW(2000);
TYPE fltarray IS VARRAY(256) OF BINARY_FLOAT;
TYPE dblarray IS VARRAY(256) OF BINARY_DOUBLE;

TYPE StatRec IS RECORD (
  epc    NUMBER,
  minval RAW(2000),
  maxval RAW(2000),
  bkvals NUMARRAY,
  novals NUMARRAY);
```

Types for listing stale tables include:

```
TYPE ObjectElem IS RECORD (
  ownname     VARCHAR2(30),     -- owner
  objtype     VARCHAR2(6),      -- 'TABLE' or 'INDEX'
  objname     VARCHAR2(30),     -- table/index
  partname    VARCHAR2(30),     -- partition
  subpartname VARCHAR2(30),     -- subpartition
  confidence  NUMBER);          -- not used
type ObjectTab is TABLE of ObjectElem;
```

Constants

Use the following constant to indicate that auto-sample size algorithms should be used:

```
AUTO_SAMPLE_SIZE CONSTANT NUMBER;
```

The constant used to determine the system default degree of parallelism, based on the initialization parameters, is:

```
DEFAULT_DEGREE CONSTANT NUMBER;
```

Use the following constant to let Oracle select the degree of parallelism based on size of the object, number of CPUs and initialization parameters:

```
AUTO_DEGREE CONSTANT NUMBER;
```

Use the following constant to let Oracle decide whether to collect statistics for indexes or not:

```
AUTO_CASCADE CONSTANT BOOLEAN;
```

Use the following constant to let oracle decide when to invalidate dependent cursors.

```
AUTO_INVALIDATE CONSTANT BOOLEAN
```

Operational Notes

The DBMS_STATS subprograms perform the following general operations:

- Gathering Optimizer Statistics
- Setting or Getting Statistics
- Deleting Statistics
- Transferring Statistics
- Locking or Unlocking Statistics
- Restoring and Purging Statistics History
- User-Defined Statistics

Most of the DBMS_STATS procedures include the three parameters statown, stattab, and statid. These parameters allow you to store statistics in your own tables (outside of the dictionary), which does not affect the optimizer. Therefore, you can maintain and experiment with *sets* of statistics.

The stattab parameter specifies the name of a table in which to hold statistics, and it is assumed that it resides in the same schema as the object for which statistics are collected (unless the statown parameter is specified). You can create multiple tables with different stattab identifiers to hold separate sets of statistics.

Additionally, you can maintain different sets of statistics within a single stattab by using the statid parameter, which avoids cluttering the user's schema.

For the SET and GET procedures, if stattab is not provided (that is, NULL), then the operation works directly on the dictionary statistics; therefore, you do not need to create these statistics tables if they only plan to modify the dictionary directly. However, if stattab is not NULL, then the SET or GET operation works on the specified user statistics table, and not the dictionary.

You can change the default values of some of the parameters of DBMS_STATS procedures using the SET_PARAM Procedure.

Most of the procedures in this package commit the current transaction, perform the operation, and then commit again.

Most of the procedures have a parameter, force which allows you to override any lock on statistics.

Whenever statistics in dictionary are modified, old versions of statistics are saved automatically for future restoring.

Gathering Optimizer Statistics

Use the following subprograms to gather certain classes of optimizer statistics, with possible performance improvements over the ANALYZE command:

GATHER_DATABASE_STATS Procedures
GATHER_DICTIONARY_STATS Procedure
GATHER_FIXED_OBJECTS_STATS Procedure
GATHER_INDEX_STATS Procedure
GATHER_SCHEMA_STATS Procedures
GATHER_SYSTEM_STATS Procedure
GATHER_TABLE_STATS Procedure

The GATHER_* procedures also collect user-defined statistics for columns and domain indexes.

The statown, stattab, and statid parameters instruct the package to back up current statistics in the specified table before gathering new statistics.

Oracle also provides the following procedure for generating statistics for derived objects when you have sufficient statistics on related objects:

GENERATE_STATS Procedure

Setting or Getting Statistics

Use the following subprograms to store and retrieve individual column-related, index-related, and table-related statistics:

PREPARE_COLUMN_VALUES Procedures
PREPARE_COLUMN_VALUES_NVARCHAR2 Procedure
PREPARE_COLUMN_VALUES_ROWID Procedure

SET_COLUMN_STATS Procedures
SET_INDEX_STATS Procedures
SET_SYSTEM_STATS Procedure
SET_TABLE_STATS Procedure

GET_COLUMN_STATS Procedures
GET_INDEX_STATS Procedures
GET_SYSTEM_STATS Procedure
GET_TABLE_STATS Procedure

In the special versions of the SET_*_STATS procedures for setting user-defined statistics, the following, if provided, are stored in the dictionary or external statistics table:

- User-defined statistics (extstats)
- The statistics type schema name (statsschema)
- The statistics type name (statsname)

The user-defined statistics and the corresponding statistics type are inserted into the USTATS$ dictionary table. You can specify user-defined statistics without specifying the statistics type name.

The special versions of the GET_*_STATS procedures return user-defined statistics and the statistics type owner and name as OUT arguments corresponding to the schema object specified. If user-defined statistics are not collected, NULL values are returned.

Deleting Statistics

The DELETE_* procedures delete both user-defined statistics and the standard statistics for the given schema object.

DELETE_COLUMN_STATS Procedure
DELETE_DATABASE_STATS Procedure
DELETE_DICTIONARY_STATS Procedure
DELETE_FIXED_OBJECTS_STATS Procedure
DELETE_INDEX_STATS Procedure
DELETE_SCHEMA_STATS Procedure
DELETE_SYSTEM_STATS Procedure

DELETE_TABLE_STATS Procedure

Transferring Statistics

Use the following procedures for creating and dropping the user statistics table.

CREATE_STAT_TABLE Procedure
DROP_STAT_TABLE Procedure

Use the following procedures to transfer statistics

- from the dictionary to a user statistics table (`EXPORT_*`)
- from a user statistics table to the dictionary (`IMPORT_*`)

EXPORT_COLUMN_STATS Procedure
EXPORT_DATABASE_STATS Procedure
EXPORT_DICTIONARY_STATS Procedure
EXPORT_FIXED_OBJECTS_STATS Procedure
EXPORT_INDEX_STATS Procedure
EXPORT_SCHEMA_STATS Procedure
EXPORT_SYSTEM_STATS Procedure
EXPORT_TABLE_STATS Procedure

IMPORT_COLUMN_STATS Procedure
IMPORT_DATABASE_STATS Procedure
IMPORT_DICTIONARY_STATS Procedure
IMPORT_FIXED_OBJECTS_STATS Procedure
IMPORT_INDEX_STATS Procedure
IMPORT_SCHEMA_STATS Procedure
IMPORT_SYSTEM_STATS Procedure
IMPORT_TABLE_STATS Procedure

Locking or Unlocking Statistics

Use the following procedures to lock and unlock statistics on objects.

LOCK_SCHEMA_STATS Procedure
LOCK_TABLE_STATS Procedure

UNLOCK_SCHEMA_STATS Procedure
UNLOCK_TABLE_STATS Procedure

The `LOCK_*` procedures either freeze the current set of the statistics or to keep the statistics empty (uncollected).When statistics on a table are locked, all the statistics depending on the table, including table statistics, column statistics, histograms and statistics on all dependent indexes, are considered to be locked.

Restoring and Purging Statistics History

Use the following procedures to restore statistics as of a specified timestamp. This is useful in case newly collected statistics leads to some sub-optimal execution plans and the administrator wants to revert to the previous set of statistics.

RESET_PARAM_DEFAULTS Procedure
RESTORE_DICTIONARY_STATS Procedure
RESTORE_FIXED_OBJECTS_STATS Procedure
RESTORE_SCHEMA_STATS Procedure
RESTORE_SYSTEM_STATS Procedure

RESTORE_TABLE_STATS Procedure

Whenever statistics in dictionary are modified, old versions of statistics are saved automatically for future restoring. The old statistics are purged automatically at regular intervals based on the statistics history retention setting and the time of recent statistics gathering performed in the system. Retention is configurable using the ALTER_STATS_HISTORY_RETENTION Procedure.

The other DBMS_STATS procedures related to restoring statistics are:

- PURGE_STATS Procedure: This procedure lets you manually purge old versions beyond a time stamp.
- GET_STATS_HISTORY_RETENTION Function: This function gets the current statistics history retention value.
- GET_STATS_HISTORY_AVAILABILITY Function: This function gets the oldest time stamp where statistics history is available. Users cannot restore statistics to a time stamp older than the oldest time stamp.

RESTORE_* operations are not supported for user defined statistics.

User-Defined Statistics

The DBMS_STATS package supports operations on user-defined statistics. When a domain index or column is associated with a statistics type (using the associate statement), operations on the index or column manipulate user-defined statistics. For example, gathering statistics for a domain index (for which an association with a statistics type exists) using the GET_INDEX_STATS Procedures invokes the user-defined statistics collection method of the associated statistics type. Similarly, delete, transfer, import, and export operations manipulate user-defined statistics.

SET_* and GET_* operations for user-defined statistics are also supported using a special version of the SET and GET interfaces for columns and indexes.

EXPORT_*, IMPORT_* and RESTORE_* operations are not supported for user defined statistics.

Deprecated Subprograms

The following subprograms are obsolete with Release 10*g*:

- ALTER_DATABASE_TAB_MONITORING Procedure
- ALTER_SCHEMA_TAB_MONITORING Procedure

In earlier releases, you could use these subprograms to change DML monitoring behavior. These subprograms are now non-operational because Oracle performs their functions automatically.

Examples

Saving Original Statistics and Gathering New Statistics

Assume many modifications have been made to the `employees` table since the last time statistics were gathered. To ensure that the cost-based optimizer is still picking the best plan, statistics should be gathered once again; however, the user is concerned that new statistics will cause the optimizer to choose bad plans when the current ones are acceptable. The user can do the following:

```
BEGIN
   DBMS_STATS.CREATE_STAT_TABLE ('hr', 'savestats');
   DBMS_STATS.GATHER_TABLE_STATS ('hr', 'employees', stattab => 'savestats');
END;
```

This operation gathers new statistics on the `employees` table, but first saves the original statistics in a user statistics table: `hr.savestats`.

If the user believes that the new statistics are causing the optimizer to generate poor plans, then the original statistics can be restored as follows:

```
BEGIN
   DBMS_STATS.DELETE_TABLE_STATS ('hr', 'employees');
   DBMS_STATS.IMPORT_TABLE_STATS ('hr', 'employees', stattab => 'savestats');
END;
```

Gathering Daytime System Statistics

Assume that you want to perform database application processing OLTP transactions during the day and run reports at night.

To collect daytime system statistics, gather statistics for 720 minutes. Store the statistics in the `MYSTATS` table.

```
BEGIN
   DBMS_STATS.GATHER_SYSTEM_STATS (
      interval => 720,
      stattab  => 'mystats',
      statid   => 'OLTP');
END;
```

To collect nighttime system statistics, gather statistics for 720 minutes. Store the statistics in the `MYSTATS` table.

```
BEGIN
   DBMS_STATS.GATHER_SYSTEM_STATS (
      interval => 720,
      stattab  => 'mystats',
      statid   => 'OLAP');
END;
```

Update the dictionary with the gathered statistics.

```
VARIABLE   jobno   number;
BEGIN
   DBMS_JOB.SUBMIT (:jobno, 'DBMS_STATS.IMPORT_SYSTEM_STATS
   (''mystats'',''OLTP'');'
```

```
   sysdate, 'sysdate + 1');
   COMMIT;
END;

BEGIN
   DBMS_JOB.SUBMIT (:jobno, 'DBMS_STATS.IMPORT_SYSTEM_STATS
   (''mystats'',''OLAP'');'
   sysdate + 0.5, 'sysdate + 1');
   COMMIT;
END;
```

Summary of DBMS_STATS Subprograms

Table 103–1 DBMS_STATS Package Subprograms

Subprogram	Description
ALTER_DATABASE_TAB_MONITORING Procedure on page 103-17	Enables or disables the DML monitoring feature of all tables in the database, except for snapshot logs and the tables, which monitoring does not support [See Deprecated Subprograms on page 103-10]
ALTER_SCHEMA_TAB_MONITORING Procedure on page 103-18	Enables or disables the DML monitoring feature of all tables in the schema, except for snapshot logs and the tables, which monitoring does not support [See Deprecated Subprograms on page 103-10]
ALTER_STATS_HISTORY_RETENTION Procedure on page 103-19	Changes the statistics history retention value
CONVERT_RAW_VALUE Procedures on page 103-20	Convert the internal representation of a minimum or maximum value into a datatype-specific value
CONVERT_RAW_VALUE_NVARCHAR Procedure on page 103-21	Convert the internal representation of a minimum or maximum value into a datatype-specific value
CONVERT_RAW_VALUE_ROWID Procedure on page 103-22	Convert the internal representation of a minimum or maximum value into a datatype-specific value
CREATE_STAT_TABLE Procedure on page 103-23	Creates a table with name `stattab` in ownname's schema which is capable of holding statistics
DELETE_COLUMN_STATS Procedure on page 103-24	Deletes column-related statistics
DELETE_DATABASE_STATS Procedure on page 103-25	Deletes statistics for the entire database
DELETE_DICTIONARY_STATS Procedure on page 103-26	Deletes statistics for all dictionary schemas ('`SYS`', '`SYSTEM`' and RDBMS component schemas)
DELETE_FIXED_OBJECTS_STATS Procedure on page 103-25	Deletes statistics of all fixed tables
DELETE_INDEX_STATS Procedure on page 103-28	Deletes index-related statistics
DELETE_SCHEMA_STATS Procedure on page 103-29	Deletes schema-related statistics
DELETE_SYSTEM_STATS Procedure on page 103-30	Deletes system statistics
DELETE_TABLE_STATS Procedure on page 103-31	Deletes table-related statistics
DROP_STAT_TABLE Procedure on page 103-33	Drops a user statistics table created by `CREATE_STAT_TABLE`
EXPORT_COLUMN_STATS Procedure on page 103-34	Retrieves statistics for a particular column and stores them in the user statistics table identified by `stattab`
EXPORT_DATABASE_STATS Procedure on page 103-35	Retrieves statistics for all objects in the database and stores them in the user statistics table identified by `statown.stattab`

Table 103–1 (Cont.) DBMS_STATS Package Subprograms

Subprogram	Description
EXPORT_DICTIONARY_STATS Procedure on page 103-36	Retrieves statistics for all dictionary schemas ('SYS', 'SYSTEM' and RDBMS component schemas) and stores them in the user statistics table identified by stattab
EXPORT_FIXED_OBJECTS_ STATS Procedure on page 103-37	Retrieves statistics for fixed tables and stores them in the user statistics table identified by stattab
EXPORT_INDEX_STATS Procedure on page 103-38	Retrieves statistics for a particular index and stores them in the user statistics table identified by stattab
EXPORT_SCHEMA_STATS Procedure on page 103-39	Retrieves statistics for all objects in the schema identified by ownname and stores them in the user statistics table identified by stattab
EXPORT_SYSTEM_STATS Procedure on page 103-40	Retrieves system statistics and stores them in the user statistics table
EXPORT_TABLE_STATS Procedure on page 103-41	Retrieves statistics for a particular table and stores them in the user statistics table
FLUSH_DATABASE_ MONITORING_INFO Procedure on page 103-42	Flushes in-memory monitoring information for all the tables to the dictionary
GATHER_DATABASE_STATS Procedures on page 103-43	Gathers statistics for all objects in the database
GATHER_DICTIONARY_STATS Procedure on page 103-43	Gathers statistics for dictionary schemas 'SYS', 'SYSTEM' and schemas of RDBMS components
GATHER_FIXED_OBJECTS_ STATS Procedure on page 103-49	Gathers statistics of fixed objects
GATHER_INDEX_STATS Procedure on page 103-50	Gathers index statistics
GATHER_SCHEMA_STATS Procedures on page 103-52	Gathers statistics for all objects in a schema
GATHER_SYSTEM_STATS Procedure on page 103-56	Gathers system statistics
GATHER_TABLE_STATS Procedure on page 103-58	Gathers table and column (and index) statistics
GENERATE_STATS Procedure on page 103-61	Generates object statistics from previously collected statistics of related objects
GET_COLUMN_STATS Procedures on page 103-62	Gets all column-related information
GET_INDEX_STATS Procedures on page 103-64	Gets all index-related information
GET_PARAM Function on page 103-67	Gets the default value of parameters of DBMS_STATS procedures
GET_STATS_HISTORY_ AVAILABILITY Function on page 103-68	Gets the oldest timestamp where statistics history is available
GET_STATS_HISTORY_ RETENTION Function on page 103-69	Returns the current retention value
GET_SYSTEM_STATS Procedure on page 103-70	Gets system statistics from stattab, or from the dictionary if stattab is NULL

Table 103–1 (Cont.) DBMS_STATS Package Subprograms

Subprogram	Description
GET_TABLE_STATS Procedure on page 103-72	Gets all table-related information
IMPORT_COLUMN_STATS Procedure on page 103-74	Retrieves statistics for a particular column from the user statistics table identified by `stattab` and stores them in the dictionary
IMPORT_DATABASE_STATS Procedure on page 103-75	Retrieves statistics for all objects in the database from the user statistics table and stores them in the dictionary
IMPORT_DICTIONARY_STATS Procedure on page 103-76	Retrieves statistics for all dictionary schemas ('`SYS`', '`SYSTEM`' and RDBMS component schemas) from the user statistics table and stores them in the dictionary
IMPORT_FIXED_OBJECTS_ STATS Procedure on page 103-77	Retrieves statistics for fixed tables from the user statistics table identified by `stattab` and stores them in the dictionary
IMPORT_INDEX_STATS Procedure on page 103-78	Retrieves statistics for a particular index from the user statistics table identified by `stattab` and stores them in the dictionary
IMPORT_SCHEMA_STATS Procedure on page 103-79	Retrieves statistics for all objects in the schema identified by `ownname` from the user statistics table and stores them in the dictionary
IMPORT_SYSTEM_STATS Procedure on page 103-80	Retrieves system statistics from the user statistics table and stores them in the dictionary
IMPORT_TABLE_STATS Procedure on page 103-81	Retrieves statistics for a particular table from the user statistics table identified by `stattab` and stores them in the dictionary
LOCK_SCHEMA_STATS Procedure on page 103-82	Locks the statistics of all tables of a schema
LOCK_TABLE_STATS Procedure on page 103-83	Locks the statistics on the table
PREPARE_COLUMN_VALUES Procedures on page 103-84	Converts user-specified minimum, maximum, and histogram endpoint datatype-specific values into Oracle's internal representation for future storage using the SET_COLUMN_STATS Procedures
PREPARE_COLUMN_VALUES_ NVARCHAR2 Procedure on page 103-86	Converts user-specified minimum, maximum, and histogram endpoint datatype-specific values into Oracle's internal representation for future storage using the SET_COLUMN_STATS Procedures
PREPARE_COLUMN_VALUES_ ROWID Procedure on page 103-88	Converts user-specified minimum, maximum, and histogram endpoint datatype-specific values into Oracle's internal representation for future storage using the SET_COLUMN_STATS Procedures
PURGE_STATS Procedure on page 103-90	Purges old versions of statistics saved in the dictionary
RESET_PARAM_DEFAULTS Procedure on page 103-91	Resets the default values of all parameters to Oracle recommended values
RESET_PARAM_DEFAULTS Procedure on page 103-91	Restores statistics of all tables of the database as of a specified timestamp
RESTORE_DICTIONARY_STATS Procedure on page 103-93	Restores statistics of all dictionary tables (tables of '`SYS`', '`SYSTEM`' and RDBMS component schemas) as of a specified timestamp

Table 103–1 (Cont.) DBMS_STATS Package Subprograms

Subprogram	Description
RESTORE_FIXED_OBJECTS_ STATS Procedure on page 103-94	Restores statistics of all fixed tables as of a specified timestamp
RESTORE_SCHEMA_STATS Procedure on page 103-95	Restores statistics of all tables of a schema as of a specified timestamp
RESTORE_SYSTEM_STATS Procedure on page 103-96	Restores statistics of all tables of a schema as of a specified timestamp
RESTORE_TABLE_STATS Procedure on page 103-97	Restores statistics of a table as of a specified timestamp (as_of_timestamp), as well as statistics of associated indexes and columns
SET_COLUMN_STATS Procedures on page 103-98	Sets column-related information
SET_INDEX_STATS Procedures on page 103-100	Sets index-related information
SET_PARAM Procedure on page 103-103	Sets default values for parameters of `DBMS_STATS` procedures
SET_SYSTEM_STATS Procedure on page 103-105	Sets system statistics
SET_TABLE_STATS Procedure on page 103-107	Sets table-related information
UNLOCK_SCHEMA_STATS Procedure on page 103-109	Unlocks the statistics on all the table in a schema
UNLOCK_TABLE_STATS Procedure on page 103-110	Unlocks the statistics on the table
UPGRADE_STAT_TABLE Procedure on page 103-111	Upgrades user statistics on an older table

ALTER_DATABASE_TAB_MONITORING Procedure

> **Note:** See Deprecated Subprograms on page 103-10.

This procedure enables or disables the DML monitoring feature of all the tables in the schema, except for snapshot logs and the tables, which monitoring does not support. Using this procedure is equivalent to issuing `ALTER TABLE...MONITORING` (or `NOMONITORING`) individually.

Syntax

```
DBMS_STATS.ALTER_DATABASE_TAB_MONITORING (
   monitoring BOOLEAN DEFAULT TRUE,
   sysobjs    BOOLEAN DEFAULT FALSE);
```

Parameters

Table 103–2 ALTER_DATABASE_TAB_MONITORING Procedure Parameters

Parameter	Description
`monitoring`	Enables monitoring if true, and disables monitoring if false
`sysobjs`	If true, changes monitoring on the dictionary objects

Exceptions

`ORA-20000`: Insufficient privileges.

ALTER_SCHEMA_TAB_MONITORING Procedure

Note: See Deprecated Subprograms on page 103-10.

This procedure enables or disables the DML monitoring feature of all the tables in the schema, except for snapshot logs and the tables, which monitoring does not support. Using this procedure is equivalent to issuing ALTER TABLE...MONITORING (or NOMONITORING) individually.

Syntax

```
DBMS_STATS.ALTER_SCHEMA_TAB_MONITORING (
   ownname    VARCHAR2 DEFAULT NULL,
   monitoring BOOLEAN DEFAULT TRUE);
```

Parameters

Table 103–3 ALTER_SCHEMA_TAB_MONITORING Procedure Parameters

Parameter	Description
ownname	The name of the schema. (NULL means the current schema.)
monitoring	Enables monitoring if TRUE, and disables monitoring if FALSE

Usage Notes

You should enable monitoring if you use GATHER_DATABASE_STATS or GATHER_SCHEMA_STATS with the GATHER AUTO or GATHER STALE options.

Exceptions

ORA-20000: Insufficient privileges.

ALTER_STATS_HISTORY_RETENTION Procedure

This procedure changes the statistics history retention value. Statistics history retention is used by both the automatic purge and PURGE_STATS Procedure.

Syntax

```
DBMS_STATS.ALTER_STATS_HISTORY_RETENTION (
   retention       IN     NUMBER);
```

Parameters

Table 103–4 ALTER_STATS_HISTORY_RETENTION Procedure Parameters

Parameter	Description
retention	The retention time in days. The statistics history will be retained for at least these many number of days.The valid range is [1,365000]. Also you can use the following values for special purposes: ■ 0 - old statistics are never saved. The automatic purge will delete all statistics history ■ 1 - statistics history is never purged by automatic purge. ■ NULL - change statistics history retention to default value

Usage Notes

To run this procedure, you must have the SYSDBA or both ANALYZE ANY DICTIONARY and ANALYZE ANY system privilege.

Exceptions

ORA-20000: Insufficient privileges.

CONVERT_RAW_VALUE Procedures

This procedure converts the internal representation of a minimum or maximum value into a datatype-specific value. The minval and maxval fields of the `StatRec` structure as filled in by `GET_COLUMN_STATS` or `PREPARE_COLUMN_VALUES` are appropriate values for input.

Syntax

```
DBMS_STATS.CONVERT_RAW_VALUE (
   rawval     RAW,
   resval OUT BINARY_FLOAT);

DBMS_STATS.CONVERT_RAW_VALUE (
   rawval     RAW,
   resval OUT BINARY_DOUBLE);

DBMS_STATS.CONVERT_RAW_VALUE (
   rawval     RAW,
   resval OUT DATE);

DBMS_STATS.CONVERT_RAW_VALUE (
   rawval     RAW,
   resval OUT NUMBER);

DBMS_STATS.CONVERT_RAW_VALUE (
   rawval     RAW,
   resval OUT VARCHAR2);
```

Pragmas

```
pragma restrict_references(convert_raw_value, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 103–5 CONVERT_RAW_VALUE Procedure Parameters

Parameter	Description
`rawval`	The raw representation of a column minimum or maximum datatype-specific output parameters
`resval`	The converted, type-specific value

CONVERT_RAW_VALUE_NVARCHAR Procedure

This procedure converts the internal representation of a minimum or maximum value into a datatype-specific value. The minval and maxval fields of the `StatRec` structure as filled in by `GET_COLUMN_STATS` or `PREPARE_COLUMN_VALUES` are appropriate values for input.

Syntax

```
DBMS_STATS.CONVERT_RAW_VALUE_NVARCHAR (
   rawval     RAW,
   resval OUT NVARCHAR2);
```

Pragmas

```
pragma restrict_references(convert_raw_value_nvarchar, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 103–6 CONVERT_RAW_VALUE_NVARCHAR Procedure Parameters

Parameter	Description
`rawval`	The raw representation of a column minimum or maximum datatype-specific output parameters
`resval`	The converted, type-specific value

CONVERT_RAW_VALUE_ROWID Procedure

This procedure converts the internal representation of a minimum or maximum value into a datatype-specific value. The minval and maxval fields of the `StatRec` structure as filled in by `GET_COLUMN_STATS` or `PREPARE_COLUMN_VALUES` are appropriate values for input.

Syntax

```
DBMS_STATS.CONVERT_RAW_VALUE_ROWID (
   rawval     RAW,
   resval OUT ROWID);
```

Pragmas

```
pragma restrict_references(convert_raw_value_rowid, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 103–7 CONVERT_RAW_VALUE_ROWID Procedure Parameters

Parameter	Description
`rawval`	The raw representation of a column minimum or maximum datatype-specific output parameters
`resval`	The converted, type-specific value

CREATE_STAT_TABLE Procedure

This procedure creates a table with name `stattab` in `ownname`'s schema which is capable of holding statistics. The columns and types that compose this table are not relevant as it should be accessed solely through the procedures in this package.

Syntax

```
DBMS_STATS.CREATE_STAT_TABLE (
   ownname  VARCHAR2,
   stattab  VARCHAR2,
   tblspace VARCHAR2 DEFAULT NULL);
```

Parameters

Table 103–8 CREATE_STAT_TABLE Procedure Parameters

Parameter	Description
`ownname`	Name of the schema
`stattab`	Name of the table to create. This value should be passed as the `stattab` parameter to other procedures when the user does not want to modify the dictionary statistics directly.
`tblspace`	Tablespace in which to create the statistics tables. If none is specified, then they are created in the user's default tablespace.

Exceptions

`ORA-20000`: Table already exists or insufficient privileges.

`ORA-20001`: Tablespace does not exist.

DELETE_COLUMN_STATS Procedure

This procedure deletes column-related statistics.

Syntax

```
DBMS_STATS.DELETE_COLUMN_STATS (
   ownname       VARCHAR2,
   tabname       VARCHAR2,
   colname       VARCHAR2,
   partname      VARCHAR2 DEFAULT NULL,
   stattab       VARCHAR2 DEFAULT NULL,
   statid        VARCHAR2 DEFAULT NULL,
   cascade_parts BOOLEAN  DEFAULT TRUE,
   statown       VARCHAR2 DEFAULT NULL,
   no_invalidate BOOLEAN  DEFAULT to_no_invalidate_type (
                                    get_param('NO_INVALIDATE')),
   force         BOOLEAN DEFAULT FALSE);
```

Parameters

Table 103–9 DELETE_COLUMN_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema
tabname	Name of the table to which this column belongs
colname	Name of the column
partname	Name of the table partition for which to delete the statistics. If the table is partitioned and if partname is NULL, then global column statistics are deleted.
stattab	User statistics table identifier describing from where to delete the statistics. If stattab is NULL, then the statistics are deleted directly from the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).
cascade_parts	If the table is partitioned and if partname is NULL, then setting this to true causes the deletion of statistics for this column for all underlying partitions as well.
statown	Schema containing stattab (if different than ownname)
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_PARAM Procedure.
force	When value of this argument is TRUE, deletes column statistics even if locked

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20005: Object statistics are locked.

DELETE_DATABASE_STATS Procedure

This procedure deletes statistics for all the tables in a database.

Syntax

```
DBMS_STATS.DELETE_DATABASE_STATS (
   stattab          VARCHAR2 DEFAULT NULL,
   statid           VARCHAR2 DEFAULT NULL,
   statown          VARCHAR2 DEFAULT NULL,
   no_invalidate    BOOLEAN  DEFAULT to_no_invalidate_type (
                                     get_param('NO_INVALIDATE')),
   force            BOOLEAN  DEFAULT FALSE);
```

Parameters

Table 103–10 DELETE_DATABASE_STATS Procedure Parameters

Parameter	Description
stattab	User statistics table identifier describing from where to delete the statistics. If stattab is NULL, then the statistics are deleted directly in the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL)
statown	Schema containing stattab (if different from current schema)
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_PARAM Procedure.
force	When the value of this argument is TRUE, deletes statistics of tables in a database even if they are locked

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

DELETE_DICTIONARY_STATS Procedure

This procedure deletes statistics for all dictionary schemas ('SYS', 'SYSTEM' and RDBMS component schemas).

Syntax

```
DBMS_STATS.DELETE_DICTIONARY_STATS (
   stattab          VARCHAR2 DEFAULT NULL,
   statid           VARCHAR2 DEFAULT NULL,
   statown          VARCHAR2 DEFAULT NULL,
   no_invalidate    BOOLEAN  DEFAULT to_no_invalidate_type (
                                     get_param('NO_INVALIDATE')),
   force            BOOLEAN  DEFAULT FALSE);
```

Parameters

Table 103–11 DELETE_DICTIONARY_STATS Procedure Parameters

Parameter	Description
stattab	User statistics table identifier describing from where to delete the statistics. If stattab is NULL, then the statistics are deleted directly in the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL)
statown	Schema containing stattab (if different from current schema)
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_PARAM Procedure.
force	When the value of this argument is TRUE, deletes statistics of tables in a database even if they are locked

Usage Notes

You must have the SYSDBA or both ANALYZE ANY DICTIONARY and ANALYZE ANY system privilege to execute this procedure.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20002: Bad user statistics table, may need to upgrade it.

DELETE_FIXED_OBJECTS_STATS Procedure

This procedure deletes statistics of all fixed tables.

Syntax

```
DBMS_STATS.DELETE_FIXED_OBJECTS_STATS (
   stattab        VARCHAR2 DEFAULT NULL,
   statid         VARCHAR2 DEFAULT NULL,
   statown        VARCHAR2 DEFAULT NULL,
   no_invalidate  BOOLEAN  DEFAULT to_no_invalidate_type (
                                    get_param('NO_INVALIDATE')),
   force          BOOLEAN  DEFAULT FALSE);
```

Parameters

Table 103–12 DELETE_FIXED_OBJECTS_STATS Procedure Parameters

Parameter	Description
stattab	The user statistics table identifier describing from where to delete the current statistics. If stattab is NULL, the statistics will be deleted directly in the dictionary.
statid	The (optional) identifier to associate with these statistics within stattab. This only applies if stattab is not NULL.
statown	Schema containing stattab (if different from current schema)
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_PARAM Procedure.
force	Ignores the statistics lock on objects and deletes the statistics if set to TRUE

Usage Notes

You must have the SYSDBA or ANALYZE ANY DICTIONARY system privilege to execute this procedure.

Exceptions

ORA-20000: Insufficient privileges.

ORA-20002: Bad user statistics table, may need to upgrade it.

DELETE_INDEX_STATS Procedure

This procedure deletes index-related statistics.

Syntax

```
DBMS_STATS.DELETE_INDEX_STATS (
   ownname          VARCHAR2,
   indname          VARCHAR2,
   partname         VARCHAR2 DEFAULT NULL,
   stattab          VARCHAR2 DEFAULT NULL,
   statid           VARCHAR2 DEFAULT NULL,
   cascade_parts    BOOLEAN  DEFAULT TRUE,
   statown          VARCHAR2 DEFAULT NULL,
   no_invalidate    BOOLEAN  DEFAULT to_no_invalidate_type (
                                     get_param('NO_INVALIDATE')),
   force            BOOLEAN DEFAULT FALSE);
```

Parameters

Table 103–13 DELETE_INDEX_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema
indname	Name of the index
partname	Name of the index partition for which to delete the statistics. If the index is partitioned and if partname is NULL, then index statistics are deleted at the global level.
stattab	User statistics table identifier describing from where to delete the statistics. If stattab is NULL, then the statistics are deleted directly from the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL)
cascade_parts	If the index is partitioned and if partname is NULL, then setting this to TRUE causes the deletion of statistics for this index for all underlying partitions as well
statown	Schema containing stattab (if different than ownname)
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_PARAM Procedure.
force	When value of this argument is TRUE, deletes index statistics even if locked

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20005: Object statistics are locked.

DELETE_SCHEMA_STATS Procedure

This procedure deletes statistics for an entire schema.

Syntax

```
DBMS_STATS.DELETE_SCHEMA_STATS (
   ownname          VARCHAR2,
   stattab          VARCHAR2 DEFAULT NULL,
   statid           VARCHAR2 DEFAULT NULL,
   statown          VARCHAR2 DEFAULT NULL,
   no_invalidate    BOOLEAN DEFAULT to_no_invalidate_type (
                                    get_param('NO_INVALIDATE')),
   force            BOOLEAN DEFAULT FALSE);
```

Parameters

Table 103–14 DELETE_SCHEMA_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema
stattab	User statistics table identifier describing from where to delete the statistics. If stattab is NULL, then the statistics are deleted directly in the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL)
statown	Schema containing stattab (if different than ownname)
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_PARAM Procedure.
force	When value of this argument is TRUE, deletes statistics of tables in a schema even if locked

Exceptions

ORA-20000: Object does not exist or insufficient privileges

DELETE_SYSTEM_STATS Procedure

This procedure deletes workload statistics (collected using the 'INTERVAL' or 'START' and 'STOP' options) and resets the default to noworkload statistics (collected using 'NOWORKLOAD' option) if stattab is not specified. If stattab is specified, the subprogram deletes all system statistics with the associated statid from the stattab.

Syntax

```
DBMS_STATS.DELETE_SYSTEM_STATS (
   stattab       VARCHAR2 DEFAULT NULL,
   statid        VARCHAR2 DEFAULT NULL,
   statown       VARCHAR2 DEFAULT NULL);
```

Parameters

Table 103–15 DELETE_SYSTEM_STATS Procedure Parameters

Parameter	Description
stattab	Identifier of the user statistics table where the statistics will be saved
statid	Optional identifier associated with the statistics saved in the stattab
statown	Schema containing stattab (if different from current schema)

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20002: Bad user statistics table; may need to be upgraded.

DELETE_TABLE_STATS Procedure

This procedure deletes table-related statistics.

Syntax

```
DBMS_STATS.DELETE_TABLE_STATS (
   ownname          VARCHAR2,
   tabname          VARCHAR2,
   partname         VARCHAR2 DEFAULT NULL,
   stattab          VARCHAR2 DEFAULT NULL,
   statid           VARCHAR2 DEFAULT NULL,
   cascade_parts    BOOLEAN  DEFAULT TRUE,
   cascade_columns  BOOLEAN  DEFAULT TRUE,
   cascade_indexes  BOOLEAN  DEFAULT TRUE,
   statown          VARCHAR2 DEFAULT NULL,
   no_invalidate    BOOLEAN  DEFAULT to_no_invalidate_type (
                                     get_param('NO_INVALIDATE')),
   force            BOOLEAN DEFAULT FALSE);
```

Parameters

Table 103–16 DELETE_TABLE_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema
tabname	Name of the table to which this column belongs
partname	Name of the table partition from which to get the statistics. If the table is partitioned and if partname is NULL, then the statistics are retrieved from the global table level.
stattab	User statistics table identifier describing from where to retrieve the statistics. If stattab is NULL, then the statistics are retrieved directly from the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL)
cascade_parts	If the table is partitioned and if partname is NULL, then setting this to TRUE causes the deletion of statistics for this table for all underlying partitions as well
cascade_columns	Indicates that DELETE_COLUMN_STATS should be called for all underlying columns (passing the cascade_parts parameter)
cascade_indexes	Indicates that DELETE_INDEX_STATS should be called for all underlying indexes (passing the cascade_parts parameter)
statown	Schema containing stattab (if different than ownname)
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_PARAM Procedure.
force	When value of this argument is TRUE, deletes table statistics even if locked

Exceptions

`ORA-20000`: Object does not exist or insufficient privileges.

`ORA-20005`: Object statistics are locked.

DROP_STAT_TABLE Procedure

This procedure drops a user statistics table.

Syntax

```
DBMS_STATS.DROP_STAT_TABLE (
   ownname VARCHAR2,
   stattab VARCHAR2);
```

Parameters

Table 103–17 DROP_STAT_TABLE Procedure Parameters

Parameter	Description
ownname	Name of the schema
stattab	User statistics table identifier

Exceptions

ORA-20000: Table does not exists or insufficient privileges.

EXPORT_COLUMN_STATS Procedure

This procedure retrieves statistics for a particular column and stores them in the user statistics table identified by `stattab`.

Syntax

```
DBMS_STATS.EXPORT_COLUMN_STATS (
   ownname  VARCHAR2,
   tabname  VARCHAR2,
   colname  VARCHAR2,
   partname VARCHAR2 DEFAULT NULL,
   stattab  VARCHAR2,
   statid   VARCHAR2 DEFAULT NULL,
   statown  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 103–18 EXPORT_COLUMN_STATS Procedure Parameters

Parameter	Description
`ownname`	Name of the schema
`tabname`	Name of the table to which this column belongs
`colname`	Name of the column
`partname`	Name of the table partition. If the table is partitioned and if `partname` is `NULL`, then global and partition column statistics are exported.
`stattab`	User statistics table identifier describing where to store the statistics
`statid`	Identifier (optional) to associate with these statistics within `stattab`
`statown`	Schema containing `stattab` (if different than `ownname`)

Exceptions

`ORA-20000`: Object does not exist or insufficient privileges.

EXPORT_DATABASE_STATS Procedure

This procedure retrieves statistics for all objects in the database and stores them in the user statistics tables identified by `statown.stattab`.

Syntax

```
DBMS_STATS.EXPORT_DATABASE_STATS (
   stattab VARCHAR2,
   statid  VARCHAR2 DEFAULT NULL,
   statown VARCHAR2 DEFAULT NULL);
```

Parameters

Table 103–19 EXPORT_DATABASE_STATS Procedure Parameters

Parameter	Description
`stattab`	User statistics table identifier describing where to store the statistics
`statid`	Identifier (optional) to associate with these statistics within `stattab`
`statown`	Schema containing `stattab` (if different from current schema)

Exceptions

`ORA-20000`: Object does not exist or insufficient privileges.

EXPORT_DICTIONARY_STATS Procedure

This procedure retrieves statistics for all dictionary schemas ('SYS', 'SYSTEM' and RDBMS component schemas) and stores them in the user statistics table identified by stattab.

Syntax

```
DBMS_STATS.EXPORT_DICTIONARY_STATS (
   stattab VARCHAR2,
   statid  VARCHAR2 DEFAULT NULL,
   statown VARCHAR2 DEFAULT NULL);
```

Parameters

Table 103–20 EXPORT_DICTIONARY_STATS Procedure Parameters

Parameter	Description
stattab	User statistics table identifier describing where to store the statistics
statid	Identifier (optional) to associate with these statistics within stattab
statown	Schema containing stattab (if different from current schema)

Usage Notes

You must have the SYSDBA or ANALYZE ANY DICTIONARY and ANALYZE ANY system privilege to execute this procedure.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20002: Bad user statistics table, may need to upgrade it.

EXPORT_FIXED_OBJECTS_STATS Procedure

This procedure retrieves statistics for fixed tables and stores them in the user statistics table identified by `stattab`.

Syntax

```
DBMS_STATS.EXPORT_FIXED_OBJECTS_STATS (
   stattab  VARCHAR2,
   statid   VARCHAR2 DEFAULT NULL,
   statown  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 103–21 EXPORT_FIXED_OBJECTS_STATS Procedure Parameters

Parameter	Description
`stattab`	User statistics table identifier describing where to store the statistics
`statid`	Identifier (optional) to associate with these statistics within `stattab`
`statown`	Schema containing `stattab` (if different from current schema)

Exceptions

`ORA-20000`: Object does not exist or insufficient privileges.

`ORA-20002`: Bad user statistics table, may need to upgrade it.

EXPORT_INDEX_STATS Procedure

This procedure retrieves statistics for a particular index and stores them in the user statistics table identified by `stattab`.

Syntax

```
DBMS_STATS.EXPORT_INDEX_STATS (
   ownname  VARCHAR2,
   indname  VARCHAR2,
   partname VARCHAR2 DEFAULT NULL,
   stattab  VARCHAR2,
   statid   VARCHAR2 DEFAULT NULL,
   statown  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 103–22 EXPORT_INDEX_STATS Procedure Parameters

Parameter	Description
`ownname`	Name of the schema
`indname`	Name of the index
`partname`	Name of the index partition. If the index is partitioned and if `partname` is `NULL`, then global and partition index statistics are exported.
`stattab`	User statistics table identifier describing where to store the statistics
`statid`	Identifier (optional) to associate with these statistics within `stattab`
`statown`	Schema containing `stattab` (if different than `ownname`)

Exceptions

`ORA-20000`: Object does not exist or insufficient privileges.

EXPORT_SCHEMA_STATS Procedure

This procedure retrieves statistics for all objects in the schema identified by `ownname` and stores them in the user statistics tables identified by `stattab`.

Syntax

```
DBMS_STATS.EXPORT_SCHEMA_STATS (
   ownname VARCHAR2,
   stattab VARCHAR2,
   statid  VARCHAR2 DEFAULT NULL,
   statown VARCHAR2 DEFAULT NULL);
```

Parameters

Table 103–23 EXPORT_SCHEMA_STATS Procedure Parameters

Parameter	Description
`ownname`	Name of the schema
`stattab`	User statistics table identifier describing where to store the statistics
`statid`	Identifier (optional) to associate with these statistics within `stattab`
`statown`	Schema containing `stattab` (if different than `ownname`)

Exceptions

`ORA-20000`: Object does not exist or insufficient privileges.

EXPORT_SYSTEM_STATS Procedure

This procedure retrieves system statistics and stores them in the user statistics table, identified by `stattab`.

Syntax

```
DBMS_STATS.EXPORT_SYSTEM_STATS (
   stattab       VARCHAR2,
   statid        VARCHAR2 DEFAULT NULL,
   statown       VARCHAR2 DEFAULT NULL);
```

Parameters

Table 103–24 EXPORT_SYSTEM_STATS Procedure Parameters

Parameter	Description
`stattab`	Identifier of the user statistics table that describes where the statistics will be stored.
`statid`	Optional identifier associated with the statistics stored from the `stattab`
`statown`	Schema containing `stattab` (if different from current schema)

Exceptions

`ORA-20000`: Object does not exist or insufficient privileges.

`ORA-20002`: Bad user statistics table; may need to be upgraded.

`ORA-20003`: Unable to export system statistics.

EXPORT_TABLE_STATS Procedure

This procedure retrieves statistics for a particular table and stores them in the user statistics table. Cascade results in all index and column statistics associated with the specified table being exported as well.

Syntax

```
DBMS_STATS.EXPORT_TABLE_STATS (
   ownname  VARCHAR2,
   tabname  VARCHAR2,
   partname VARCHAR2 DEFAULT NULL,
   stattab  VARCHAR2,
   statid   VARCHAR2 DEFAULT NULL,
   cascade  BOOLEAN  DEFAULT TRUE,
   statown  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 103–25 EXPORT_TABLE_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema
tabname	Name of the table
partname	Name of the table partition. If the table is partitioned and if partname is NULL, then global and partition table statistics are exported.
stattab	User statistics table identifier describing where to store the statistics
statid	Identifier (optional) to associate with these statistics within stattab
cascade	If true, then column and index statistics for this table are also exported
statown	Schema containing stattab (if different than ownname)

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

FLUSH_DATABASE_MONITORING_INFO Procedure

This procedure flushes in-memory monitoring information for all tables in the dictionary. Corresponding entries in the `*_TAB_MODIFICATIONS`, `*_TAB_STATISTICS` and `*_IND_STATISTICS` views are updated immediately, without waiting for the Oracle database to flush them periodically. This procedure is useful when you need up-to-date information in those views. Because the `GATHER_*_STATS` procedures internally flush monitoring information, it is not necessary to run this procedure before gathering the statistics.

Syntax

```
DBMS_STATS.FLUSH_DATABASE_MONITORING_INFO;
```

Exceptions

`ORA-20000`: Insufficient privileges.

GATHER_DATABASE_STATS Procedures

This procedure gathers statistics for all objects in the database.

Syntax

```
DBMS_STATS.GATHER_DATABASE_STATS (
   estimate_percent NUMBER   DEFAULT to_estimate_percent_type
                                        (get_param('ESTIMATE_PERCENT')),
   block_sample     BOOLEAN  DEFAULT FALSE,
   method_opt       VARCHAR2 DEFAULT get_param('METHOD_OPT'),
   degree           NUMBER   DEFAULT to_degree_type(get_param('DEGREE')),
   granularity      VARCHAR2 DEFAULT GET_PARAM('GRANULARITY'),
   cascade          BOOLEAN  DEFAULT to_cascade_type(get_param('CASCADE')),
   stattab          VARCHAR2 DEFAULT NULL,
   statid           VARCHAR2 DEFAULT NULL,
   options          VARCHAR2 DEFAULT 'GATHER',
   objlist          OUT      ObjectTab,
   statown          VARCHAR2 DEFAULT NULL,
   gather_sys       BOOLEAN  DEFAULT TRUE,
   no_invalidate    BOOLEAN  DEFAULT to_no_invalidate_type (
                                    get_param('NO_INVALIDATE')));

DBMS_STATS.GATHER_DATABASE_STATS (
   estimate_percent NUMBER   DEFAULT to_estimate_percent_type
                                        (get_param('ESTIMATE_PERCENT')),
   block_sample     BOOLEAN  DEFAULT FALSE,
   method_opt       VARCHAR2 DEFAULT get_param('METHOD_OPT'),
   degree           NUMBER   DEFAULT to_degree_type(get_param('DEGREE')),
   granularity      VARCHAR2 DEFAULT GET_PARAM('GRANULARITY'),
   cascade          BOOLEAN  DEFAULT to_cascade_type(get_param('CASCADE')),
   stattab          VARCHAR2 DEFAULT NULL,
   statid           VARCHAR2 DEFAULT NULL,
   options          VARCHAR2 DEFAULT 'GATHER',
   statown          VARCHAR2 DEFAULT NULL,
   gather_sys       BOOLEAN  DEFAULT TRUE,
   no_invalidate    BOOLEAN  DEFAULT to_no_invalidate_type (
                                    get_param('NO_INVALIDATE')));
```

Parameters

Table 103–26 GATHER_DATABASE_STATS Procedure Parameters

Parameter	Description
estimate_percent	Percentage of rows to estimate (NULL means compute): The valid range is [0.000001,100]. Use the constant DBMS_STATS.AUTO_SAMPLE_SIZE to have Oracle determine the appropriate sample size for good statistics. This is the default.The default value can be changed using the SET_PARAM Procedure.
block_sample	Whether or not to use random block sampling instead of random row sampling. Random block sampling is more efficient, but if the data is not randomly distributed on disk, then the sample values may be somewhat correlated. Only pertinent when doing an estimate statistics.

Table 103–26 (Cont.) GATHER_DATABASE_STATS Procedure Parameters

Parameter	Description
method_opt	Accepts: ▪ FOR ALL [INDEXED | HIDDEN] COLUMNS [size_clause] ▪ FOR COLUMNS [size clause] column|attribute [size_clause] [,column|attribute [size_clause]...] size_clause is defined as size_clause := SIZE {integer | REPEAT | AUTO | SKEWONLY} – integer : Number of histogram buckets. Must be in the range [1,254]. – REPEAT : Collects histograms only on the columns that already have histograms. – AUTO : Oracle determines the columns to collect histograms based on data distribution and the workload of the columns. – SKEWONLY : Oracle determines the columns to collect histograms based on the data distribution of the columns. The default is FOR ALL COLUMNS SIZE AUTO.The default value can be changed using the SET_PARAM Procedure.
degree	Degree of parallelism. The default for degree is NULL. The default value can be changed using the SET_PARAM Procedure. NULL means use the table default value specified by the DEGREE clause in the CREATE TABLE or ALTER TABLE statement. Use the constant DBMS_STATS.DEFAULT_DEGREE to specify the default value based on the initialization parameters. The AUTO_DEGREE value determines the degree of parallelism automatically. This is either 1 (serial execution) or DEFAULT_DEGREE (the system default value based on number of CPUs and initialization parameters) according to size of the object.
granularity	Granularity of statistics to collect (only pertinent if the table is partitioned). 'ALL' - gathers all (subpartition, partition, and global) statistics 'AUTO'- determines the granularity based on the partitioning type. This is the default value. 'DEFAULT' - gathers global and partition-level statistics. This option is obsolete, and while currently supported, it is included in the documentation for legacy reasons only. You should use the 'GLOBAL AND PARTITION' for this functionality. Note that the default value is now 'AUTO'. 'GLOBAL' - gathers global statistics 'GLOBAL AND PARTITION' - gathers the global and partition level statistics. No subpartition level statistics are gathered even if it is a composite partitioned object. 'PARTITION'- gathers partition-level statistics 'SUBPARTITION' - gathers subpartition-level statistics.
cascade	Gather statistics on the indexes as well. Index statistics gathering is not parallelized. Using this option is equivalent to running the GATHER_INDEX_STATS Procedure on each of the indexes in the database in addition to gathering table and column statistics. Use the constant DBMS_STATS.AUTO_CASCADE to have Oracle determine whether index statistics to be collected or not. This is the default. The default value can be changed using the SET_PARAM Procedure.

Table 103–26 (Cont.) GATHER_DATABASE_STATS Procedure Parameters

Parameter	Description
stattab	User statistics table identifier describing where to save the current statistics. The statistics table is assumed to reside in the same schema as the object being analyzed, so there must be one such table in each schema to use this option.
statid	Identifier (optional) to associate with these statistics within stattab.
options	Further specification of which objects to gather statistics for: GATHER: Gathers statistics on all objects in the schema. GATHER AUTO: Gathers all necessary statistics automatically. Oracle implicitly determines which objects need new statistics, and determines how to gather those statistics. When GATHER AUTO is specified, the only additional valid parameters are stattab, statid, objlist and statown; all other parameter settings are ignored. Returns a list of processed objects. GATHER STALE: Gathers statistics on stale objects as determined by looking at the *_tab_modifications views. Also, return a list of objects found to be stale. GATHER EMPTY: Gathers statistics on objects which currently have no statistics. Return a list of objects found to have no statistics. LIST AUTO: Returns a list of objects to be processed with GATHER AUTO. LIST STALE: Returns a list of stale objects as determined by looking at the *_tab_modifications views. LIST EMPTY: Returns a list of objects which currently have no statistics.
objlist	List of objects found to be stale or empty
statown	Schema containing stattab (if different from current schema)
gather_sys	Gathers statistics on the objects owned by the 'SYS' user
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_PARAM Procedure.

Usage Notes

Statistics for external tables are not collected by this procedure.

Exceptions

ORA-20000: Insufficient privileges.

ORA-20001: Bad input value.

GATHER_DICTIONARY_STATS Procedure

This procedure gathers statistics for dictionary schemas 'SYS', 'SYSTEM' and schemas of RDBMS components.

Syntax

```
DBMS_STATS.GATHER_DICTIONARY_STATS (
   comp_id          VARCHAR2 DEFAULT NULL,
   estimate_percent NUMBER   DEFAULT to_estimate_percent_type
                                          (get_param('ESTIMATE_PERCENT')),
   block_sample     BOOLEAN  DEFAULT FALSE,
   method_opt       VARCHAR2 DEFAULT get_param('METHOD_OPT'),
   degree           NUMBER   DEFAULT to_degree_type(get_param('DEGREE')),
   granularity      VARCHAR2 DEFAULT GET_PARAM('GRANULARITY'),
   cascade          BOOLEAN  DEFAULT to_cascade_type(get_param('CASCADE')),
   stattab          VARCHAR2 DEFAULT NULL,
   statid           VARCHAR2 DEFAULT NULL,
   options          VARCHAR2 DEFAULT 'GATHER AUTO',
   objlist    OUT   ObjectTab,
   statown          VARCHAR2 DEFAULT NULL,
   no_invalidate    BOOLEAN  DEFAULT to_no_invalidate_type (
                                     get_param('NO_INVALIDATE')));

DBMS_STATS.GATHER_DICTIONARY_STATS (
   comp_id           VARCHAR2 DEFAULT NULL,
   estimate_percent  NUMBER DEFAULT
                        to_estimate_percent_type(GET_PARAM('ESTIMATE_PERCENT')),
   block_sample      BOOLEAN DEFAULT FALSE,
   method_opt        VARCHAR2 DEFAULT GET_PARAM('METHOD_OPT'),
   degree            NUMBER DEFAULT to_degree_type(GET_PARAM('DEGREE')),
   granularity       VARCHAR2 DEFAULT GET_PARAM('GRANULARITY'),
   cascade           BOOLEAN DEFAULT to_cascade_type(GET_PARAM('CASCADE')),
   stattab           VARCHAR2 DEFAULT NULL,
   statid            VARCHAR2 DEFAULT NULL,
   options           VARCHAR2 DEFAULT 'GATHER AUTO',
   statown           VARCHAR2 DEFAULT NULL,
   no_invalidate     BOOLEAN DEFAULT
                             to_no_invalidate_type(get_param('NO_INVALIDATE')));
```

Parameters

Table 103–27 GATHER_DICTIONARY_STATS Procedure Parameters

Parameter	Description
comp_id	The component id of the schema to analyze (NULL will result in analyzing schemas of all RDBMS components).Please refer to comp_id column of DBA_REGISTRY view. The procedure always gather statistics on 'SYS' and 'SYSTEM' schemas regardless of this argument.
estimate_percent	Percentage of rows to estimate (NULL means compute). The valid range is [0.000001,100]. Use the constant DBMS_STATS.AUTO_SAMPLE_SIZE to have Oracle determine the appropriate sample size for good statistics. This is the default.The default value can be changed using the SET_PARAM Procedure .

Table 103–27 (Cont.) GATHER_DICTIONARY_STATS Procedure Parameters

Parameter	Description
block_sample	Determines whether or not to use random block sampling instead of random row sampling. Random block sampling is more efficient, but if the data is not randomly distributed on disk then the sample values may be somewhat correlated. Only pertinent when performing estimate statistics.
method_opt	Accepts: ▪ FOR ALL [INDEXED | HIDDEN] COLUMNS [size_clause] ▪ FOR COLUMNS [size clause] column|attribute [size_clause] [,column|attribute [size_clause]...] size_clause is defined as size_clause := SIZE {integer | REPEAT | AUTO | SKEWONLY} – integer : Number of histogram buckets. Must be in the range [1,254]. – REPEAT : Collects histograms only on the columns that already have histograms. – AUTO : Oracle determines the columns to collect histograms based on data distribution and the workload of the columns. – SKEWONLY : Oracle determines the columns to collect histograms based on the data distribution of the columns. The default is FOR ALL COLUMNS SIZE AUTO.The default value can be changed using the SET_PARAM Procedure.
degree	Degree of parallelism. The default for degree is NULL. The default value can be changed using the SET_PARAM Procedure. NULL means use of table default value that was specified by the DEGREE clause in the CREATE or ALTER INDEX statement. Use the constant DBMS_STATS.DEFAULT_DEGREE for the default value based on the initialization parameters. The AUTO_DEGREE value determines the degree of parallelism automatically. This is either 1 (serial execution) or DEFAULT_DEGREE (the system default value based on number of CPUs and initialization parameters) according to size of the object.
granularity	Granularity of statistics to collect (only pertinent if the table is partitioned). 'ALL' - gathers all (subpartition, partition, and global) statistics 'AUTO'- determines the granularity based on the partitioning type. This is the default value. 'DEFAULT' - gathers global and partition-level statistics. This option is obsolete, and while currently supported, it is included in the documentation for legacy reasons only. You should use the 'GLOBAL AND PARTITION' for this functionality. Note that the default value is now 'AUTO'. 'GLOBAL' - gathers global statistics 'GLOBAL AND PARTITION' - gathers the global and partition level statistics. No subpartition level statistics are gathered even if it is a composite partitioned object. 'PARTITION'- gathers partition-level statistics 'SUBPARTITION' - gathers subpartition-level statistics.

Table 103–27 (Cont.) GATHER_DICTIONARY_STATS Procedure Parameters

Parameter	Description
cascade	Gathers statistics on indexes also.Index statistics gathering will not be parallelized. Using this option is equivalent to running the GATHER_INDEX_STATS Procedure on each of the indexes in the schema in addition to gathering table and column statistics. Use the constant DBMS_STATS.AUTO_CASCADE to have Oracle determine whether index statistics to be collected or not. This is the default.The default value can be changed using the SET_PARAM Procedure.
stattab	User statistics table identifier describing where to save the current statistics
statid	The (optional) identifier to associate with these statistics within stattab
options	Further specification of objects for which to gather statistics: ■ 'GATHER' - gathers statistics on all objects in the schema ■ 'GATHER AUTO' - gathers all necessary statistics automatically. Oracle implicitly determines which objects need new statistics and determines how to gather those statistics. When 'GATHER AUTO' is specified, the only additional valid parameters are comp_id, stattab, statid and statown; all other parameter settings will be ignored. Also, returns a list of objects processed. ■ 'GATHER STALE' - gathers statistics on stale objects as determined by looking at the *_tab_modifications views. Also, returns a list of objects found to be stale. ■ 'GATHER EMPTY' - gathers statistics on objects which currently have no statistics. Also, returns a list of objects found to have no statistics. ■ 'LIST AUTO' - returns list of objects to be processed with 'GATHER AUTO' ■ 'LIST STALE' - returns list of stale objects as determined by looking at the *_tab_modifications views ■ 'LIST EMPTY' - returns list of objects which currently have no statistics
objlist	The list of objects found to be stale or empty
statown	Schema containing stattab (if different from current schema)
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_PARAM Procedure.

Usage Notes

You must have the SYSDBA or both ANALYZE ANY DICTIONARY and ANALYZE ANY system privilege to execute this procedure.

Exceptions

ORA-20000: Index does not exist or insufficient privileges.

ORA-20001: Bad input value.

ORA-20002: Bad user statistics table, may need to upgrade it.

GATHER_FIXED_OBJECTS_STATS Procedure

This procedure gathers statistics for all fixed objects (dynamic performance tables).

Syntax

```
DBMS_STATS.GATHER_FIXED_OBJECTS_STATS (
   stattab        VARCHAR2 DEFAULT NULL,
   statid         VARCHAR2 DEFAULT NULL,
   statown        VARCHAR2 DEFAULT NULL,
   no_invalidate  BOOLEAN  DEFAULT to_no_invalidate_type (
                                      get_param('NO_INVALIDATE')));
```

Parameters

Table 103–28 GATHER_FIXED_OBJECTS_STATS Procedure Parameters

Parameter	Description
stattab	The user statistics table identifier describing where to save the current statistics
statid	The (optional) identifier to associate with these statistics within stattab
statown	Schema containing stattab (if different from current schema)
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_PARAM Procedure.

Usage Notes

You must have the SYSDBA or ANALYZE ANY DICTIONARY system privilege to execute this procedure.

Exceptions

ORA-20000: Insufficient privileges.

ORA-20001: Bad input value.

ORA-20002: Bad user statistics table, may need to upgrade it.

GATHER_INDEX_STATS Procedure

This procedure gathers index statistics. It attempts to parallelize as much of the work as possible. Restrictions are described in the individual parameters. This operation will not parallelize with certain types of indexes, including cluster indexes, domain indexes, and bitmap join indexes. The `granularity` and `no_invalidate` arguments are not relevant to these types of indexes.

Syntax

```
DBMS_STATS.GATHER_INDEX_STATS (
   ownname          VARCHAR2,
   indname          VARCHAR2,
   partname         VARCHAR2 DEFAULT NULL,
   estimate_percent NUMBER   DEFAULT to_estimate_percent_type
                                          (GET_PARAM('ESTIMATE_PERCENT')),
   stattab          VARCHAR2 DEFAULT NULL,
   statid           VARCHAR2 DEFAULT NULL,
   statown          VARCHAR2 DEFAULT NULL,
   degree           NUMBER   DEFAULT to_degree_type(get_param('DEGREE')),
   granularity      VARCHAR2 DEFAULT GET_PARAM('GRANULARITY'),
   no_invalidate    BOOLEAN  DEFAULT to_no_invalidate_type
                                          (GET_PARAM('NO_INVALIDATE')),
   force            BOOLEAN DEFAULT FALSE);
```

Parameters

Table 103–29 GATHER_INDEX_STATS Procedure Parameters

Parameter	Description
`ownname`	Schema of index to analyze
`indname`	Name of index
`partname`	Name of partition
`estimate_percent`	Percentage of rows to estimate (`NULL` means compute). The valid range is `[0.000001,100]`. Use the constant `DBMS_STATS.AUTO_SAMPLE_SIZE` to have Oracle determine the appropriate sample size for good statistics. This is the default.The default value can be changed using the SET_PARAM Procedure.
`stattab`	User statistics table identifier describing where to save the current statistics
`statid`	Identifier (optional) to associate with these statistics within `stattab`
`statown`	Schema containing `stattab` (if different than `ownname`)
`degree`	Degree of parallelism. The default for `degree` is `NULL`. The default value can be changed using the SET_PARAM Procedure. `NULL` means use of table default value that was specified by the `DEGREE` clause in the `CREATE/ALTER INDEX` statement. Use the constant `DBMS_STATS.DEFAULT_DEGREE` for the default value based on the initialization parameters. The AUTO_DEGREE value determines the degree of parallelism automatically. This is either 1 (serial execution) or `DEFAULT_DEGREE (the system default value based on number of CPUs and initialization parameters) according to size of the object.`

Table 103–29 (Cont.) GATHER_INDEX_STATS Procedure Parameters

Parameter	Description
granularity	Granularity of statistics to collect (only pertinent if the table is partitioned). 'ALL' - gathers all (subpartition, partition, and global) statistics 'AUTO'- determines the granularity based on the partitioning type. This is the default value. 'DEFAULT' - gathers global and partition-level statistics. This option is obsolete, and while currently supported, it is included in the documentation for legacy reasons only. You should use the 'GLOBAL AND PARTITION' for this functionality. Note that the default value is now 'AUTO'. 'GLOBAL' - gathers global statistics 'GLOBAL AND PARTITION' - gathers the global and partition level statistics. No subpartition level statistics are gathered even if it is a composite partitioned object. 'PARTITION'- gathers partition-level statistics 'SUBPARTITION' - gathers subpartition-level statistics.
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_PARAM Procedure.
force	Gather statistics on object even if it is locked

Exceptions

ORA-20000: Index does not exist or insufficient privileges.

ORA-20001: Bad input value.

GATHER_SCHEMA_STATS Procedures

This procedure gathers statistics for all objects in a schema.

Syntax

```
DBMS_STATS.GATHER_SCHEMA_STATS (
   ownname          VARCHAR2,
   estimate_percent NUMBER   DEFAULT to_estimate_percent_type
                                        (get_param('ESTIMATE_PERCENT')),
   block_sample     BOOLEAN  DEFAULT FALSE,
   method_opt       VARCHAR2 DEFAULT get_param('METHOD_OPT'),
   degree           NUMBER   DEFAULT to_degree_type(get_param('DEGREE')),
   granularity      VARCHAR2 DEFAULT GET_PARAM('GRANULARITY'),
   cascade          BOOLEAN  DEFAULT to_cascade_type(get_param('CASCADE')),
   stattab          VARCHAR2 DEFAULT NULL,
   statid           VARCHAR2 DEFAULT NULL,
   options          VARCHAR2 DEFAULT 'GATHER',
   objlist          OUT      ObjectTab,
   statown          VARCHAR2 DEFAULT NULL,
   no_invalidate    BOOLEAN  DEFAULT to_no_invalidate_type (
                                   get_param('NO_INVALIDATE')),
  force             BOOLEAN DEFAULT FALSE);

DBMS_STATS.GATHER_SCHEMA_STATS (
   ownname          VARCHAR2,
   estimate_percent NUMBER   DEFAULT to_estimate_percent_type
                                        (get_param('ESTIMATE_PERCENT')),
   block_sample     BOOLEAN  DEFAULT FALSE,
   method_opt       VARCHAR2 DEFAULT get_param('METHOD_OPT'),
   degree           NUMBER   DEFAULT to_degree_type(get_param('DEGREE')),
   granularity      VARCHAR2 DEFAULT GET_PARAM('GRANULARITY'),
   cascade          BOOLEAN  DEFAULT to_cascade_type(get_param('CASCADE')),
   stattab          VARCHAR2 DEFAULT NULL,
   statid           VARCHAR2 DEFAULT NULL,
   options          VARCHAR2 DEFAULT 'GATHER',
   statown          VARCHAR2 DEFAULT NULL,
   no_invalidate    BOOLEAN  DEFAULT to_no_invalidate_type (
                                   get_param('NO_INVALIDATE'),
   force            BOOLEAN DEFAULT FALSE);
```

Parameters

Table 103–30 GATHER_SCHEMA_STATS Procedure Parameters

Parameter	Description
ownname	Schema to analyze (NULL means current schema)
estimate_percent	Percentage of rows to estimate (NULL means compute): The valid range is [0.000001,100]. Use the constant DBMS_STATS.AUTO_SAMPLE_SIZE to have Oracle determine the appropriate sample size for good statistics. This is the default.The default value can be changed using the SET_PARAM Procedure.
block_sample	Whether or not to use random block sampling instead of random row sampling. Random block sampling is more efficient, but if the data is not randomly distributed on disk, then the sample values may be somewhat correlated. Only pertinent when doing an estimate statistics.

Table 103–30 (Cont.) GATHER_SCHEMA_STATS Procedure Parameters

Parameter	Description
method_opt	Accepts: ▪ FOR ALL [INDEXED | HIDDEN] COLUMNS [size_clause] ▪ FOR COLUMNS [size clause] column|attribute [size_clause] [,column|attribute [size_clause]...] size_clause is defined as size_clause := SIZE {integer | REPEAT | AUTO | SKEWONLY} – integer : Number of histogram buckets. Must be in the range [1,254]. – REPEAT : Collects histograms only on the columns that already have histograms. – AUTO : Oracle determines the columns to collect histograms based on data distribution and the workload of the columns. – SKEWONLY : Oracle determines the columns to collect histograms based on the data distribution of the columns. The default is FOR ALL COLUMNS SIZE AUTO.The default value can be changed using the SET_PARAM Procedure.
degree	Degree of parallelism. The default for degree is NULL. The default value can be changed using the SET_PARAM Procedure. NULL means use the table default value specified by the DEGREE clause in the CREATE TABLE or ALTER TABLE statement. Use the constant DBMS_STATS.DEFAULT_DEGREE to specify the default value based on the initialization parameters.The AUTO_DEGREE value determines the degree of parallelism automatically. This is either 1 (serial execution) or DEFAULT_DEGREE (the system default value based on number of CPUs and initialization parameters) according to size of the object.
granularity	Granularity of statistics to collect (only pertinent if the table is partitioned). 'ALL' - gathers all (subpartition, partition, and global) statistics 'AUTO'- determines the granularity based on the partitioning type. This is the default value. 'DEFAULT' - gathers global and partition-level statistics. This option is obsolete, and while currently supported, it is included in the documentation for legacy reasons only. You should use the 'GLOBAL AND PARTITION' for this functionality. Note that the default value is now 'AUTO'. 'GLOBAL' - gathers global statistics 'GLOBAL AND PARTITION' - gathers the global and partition level statistics. No subpartition level statistics are gathered even if it is a composite partitioned object. 'PARTITION'- gathers partition-level statistics 'SUBPARTITION' - gathers subpartition-level statistics.
cascade	Gather statistics on the indexes as well.Index statistics gathering is not parallelized. Using this option is equivalent to running the GATHER_INDEX_STATS Procedure on each of the indexes in the schema in addition to gathering table and column statistics. Use the constant DBMS_STATS.AUTO_CASCADE to have Oracle determine whether index statistics to be collected or not. This is the default. The default value can be changed using the SET_PARAM Procedure.

Table 103–30 (Cont.) GATHER_SCHEMA_STATS Procedure Parameters

Parameter	Description
stattab	User statistics table identifier describing where to save the current statistics
statid	Identifier (optional) to associate with these statistics within stattab
options	Further specification of which objects to gather statistics for: GATHER: Gathers statistics on all objects in the schema. GATHER AUTO: Gathers all necessary statistics automatically. Oracle implicitly determines which objects need new statistics, and determines how to gather those statistics. When GATHER AUTO is specified, the only additional valid parameters are ownname, stattab, statid, objlist and statown; all other parameter settings are ignored. Returns a list of processed objects. GATHER STALE: Gathers statistics on stale objects as determined by looking at the *_tab_modifications views. Also, return a list of objects found to be stale. GATHER EMPTY: Gathers statistics on objects which currently have no statistics. also, return a list of objects found to have no statistics. LIST AUTO: Returns a list of objects to be processed with GATHER AUTO. LIST STALE: Returns list of stale objects as determined by looking at the *_tab_modifications views. LIST EMPTY: Returns list of objects which currently have no statistics.
objlist	List of objects found to be stale or empty
statown	Schema containing stattab (if different than ownname)
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_PARAM Procedure.
force	Gather statistics on objects even if they are locked

Usage Notes

DBMS_STATS.GATHER_SCHEMA_STATS generates differing sampling rates on partitioned tables when you use the auto_sample_size constant. DBMS_STATS tries to determine an adequate sample size for each type of statistic, which is different for each table or column (and each partition, if partitioned). It starts with a sampling rate to get approximately 5000 rows and examines the result based on statistical equations. This process is repeated with increased sampling rate for unsatisfactory results.

In general, the number of distinct values column statistics requires the highest sampling rate among the others, especially when each distinct value repeats a small number of times.

When you use a specific value for the sampling percentage, DBMS_STATS honors it except for when:

- The result is less than 2500 rows (too small a sample) and
- The specified percentage is more than the certain percentage.

Statistics for external tables are not collected by this procedure.

Exceptions

`ORA-20000`: Schema does not exist or insufficient privileges.

`ORA-20001`: Bad input value.

GATHER_SYSTEM_STATS Procedure

This procedure gathers system statistics.

Syntax

```
DBMS_STATS.GATHER_SYSTEM_STATS (
   gathering_mode   VARCHAR2 DEFAULT 'NOWORKLOAD',
   interval         INTEGER  DEFAULT NULL,
   stattab          VARCHAR2 DEFAULT NULL,
   statid           VARCHAR2 DEFAULT NULL,
   statown          VARCHAR2 DEFAULT NULL);
```

Parameters

Table 103–31 GATHER_SYSTEM_STATS Procedure Parameters

Parameter	Description
gathering_mode	Mode values are: NOWORKLOAD: Will capture characteristics of the I/O system. Gathering may take a few minutes and depends on the size of the database. During this period Oracle will estimate the average read seek time and transfer speed for the I/O system. This mode is suitable for the all workloads. Oracle recommends to run GATHER_SYSTEM_STATS ('noworkload') after creation of the database and tablespaces. To fine tune system statistics for the workload use 'START' and 'STOP' or 'INTERVAL' options. If you gather both 'NOWORKLOAD' and workload specific (statistics collected using 'INTERVAL' or 'START' and 'STOP'), the workload statistics will be used by optimizer. Collected components: cpuspeednw, ioseektim, iotfrspeed. INTERVAL: Captures system activity during a specified interval. This works in combination with the interval parameter. You should provide an interval value in minutes, after which system statistics are created or updated in the dictionary or stattab. You can use GATHER_SYSTEM_STATS (gathering_mode=>'STOP') to stop gathering earlier than scheduled. Collected components: maxthr, slavethr, cpuspeed, sreadtim, mreadtim, mbrc. START | STOP: Captures system activity during specified start and stop times and refreshes the dictionary or stattab with statistics for the elapsed period. Interval value is ignored. Collected components: maxthr, slavethr, cpuspeed, sreadtim, mreadtim, mbrc.
interval	Time, in minutes, to gather statistics. This parameter applies only when gathering_mode='INTERVAL'
stattab	Identifier of the user statistics table where the statistics will be saved
statid	Optional identifier associated with the statistics saved in the stattab
statown	Schema containing stattab (if different from current schema)

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20001: Invalid input value.

ORA-20002: Bad user statistics table; may need to be upgraded.

ORA-20003: Unable to gather system statistics.

ORA-20004: Error in the INTERVAL mode: system parameter job_queue_processes must be >0.

GATHER_TABLE_STATS Procedure

This procedure gathers table and column (and index) statistics. It attempts to parallelize as much of the work as possible, but there are some restrictions as described in the individual parameters.

Syntax

```
DBMS_STATS.GATHER_TABLE_STATS (
   ownname          VARCHAR2,
   tabname          VARCHAR2,
   partname         VARCHAR2 DEFAULT NULL,
   estimate_percent NUMBER   DEFAULT to_estimate_percent_type
                                        (get_param('ESTIMATE_PERCENT')),
   block_sample     BOOLEAN  DEFAULT FALSE,
   method_opt       VARCHAR2 DEFAULT get_param('METHOD_OPT'),
   degree           NUMBER   DEFAULT to_degree_type(get_param('DEGREE')),
   granularity      VARCHAR2 DEFAULT GET_PARAM('GRANULARITY'),
   cascade          BOOLEAN  DEFAULT to_cascade_type(get_param('CASCADE')),
   stattab          VARCHAR2 DEFAULT NULL,
   statid           VARCHAR2 DEFAULT NULL,
   statown          VARCHAR2 DEFAULT NULL,
   no_invalidate    BOOLEAN  DEFAULT  to_no_invalidate_type (
                                 get_param('NO_INVALIDATE')),
   force            BOOLEAN DEFAULT FALSE);
```

Parameters

Table 103–32 GATHER_TABLE_STATS Procedure Parameters

Parameter	Description
ownname	Schema of table to analyze
tabname	Name of table
partname	Name of partition
estimate_percent	Percentage of rows to estimate (NULL means compute) The valid range is [0.000001,100]. Use the constant DBMS_STATS.AUTO_SAMPLE_SIZE to have Oracle determine the appropriate sample size for good statistics. This is the default.The default value can be changed using the SET_PARAM Procedure.
block_sample	Whether or not to use random block sampling instead of random row sampling. Random block sampling is more efficient, but if the data is not randomly distributed on disk, then the sample values may be somewhat correlated. Only pertinent when doing an estimate statistics.

Table 103–32 (Cont.) GATHER_TABLE_STATS Procedure Parameters

Parameter	Description
method_opt	Accepts: ▪ FOR ALL [INDEXED | HIDDEN] COLUMNS [size_clause] ▪ FOR COLUMNS [size clause] column|attribute [size_clause] [,column|attribute [size_clause]...] size_clause is defined as size_clause := SIZE {integer | REPEAT | AUTO | SKEWONLY} – integer : Number of histogram buckets. Must be in the range [1,254]. – REPEAT : Collects histograms only on the columns that already have histograms. – AUTO : Oracle determines the columns to collect histograms based on data distribution and the workload of the columns. – SKEWONLY : Oracle determines the columns to collect histograms based on the data distribution of the columns. The default is FOR ALL COLUMNS SIZE AUTO.The default value can be changed using the SET_PARAM Procedure.
degree	Degree of parallelism. The default for degree is NULL. The default value can be changed using the SET_PARAM Procedure NULL means use the table default value specified by the DEGREE clause in the CREATE TABLE or ALTER TABLE statement. Use the constant DBMS_STATS.DEFAULT_DEGREE to specify the default value based on the initialization parameters. The AUTO_DEGREE value determines the degree of parallelism automatically. This is either 1 (serial execution) or DEFAULT_DEGREE (the system default value based on number of CPUs and initialization parameters) according to size of the object.
granularity	Granularity of statistics to collect (only pertinent if the table is partitioned). 'ALL' - gathers all (subpartition, partition, and global) statistics 'AUTO'- determines the granularity based on the partitioning type. This is the default value. 'DEFAULT' - gathers global and partition-level statistics. This option is obsolete, and while currently supported, it is included in the documentation for legacy reasons only. You should use the 'GLOBAL AND PARTITION' for this functionality. Note that the default value is now 'AUTO'. 'GLOBAL' - gathers global statistics 'GLOBAL AND PARTITION' - gathers the global and partition level statistics. No subpartition level statistics are gathered even if it is a composite partitioned object. 'PARTITION'- gathers partition-level statistics 'SUBPARTITION' - gathers subpartition-level statistics.
cascade	Gather statistics on the indexes for this table. Index statistics gathering is not parallelized. Using this option is equivalent to running the GATHER_INDEX_STATS Procedure on each of the table's indexes. Use the constant DBMS_STATS.AUTO_CASCADE to have Oracle determine whether index statistics to be collected or not. This is the default. The default value can be changed using theSET_PARAM Procedure .

Table 103–32 (Cont.) GATHER_TABLE_STATS Procedure Parameters

Parameter	Description
stattab	User statistics table identifier describing where to save the current statistics
statid	Identifier (optional) to associate with these statistics within stattab
statown	Schema containing stattab (if different than ownname)
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_PARAM Procedure.
force	Gather statistics of table even if it is locked

Usage Notes

This operation does not parallelize if the user does not have select privilege on the table being analyzed.

Exceptions

ORA-20000: Table does not exist or insufficient privileges.

ORA-20001: Bad input value.

GENERATE_STATS Procedure

This procedure generates object statistics from previously collected statistics of related objects. The currently supported objects are b-tree and bitmap indexes.

Syntax

```
DBMS_STATS.GENERATE_STATS (
   ownname   VARCHAR2,
   objname   VARCHAR2,
   organized NUMBER DEFAULT 7);
```

Parameters

Table 103–33 GENERATE_STATS Procedure Parameters

Parameter	Description
ownname	Schema of object
objname	Name of object
organized	Amount of ordering associated between the index and its underlying table. A heavily organized index would have consecutive index keys referring to consecutive rows on disk for the table (the same block). A heavily disorganized index would have consecutive keys referencing different table blocks on disk. This parameter is only used for b-tree indexes. The number can be in the range of 0-10, with 0 representing a completely organized index and 10 a completely disorganized one.

Usage Notes

For fully populated schemas, the gather procedures should be used instead when more accurate statistics are desired.

Exceptions

ORA-20000: Unsupported object type of object does not exist.

ORA-20001: Invalid option or invalid statistics.

GET_COLUMN_STATS Procedures

These procedures gets all column-related information. In the form of this procedure that deals with user-defined statistics, the statistics type returned is the type stored, in addition to the user-defined statistics.

Syntax

```
DBMS_STATS.GET_COLUMN_STATS (
   ownname     VARCHAR2,
   tabname     VARCHAR2,
   colname     VARCHAR2,
   partname    VARCHAR2 DEFAULT NULL,
   stattab     VARCHAR2 DEFAULT NULL,
   statid      VARCHAR2 DEFAULT NULL,
   distcnt OUT NUMBER,
   density OUT NUMBER,
   nullcnt OUT NUMBER,
   srec    OUT StatRec,
   avgclen OUT NUMBER,
   statown     VARCHAR2 DEFAULT NULL);
```

Use the following for user-defined statistics:

```
DBMS_STATS.GET_COLUMN_STATS (
   ownname          VARCHAR2,
   tabname          VARCHAR2,
   colname          VARCHAR2,
   partname         VARCHAR2 DEFAULT NULL,
   stattab          VARCHAR2 DEFAULT NULL,
   statid           VARCHAR2 DEFAULT NULL,
   ext_stats   OUT  RAW,
   stattypown  OUT  VARCHAR2 DEFAULT NULL,
   stattypname OUT  VARCHAR2 DEFAULT NULL,
   statown          VARCHAR2 DEFAULT NULL);
```

Parameters

Table 103–34 GET_COLUMN_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema
tabname	Name of the table to which this column belongs
colname	Name of the column
partname	Name of the table partition from which to get the statistics. If the table is partitioned and if partname is NULL, then the statistics are retrieved from the global table level.
stattab	User statistics table identifier describing from where to retrieve the statistics. If stattab is NULL, then the statistics are retrieved directly from the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL)
ext_stats	The user-defined statistics
stattypown	Schema of the statistics type

Table 103–34 (Cont.) GET_COLUMN_STATS Procedure Parameters

Parameter	Description
stattypname	Name of the statistics type
distcnt	Number of distinct values
density	Column density
nullcnt	Number of NULLs
srec	Structure holding internal representation of column minimum, maximum, and histogram values
avgclen	Average length of the column (in bytes)
statown	Schema containing stattab (if different than ownname)

Exceptions

ORA-20000: Object does not exist or insufficient privileges or no statistics have been stored for requested object.

GET_INDEX_STATS Procedures

These procedures get all index-related information. In the form of this procedure that deals with user-defined statistics, the statistics type returned is the type stored, in addition to the user-defined statistics.

Syntax

```
DBMS_STATS.GET_INDEX_STATS (
   ownname        VARCHAR2,
   indname        VARCHAR2,
   partname       VARCHAR2 DEFAULT NULL,
   stattab        VARCHAR2 DEFAULT NULL,
   statid         VARCHAR2 DEFAULT NULL,
   numrows    OUT NUMBER,
   numlblks   OUT NUMBER,
   numdist    OUT NUMBER,
   avglblk    OUT NUMBER,
   avgdblk    OUT NUMBER,
   clstfct    OUT NUMBER,
   indlevel   OUT NUMBER,
   statown        VARCHAR2 DEFAULT NULL,
   cachedblk  OUT NUMBER,
   cachehit   OUT NUMBER);

DBMS_STATS.GET_INDEX_STATS (
   ownname        VARCHAR2,
   indname        VARCHAR2,
   partname       VARCHAR2 DEFAULT NULL,
   stattab        VARCHAR2 DEFAULT NULL,
   statid         VARCHAR2 DEFAULT NULL,
   numrows    OUT NUMBER,
   numlblks   OUT NUMBER,
   numdist    OUT NUMBER,
   avglblk    OUT NUMBER,
   avgdblk    OUT NUMBER,
   clstfct    OUT NUMBER,
   indlevel   OUT NUMBER,
   statown        VARCHAR2 DEFAULT NULL,
   guessq     OUT NUMBER,
   cachedblk  OUT NUMBER,
   cachehit   OUT NUMBER);
```

Use the following for user-defined statistics:

```
DBMS_STATS.GET_INDEX_STATS (
   ownname          VARCHAR2,
   indname          VARCHAR2,
   partname         VARCHAR2 DEFAULT NULL,
   stattab          VARCHAR2 DEFAULT NULL,
   statid           VARCHAR2 DEFAULT NULL,
   ext_stats   OUT  RAW,
   stattypown  OUT  VARCHAR2 DEFAULT NULL,
   stattypname OUT  VARCHAR2 DEFAULT NULL,
   statown          VARCHAR2 DEFAULT NULL,
   cachedblk        OUT NUMBER,
   cachehit         OUT NUMBER);
```

Parameters

Table 103–35 GET_INDEX_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema
indname	Name of the index
partname	Name of the index partition for which to get the statistics. If the index is partitioned and if partname is NULL, then the statistics are retrieved for the global index level.
stattab	User statistics table identifier describing from where to retrieve the statistics. If stattab is NULL, then the statistics are retrieved directly from the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL)
ext_stats	The user-defined statistics
stattypown	Schema of the statistics type
stattypname	Name of the statistics type
numrows	Number of rows in the index (partition)
numlblks	Number of leaf blocks in the index (partition)
numdist	Number of distinct keys in the index (partition)
avglblk	Average integral number of leaf blocks in which each distinct key appears for this index (partition)
avgdblk	Average integral number of data blocks in the table pointed to by a distinct key for this index (partition)
clstfct	Clustering factor for the index (partition)
indlevel	Height of the index (partition)
statown	Schema containing stattab (if different than ownname)
guessq	Guess quality for the index (partition)
cachedblk	The average number of blocks in the buffer cache for the segment (index/table/index partition/table partition)
cachehit	The average cache hit ratio for the segment (index/table/index partition/table partition)

Usage Notes

- The Optimizer uses the cached data to estimate number of cached blocks for index or statistics table access. The total cost of the operation will be combined from the I/O cost of reading not cached blocks from disk, the CPU cost of getting cached blocks from the buffer cache, and the CPU cost of processing the data.
- Oracle maintains cachedblk and cachehit at all times but uses correspondent caching statistics for optimization as part of the table and index statistics only when the user calls DBMS_STATS.GATHER_[TABLE/INDEX/SCHEMA/DATABASE]_STATS procedure for auto mode or DBMS_STATS.GATHER_SYSTEM_STATS for manual mode. In order to prevent the user from utilizing inaccurate and unreliable data, the optimizer will compute a 'confidence factor' for each cachehit and a cachedblk for each object. If the 'confidence factor' for the value meets confidence criteria, this value will be used, otherwise the defaults will be used.

- The automatic maintenance algorithm for object caching statistics assumes that there is only one major workload for the system and adjusts statistics to this workload, ignoring other "minor" workloads. If this is not the case, you must use manual mode for maintaining object caching statistics.
- The object caching statistics maintenance algorithm for auto mode prevents you from using statistics in the following situations
  - When not enough data has been analyzed, such as when an object has been recently create
  - When the system does not have one major workload resulting in averages not corresponding to real values.

Exceptions

`ORA-20000`: Object does not exist or insufficient privileges or no statistics have been stored for requested object.

GET_PARAM Function

This function returns the default value of parameters of DBMS_STATS procedures.

Syntax

```
DBMS_STATS.GET_PARAM (
   pname      IN    VARCHAR2)
 RETURN VARCHAR2;
```

Parameters

Table 103–36 GET_PARAM Function Parameters

Parameter	Description
pname	The parameter name

Exceptions

ORA-20001: Invalid input values

GET_STATS_HISTORY_AVAILABILITY Function

This function returns oldest timestamp where statistics history is available.Users cannot restore statistics to a timestamp older than this one.

Syntax

```
DBMS_STATS.GET_STATS_HISTORY_AVAILABILITY
 RETURN TIMESTAMP WITH TIMEZONE;
```

GET_STATS_HISTORY_RETENTION Function

This function returns the current retention value.

Syntax

```
DBMS_STATS.GET_STATS_HISTORY_RETENTION
 RETURN NUMBER;
```

GET_SYSTEM_STATS Procedure

This procedure gets system statistics from `stattab`, or from the dictionary if `stattab` is `NULL`.

Syntax

```
DBMS_STATS.GET_SYSTEM_STATS (
   status    OUT  VARCHAR2,
   dstart    OUT  DATE,
   dstop     OUT  DATE,
   pname          VARCHAR2,
   pvalue    OUT  NUMBER,
   stattab   IN   VARCHAR2 DEFAULT NULL,
   statid    IN   VARCHAR2 DEFAULT NULL,
   statown   IN   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 103–37 GET_SYSTEM_STATS Procedure Parameters

Parameter	Description
`status`	Output is one of the following: ■ `COMPLETED:` ■ `AUTOGATHERING:` ■ `MANUALGATHERING:` ■ `BADSTATS:`
`dstart`	Date when statistics gathering started. If `status` = `MANUALGATHERING`, the start date is returned.
`dstop`	Date when statistics gathering stopped. ■ If `status` = `COMPLETE`, the finish date is returned. ■ If `status` = `AUTOGATHERING`, the future finish date is returned. ■ If `status` = `BADSTATS`, the must-finished-by date is returned.

Table 103–37 (Cont.) GET_SYSTEM_STATS Procedure Parameters

Parameter	Description
pname	The parameter name to get, which can have one of the following values: ■ iotfrspeed - I/O transfer speed in bytes for each millisecond ■ ioseektim - seek time + latency time + operating system overhead time, in milliseconds ■ sreadtim - average time to read single block (random read), in milliseconds ■ mreadtim - average time to read an mbrc block at once (sequential read), in milliseconds ■ cpuspeed - average number of CPU cycles for each second, in millions, captured for the workload (statistics collected using 'INTERVAL' or 'START' and 'STOP' options) ■ cpuspeednw - average number of CPU cycles for each second, in millions, captured for the noworkload (statistics collected using 'NOWORKLOAD' option. ■ mbrc - average multiblock read count for sequential read, in blocks ■ maxthr - maximum I/O system throughput, in bytes/second ■ slavethr - average slave I/O throughput, in bytes/second
pvalue	The parameter value to get
stattab	Identifier of the user statistics table where the statistics will be obtained. If stattab is null, the statistics will be obtained from the dictionary.
statid	Optional identifier associated with the statistics saved in the stattab
statown	Schema containing stattab (if different from current schema)

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20002: Bad user statistics table; may need to be upgraded.

ORA-20003: Unable to gather system statistics.

ORA-20004: Parameter does not exist.

GET_TABLE_STATS Procedure

This procedure gets all table-related information.

Syntax

```
DBMS_STATS.GET_TABLE_STATS (
   ownname         VARCHAR2,
   tabname         VARCHAR2,
   partname        VARCHAR2 DEFAULT NULL,
   stattab         VARCHAR2 DEFAULT NULL,
   statid          VARCHAR2 DEFAULT NULL,
   numrows     OUT NUMBER,
   numblks     OUT NUMBER,
   avgrlen     OUT NUMBER,
   statown         VARCHAR2 DEFAULT NULL,
   cachedblk   OUT NUMBER,
   cachehit    OUT NUMBER);
```

Parameters

Table 103–38 GET_TABLE_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
tabname	Name of the table to which this column belongs.
partname	Name of the table partition from which to get the statistics. If the table is partitioned and if partname is NULL, then the statistics are retrieved from the global table level.
stattab	User statistics table identifier describing from where to retrieve the statistics. If stattab is NULL, then the statistics are retrieved directly from the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).
numrows	Number of rows in the table (partition).
numblks	Number of blocks the table (partition) occupies.
avgrlen	Average row length for the table (partition).
statown	Schema containing stattab (if different than ownname).
cachedblk	
cachehit	

Usage Notes

- The Optimizer uses the cached data to estimate number of cached blocks for index or statistics table access. The total cost of the operation will be combined from the I/O cost of reading not cached blocks from disk, the CPU cost of getting cached blocks from the buffer cache, and the CPU cost of processing the data.
- Oracle maintains cachedblk and cachehit at all times but uses correspondent caching statistics for optimization as part of the table and index statistics only when the user calls DBMS_STATS.GATHER_ [TABLE/INDEX/SCHEMA/DATABASE]_STATS procedure for auto mode or DBMS_STATS.GATHER_SYSTEM_STATS for manual mode. In order to prevent the

user from utilizing inaccurate and unreliable data, the optimizer will compute a 'confidence factor' for each `cachehit` and a `cachedblk` for each object. If the 'confidence factor' for the value meets confidence criteria, this value will be used, otherwise the defaults will be used.

- The automatic maintenance algorithm for object caching statistics assumes that there is only one major workload for the system and adjusts statistics to this workload, ignoring other "minor" workloads. If this is not the case, you must use manual mode for maintaining object caching statistics.
- The object caching statistics maintenance algorithm for auto mode prevents you from using statistics in the following situations
  - When not enough data has been analyzed, such as when an object has been recently create
  - When the system does not have one major workload resulting in averages not corresponding to real values.

Exceptions

`ORA-20000`: Object does not exist or insufficient privileges or no statistics have been stored for requested object

IMPORT_COLUMN_STATS Procedure

This procedure retrieves statistics for a particular column from the user statistics table identified by `stattab` and stores them in the dictionary.

Syntax

```
DBMS_STATS.IMPORT_COLUMN_STATS (
   ownname       VARCHAR2,
   tabname       VARCHAR2,
   colname       VARCHAR2,
   partname      VARCHAR2 DEFAULT NULL,
   stattab       VARCHAR2,
   statid        VARCHAR2 DEFAULT NULL,
   statown       VARCHAR2 DEFAULT NULL,
   no_invalidate BOOLEAN DEFAULT to_no_invalidate_type (
                                    get_param('NO_INVALIDATE')),
   force         BOOLEAN DEFAULT FALSE);
```

Parameters

Table 103–39 IMPORT_COLUMN_STATS Procedure Parameters

Parameter	Description
`ownname`	The name of the schema
`tabname`	The name of the table to which this column belongs
`colname`	The name of the column
`partname`	The name of the table partition. If the table is partitioned and if `partname` is `NULL`, then global and partition column statistics are imported.
`stattab`	The user statistics table identifier describing from where to retrieve the statistics
`statid`	The (optional) identifier to associate with these statistics within `stattab`
`statown`	The schema containing `stattab` (if different than `ownname`)
`no_invalidate`	Does not invalidate the dependent cursors if set to `TRUE`. The procedure invalidates the dependent cursors immediately if set to `FALSE`. Use `DBMS_STATS.AUTO_INVALIDATE`. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_PARAM Procedure.
`force`	If set to `TRUE`, imports statistics even if statistics are locked

Exceptions

`ORA-20000`: Object does not exist or insufficient privileges.

`ORA-20001`: Invalid or inconsistent values in the user statistics table.

`ORA-20005`: Object statistics are locked.

IMPORT_DATABASE_STATS Procedure

This procedure retrieves statistics for all objects in the database from the user statistics table(s) and stores them in the dictionary.

Syntax

```
DBMS_STATS.IMPORT_DATABASE_STATS (
   stattab       VARCHAR2,
   statid        VARCHAR2 DEFAULT NULL,
   statown       VARCHAR2 DEFAULT NULL,
   no_invalidate BOOLEAN DEFAULT to_no_invalidate_type(
                                  get_param('NO_INVALIDATE')),
   force         BOOLEAN DEFAULT FALSE);
```

Parameters

Table 103–40 IMPORT_DATABASE_STATS Procedure Parameters

Parameter	Description
stattab	User statistics table identifier describing from where to retrieve the statistics
statid	Identifier (optional) to associate with these statistics within stattab
statown	Schema containing stattab (if different from current schema)
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_PARAM Procedure.
force	Overrides statistics locked at the object (table) level: ■ TRUE - Ignores the statistics lock and imports the statistics. ■ FALSE - The statistics will be imported only if they are not locked.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20001: Invalid or inconsistent values in the user statistics table.

IMPORT_DICTIONARY_STATS Procedure

This procedure retrieves statistics for all dictionary schemas ('SYS', 'SYSTEM' and RDBMS component schemas) from the user statistics table and stores them in the dictionary.

Syntax

```
DBMS_STATS.IMPORT_DICTIONARY_STATS (
   stattab       VARCHAR2,
   statid        VARCHAR2 DEFAULT NULL,
   statown       VARCHAR2 DEFAULT NULL,
   no_invalidate BOOLEAN DEFAULT to_no_invalidate_type(
                                  get_param('NO_INVALIDATE')),
   force         BOOLEAN DEFAULT FALSE);
```

Parameters

Table 103–41 IMPORT_DICTIONARY_STATS Procedure Parameters

Parameter	Description
stattab	User statistics table identifier describing from where to retrieve the statistics
statid	The (optional) identifier to associate with these statistics within stattab
statown	Schema containing stattab (if different from current schema)
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_PARAM Procedure.
force	Overrides statistics lock at the object (table) level: ▪ TRUE - Ignores the statistics lock and imports the statistics. ▪ FALSE - The statistics will be imported only if there is no lock.

Usage Notes

You must have the SYSDBA or both ANALYZE ANY DICTIONARY and ANALYZE ANY system privilege to execute this procedure.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20001: Invalid or inconsistent values in the user statistics table.

ORA-20002: Bad user statistics table, may need to upgrade it.

IMPORT_FIXED_OBJECTS_STATS Procedure

This procedure retrieves statistics for fixed tables from the user statistics table(s) and stores them in the dictionary.

Syntax

```
DBMS_STATS.IMPORT_FIXED_OBJECTS_STATS (
   stattab       VARCHAR2,
   statid        VARCHAR2 DEFAULT NULL,
   statown       VARCHAR2 DEFAULT NULL,
   no_invalidate BOOLEAN DEFAULT to_no_invalidate_type(
                                  get_param('NO_INVALIDATE')),
   force         BOOLEAN DEFAULT FALSE);
```

Parameters

Table 103–42 IMPORT_FIXED_OBJECTS_STATS Procedure Parameters

Parameter	Description
stattab	User statistics table identifier describing from where to retrieve the statistics
statid	Identifier (optional) to associate with these statistics within stattab
statown	Schema containing stattab (if different from current schema)
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_PARAM Procedure.
force	Overrides statistics lock: ■ TRUE - Ignores the statistics lock and imports the statistics ■ FALSE - The statistics will be imported only if there is no lock

Usage Notes

You must have the SYSDBA or ANALYZE ANY DICTIONARY system privilege to execute this procedure.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20001: Invalid or inconsistent values in the user statistics table.

ORA-20002: Bad user statistics table, may need to upgrade it.

IMPORT_INDEX_STATS Procedure

http://usunnab06.us.oracle.com:80/servers/MifChecker/Out/Y10312_01.htm retrieves statistics for a particular index from the user statistics table identified by `stattab` and stores them in the dictionary.

Syntax

```
DBMS_STATS.IMPORT_INDEX_STATS (
   ownname       VARCHAR2,
   indname       VARCHAR2,
   partname      VARCHAR2 DEFAULT NULL,
   stattab       VARCHAR2,
   statid        VARCHAR2 DEFAULT NULL,
   statown       VARCHAR2 DEFAULT NULL,
   no_invalidate BOOLEAN DEFAULT to_no_invalidate_type(
                                  get_param('NO_INVALIDATE')),
   force         BOOLEAN DEFAULT FALSE);
```

Parameters

Table 103–43 IMPORT_INDEX_STATS Procedure Parameters

Parameter	Description
`ownname`	Name of the schema
`indname`	Name of the index
`partname`	Name of the index partition. If the index is partitioned and if `partname` is `NULL`, then global and partition index statistics are imported.
`stattab`	User statistics table identifier describing from where to retrieve the statistics
`statid`	Identifier (optional) to associate with these statistics within `stattab`.
`statown`	Schema containing `stattab` (if different than `ownname`)
`no_invalidate`	Does not invalidate the dependent cursors if set to `TRUE`. The procedure invalidates the dependent cursors immediately if set to `FALSE`. Use `DBMS_STATS.AUTO_INVALIDATE`. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_PARAM Procedure.
`force`	Imports statistics even if index statistics are locked

Exceptions

`ORA-20000`: Object does not exist or insufficient privileges.

`ORA-20001`: Invalid or inconsistent values in the user statistics table.

`ORA-20005`: Object statistics are locked.

IMPORT_SCHEMA_STATS Procedure

http://usunnab06.us.oracle.com:80/servers/MifChecker/Out/Y10312_01.htm retrieves statistics for all objects in the schema identified by ownname from the user statistics table and stores them in the dictionary.

Syntax

```
DBMS_STATS.IMPORT_SCHEMA_STATS (
   ownname       VARCHAR2,
   stattab       VARCHAR2,
   statid        VARCHAR2 DEFAULT NULL,
   statown       VARCHAR2 DEFAULT NULL,
   no_invalidate BOOLEAN DEFAULTto_no_invalidate_type(
                                  get_param('NO_INVALIDATE')),
   force         BOOLEAN DEFAULT FALSE);
```

Parameters

Table 103–44 IMPORT_SCHEMA_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema
stattab	User statistics table identifier describing from where to retrieve the statistics
statid	Identifier (optional) to associate with these statistics within stattab
statown	Schema containing stattab (if different than ownname)
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_PARAM Procedure.
force	Overrides statistics locked at the object (table) level: ■ TRUE - Ignores the statistics lock and imports the statistics. ■ FALSE - The statistics will be imported only if there is no lock.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20001: Invalid or inconsistent values in the user statistics table.

IMPORT_SYSTEM_STATS Procedure

http://usunnab06.us.oracle.com:80/servers/MifChecker/Out/Y10312_01.htm retrieves system statistics from the user statistics table, identified by stattab, and stores the statistics in the dictionary.

Syntax

```
DBMS_STATS.IMPORT_SYSTEM_STATS (
   stattab       VARCHAR2,
   statid        VARCHAR2 DEFAULT NULL,
   statown       VARCHAR2 DEFAULT NULL);
```

Parameters

Table 103–45 IMPORT_SYSTEM_STATS Procedure Parameters

Parameter	Description
`stattab`	Identifier of the user statistics table where the statistics will be retrieved
`statid`	Optional identifier associated with the statistics retrieved from the stattab
`statown`	Schema containing `stattab` (if different from current schema)

Exceptions

`ORA-20000`: Object does not exist or insufficient privileges.

`ORA-20001`: Invalid or inconsistent values in the user statistics table.

`ORA-20002`: Bad user statistics table; may need to be upgraded.

`ORA-20003`: Unable to import system statistics.

IMPORT_TABLE_STATS Procedure

http://usunnab06.us.oracle.com:80/servers/MifChecker/Out/Y10312_01.htm retrieves statistics for a particular table from the user statistics table identified by `stattab` and stores them in the dictionary. Cascade results in all index and column statistics associated with the specified table being imported as well.

Syntax

```
DBMS_STATS.IMPORT_TABLE_STATS (
   ownname       VARCHAR2,
   tabname       VARCHAR2,
   partname      VARCHAR2 DEFAULT NULL,
   stattab       VARCHAR2,
   statid        VARCHAR2 DEFAULT NULL,
   cascade       BOOLEAN  DEFAULT TRUE,
   statown       VARCHAR2 DEFAULT NULL,
   no_invalidate BOOLEAN DEFAULT to_no_invalidate_type(
                                  get_param('NO_INVALIDATE')),
   force         BOOLEAN DEFAULT FALSE);
```

Parameters

Table 103–46 IMPORT_TABLE_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema
tabname	Name of the table
partname	Name of the table partition. If the table is partitioned and if partname is NULL, then global and partition table statistics are imported.
stattab	User statistics table identifier describing from where to retrieve the statistics
statid	Identifier (optional) to associate with these statistics within stattab
cascade	If true, then column and index statistics for this table are also imported
statown	Schema containing stattab (if different than ownname)
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_PARAM Procedure.
force	Imports statistics even if table statistics are locked

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20001: Invalid or inconsistent values in the user statistics table.

LOCK_SCHEMA_STATS Procedure

This procedure locks the statistics of all tables of a schema.

Syntax

```
DBMS_STATS.LOCK_SCHEMA_STATS (
   ownname     VARCHAR2);
```

Parameters

Table 103–47 LOCK_SCHEMA_STATS Procedure Parameters

Parameter	Description
ownname	The name of the schema to lock

Usage Notes

See "Usage Notes" for LOCK_TABLE_STATS Procedure.

LOCK_TABLE_STATS Procedure

This procedure locks the statistics on the table.

Syntax

```
DBMS_STATS.LOCK_TABLE_STATS (
   ownname    VARCHAR2,
   tabname    VARCHAR2);
```

Parameters

Table 103–48 LOCK_TABLE_STATS Procedure Parameters

Parameter	Description
ownname	The name of the schema
tabname	The name of the table

Usage Notes

- When statistics on a table are locked, all the statistics depending on the table, including table statistics, column statistics, histograms and statistics on all dependent indexes, are considered to be locked.
- The SET_*, DELETE_*, IMPORT_*, GATHER_* procedures that modify statistics in the dictionary of an individual table, index or column will raise an error if statistics of the object is locked.
- Procedures that operates on multiple objects (such as GATHER_SCHEMA_STATS) will skip modifying the statistics of an object if it is locked. Many procedures have force argument to override the lock.
- This procedure either freezes the current set of the statistics or keeps the statistics empty (uncollected) to use Dynamic Sampling.
- The locked or unlocked state is not exported along with the table statistics when using EXPORT_*_STATS procedures.

PREPARE_COLUMN_VALUES Procedures

These procedures convert user-specified minimum, maximum, and histogram endpoint datatype-specific values into Oracle's internal representation for future storage using SET_COLUMN_STATS.

Syntax

```
DBMS_STATS.PREPARE_COLUMN_VALUES (
   srec     IN OUT StatRec,
   charvals        CHARARRAY);

DBMS_STATS.PREPARE_COLUMN_VALUES (
   srec      IN OUT StatRec,
   datevals         DATEARRAY);

DBMS_STATS.PREPARE_COLUMN_VALUES (
   srec     IN OUT StatRec,
   dblvals         DBLARRAY);

DBMS_STATS.PREPARE_COLUMN_VALUES (
   srec     IN OUT StatRec,
   fltvals         FLTARRAY);

DBMS_STATS.PREPARE_COLUMN_VALUES (
   srec     IN OUT StatRec,
   numvals         NUMARRAY);

DBMS_STATS.PREPARE_COLUMN_VALUES (
   srec     IN OUT StatRec,
   rawvals         RAWARRAY);
```

Pragmas

```
pragma restrict_references(prepare_column_values, WNDS, RNDS, WNPS, RNPS);
pragma restrict_references(prepare_column_values_nvarchar, WNDS, RNDS, WNPS,
RNPS);
pragma restrict_references(prepare_column_values_rowid, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 103–49 PREPARE_COLUMN_VALUES Procedure Parameters

Parameter	Description
srec.epc	Number of values specified in charvals, datevals, dblvals, fltvals, numvals, or rawvals. This value must be between 2 and 256, inclusive, and it should be set to 2 for procedures which do not allow histogram information (nvarchar and rowid). The first corresponding array entry should hold the minimum value for the column, and the last entry should hold the maximum. If there are more than two entries, then all the others hold the remaining height-balanced or frequency histogram endpoint values (with in-between values ordered from next-smallest to next-largest). This value may be adjusted to account for compression, so the returned value should be left as is for a call to SET_COLUMN_STATS.

Table 103–49 (Cont.) PREPARE_COLUMN_VALUES Procedure Parameters

Parameter	Description
srec.bkvals	If you want a frequency distribution, then this array contains the number of occurrences of each distinct value specified in charvals, datevals, dblvals, fltvals, numvals, or rawvals. Otherwise, it is merely an output parameter, and it must be set to NULL when this procedure is called.

Datatype-specific input parameters (use one) are shown in Table 103–50.

Table 103–50 Datatype-Specific Input Parameters

Type	Description
charvals	The array of values when the column type is character-based. Up to the first 32 bytes of each string should be provided. Arrays must have between 2 and 256 entries, inclusive. If the datatype is fixed CHAR, the strings must be space-padded to 15 characters for correct normalization.
datevals	The array of values when the column type is date-based
dblvals	The array of values when the column type is double-based
fltvals	The array of values when the column type is float-based
numvals	The array of values when the column type is numeric-based
rawvals	The array of values when the column type is RAW. Up to the first 32 bytes of each strings should be provided.
nvmin, nvmax	The minimum and maximum values when the column type is national character set based. No histogram information can be provided for a column of this type. If the datatype is fixed CHAR, the strings must be space-padded to 15 characters for correct normalization.
rwmin, rwmax	The minimum and maximum values when the column type is rowid. No histogram information is provided for a column of this type.

Output Parameters

Table 103–51 PREPARE_COLUMN_VALUES Procedure Output Parameters

Parameter	Description
srec.minval	Internal representation of the minimum suitable for use in a call to SET_COLUMN_STATS
srec.maxval	Internal representation of the maximum suitable for use in a call to SET_COLUMN_STATS
srec.bkvals	Array suitable for use in a call to SET_COLUMN_STATS
srec.novals	Array suitable for use in a call to SET_COLUMN_STATS

Exceptions

ORA-20001: Invalid or inconsistent input values.

PREPARE_COLUMN_VALUES_NVARCHAR2 Procedure

This procedure converts user-specified minimum, maximum, and histogram endpoint datatype-specific values into Oracle's internal representation for future storage using SET_COLUMN_STATS.

Syntax

```
DBMS_STATS.PREPARE_COLUMN_VALUES_NVARCHAR2 (
   srec     IN OUT StatRec,
   nvmin           NVARCHAR2,
   nvmax           NVARCHAR2);
```

Pragmas

```
pragma restrict_references(prepare_column_values_nvarchar, WNDS, RNDS, WNPS,
RNPS);
```

Parameters

Table 103–52 PREPARE_COLUMN_VALUES_NVARCHAR2 Procedure Parameters

Parameter	Description
srec.epc	Number of values specified in charvals, datevals, dblvals, fltvals, numvals, or rawvals. This value must be between 2 and 256, inclusive, and it should be set to 2 for procedures which do not allow histogram information (nvarchar and rowid). The first corresponding array entry should hold the minimum value for the column, and the last entry should hold the maximum. If there are more than two entries, then all the others hold the remaining height-balanced or frequency histogram endpoint values (with in-between values ordered from next-smallest to next-largest). This value may be adjusted to account for compression, so the returned value should be left as is for a call to SET_COLUMN_STATS.
srec.bkvals	If you want a frequency distribution, then this array contains the number of occurrences of each distinct value specified in charvals, datevals, dblvals, fltvals, numvals, or rawvals. Otherwise, it is merely an output parameter, and it must be set to NULL when this procedure is called.

Datatype-specific input parameters (use one) are shown in Table 103–50.

Table 103–53 Datatype-Specific Input Parameters

Type	Description
nvmin, nvmax	The minimum and maximum values when the column type is national character set based. No histogram information can be provided for a column of this type. If the datatype is fixed CHAR, the strings must be space-padded to 15 characters for correct normalization.

Output Parameters

Table 103–54 PREPARE_COLUMN_VALUES Procedure Output Parameters

Parameter	Description
srec.minval	Internal representation of the minimum suitable for use in a call to SET_COLUMN_STATS
srec.maxval	Internal representation of the maximum suitable for use in a call to SET_COLUMN_STATS
srec.bkvals	Array suitable for use in a call to SET_COLUMN_STATS.
srec.novals	Array suitable for use in a call to SET_COLUMN_STATS

Exceptions

ORA-20001: Invalid or inconsistent input values.

PREPARE_COLUMN_VALUES_ROWID Procedure

This procedure converts user-specified minimum, maximum, and histogram endpoint datatype-specific values into Oracle's internal representation for future storage using SET_COLUMN_STATS.

Syntax

```
DBMS_STATS.PREPARE_COLUMN_VALUES_ROWID (
   srec  IN OUT StatRec,
   rwmin        ROWID,
   rwmax        ROWID);
```

Pragmas

```
pragma restrict_references(prepare_column_values_rowid, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 103–55 PREPARE_COLUMN_VALUES_ROWID Procedure Parameters

Parameter	Description
srec.epc	Number of values specified in charvals, datevals, dblvals, fltvals, numvals, or rawvals. This value must be between 2 and 256, inclusive, and it should be set to 2 for procedures which do not allow histogram information (nvarchar and rowid). The first corresponding array entry should hold the minimum value for the column, and the last entry should hold the maximum. If there are more than two entries, then all the others hold the remaining height-balanced or frequency histogram endpoint values (with in-between values ordered from next-smallest to next-largest). This value may be adjusted to account for compression, so the returned value should be left as is for a call to SET_COLUMN_STATS.
srec.bkvals	If you want a frequency distribution, then this array contains the number of occurrences of each distinct value specified in charvals, datevals, dblvals, fltvals, numvals, or rawvals. Otherwise, it is merely an output parameter, and it must be set to NULL when this procedure is called.

Datatype-specific input parameters (use one) are shown in Table 103–50.

Table 103–56 Datatype-Specific Input Parameters

Type	Description
rwmin, rwmax	The minimum and maximum values when the column type is rowid. No histogram information is provided for a column of this type.

Output Parameters

Table 103–57 PREPARE_COLUMN_VALUES Procedure Output Parameters

Parameter	Description
srec.minval	Internal representation of the minimum suitable for use in a call to SET_COLUMN_STATS.

Table 103–57 (Cont.) PREPARE_COLUMN_VALUES Procedure Output Parameters

Parameter	Description
srec.maxval	Internal representation of the maximum suitable for use in a call to SET_COLUMN_STATS.
srec.bkvals	Array suitable for use in a call to SET_COLUMN_STATS.
srec.novals	Array suitable for use in a call to SET_COLUMN_STATS.

Exceptions

ORA-20001: Invalid or inconsistent input values.

PURGE_STATS Procedure

This procedure purges old versions of statistics saved in the dictionary. To run this procedure, you must have the SYSDBA or both ANALYZE ANY DICTIONARY and ANALYZE ANY system privilege.

Syntax

```
DBMS_STATS.PURGE_STATS(
    before_timestamp       TIMESTAMP WITH TIME ZONE);
```

Parameters

Table 103–58 PURGE_STATS Procedure Parameters

Parameter	Description
before_timestamp	Versions of statistics saved before this timestamp are purged. If NULL, it uses the purging policy used by automatic purge. The automatic purge deletes all history older than the older of (current time - statistics history retention) and (time of recent analyze in the system - 1). The statistics history retention value can be changed using ALTER_STATS_HISTORY_RETENTION Procedure.The default is 31 days.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20001: Invalid or inconsistent values.

RESET_PARAM_DEFAULTS Procedure

This procedure resets the default values of all parameters to Oracle recommended values.

Syntax

```
DBMS_STATS.RESET_PARAM_DEFAULTS;
```

RESTORE_DATABASE_STATS Procedure

This procedure restores statistics of all tables of the database as of a specified timestamp (`as_of_timestamp`).

Syntax

```
DBMS_STATS.RESTORE_DATABSE_STATS(
   as_of_timestamp        TIMESTAMP WITH TIME ZONE,
   force                  BOOLEAN DEFAULT FALSE,
   no_invalidate          BOOLEAN DEFAULT to_no_invalidate_type
                                              (GET_PARAM('NO_INVALIDATE')));
```

Parameters

Table 103–59 RESTORE_DATABASE_STATS Procedure Parameters

Parameter	Description
`as_of_timestamp`	The timestamp to which to restore statistics
`force`	Restores statistics even if their statistics are locked
`no_invalidate`	Does not invalidate the dependent cursors if set to `TRUE`. The procedure invalidates the dependent cursors immediately if set to `FALSE`. Use `DBMS_STATS.AUTO_INVALIDATE`. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_PARAM Procedure.

Exceptions

`ORA-20000`: Object does not exist or insufficient privileges.

`ORA-20001`: Invalid or inconsistent values.

`ORA-20006`: Unable to restore statistics, statistics history not available.

RESTORE_DICTIONARY_STATS Procedure

This procedure restores statistics of all dictionary tables (tables of 'SYS', 'SYSTEM' and RDBMS component schemas) as of a specified timestamp (as_of_timestamp).

Syntax

```
DBMS_STATS.RESTORE_DICTIONARY_STATS(
   as_of_timestamp        TIMESTAMP WITH TIME ZONE,
   force                  BOOLEAN DEFAULT FALSE,
   no_invalidate          BOOLEAN DEFAULT to_no_invalidate_type
                                              (GET_PARAM('NO_INVALIDATE')));
```

Parameters

Table 103–60 RESTORE_DICTIONARY_STATS Procedure Parameters

Parameter	Description
as_of_timestamp	The timestamp to which to restore statistics
force	Restores statistics even if their statistics are locked
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_PARAM Procedure.

Usage Notes

To run this procedure, you must have the SYSDBA or both ANALYZE ANY DICTIONARY and ANALYZE ANY system privilege.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20001: Invalid or inconsistent values.

ORA-20006: Unable to restore statistics, statistics history not available.

RESTORE_FIXED_OBJECTS_STATS Procedure

This procedure restores statistics of all fixed tables as of a specified timestamp (`as_of_timestamp`).

Syntax

```
DBMS_STATS.RESTORE_FIXED_OBJECTS_STATS(
   as_of_timestamp        TIMESTAMP WITH TIME ZONE,
   force                  BOOLEAN DEFAULT FALSE,
   no_invalidate          BOOLEAN DEFAULT to_no_invalidate_type
                                              (GET_PARAM('NO_INVALIDATE')));
```

Parameters

Table 103–61 RESTORE_FIXED_OBJECTS_STATS Procedure Parameters

Parameter	Description
`as_of_timestamp`	The timestamp to which to restore statistics
`force`	Restores statistics even if their statistics are locked
`no_invalidate`	Does not invalidate the dependent cursors if set to `TRUE`. The procedure invalidates the dependent cursors immediately if set to `FALSE`. Use `DBMS_STATS.AUTO_INVALIDATE`. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_PARAM Procedure.

Usage Notes

To run this procedure, you must have the `SYSDBA` or `ANALYZE ANY DICTIONARY` system privilege.

Exceptions

`ORA-20000`: Object does not exist or insufficient privileges.

`ORA-20001`: Invalid or inconsistent values.

`ORA-20006`: Unable to restore statistics, statistics history not available.

RESTORE_SCHEMA_STATS Procedure

This procedure restores statistics of all tables of a schema as of a specified timestamp (as_of_timestamp).

Syntax

```
DBMS_STATS.RESTORE_SCHEMA_STATS(
   ownname                VARCHAR2,
   as_of_timestamp        TIMESTAMP WITH TIME ZONE,
   force                  BOOLEAN DEFAULT FALSE,
   no_invalidate          BOOLEAN DEFAULT to_no_invalidate_type
                                              (GET_PARAM('NO_INVALIDATE')));
```

Parameters

Table 103–62 RESTORE_SCHEMA_STATS Procedure Parameters

Parameter	Description
ownname	The schema of the tables for which the statistics are to be restored
as_of_timestamp	The timestamp to which to restore statistics
force	Restores statistics even if their statistics are locked
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_PARAM Procedure.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20001: Invalid or inconsistent values.

ORA-20006: Unable to restore statistics, statistics history not available.

RESTORE_SYSTEM_STATS Procedure

This procedure restores system statistics as of a specified timestamp (`as_of_timestamp`).

Syntax

```
DBMS_STATS.RESTORE_SCHEMA_STATS(
   as_of_timestamp        TIMESTAMP WITH TIME ZONE);
```

Parameters

Table 103–63 RESTORE_SYSTEM_STATS Procedure Parameters

Parameter	Description
`as_of_timestamp`	The timestamp to which to restore statistics

Exceptions

`ORA-20000`: Object does not exist or insufficient privileges.

`ORA-20001`: Invalid or inconsistent values.

`ORA-20006`: Unable to restore statistics, statistics history not available.

RESTORE_TABLE_STATS Procedure

This procedure restores statistics of a table as of a specified timestamp (as_of_timestamp). The procedure will restore statistics of associated indexes and columns as well. If the table statistics were locked at the specified timestamp the procedure will lock the statistics. The procedure will not restore user defined statistics.

Syntax

```
DBMS_STATS.RESTORE_TABLE_STATS (
   ownname                  VARCHAR2,
   tabname                  VARCHAR2,
   as_of_timestamp          TIMESTAMP WITH TIME ZONE,
   restore_cluster_index    BOOLEAN DEFAULT FALSE,
   force                    BOOLEAN DEFAULT FALSE,
   no_invalidate            BOOLEAN DEFAULT to_no_invalidate_type
                                          (GET_PARAM('NO_INVALIDATE')));
```

Parameters

Table 103–64 RESTORE_TABLE_STATS Procedure Parameters

Parameter	Description
ownname	The schema of the table for which the statistics are to be restored
tabname	The table name
as_of_timestamp	The timestamp to which to restore statistics
restore_cluster_index	If the table is part of a cluster, restore statistics of the cluster index if set to TRUE
force	Restores statistics even if the table statistics are locked. If the table statistics were not locked at the specified timestamp, it unlocks the statistics.
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_PARAM Procedure.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20001: Invalid or inconsistent values.

ORA-20006: Unable to restore statistics, statistics history not available.

SET_COLUMN_STATS Procedures

This procedure sets column-related information. In the version of this procedure that deals with user-defined statistics, the statistics type specified is the type to store in the dictionary, in addition to the actual user-defined statistics. If this statistics type is NULL, the statistics type associated with the index or column is stored.

Syntax

```
DBMS_STATS.SET_COLUMN_STATS (
   ownname       VARCHAR2,
   tabname       VARCHAR2,
   colname       VARCHAR2,
   partname      VARCHAR2 DEFAULT NULL,
   stattab       VARCHAR2 DEFAULT NULL,
   statid        VARCHAR2 DEFAULT NULL,
   distcnt       NUMBER DEFAULT NULL,
   density       NUMBER DEFAULT NULL,
   nullcnt       NUMBER DEFAULT NULL,
   srec          StatRec DEFAULT NULL,
   avgclen       NUMBER DEFAULT NULL,
   flags         NUMBER DEFAULT NULL,
   statown       VARCHAR2 DEFAULT NULL,
   no_invalidate BOOLEAN DEFAULT to_no_invalidate_type(
                                  get_param('NO_INVALIDATE')),
   force         BOOLEAN DEFAULT FALSE);
```

Use the following for user-defined statistics:

```
DBMS_STATS.SET_COLUMN_STATS (
   ownname       VARCHAR2,
   tabname       VARCHAR2,
   colname       VARCHAR2,
   partname      VARCHAR2 DEFAULT NULL,
   stattab       VARCHAR2 DEFAULT NULL,
   statid        VARCHAR2 DEFAULT NULL,
   ext_stats     RAW,
   stattypown    VARCHAR2 DEFAULT NULL,
   stattypname   VARCHAR2 DEFAULT NULL,
   statown       VARCHAR2 DEFAULT NULL,
   no_invalidate BOOLEAN DEFAULT to_no_invalidate_type(
                                  get_param('NO_INVALIDATE')),
   force         BOOLEAN DEFAULT FALSE);
```

Parameters

Table 103–65 SET_COLUMN_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
tabname	Name of the table to which this column belongs.
colname	Name of the column.
partname	Name of the table partition in which to store the statistics. If the table is partitioned and partname is NULL, then the statistics are stored at the global table level.

Table 103–65 (Cont.) SET_COLUMN_STATS Procedure Parameters

Parameter	Description
stattab	User statistics table identifier describing where to store the statistics. If stattab is NULL, then the statistics are stored directly in the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL)
ext_stats	The user-defined statistics
stattypown	Schema of the statistics type
stattypname	Name of the statistics type
distcnt	Number of distinct values
density	Column density. If this value is NULL and if distcnt is not NULL, then density is derived from distcnt.
nullcnt	Number of NULLs
srec	StatRec structure filled in by a call to PREPARE_COLUMN_VALUES or GET_COLUMN_STATS
avgclen	Average length for the column (in bytes)
flags	For internal Oracle use (should be left as NULL)
statown	Schema containing stattab (if different than ownname)
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_PARAM Procedure.
force	Sets the values even if statistics of the column are locked

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20001: Invalid or inconsistent input values.

ORA-20005: Object statistics are locked.

SET_INDEX_STATS Procedures

These procedures set index-related information. In the version of this procedure that deals with user-defined statistics, the statistics type specified is the type to store in the dictionary, in addition to the actual user-defined statistics. If this statistics type is NULL, the statistics type associated with the index or column is stored.

Syntax

```
DBMS_STATS.SET_INDEX_STATS (
   ownname       VARCHAR2,
   indname       VARCHAR2,
   partname      VARCHAR2  DEFAULT NULL,
   stattab       VARCHAR2  DEFAULT NULL,
   statid        VARCHAR2  DEFAULT NULL,
   numrows       NUMBER    DEFAULT NULL,
   numlblks      NUMBER    DEFAULT NULL,
   numdist       NUMBER    DEFAULT NULL,
   avglblk       NUMBER    DEFAULT NULL,
   avgdblk       NUMBER    DEFAULT NULL,
   clstfct       NUMBER    DEFAULT NULL,
   indlevel      NUMBER    DEFAULT NULL,
   flags         NUMBER    DEFAULT NULL,
   statown       VARCHAR2  DEFAULT NULL,
   no_invalidate BOOLEAN   DEFAULT to_no_invalidate_type(
                                    get_param('NO_INVALIDATE')),
   guessq        NUMBER    DEFAULT NULL,
   cachedblk     NUMBER    DEFAULT NULL,
   cachehit      NUMBER    DEFUALT NULL,
   force         BOOLEAN   DEFAULT FALSE);
```

Use the following for user-defined statistics:

```
DBMS_STATS.SET_INDEX_STATS (
   ownname       VARCHAR2,
   indname       VARCHAR2,
   partname      VARCHAR2  DEFAULT NULL,
   stattab       VARCHAR2  DEFAULT NULL,
   statid        VARCHAR2  DEFAULT NULL,
   ext_stats     RAW,
   stattypown    VARCHAR2 DEFAULT NULL,
   stattypname   VARCHAR2 DEFAULT NULL,
   statown       VARCHAR2 DEFAULT NULL,
   no_invalidate BOOLEAN DEFAULT to_no_invalidate_type(
                                    get_param('NO_INVALIDATE')),
   cachedblk     NUMBER    DEFAULT NULL,
   cachehit      NUMBER    DEFUALT NULL,
   force         BOOLEAN   DEFAULT FALSE);
```

Parameters

Table 103–66 SET_INDEX_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema
indname	Name of the index

Table 103–66 (Cont.) SET_INDEX_STATS Procedure Parameters

Parameter	Description
partname	Name of the index partition in which to store the statistics. If the index is partitioned and if partname is NULL, then the statistics are stored at the global index level.
stattab	User statistics table identifier describing where to store the statistics. If stattab is NULL, then the statistics are stored directly in the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL)
ext_stats	The user-defined statistics
stattypown	Schema of the statistics type
stattypname	Name of the statistics type
numrows	Number of rows in the index (partition)
numlblks	Number of leaf blocks in the index (partition)
numdist	Number of distinct keys in the index (partition)
avglblk	Average integral number of leaf blocks in which each distinct key appears for this index (partition). If not provided, then this value is derived from numlblks and numdist.
avgdblk	Average integral number of data blocks in the table pointed to by a distinct key for this index (partition). If not provided, then this value is derived from clstfct and numdist.
clstfct	See clustering_factor column of the all_indexes view for a description
indlevel	Height of the index (partition)
flags	For internal Oracle use (should be left as NULL)
statown	Schema containing stattab (if different than ownname)
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_PARAM Procedure.
guessq	Guess quality. See the pct_direct_access column of the all_indexes view for a description.
cachedblk	The average number of blocks in the buffer cache for the segment (index/table/index partition/table partition)
cachehit	The average cache hit ratio for the segment (index/table/index partition/table partition)
force	Sets the values even if statistics of the index are locked

Usage Notes

- The Optimizer uses the cached data to estimate number of cached blocks for index or statistics table access. The total cost of the operation will be combined from the I/O cost of reading not cached blocks from disk, the CPU cost of getting cached blocks from the buffer cache, and the CPU cost of processing the data.
- Oracle maintains cachedblk and cachehit at all times but uses correspondent caching statistics for optimization as part of the table and index statistics only

when the user calls DBMS_STATS.GATHER_[TABLE/INDEX/SCHEMA/DATABASE]_STATS procedure for auto mode or DBMS_STATS.GATHER_SYSTEM_STATS for manual mode. In order to prevent the user from utilizing inaccurate and unreliable data, the optimizer will compute a 'confidence factor' for each cachehit and a cachedblk for each object. If the 'confidence factor' for the value meets confidence criteria, this value will be used, otherwise the defaults will be used.

- The automatic maintenance algorithm for object caching statistics assumes that there is only one major workload for the system and adjusts statistics to this workload, ignoring other "minor" workloads. If this is not the case, you must use manual mode for maintaining object caching statistics.
- The object caching statistics maintenance algorithm for auto mode prevents you from using statistics in the following situations
  - When not enough data has been analyzed, such as when an object has been recently create
  - When the system does not have one major workload resulting in averages not corresponding to real values.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20001: Invalid input value.

ORA-20005: Object statistics are locked.

SET_PARAM Procedure

This procedure sets default values for parameters of DBMS_STATS procedures. You can use the GET_PARAM Function to get the current default value of a parameter.

Syntax

```
DBMS_STATS.SET_PARAM (
   pname      IN    VARCHAR2,
   pval       IN    VARCHAR2);
```

Parameters

Table 103–67 SET_PARAM Procedure Parameters

Parameter	Description
pname	The parameter name The default value for following parameters can be set. ■ CASCADE - The default value for CASCADE set by SET_PARAM is not used by export/import procedures.It is used only by gather procedures. ■ DEGREE ■ ESTIMATE_PERCENT ■ METHOD_OPT ■ NO_INVALIDATE ■ GRANULARITY ■ AUTOSTATS_TARGET - This parameter is applicable only for auto statistics collection. The value of this parameter controls the objects considered for statistics collection (see pval)
pval	The parameter value. If NULL is specified, it will set the default value determined by Oracle. When pname is AUTOSTATS_TARGET, the following are valid values: ■ 'ALL' - Statistics are collected for all objects in the system ■ 'ORACLE' - Statistics are collected for all Oracle owned objects ■ 'AUTO' - Oracle decides for which objects to collect statistics

Usage Notes

- To run this procedure, you must have the SYSDBA or both the ANALYZE ANY DICTIONARY and ANALYZE ANY system privileges.
- Note that both arguments are of type VARCHAR2 and the values need to be enclosed in quotes.
- Note also the difference between NULL and 'NULL':
  - When NULL is unquoted, this sets the parameter to the value Oracle recommends.
  - In the case of the quoted 'NULL', this sets the value of the parameter to NULL.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20001: Invalid or illegal input value.

Examples

```
DBMS_STATS.SET_PARAM('CASCADE','DBMS_STATS.AUTO_CASCADE');
DBMS_STATS.SET_PARAM('ESTIMATE_PERCENT','5');
DBMS_STATS.SET_PARAM('DEGREE','NULL');
```

SET_SYSTEM_STATS Procedure

This procedure sets systems statistics.

Syntax

```
DBMS_STATS.SET_SYSTEM_STATS (
   pname          VARCHAR2,
   pvalue         NUMBER,
   stattab   IN   VARCHAR2 DEFAULT NULL,
   statid    IN   VARCHAR2 DEFAULT NULL,
   statown   IN   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 103–68 SET_SYSTEM_STATS Procedure Parameters

Parameter	Description
pname	The parameter name to get, which can have one of the following values: ■ iotfrspeed—I/O transfer speed in bytes for each millisecond ■ ioseektim - seek time + latency time + operating system overhead time, in milliseconds ■ sreadtim - average time to read single block (random read), in milliseconds ■ mreadtim - average time to read an mbrc block at once (sequential read), in milliseconds ■ cpuspeed - average number of CPU cycles for each second, in millions, captured for the workload (statistics collected using 'INTERVAL' or 'START' and 'STOP' options) ■ cpuspeednw - average number of CPU cycles for each second, in millions, captured for the noworkload (statistics collected using 'NOWORKLOAD' option. ■ mbrc - average multiblock read count for sequential read, in blocks ■ maxthr - maximum I/O system throughput, in bytes/second ■ slavethr - average slave I/O throughput, in bytes/second
pvalue	Parameter value to get
stattab	Identifier of the user statistics table where the statistics will be obtained. If stattab is null, the statistics will be obtained from the dictionary.
statid	Optional identifier associated with the statistics saved in the stattab
statown	Schema containing stattab (if different from current schema)
cachedblk	The average number of blocks in the buffer cache for the segment (index/table/index partition/table partition)
cachehit	The average cache hit ratio for the segment (index/table/index partition/table partition)

Usage Notes

- The Optimizer uses the cached data to estimate number of cached blocks for index or statistics table access. The total cost of the operation will be combined from the I/O cost of reading not cached blocks from disk, the CPU cost of getting cached blocks from the buffer cache, and the CPU cost of processing the data.
- Oracle maintains `cachedblk` and `cachehit` at all times but uses correspondent caching statistics for optimization as part of the table and index statistics only when the user calls `DBMS_STATS.GATHER_[TABLE/INDEX/SCHEMA/DATABASE]_STATS` procedure for auto mode or `DBMS_STATS.GATHER_SYSTEM_STATS` for manual mode. In order to prevent the user from utilizing inaccurate and unreliable data, the optimizer will compute a 'confidence factor' for each `cachehit` and a `cachedblk` for each object. If the 'confidence factor' for the value meets confidence criteria, this value will be used, otherwise the defaults will be used.
- The automatic maintenance algorithm for object caching statistics assumes that there is only one major workload for the system and adjusts statistics to this workload, ignoring other "minor" workloads. If this is not the case, you must use manual mode for maintaining object caching statistics.
- The object caching statistics maintenance algorithm for auto mode prevents you from using statistics in the following situations
  - When not enough data has been analyzed, such as when an object has been recently create
  - When the system does not have one major workload resulting in averages not corresponding to real values.

Exceptions

`ORA-20000`: Object does not exist or insufficient privileges.

`ORA-20001`: Invalid input value.

`ORA-20002`: Bad user statistics table; may need to be upgraded.

`ORA-20003`: Unable to set system statistics.

`ORA-20004`: Parameter does not exist.

SET_TABLE_STATS Procedure

This procedure sets table-related information.

Syntax

```
DBMS_STATS.SET_TABLE_STATS (
   ownname       VARCHAR2,
   tabname       VARCHAR2,
   partname      VARCHAR2 DEFAULT NULL,
   stattab       VARCHAR2 DEFAULT NULL,
   statid        VARCHAR2 DEFAULT NULL,
   numrows       NUMBER   DEFAULT NULL,
   numblks       NUMBER   DEFAULT NULL,
   avgrlen       NUMBER   DEFAULT NULL,
   flags         NUMBER   DEFAULT NULL,
   statown       VARCHAR2 DEFAULT NULL,
   no_invalidate BOOLEAN  DEFAULT to_no_invalidate_type (
                                    get_param('NO_INVALIDATE')),
   cachedblk     NUMBER    DEFAULT NULL,
   cachehit      NUMBER    DEFUALT NULL,
   force         BOOLEAN   DEFAULT FALSE);
```

Parameters

Table 103–69 SET_TABLE_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema
tabname	Name of the table
partname	Name of the table partition in which to store the statistics. If the table is partitioned and partname is NULL, then the statistics are stored at the global table level.
stattab	User statistics table identifier describing where to store the statistics. If stattab is NULL, then the statistics are stored directly in the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL)
numrows	Number of rows in the table (partition)
numblks	Number of blocks the table (partition) occupies
avgrlen	Average row length for the table (partition)
flags	For internal Oracle use (should be left as NULL)
statown	Schema containing stattab (if different than ownname)
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_PARAM Procedure.
cachedblk	The average number of blocks in the buffer cache for the segment (index/table/index partition/table partition)
cachehit	The average cache hit ratio for the segment (index/table/index partition/table partition)

Table 103–69 (Cont.) SET_TABLE_STATS Procedure Parameters

Parameter	Description
force	Sets the values even if statistics of the table are locked

Usage Notes

- The Optimizer uses the cached data to estimate number of cached blocks for index or statistics table access. The total cost of the operation will be combined from the I/O cost of reading not cached blocks from disk, the CPU cost of getting cached blocks from the buffer cache, and the CPU cost of processing the data.
- Oracle maintains `cachedblk` and `cachehit` at all times but uses correspondent caching statistics for optimization as part of the table and index statistics only when the user calls `DBMS_STATS.GATHER_[TABLE/INDEX/SCHEMA/DATABASE]_STATS` procedure for auto mode or `DBMS_STATS.GATHER_SYSTEM_STATS` for manual mode. In order to prevent the user from utilizing inaccurate and unreliable data, the optimizer will compute a 'confidence factor' for each `cachehit` and a `cachedblk` for each object. If the 'confidence factor' for the value meets confidence criteria, this value will be used, otherwise the defaults will be used.
- The automatic maintenance algorithm for object caching statistics assumes that there is only one major workload for the system and adjusts statistics to this workload, ignoring other "minor" workloads. If this is not the case, you must use manual mode for maintaining object caching statistics.
- The object caching statistics maintenance algorithm for auto mode prevents you from using statistics in the following situations
  - When not enough data has been analyzed, such as when an object has been recently create
  - When the system does not have one major workload resulting in averages not corresponding to real values.

Exceptions

`ORA-20000`: Object does not exist or insufficient privileges.

`ORA-20001`: Invalid input value.

`ORA-20005`: Object statistics are locked.

UNLOCK_SCHEMA_STATS Procedure

This procedure unlocks the statistics on all the tables in schema.

Syntax

```
DBMS_STATS.UNLOCK_SCHEMA_STATS (
   ownname     VARCHAR2);
```

Parameters

Table 103–70 UNLOCK_SCHEMA_STATS Procedure Parameters

Parameter	Description
ownname	The name of the schema

Usage Notes

- When statistics on a table is locked, all the statistics depending on the table, including table statistics, column statistics, histograms and statistics on all dependent indexes, are considered to be locked.
- The SET_*, DELETE_*, IMPORT_*, GATHER_* procedures that modify statistics in the dictionary of an individual table, index or column will raise an error if statistics of the object is locked.
- Procedures that operates on multiple objects (such as GATHER_SCHEMA_STATS) will skip modifying the statistics of an object if it is locked. Many procedures have force argument to override the lock.

UNLOCK_TABLE_STATS Procedure

This procedure unlocks the statistics on the table.

Syntax

```
DBMS_STATS.UNLOCK_TABLE_STATS (
   ownname    VARCHAR2,
   tabname    VARCHAR2);
```

Parameters

Table 103–71 UNLOCK_TABLE_STATS Procedure Parameters

Parameter	Description
ownname	The name of the schema
tabname	The name of the table

Usage Notes

- When statistics on a table is locked, all the statistics depending on the table, including table statistics, column statistics, histograms and statistics on all dependent indexes, are considered to be locked.
- The SET_*, DELETE_*, IMPORT_*, GATHER_* procedures that modify statistics in the dictionary of an individual table, index or column will raise an error if statistics of the object is locked.
- Procedures that operates on multiple objects (such as GATHER_SCHEMA_STATS) will skip modifying the statistics of an object if it is locked. Many procedures have force argument to override the lock.

UPGRADE_STAT_TABLE Procedure

This procedure upgrades a user statistics table from an older version.

Syntax

```
DBMS_STATS.UPGRADE_STAT_TABLE (
   ownname    VARCHAR2,
   stattab    VARCHAR2);
```

Parameters

Table 103–72 UPGRADE_STAT_TABLE Procedure Parameters

Parameter	Description
ownname	Name of the schema
stattab	Name of the table

Exceptions

ORA-20000: Unable to upgrade table.

104

DBMS_STORAGE_MAP

With the DBMS_STORAGE_MAP package, you can communicate with the Oracle background process FMON to invoke mapping operations that populate mapping views. FMON communicates with operating and storage system vendor-supplied mapping libraries.

This chapter contains the following topics:

Using DBMS_STORAGE_MAP

Overview

The following terminology and descriptions will help you understand the DBMS_STORAGE_MAP API:

- Mapping libraries

  Mapping libraries help you map the components of I/O processing stack elements. Examples of I/O processing components include files, logical volumes, and storage array I/O targets. The mapping libraries are identified in filemap.ora.

- Mapping files

  A mapping file is a mapping structure that describes a file. It provides a set of attributes, including file size, number of extents that the file is composed of, and file type.

- Mapping elements and sub-elements

  A mapping element is the abstract mapping structure that describes a storage component within the I/O stack. Examples of elements include mirrors, stripes, partitions, raid5, concatenated elements, and disks—structures that are the mapping building blocks. A mapping sub-element describes the link between an element and the next elements in the I/O mapping stack

- Mapping file extents

  A mapping file extent describes a contiguous chunk of blocks residing on one element. This includes the device offset, the extent size, the file offset, the type (data or parity), and the name of the element where the extent resides. In the case of a raw device or volume, the file is composed of only one file extent component. A mapping file extent is different from Oracle extents.

See Also:

- *Oracle Database Administrator's Guide* for more information
- *Oracle Database Reference* for V$MAP views, including V$MAP_FILE, V$MAP_ELEMENT, V$MAP_SUBELEMENT, V$MAP_FILE_EXTENT

Operational Notes

For `MAP_ELEMENT`, `MAP_FILE`, and `MAP_ALL`: Invoking these functions when mapping information already exists will refresh the mapping if configuration IDs are supported. If configuration IDs are not supported, then invoking these functions again will rebuild the mapping.

> **See Also:** *Oracle Database Administrator's Guide* for a discussion of the configuration ID, an attribute of the element or file that is changed.

Summary of DBMS_STORAGE_MAP Subprograms

Table 104–1 DBMS_STORAGE_MAP Package Subprograms

Subprogram	Description
DROP_ALL Function on page 104-6	Drops all mapping information in the shared memory of the instance
DROP_ELEMENT Function on page 104-7	Drops the mapping information for the element defined by `elemname`
DROP_FILE Function on page 104-8	Drops the file mapping information defined by `filename`
LOCK_MAP Procedure on page 104-9	Locks the mapping information in the shared memory of the instance
MAP_ALL Function on page 104-10	Builds the entire mapping information for all types of Oracle files (except archive logs), including all directed acyclic graph (DAG) elements
MAP_ELEMENT Function on page 104-11	Builds mapping information for the element identified by `elemname`
MAP_FILE Function on page 104-12	Builds mapping information for the file identified by `filename`
MAP_OBJECT Function on page 104-13	Builds the mapping information for the Oracle object identified by the object name, owner, and type
RESTORE Function on page 104-14	Loads the entire mapping information from the data dictionary into the shared memory of the instance
SAVE Function on page 104-15	Saves information needed to regenerate the entire mapping into the data dictionary
UNLOCK_MAP Procedure on page 104-16	Unlocks the mapping information in the shared memory of the instance.

DROP_ALL Function

This function drops all mapping information in the shared memory of the instance.

Syntax

```
DBMS_STORAGE_MAP.DROP_ALL(
      dictionary_update IN BOOLEAN DEFAULT TRUE);
```

Parameters

Table 104–2 DROP_ALL Function Parameters

Parameter	Description
`dictionary_update`	If `TRUE`, mapping information in the data dictionary is updated to reflect the changes. The default value is `TRUE`; `dictionary_update` is an overloaded argument.

DROP_ELEMENT Function

This function drops the mapping information for the element defined by `elemname`.

Syntax

```
DBMS_STORAGE_MAP.DROP_ELEMENT(
   elemname          IN VARCHAR2,
   cascade           IN BOOLEAN,
   dictionary_update IN BOOLEAN DEFAULT TRUE);
```

Parameters

Table 104–3 DROP_ELEMENT Function Parameters

Parameter	Description
`elemname`	The element for which mapping information is dropped.
`cascade`	If `TRUE`, then `DROP_ELEMENT` is invoked recursively on all elements of the DAG defined by `elemname`, if possible.
`dictionary_update`	If `TRUE`, mapping information in the data dictionary is updated to reflect the changes. The default value is `TRUE`; `dictionary_update` is an overloaded argument.

DROP_FILE Function

This function drops the file mapping information defined by `filename`.

Syntax

```
DBMS_STORAGE_MAP.DROP_FILE(
   filename          IN VARCHAR2,
   cascade           IN BOOLEAN,
   dictionary_update IN BOOLEAN DEFAULT TRUE);
```

Parameters

Table 104–4 DROP_FILE Function Parameters

Parameter	Description
`filename`	The file for which file mapping information is dropped.
`cascade`	If `TRUE`, then the mapping DAGs for the elements where the file resides are also dropped, if possible.
`dictionary_update`	If `TRUE`, mapping information in the data dictionary is updated to reflect the changes. The default value is `TRUE`; `dictionary_update` is an overloaded argument.

LOCK_MAP Procedure

This procedure locks the mapping information in the shared memory of the instance. This is useful when you need a consistent snapshot of the V$MAP tables. Without locking the mapping information, `V$MAP_ELEMENT` and `V$MAP_SUBELEMENT`, for example, may be inconsistent.

Syntax

```
DBMS_STORAGE_MAP.LOCK_MAP;
```

MAP_ALL Function

This function builds the entire mapping information for all types of Oracle files (except archive logs), including all directed acyclic graph (DAG) elements. It obtains the latest mapping information because it explicitly synchronizes all mapping libraries.

Syntax

```
DBMS_STORAGE_MAP.MAP_ALL(
   max_num_fileext   IN NUMBER DEFAULT 100,
   dictionary_update IN BOOLEAN DEFAULT TRUE);
```

Parameters

Table 104–5 MAP_ALL Function Parameters

Parameter	Description
`max_num_fileext`	Defines the maximum number of file extents to be mapped. This limits the amount of memory used when mapping file extents. The default value is `100`; `max_num_fileextent` is an overloaded argument.
`dictionary_update`	If `TRUE`, mapping information in the data dictionary is updated to reflect the changes. The default value is `TRUE`; `dictionary_update` is an overloaded argument.

Usage Notes

You must explicitly call `MAP_ALL` in a cold startup scenario.

MAP_ELEMENT Function

This function builds mapping information for the element identified by `elemname`. It may not obtain the latest mapping information if the element being mapped, or any one of the elements within its I/O stack (if `cascade` is `TRUE`), is owned by a library that must be explicitly synchronized.

Syntax

```
DBMS_STORAGE_MAP.MAP_ELEMENT(
   elemname          IN VARCHAR2,
   cascade           IN BOOLEAN,
   dictionary_update IN BOOLEAN DEFAULT TRUE);
```

Parameters

Table 104–6 MAP_ELEMENT Function Parameters

Parameter	Description
`elemname`	The element for which mapping information is built.
`cascade`	If `TRUE`, all elements within the elemname I/O stack DAG are mapped.
`dictionary_update`	If `TRUE`, mapping information in the data dictionary is updated to reflect the changes. The default value is `TRUE`; `dictionary_update` is an overloaded argument.

MAP_FILE Function

This function builds mapping information for the file identified by `filename`. Use this function if the mapping of one particular file has changed. The Oracle database server does not have to rebuild the entire mapping.

Syntax

```
DBMS_STORAGE_MAP.MAP_FILE(
   filename           IN VARCHAR2,
   filetype           IN VARCHAR2,
   cascade            IN BOOLEAN,
   max_num_fileextent IN NUMBER DEFAULT 100,
   dictionary_update  IN BOOLEAN DEFAULT TRUE);
```

Parameters

Table 104–7 MAP_FILE Function Parameters

Parameter	Description
`filename`	The file for which mapping information is built.
`filetype`	Defines the type of the file to be mapped. It can be "`DATAFILE`", "`SPFILE`", "`TEMPFILE`", "`CONTROLFILE`", "`LOGFILE`", or "`ARCHIVEFILE`".
`cascade`	Should be `TRUE` only if a storage reconfiguration occurred. For all other instances, such as file resizing (either through an `ALTER SYSTEM` command or DML operations on extended files), `cascade` can be set to `FALSE` because the mapping changes are limited to the file extents only. If `TRUE`, mapping DAGs are also built for the elements where the file resides.
`max_num_fileextent`	Defines the maximum number of file extents to be mapped. This limits the amount of memory used when mapping file extents. The default value is `100`; `max_num_fileextent` is an overloaded argument.
`dictionary_update`	If `TRUE`, mapping information in the data dictionary is updated to reflect the changes. The default value is `TRUE`; `dictionary_update` is an overloaded argument.

Usage Notes

This function may not obtain the latest mapping information if the file being mapped, or any one of the elements within its I/O stack (if `cascade` is `TRUE`), is owned by a library that must be explicitly synchronized.

MAP_OBJECT Function

This function builds the mapping information for the Oracle object identified by the object name, owner, and type.

Syntax

```
DBMS_STORAGE_MAP.MAP_OBJECT(
   objname  IN  VARCHAR2,
   owner    IN  VARCHAR2,
   objtype  IN  VARCHAR2);
```

Parameters

Table 104–8 MAP_OBJECT Function Parameters

Parameter	Description
objname	The name of the object.
owner	The owner of the object.
objtype	The type of the object.

RESTORE Function

This function loads the entire mapping information from the data dictionary into the shared memory of the instance. You can invoke `RESTORE` only after a `SAVE` operation. You must explicitly call `RESTORE` in a warm startup scenario.

Syntax

```
DBMS_STORAGE_MAP.RESTORE;
```

SAVE Function

This function saves information needed to regenerate the entire mapping into the data dictionary.

Syntax

```
DBMS_STORAGE_MAP.SAVE;
```

UNLOCK_MAP Procedure

This procedure unlocks the mapping information in the shared memory of the instance.

Syntax

```
DBMS_STORAGE_MAP.UNLOCK_MAP;
```

105

DBMS_STREAMS

The DBMS_STREAMS package, one of a set of Streams packages, provides subprograms to convert ANYDATA objects into logical change record (LCR) objects, to return information about Streams attributes and Streams clients, and to annotate redo entries generated by a session with a binary tag. This tag affects the behavior of a capture process, a propagation, or an apply process whose rules include specifications for these binary tags in redo entries or LCRs.

> **See Also:** *Oracle Streams Concepts and Administration* and *Oracle Streams Replication Administrator's Guide* for more information about this package and Streams

This chapter contains the following topics:

Using DBMS_STREAMS

This section contains topics which relate to using the `DBMS_STREAMS` package.

Security Model

User group PUBLIC is granted EXECUTE privilege on this package.

See Also: *Oracle Database Security Guide* for more information about user group PUBLIC

Summary of DBMS_STREAMS Subprograms

Table 105–1 DBMS_STREAMS Package Subprograms

Subprogram	Description
COMPATIBLE_10_2 Function on page 105-5	Returns the `DBMS_STREAMS.COMPATIBLE_10_2` constant
COMPATIBLE_10_1 Function on page 105-6	Returns the `DBMS_STREAMS.COMPATIBLE_10_1` constant
COMPATIBLE_9_2 Function on page 105-7	Returns the `DBMS_STREAMS.COMPATIBLE_9_2` constant
COMPATIBLE_10_1 Function on page 105-6	Converts a `ANYDATA` object to a `SYS.LCR$_DDL_RECORD` object
CONVERT_ANYDATA_TO_LCR_ROW Function on page 105-9	Converts a `ANYDATA` object to a `SYS.LCR$_ROW_RECORD` object
CONVERT_LCR_TO_XML Function on page 105-10	Converts a logical change record (LCR) encapsulated in a `ANYDATA` object into an XML object that conforms to the XML schema for LCRs
CONVERT_XML_TO_LCR Function on page 105-11	Converts an XML object that conforms to the XML schema for LCRs into a logical change record (LCR) encapsulated in a `ANYDATA` object
GET_INFORMATION Function on page 105-12	Returns information about various Streams attributes
GET_STREAMS_NAME Function on page 105-13	Returns the name of the invoker
GET_STREAMS_TYPE Function on page 105-14	Returns the type of the invoker
GET_TAG Function on page 105-15	Gets the binary tag for all redo entries generated by the current session
SET_TAG Procedure on page 105-16	Sets the binary tag for all redo entries subsequently generated by the current session

Note: The subprograms in this package do not commit.

COMPATIBLE_10_2 Function

This function returns the DBMS_STREAMS.COMPATIBLE_10_2 constant.

Syntax

```
DBMS_STREAMS.COMPATIBLE_10_2
 RETURN INTEGER;
```

Usage Notes

You can use this function with the GET_COMPATIBLE member function for logical change records (LCRs) to specify behavior based on compatibility.

The constant value returned by this function corresponds to 10.2.0 compatibility in a database. You control the compatibility of an Oracle database using the COMPATIBLE initialization parameter.

See Also:

- GET_COMPATIBLE Member Function on page 188-27
- *Oracle Streams Concepts and Administration* for information about creating rules that discard changes that are not supported by Streams
- *Oracle Database Reference* and *Oracle Database Upgrade Guide* for more information about the COMPATIBLE initialization parameter

COMPATIBLE_10_1 Function

This function returns the DBMS_STREAMS.COMPATIBLE_10_1 constant.

Syntax

```
DBMS_STREAMS.COMPATIBLE_10_1
 RETURN INTEGER;
```

Usage Notes

You can use this function with the GET_COMPATIBLE member function for logical change records (LCRs) to specify behavior based on compatibility.

The constant value returned by this function corresponds to 10.1.0 compatibility in a database. You control the compatibility of an Oracle database using the COMPATIBLE initialization parameter.

See Also:

- GET_COMPATIBLE Member Function on page 188-27
- *Oracle Streams Concepts and Administration* for information about creating rules that discard changes that are not supported by Streams
- *Oracle Database Reference* and *Oracle Database Upgrade Guide* for more information about the COMPATIBLE initialization parameter

COMPATIBLE_9_2 Function

This function returns the DBMS_STREAMS.COMPATIBLE_9_2 constant.

Syntax

```
DBMS_STREAMS.COMPATIBLE_9_2
 RETURN INTEGER;
```

Usage Notes

You can use this function with the GET_COMPATIBLE member function for logical change records (LCRs) to specify behavior based on compatibility.

The constant value returned by this function corresponds to 9.2.0 compatibility in a database. You control the compatibility of an Oracle database using the COMPATIBLE initialization parameter.

See Also:

- GET_COMPATIBLE Member Function on page 188-27
- *Oracle Streams Concepts and Administration* for information about creating rules that discard changes that are not supported by Streams
- *Oracle Database Reference* and *Oracle Database Upgrade Guide* for more information about the COMPATIBLE initialization parameter

CONVERT_ANYDATA_TO_LCR_DDL Function

This function converts a ANYDATA object into a SYS.LCR$_DDL_RECORD object.

Syntax

```
DBMS_STREAMS.CONVERT_ANYDATA_TO_LCR_DDL(
   source  IN  ANYDATA)
 RETURN SYS.LCR$_DDL_RECORD;
```

Parameters

Table 105–2 CONVERT_ANYDATA_TO_LCR_DDL Function Parameters

Parameter	Description
source	The ANYDATA object to be converted. If this object is not a DDL logical change record (DDL LCR), then the function raises an exception.

Usage Notes

You can use this function in a transformation created by the CREATE_TRANSFORMATION procedure in the DBMS_TRANSFORM package. Use the transformation you create when you add a subscriber for propagation of DDL LCRs from a ANYDATA queue to a SYS.LCR$_DDL_RECORD typed queue.

> **See Also:** *Oracle Streams Concepts and Administration* for more information about this function

CONVERT_ANYDATA_TO_LCR_ROW Function

This function converts a ANYDATA object into a SYS.LCR$_ROW_RECORD object.

Syntax

```
DBMS_STREAMS.CONVERT_ANYDATA_TO_LCR_ROW(
   source  IN  ANYDATA)
 RETURN SYS.LCR$_ROW_RECORD;
```

Parameters

Table 105–3 CONVERT_ANYDATA_TO_LCR_ROW Function Parameters

Parameter	Description
source	The ANYDATA object to be converted. If this object is not a row logical change record (row LCR), then the function raises an exception.

Usage Notes

You can use this function in a transformation created by the CREATE_TRANSFORMATION procedure in the DBMS_TRANSFORM package. Use the transformation you create when you add a subscriber for propagation of row LCRs from a ANYDATA queue to a SYS.LCR$_ROW_RECORD typed queue.

> **See Also:** *Oracle Streams Concepts and Administration* for more information about this function

CONVERT_LCR_TO_XML Function

This function converts a logical change record (LCR) encapsulated in a ANYDATA object into an XML object that conforms to the XML schema for LCRs. The LCR can be a row LCR or a DDL LCR.

> **See Also:** *Oracle Streams Concepts and Administration* for more information about the XML schema for LCRs

Syntax

```
DBMS_STREAMS.CONVERT_LCR_TO_XML(
   anylcr  IN  ANYDATA)
 RETURN SYS.XMLTYPE;
```

Parameters

Table 105–4 CONVERT_LCR_TO_XML Function Parameters

Parameter	Description
anylcr	The ANYDATA encapsulated LCR to be converted. If this object is not a ANYDATA encapsulated LCR, then the function raises an exception.

CONVERT_XML_TO_LCR Function

This function converts an XML object that conforms to the XML schema for logical change records (LCRs) into an LCR encapsulated in a `ANYDATA` object. The LCR can be a row or DDL LCR.

> **See Also:** *Oracle Streams Concepts and Administration* for more information about the XML schema for LCRs

Syntax

```
DBMS_STREAMS.CONVERT_XML_TO_LCR(
   xmldat  IN  SYS.XMLTYPE)
 RETURN ANYDATA;
```

Parameters

Table 105–5 CONVERT_XML_TO_LCR Function Parameters

Parameter	Description
`xmldat`	The XML LCR object to be converted. If this object does not conform to XML schema for LCRs, then the function raises an exception.

GET_INFORMATION Function

This function returns information about various Streams attributes.

Syntax

```
DBMS_STREAMS.GET_INFORMATION(
   name  IN  VARCHAR2)
 RETURN ANYDATA;
```

Parameters

Table 105–6 GET_INFORMATION Function Parameters

Parameter	Description
name	The type of information you want to retrieve. Currently, the following names are available: ■ SENDER: Returns the name of the sender for the current logical change record (LCR) from its AQ message properties. This function is called inside an apply handler. An apply handler is a DML handler, a DDL handler, an error handler, or a message handler. Returns NULL if called outside of an apply handler. The return value is to be interpreted as a VARCHAR2. ■ CONSTRAINT_NAME: Returns the name of the constraint that was violated for an LCR that raised an error. This function is called inside a DML handler or error handler for an apply process. Returns NULL if called outside of a DML handler or error handler. The return value is to be interpreted as a VARCHAR2.

GET_STREAMS_NAME Function

This function gets the Streams name of the invoker if the invoker is one of the following Streams types:

- CAPTURE
- APPLY
- ERROR_EXECUTION

If the invoker is not one of these types, then this function returns a NULL.

Syntax

```
DBMS_STREAMS.GET_STREAMS_NAME
 RETURN VARCHAR2;
```

Usage Notes

You can use this function in rule conditions, rule-based transformations, apply handlers, and error handlers. For example, if you use one error handler for multiple apply processes, then you can use the GET_STREAMS_NAME function to determine the name of the apply process that raised the error.

GET_STREAMS_TYPE Function

This function gets the Streams type of the invoker and returns one of the following types:

- CAPTURE
- APPLY
- ERROR_EXECUTION

If the invoker is not one of these types, then this function returns a NULL.

Syntax

```
DBMS_STREAMS.GET_STREAMS_TYPE
 RETURN VARCHAR2;
```

Usage Notes

This function can be used in rule conditions, rule-based transformations, apply handlers, and error handlers. For example, you can use the GET_STREAMS_TYPE function to instruct a DML handler to operate differently if it is processing messages from the error queue (ERROR_EXECUTION type) instead of the apply process queue (APPLY type).

GET_TAG Function

This function gets the binary tag for all redo entries generated by the current session.

> **See Also:** *Oracle Streams Replication Administrator's Guide* for more information about tags

Syntax

```
DBMS_STREAMS.GET_TAG
 RETURN RAW;
```

Examples

The following example illustrates how to display the current logical change record (LCR) tag as output:

```
SET SERVEROUTPUT ON
DECLARE
   raw_tag RAW(2000);
BEGIN
   raw_tag := DBMS_STREAMS.GET_TAG();
   DBMS_OUTPUT.PUT_LINE('Tag Value = ' || RAWTOHEX(raw_tag));
END;
/
```

You can also display the value by querying the `DUAL` view:

```
SELECT DBMS_STREAMS.GET_TAG FROM DUAL;
```

SET_TAG Procedure

This procedure sets the binary tag for all redo entries subsequently generated by the current session. Each redo entry generated by DML or DDL statements in the current session will have this tag. This procedure affects only the current session.

Note: This procedure is not transactional. That is, the effects of `SET_TAG` cannot be rolled back.

See Also: *Oracle Streams Replication Administrator's Guide* for more information about tags

Syntax

```
DBMS_STREAMS.SET_TAG(
   tag  IN RAW  DEFAULT NULL);
```

Parameters

Table 105–7 SET_TAG Procedure Parameters

Parameter	Description
`tag`	The binary tag for all subsequent redo entries generated by the current session. A raw value is a sequence of bytes, and a byte is a sequence of bits. By default, the tag for a session is `NULL`. The size limit for a tag value is 2000 bytes.

Usage Notes

To set the tag to the hexadecimal value of `'17'` in the current session, run the following procedure:

```
EXEC DBMS_STREAMS.SET_TAG(tag => HEXTORAW('17'));
```

106

DBMS_STREAMS_ADM

The DBMS_STREAMS_ADM package, one of a set of Streams packages, provides subprograms for adding and removing simple rules for capture, propagation, apply, and dequeue at the table, schema, and database level.

See Also:

- *Oracle Streams Concepts and Administration* and *Oracle Streams Replication Administrator's Guide* for more information about this package and Streams
- Chapter 91, "DBMS_RULE"

This chapter contains the following topics:

Using DBMS_STREAMS_ADM

This section contains topics which relate to using the `DBMS_STREAMS_ADM` package.

Overview

The DBMS_STREAMS_ADM package, one of a set of Streams packages, provides subprograms for adding and removing simple rules for capture, propagation, apply, and dequeue at the table, schema, and database level. These rules support logical change records (LCRs), which include row LCRs and data definition language (DDL) LCRs. This package also contains subprograms for creating message rules for specific message types. This package also contains subprograms for configuring a Streams replication environment, creating queues, and for managing Streams metadata, such as data dictionary information.

If you require more sophisticated rules, refer to Chapter 91, "DBMS_RULE" package.

Deprecated Subprograms

> **Note:** Oracle recommends that you do not use deprecated subprograms. Support for deprecated features is for backward compatibility only.

The following subprograms are deprecated with Oracle Database 10*g* Release 2:

- `MAINTAIN_SIMPLE_TABLESPACE`

  This procedure is replaced by the `MAINTAIN_SIMPLE_TTS` procedure.

  See Also: MAINTAIN_SIMPLE_TTS Procedure on page 106-94

- `MAINTAIN_TABLESPACES`

  This procedure is replaced by the `MAINTAIN_TTS` procedure.

  See Also: MAINTAIN_TTS Procedure on page 106-108

Security Model

A user is associated with each Streams client. The following sections describe these users.

Capture User

The following procedures can create a capture process:

- ADD_GLOBAL_RULES Procedure
- ADD_SCHEMA_RULES Procedure
- ADD_SUBSET_RULES Procedure
- ADD_TABLE_RULES Procedure

If one of these procedures creates a capture process, then it configures the current user as the `capture_user`. The capture user is the user in whose security domain a capture process captures changes that satisfy its rule sets and runs custom rule-based transformations configured for capture process rules. This user must have the necessary privileges to capture changes. The procedure grants the capture user enqueue privilege on the queue used by the capture process and configures the user as a secure queue user of the queue.

> **See Also:** CREATE_CAPTURE Procedure on page 20-12 for information about the privileges required to capture changes

Propagation User

The following procedures can create a propagation:

- ADD_GLOBAL_PROPAGATION_RULES Procedure
- ADD_MESSAGE_PROPAGATION_RULE Procedure
- ADD_SCHEMA_PROPAGATION_RULES Procedure
- ADD_SUBSET_PROPAGATION_RULES Procedure
- ADD_TABLE_PROPAGATION_RULES Procedure

When a propagation is created, a propagation job also might be created. If a propagation job is created when one of these procedures is run, then the user who runs the procedure owns the propagation job.

Note:

- The source queue owner performs the propagation, but the propagation job is owned by the user who creates it. These two users might or might not be the same.
- For a propagation to work properly, the owner of the source queue must have the necessary privileges to propagate messages.

See Also:

- CREATE_PROPAGATION Procedure on page 74-5 for more information about the required privileges
- "Propagation Rules for LCRs" on page 106-9 for information about when a propagation job is created

Apply User

The following procedures can create an apply process:

- ADD_GLOBAL_RULES Procedure
- ADD_MESSAGE_RULE Procedure
- ADD_SCHEMA_RULES Procedure
- ADD_SUBSET_RULES Procedure
- ADD_TABLE_RULES Procedure

If one of these procedures creates an apply process, then it configures the current user as the `apply_user`. The apply user is the user in whose security domain an apply process dequeues messages that satisfy its rule sets, applies messages directly to database objects, runs custom rule-based transformations configured for apply process rules, and runs apply handlers configured for the apply process. This user must have the necessary privileges to apply changes. The procedure grants the apply user dequeue privilege on the queue used by the apply process and configures the user as a secure queue user of the queue.

See Also: CREATE_APPLY Procedure on page 15-11 for information about the privileges required to apply changes (refer to the `apply_user` parameter)

Messaging Client User

The following procedures can create a messaging client:

- ADD_GLOBAL_RULES Procedure
- ADD_MESSAGE_RULE Procedure
- ADD_SCHEMA_RULES Procedure
- ADD_SUBSET_RULES Procedure
- ADD_TABLE_RULES Procedure

If one of these procedures creates a messaging client, then the user who runs this procedure is granted the privileges to dequeue from the queue using the messaging client. The procedure configures this user as a secure queue user of the queue, and only this user can use the messaging client.

Operational Notes

Several procedures in this package create rules for Streams clients, and several procedures configure and maintain a Streams replication environment. The following sections provide information about using these procedures:

- Procedures That Create Rules and Streams Clients
- Procedures That Configure a Streams Replication Environment

Procedures That Create Rules and Streams Clients

Streams clients include capture processes, propagations, apply processes, and messaging clients. Some of the procedures in the `DBMS_STREAMS_ADM` package add rules to the rule sets of Streams clients. The rules can pertain to changes in the redo log, to logical change records (LCRs), or to user messages.

An LCR represents either a row change that results from a data manipulation language (DML) change or a data definition language (DDL) change. An LCR that represents a row change is a row LCR, and an LCR that represents a DDL change is a DDL LCR. LCRs can either represent changes in the redo record that were captured by a capture process, or they can represent changes created by a user or application. A user message is a custom message that is based on a user-defined type and created by users or applications.

For all of the procedures except the ones that create subset rules, you use the `inclusion_rule` parameter to specify the type of rule set (either positive or negative) for the created rules. If the Streams client does not have a rule set of the specified type, then a rule set is created automatically, and the rules are added to the rule set. Other rules in an existing rule set for the Streams client are not affected. Additional rules can be added to a rule set using either the `DBMS_STREAMS_ADM` package or the `DBMS_RULE_ADM` package. If a Streams client has both a positive and a negative rule set, then the negative rule set is always evaluated first.

The following sections describe each type of rule in detail:

- Capture Process Rules for Changes in the Redo Log
- Propagation Rules for LCRs
- Propagation Rules for User Messages
- Apply Process Rules for LCRs
- Apply Process Rules for User Messages
- Messaging Client Rules for LCRs
- Messaging Client Rules for User Messages

> **See Also:** *Oracle Streams Concepts and Administration* for more information about how rules are used in Streams

Capture Process Rules for Changes in the Redo Log

The following procedures add rules to a rule set of a capture process when you specify `capture` for the `streams_type` parameter:

- The `ADD_GLOBAL_RULES` procedure adds rules whose rule condition evaluates to `TRUE` for all changes made to a source database. See ADD_GLOBAL_RULES Procedure on page 106-33.

- The ADD_SCHEMA_RULES procedure adds rules whose rule condition evaluates to TRUE for changes made to a specified schema. See ADD_SCHEMA_RULES Procedure on page 106-50.
- The ADD_SUBSET_RULES procedure adds rules whose rule condition evaluates to TRUE for DML changes made to a subset of rows in a specified table. See ADD_SUBSET_RULES Procedure on page 106-59.
- The ADD_TABLE_RULES procedure adds rules whose rule condition evaluates to TRUE for changes made to a specified table. See ADD_TABLE_RULES Procedure on page 106-69.

If one of these procedures adds rules to the positive rule set for a capture process, then the capture process captures row changes resulting from DML changes, or DDL changes, or both from a source database and enqueues these changes into the specified queue. If one of these procedures adds rules to the negative rule set for a capture process, then the capture process discards row changes, or DDL changes, or both from a source database.

A capture process can capture changes locally at a source database or remotely at a downstream database. Therefore, for capture process rules, you should execute the procedure either at the source database or at a downstream database.

If the capture process is a local capture process, or if the capture process is a downstream capture process that uses a database link to the source database, then these procedures automatically prepare the appropriate database objects for instantiation:

- ADD_GLOBAL_RULES invokes the PREPARE_GLOBAL_INSTANTIATION procedure in the DBMS_CAPTURE_ADM package at the source database.
- ADD_SCHEMA_RULES invokes the PREPARE_SCHEMA_INSTANTIATION procedure in the DBMS_CAPTURE_ADM package at the source database.
- ADD_SUBSET_RULES and ADD_TABLE_RULES invoke the PREPARE_TABLE_INSTANTIATION procedure in the DBMS_CAPTURE_ADM package at the source database.

These procedures also enable supplemental logging for the primary key, unique key, foreign key, and bitmap index columns in the tables prepared for instantiation. The primary key columns are unconditionally logged. The unique key, foreign key, and bitmap index columns are conditionally logged.

If the capture process is a downstream capture process that does not use a database link to the source database, then you must prepare the appropriate objects for instantiation and specify the necessary supplemental logging manually at the source database.

If one of these procedures is executed at a downstream database, then you specify the source database using the source_database parameter, and the specified capture process must exist. The procedure cannot create a capture process if it is run at a downstream database. You can create a capture process at a downstream database using the CREATE_CAPTURE procedure in the DBMS_CAPTURE_ADM package.

> **See Also:** Chapter , "Summary of DBMS_CAPTURE_ADM Subprograms" on page 20-2 for more information about the CREATE_CAPTURE procedure and the procedures that prepare database objects for instantiation

Propagation Rules for LCRs

The following procedures add propagation rules for LCRs to a rule set of a propagation:

- The ADD_GLOBAL_PROPAGATION_RULES procedure adds rules whose rule condition evaluates to TRUE for all LCRs in a source queue. See ADD_GLOBAL_PROPAGATION_RULES Procedure on page 106-28.
- The ADD_SCHEMA_PROPAGATION_RULES procedure adds rules whose rule condition evaluates to TRUE for LCRs in a source queue containing changes made to a specified schema. See ADD_SCHEMA_PROPAGATION_RULES Procedure on page 106-45.
- The ADD_SUBSET_PROPAGATION_RULES procedure adds rules whose rule condition evaluates to TRUE for row LCRs in a source queue containing the results of DML changes made to a subset of rows in a specified table. See "ADD_SUBSET_PROPAGATION_RULES Procedure" on page 106-55.
- The ADD_TABLE_PROPAGATION_RULES procedure adds rules whose rule condition evaluates to TRUE for LCRs in a source queue containing changes made to a specified table. See "ADD_TABLE_PROPAGATION_RULES Procedure" on page 106-64.

If one of these procedures adds rules to the positive rule set for the propagation, then the rules specify that the propagation propagates LCRs in a source queue to a destination queue. If one of these procedures adds rules to the negative rule set for the propagation, then the rules specify that the propagation discards LCRs in a source queue. When you create rules with one of these procedures, and you specify a value for the source_databse parameter, then the rules include conditions for the specified source database.

Propagation Rules for User Messages

The ADD_MESSAGE_PROPAGATION_RULE procedure adds a message rule to a rule set of a propagation. If this procedure adds a rule to the positive rule set for the propagation, then the rule specifies that the propagation propagates the user-enqueued messages of a specific message type that evaluate to TRUE for the rule condition in a source queue to a destination queue. If this procedure adds a rule to the negative rule set for the propagation, then the rule specifies that the propagation discards the user-enqueued messages of a specific message type that evaluate to TRUE for the rule condition in a source queue. This procedure generates a rule name for the rule.

> **See Also:** "ADD_MESSAGE_PROPAGATION_RULE Procedure" on page 106-38

Apply Process Rules for LCRs

The following procedures add rules to a rule set of an apply process when you specify apply for the streams_type parameter:

- The ADD_GLOBAL_RULES procedure adds rules whose rule condition evaluates to TRUE for all LCRs in the apply process queue. See "ADD_GLOBAL_RULES Procedure" on page 106-33.
- The ADD_SCHEMA_RULES procedure adds rules whose rule condition evaluates to TRUE for LCRs in the apply process queue containing changes made to a specified schema. See "ADD_SCHEMA_RULES Procedure" on page 106-50.
- The ADD_SUBSET_RULES procedure adds rules whose rule condition evaluates to TRUE for row LCRs in the apply process queue containing the results of DML

changes made to a subset of rows in a specified table. See "ADD_SUBSET_RULES Procedure" on page 106-59.

- The ADD_TABLE_RULES procedure adds rules whose rule condition evaluates to TRUE for LCRs in the apply process queue containing changes made to a specified table. See "ADD_TABLE_RULES Procedure" on page 106-69.

If one of these procedures adds rules to the positive rule set for the apply process, then the rules specify that the apply process applies LCRs in its queue. If one of these procedures adds rules to the negative rule set for the apply process, then the rules specify that the apply process discards LCRs in its queue. For apply process rules, you should execute these procedures at the destination database.

An apply process can apply captured LCRs from only one source database. If one of these procedures creates an apply process, then specify the source database for the apply process using the source_database parameter. If the source_database parameter is NULL, and one of these procedures creates an apply process, then the source database name of the first LCR received by the apply process is used for the source database.

The rules in the apply process rule sets determine which messages are dequeued by the apply process. When you create rules with one of these procedures, and you specify a value for the source_databse parameter, then the rules include conditions for the specified source database. If the apply process dequeues an LCR with a source database that is different than the source database for the apply process, then an error is raised. In addition, when adding rules to an existing apply process, the database specified in the source_database parameter cannot be different than the source database for the apply process. You can determine the source database for an apply process by querying the DBA_APPLY_PROGRESS data dictionary view.

Changes applied by an apply process created by one of these procedures generate tags in the redo log at the destination database with a value of '00' (double zero). You can use the ALTER_APPLY procedure in the DBMS_APPLY_ADM package to alter the tag value after the apply process is created, if necessary.

An apply process created by one of these procedures can apply messages only at the local database and can apply only captured messages. To create an apply process that applies messages at a remote database or an apply process that applies user-enqueued messages, use the CREATE_APPLY procedure in the DBMS_APPLY_ADM package.

You can also use the DBMS_APPLY_ADM.CREATE_APPLY procedure to specify nondefault values for the apply_captured, apply_user, apply_database_link, and apply_tag parameters when you run that procedure. You can use one of the procedures in the DBMS_STREAMS_ADM package to add rules to a rule set used by the apply process after you create it.

> **See Also:** "ALTER_APPLY Procedure" on page 15-4 and "CREATE_APPLY Procedure" on page 15-11

Apply Process Rules for User Messages

The ADD_MESSAGE_RULE procedure adds a message rule to a rule set of an apply process when you specify apply for the streams_type parameter. For an apply process rule, you should execute this procedure at the destination database.

If this procedure adds a rule to the positive rule set for an apply process, then the apply process dequeues user-enqueued messages of a specific message type that satisfy the apply process rule and sends these messages to its message handler. If no message handler is specified for the apply process, then use the ALTER_APPLY procedure in the DBMS_APPLY_ADM package to set the message handler. If this

procedure adds a rule to the negative rule set for an apply process, then the apply process discards user-enqueued messages of a specific message type that satisfy the apply process rule.

See Also:

- ADD_MESSAGE_RULE Procedure on page 106-42
- ALTER_APPLY Procedure on page 15-4

Messaging Client Rules for LCRs

The following procedures add rules to a rule set of a messaging client when you specify `dequeue` for the `streams_type` parameter:

- The `ADD_GLOBAL_RULES` procedure adds rules whose rule condition evaluates to `TRUE` for all LCRs in the messaging client queue. See "ADD_GLOBAL_RULES Procedure" on page 106-33.
- The `ADD_SCHEMA_RULES` procedure adds rules whose rule condition evaluates to `TRUE` for LCRs in the messaging client queue containing changes made to a specified schema. See "ADD_SCHEMA_RULES Procedure" on page 106-50.
- The `ADD_SUBSET_RULES` procedure adds rules whose rule condition evaluates to `TRUE` for row LCRs in the messaging client queue containing the results of DML changes made to a subset of rows in a specified table. See "ADD_SUBSET_RULES Procedure" on page 106-59.
- The `ADD_TABLE_RULES` procedure adds rules whose rule condition evaluates to `TRUE` for LCRs in the messaging client queue containing changes made to a specified table. See "ADD_TABLE_RULES Procedure" on page 106-69.

If one of these procedures adds rules to the positive rule set for a messaging client, then the messaging client can dequeue user-enqueued row LCRs, or DDL LCRs, or both that originated at the source database matching the `source_database` parameter. If one of these procedures adds rules to the negative rule set for a messaging client, then the messaging client discards user-enqueued row LCRs, or DDL LCRs, or both that originated at the source database matching the `source_database` parameter. You should execute these procedures at the database where you want to dequeue the messages with the messaging client.

Messaging Client Rules for User Messages

The `ADD_MESSAGE_RULE` procedure adds a message rule to a rule set of a messaging client when you specify `dequeue` for the `streams_type` parameter. You should execute this procedure at the database that will dequeue messages.

If this procedure adds a rule to the positive rule set for a messaging client, then the messaging client dequeues user-enqueued messages of a specific message type that satisfy the message rule. If this procedure adds a rule to the negative rule set for a messaging client, then the messaging client discards user-enqueued messages of a specific message type that satisfy the message rule.

See Also: "ADD_MESSAGE_RULE Procedure" on page 106-42

Procedures That Configure a Streams Replication Environment

The following procedures in this package configure a replication environment that is maintained by Streams:

- MAINTAIN_GLOBAL Procedure
- MAINTAIN_SCHEMAS Procedure

- MAINTAIN_SIMPLE_TTS Procedure
- MAINTAIN_TABLES Procedure
- MAINTAIN_TTS Procedure
- PRE_INSTANTIATION_SETUP Procedure and POST_INSTANTIATION_SETUP Procedure

The `PRE_INSTANTIATION_SETUP` and `POST_INSTANTIATION_SETUP` procedures must be used together to complete the Streams replication configuration.

The following sections contain information about using these procedures:

- Local Capture or Downstream Capture for the Source Database
- Single Source and Bi-Directional Configurations
- Streams Clients and Queues Configured By These Procedures
- Change Cycling
- Automatic Platform Conversion
- Actions Performed by These Procedures
- Configuration Progress and Recoverability
- Requirements for Running These Procedures
- Common Parameters for the Configuration Procedures

Local Capture or Downstream Capture for the Source Database

Local capture means that a capture process runs on the source database. Downstream capture means that a capture process runs on a database other than the source database. These procedures can either configure local capture or downstream capture for the database specified in the `source_database` parameter.

The database that captures changes made to the source database is called the capture database. These procedures can configure one of the following databases as the capture database:

- Source database (local capture)
- Destination database (downstream capture)
- A third database (downstream capture)

The database on which the procedure is run is configured as the capture database for the source database. Therefore, to configure local capture at the source database, run the procedure at the source database. To configure downstream capture at the destination database or a third database, run the procedure at the destination database or third database.

Note:

- When these procedures configure downstream capture, they always configure archived-log downstream capture. These procedures do not configure real-time downstream capture. However, the scripts generated by these procedures can be modified to configure real-time downstream capture.
- If these procedures configure bi-directional replication, then the capture process for the destination database always is a local capture process. That is, these procedures always configure the capture process for changes made to the destination database to run on the destination database.
- When the RMAN `DUPLICATE` or `CONVERT DATABASE` command is used for database instantiation with one of these procedures, the destination database cannot be the capture database.

See Also:

- *Oracle Streams Concepts and Administration* for information about local capture and downstream capture
- "Single Source and Bi-Directional Configurations" on page 106-13

Single Source and Bi-Directional Configurations

These procedures either set up a single source Streams configuration with the database specified in the `source_database` parameter acting as the only source database, or these procedures set up a bi-directional Streams configuration with both databases acting as source and destination databases. The `bi_directional` parameter controls whether the Streams configuration is single source or bi-directional:

- If `bi_directional` is `FALSE`, then a capture process captures changes made to the source database and an apply process at the destination database applies these changes. If the destination database is not the capture database, then a propagation propagates the captured changes to the destination database. The default value for this parameter is `FALSE`.
- If `bi_directional` is `TRUE`, then a separate capture process captures changes made to each database, propagations propagate these changes to the other database, and each database applies changes from the other database.

If `bi_directional` is set to `FALSE`, then these procedures do not configure bi-directional replication. Therefore, changes made to the shared database objects at the destination database are not replicated to the source database, and the shared database objects are not kept synchronized at the two databases, unless no changes are made to the shared database objects at the destination database. However, if `bi_directional` is set to `TRUE`, then Streams is configured to keep the shared database objects synchronized at the two databases, even if both databases allow changes to the database objects.

Note: You might need to configure conflict resolution if bi-directional replication is configured.

See Also:

- SET_UPDATE_CONFLICT_HANDLER Procedure on page 15-50
- "Local Capture or Downstream Capture for the Source Database" on page 106-12
- *Oracle Streams Replication Administrator's Guide* for more information about conflict resolution

Streams Clients and Queues Configured By These Procedures

These procedures configure the following Streams clients:

- These procedures configure a capture process that captures changes to the source database. If bi-directional replication is configured, then these procedures also configure a capture process that captures changes to the destination database.
- If the capture database and the destination database are different databases, then these procedures configure a propagation that propagates changes from the capture database to the destination database.
- If the capture database and the destination database are the same database, then the queue names determine whether a propagation is created:
  - If the `capture_queue_name` and `apply_queue_name` parameters specify different queue names, then a propagation is created between the two queues within the destination database.
  - If the `capture_queue_name` and `apply_queue_name` parameters specify the same queue name, then a propagation is not created, and the downstream capture process and the apply process use the same queue. This configuration is possible only if the `bi_directional` parameter is set to `FALSE` to configure a single source replication environment.
- If bi-directional replication is configured, then these procedures configure a propagation that propagates changes from the destination database to the source database.
- These procedures configure an apply process that applies changes at the destination database. These changes originated at the source database. If bi-directional replication is configured, then these procedures also configure an apply process that applies changes to the source database. These changes originated at the destination database.

By default, the `capture_queue_name` and `apply_queue_name` parameters are set to `NULL`. When these parameters are set to `NULL`, these procedures configure a separate queue for each capture process and apply process. The Streams replication environment might operate more efficiently if each Streams client has its own separate queue.

However, two Streams clients share a queue in the following configurations:

- The configuration described previously in this section in which the downstream capture process and the apply process at the destination database share a queue.
- A configuration in which all of the following conditions are met:
  - The capture database is the source database or a third database.
  - The `bi_directional` parameter is set to `TRUE`.

  - The same queue name is specified for the `capture_queue_name` and `apply_queue_name` parameters.

  In this case, the local capture process and the apply process at the destination database share the same queue. If the source database is the capture database, then the local capture process and the apply process at the source database also share the same queue.

The `capture_name` and `capture_queue_name` parameters must be set to `NULL` when both of the following conditions are met:

- The destination database is the capture database.
- The `bi_directional` parameter is set to `TRUE`.

When both of these conditions are met, these procedure configure two capture processes at the destination database, and these capture processes must have different names. When the `capture_name` and `capture_queue_name` parameters are set to `NULL`, the system generates a different name for the capture processes and queues. These procedures raise an error if both conditions are met and either the `capture_name` parameter or the `capture_queue_name` parameter is set to a non-`NULL` value.

See Also:

- "Single Source and Bi-Directional Configurations" on page 106-13
- "Local Capture or Downstream Capture for the Source Database" on page 106-12

Change Cycling

Change cycling happens when a change is sent back to the database where it originated. Typically, change cycling should be avoided because it can result in each change going through endless loops back to the database where it originated. Such loops can result in unintended data in the database and tax the networking and computer resources of an environment.

If the `bi_directional` parameter is set to `TRUE`, then these procedures configure bi-directional replication. To prevent change cycling in a bi-directional Streams replication environment, these procedures configure the environment in the following way:

- The apply process at each database applies each change with an apply tag that is unique to the environment. An apply tag is a Streams tag that is part of each redo record created by the apply process. For example, if a procedure configures databases `sfdb.net` and `nydb.net` for bi-directional replication, then the apply tag for the apply process at `sfdb.net` can be the hexidecimal equivalent of `'1'`, and the apply tag for the apply process at `nydb.net` can be the hexidecimal equivalent of `'2'`.
- The capture process at each database captures all changes to the shared database objects, regardless of tags in the redo records for the changes to these database objects.
- Each propagation propagates all changes made to the shared database objects to the other database in the bi-directional replication environment, except for changes that originated at the other database. Continuing the example, the propagation at `sfdb.net` propagates all changes to `nydb.net`, except for changes with a tag value that is the hexidecimal equivalent of `'1'`, because these changes originated at `nydb.net`. Similarly, the propagation at `nydb.net` propagates all changes to `sfdb.net`, except for changes with a tag value that is

the hexidecimal equivalent of `'2'`. A change that is not propagated because of its tag value is discarded.

These procedures cannot be used to configure multi-directional replication where changes can be cycled back to a source database by a third database in the environment. For example, these procedures cannot be used to configure a Streams replication environment with three databases where each database shares changes with the other two databases in the environment. If these procedures were used to configure a three way replication environment such as this, then changes made at a source database would be cycled back to the same source database. In a valid three way replication environment, a particular change is made only once at each database.

These procedures can be used to configure a Streams replication environment that includes more than two databases, as long as changes made at a source database cannot cycle back to the same source database. For example, a procedure can be run multiple times to configure an environment in which a primary database shares changes with multiple secondary databases. Such an environment is sometimes called a "hub and spoke" environment.

You can configure the Streams environment manually to replicate changes in a multiple source environment where each source database shares changes with the other source databases, or you can modify generated scripts to achieve this.

See Also:

- "Single Source and Bi-Directional Configurations" on page 106-13
- *Oracle Streams Replication Administrator's Guide* for more information about tags, for an example of a hub and spoke environment, and for information about configuring a multiple source environment manually

Automatic Platform Conversion

If the source and destination databases are running on different platforms, then these procedures, or the scripts generated by these procedures, convert transferred datafiles to the appropriate platform automatically.

Actions Performed by These Procedures

To view all of the actions performed by one of these procedures in detail, use the procedure to generate a script, and view the script in a text editor.

Configuration Progress and Recoverability

When one of these procedures is run with the `perform_actions` parameter set to `TRUE`, metadata about its configuration actions is recorded in the following data dictionary views: `DBA_RECOVERABLE_SCRIPT`, `DBA_RECOVERABLE_SCRIPT_PARAMS`, `DBA_RECOVERABLE_SCRIPT_BLOCKS`, and `DBA_RECOVERABLE_SCRIPT_ERRORS`. If the procedure stops because it encounters an error, then you can use the `RECOVER_OPERATION` procedure to complete the configuration after you correct the conditions that caused the error.

Note: When one of these procedures is run with the `perform_actions` parameter set to `FALSE`, these views are not populated. Also, the views are not populated when a script generated by one of these procedures is run.

See Also: "RECOVER_OPERATION Procedure" on page 106-120

Requirements for Running These Procedures

Meet the following requirements when you use one of these procedures:

- Run the procedure at the capture database.
- If the `bi_directional` parameter is set to `TRUE`, or if the source database is not the capture database, then the `source_database` parameter must specify a database that contains the database objects to be shared. The database specified in the `destination_database` parameter might or might not contain these database objects. If the destination database does not contain the shared database objects, then these procedures will instantiate the database objects at the destination database (excluding the `PRE_INSTANTIATION_SETUP` and `POST_INSTANTIATION_SETUP` procedures).
- Both databases must be open during configuration. If the procedure is generating a script only, then the database specified in the `destination_database` parameter does not need to be open when you run the procedure, but both databases must be open when you run the generated script.
- The user who runs one of these procedures should be granted `DBA` role. This user must have the necessary privileges to complete the following actions:
  - Create `ANYDATA` queues, capture processes, propagations, and apply processes.
  - Specify supplemental logging
  - Run subprograms in the `DBMS_STREAMS_ADM` and `DBMS_AQADM` packages.
  - Access the database specified in the `destination_database` parameter through a database link. This database link should have the same name as the global name of the destination database.
- If the `bi_directional` parameter is set to `TRUE` or if a network instantiation will be performed, then the corresponding user at the destination database must be able to use a database link to access the source database. This database link should have the same name as the global name of the source database.
- If these procedures configure downstream capture, then the corresponding user at the capture database must be able to use a database link to access the source database. This database link should have the same name as the global name of the source database.
- If these procedures configure downstream capture, then the corresponding user at the capture database must be able to use a database link to access the destination database. This database link should have the same name as the global name of the destination database.
- Each specified directory object must be created using the SQL statement `CREATE DIRECTORY`, and the user who invokes one of these procedures must have `READ` and `WRITE` privilege on each one.
- These procedures, or the scripts generated by these procedures, must be run at an Oracle Database 10*g* Release 2 database.
- If the `perform_actions` parameter is set to `TRUE` in one of these procedures to configure the Streams replication environment directly, then all of the databases configured by the procedure must be Oracle Database 10*g* Release 2 databases.

- If the `perform_actions` parameter is set to `FALSE` in one of these procedures, and the replication environment is configured with a generated script, then the databases configured by the script must be Oracle Database 10*g* Release 1 or later databases. If the script configures an Oracle Database 10*g* Release 1 database, then the script must be modified so that it does not configure features that are available only in Oracle Database 10*g* Release 2, such as queue-to-queue propagation.

To ensure that the user who runs these procedures has the necessary privileges, you should configure a Streams administrator at each database, and each database link should be should be created in the Streams administrator's schema.

> **See Also:** *Oracle Streams Concepts and Administration* for information about configuring a Streams administrator

Common Parameters for the Configuration Procedures

Table 106–1 describes the common parameters for the procedures in this package that configure a Streams replication environment.

Table 106–1 Common Parameters for Configuration Procedures

Parameter	Description
`source_database`	The global name of the source database. If the specified global name is the same as the global name of the local database, then the procedure configures a local capture process for the source database. If the specified global name is different than the global name of the local database, then the procedure configures a downstream capture process at the local database. In this case, a database link from the local database to the source database with the same name as the global name of the source database must exist and must be accessible to the user who runs the procedure. If `NULL`, then the procedure uses the global name of the local database.
`destination_database`	The global name of the destination database. If the local database is not the destination database, then a database link from the local database to the destination database with the same name as the global name of the destination database must exist and must be accessible to the user who runs the procedure. If `NULL`, then the procedure raises an error.
`perform_actions`	If `TRUE`, then the procedure performs the necessary actions to configure the replication environment directly. If `FALSE`, then the procedure does not perform the necessary actions to configure the replication environment directly. Specify `FALSE` when this procedure is generating a script that you can edit and then run. The procedure raises an error if you specify `FALSE` and either of the following parameters is `NULL`: - `script_name` - `script_directory_object`

Table 106–1 (Cont.) Common Parameters for Configuration Procedures

Parameter	Description
script_name	If non-NULL and the perform_actions parameter is FALSE, then specify the name of the script generated by this procedure. The script contains all of the statements used to configure the replication environment. If a file with the specified script name exists in the specified directory for the script_directory_object parameter, then the procedure appends the statements to the existing file. If non-NULL and the perform_actions parameter is TRUE, then the procedure generates the specified script and performs the actions to configure the replication environment directly. If NULL and the perform_actions parameter is TRUE, then the procedure performs the actions to configure the replication environment directly and does not generate a script. If NULL and the perform_actions parameter is FALSE, then the procedure raises an error.
script_directory_object	The directory object for the directory on the local computer system into which the generated script is placed. If the script_name parameter is NULL, then the procedure ignores this parameter and does not generate a script. If NULL and the script_name parameter is non-NULL, then the procedure raises an error.
capture_name	The name of each capture process configured to capture changes. Do not specify an owner. If the bi_directional parameter is set to TRUE, then each capture process created by this procedure has the specified name. If the specified name matches the name of an existing capture process, then the procedure uses the existing capture process and adds the rules for capturing changes to the database to the positive capture process rule set. If NULL, then the system generates a name for each capture process it creates. **Note:** The capture process name cannot be altered after the capture process is created.
capture_queue_table	The name of the queue table for each queue used by a capture process, specified as [*schema_name.*]*queue_table_name*. For example, strmadmin.streams_queue_table. If the schema is not specified, then the current user is the default. If NULL, then the system generates a name for the queue table of each queue used by a capture process, and the current user is the owner of each queue table.

Table 106–1 (Cont.) Common Parameters for Configuration Procedures

Parameter	Description
capture_queue_name	The name of each queue used by a capture process, specified as [*schema_name.*]*queue_name*. For example, strmadmin.streams_queue. If the schema is not specified, then the queue table owner is the default. The queue owner automatically has privileges to perform all queue operations on the queue. If NULL, then the system generates a name for each queue used by a capture process.
capture_queue_user	The name of the user who requires ENQUEUE and DEQUEUE privileges for the queue at the source database. This user also is configured as a secure queue user of the queue. The queue user cannot grant these privileges to other users because they are not granted with the GRANT option. If NULL, then the procedure does not grant any privileges. You can also grant queue privileges to the appropriate users using the DBMS_AQADM package.
propagation_name	The name of each propagation configured to propagate changes. Do not specify an owner. If the specified name matches the name of an existing propagation, then the procedure uses the existing propagation and adds the rules for propagating changes to the positive propagation rule set. If NULL, then the system generates a name for each propagation it creates. **Note:** The propagation name cannot be altered after the propagation is created.
apply_name	The name of each apply process configured to apply changes. Do not specify an owner. If the specified name matches the name of an existing apply process, then the procedure uses the existing apply process and adds the rules for applying changes to the positive apply process rule set. The specified name must not match the name of an existing messaging client at the destination database. If NULL, then the system generates a name for each apply process it creates. **Note:** The apply process name cannot be altered after the apply process is created.
apply_queue_table	The name of the queue table for each queue used by an apply process, specified as [*schema_name.*]*queue_table_name*. For example, strmadmin.streams_queue_table. If the schema is not specified, then the current user is the default. If NULL, then the system generates a name for the queue table of each queue used by an apply process, and the current user is the owner of each queue table.

Table 106–1 (Cont.) Common Parameters for Configuration Procedures

Parameter	Description
apply_queue_name	The name of each queue used by an apply process, specified as [*schema_name*.]*queue_name*. For example, strmadmin.streams_queue. If the schema is not specified, then the queue table owner is the default. The queue owner automatically has privileges to perform all queue operations on the queue. If NULL, then the system generates a name for each queue used by an apply process.
apply_queue_user	The name of the user who requires ENQUEUE and DEQUEUE privileges for the queue at the destination database. This user also is configured as a secure queue user of the queue. The queue user cannot grant these privileges to other users because they are not granted with the GRANT option. If NULL, then the procedure does not grant any privileges. You can also grant queue privileges to the appropriate users using the DBMS_AQADM package.
bi_directional	Specify TRUE to configure bi-directional replication between the database specified in source_database and the database specified in destination_database. Both databases are configured as source and destination databases, a capture and apply process is configured to capture changes to both databases, and propagations are configured to propagate these changes. If TRUE, then a database link from the destination database to the source database with the same global name as the source database must exist. Specify FALSE to configure one way replication from the database specified in source_database and the database specified in destination_database. A capture process is configured at the current database and an apply process is configured at the destination database. A propagation is configured if necessary. **See Also:** "Streams Clients and Queues Configured By These Procedures" on page 106-14 for information about when propagations are configured
include_ddl	Specify TRUE to configure a Streams replication environment that maintains both DML and DDL changes. Specify FALSE to configure a Streams replication environment that maintains DML changes only. When this parameter is set to FALSE, DDL changes, such as ALTER TABLE, are not replicated.

Summary of DBMS_STREAMS_ADM Subprograms

Table 106–2 DBMS_STREAMS_ADM Package Subprograms

Subprogram	Description
ADD_COLUMN Procedure on page 106-25	Either adds or removes a declarative rule-based transformation which adds a column to a row logical change record (row LCR) that satisfies the specified rule
ADD_GLOBAL_PROPAGATION_RULES Procedure on page 106-28	Either adds global rules to the positive rule set for a propagation, or adds global rules to the negative rule set for a propagation, and creates the specified propagation if it does not exist
ADD_GLOBAL_RULES Procedure on page 106-33	Adds global rules to either the positive or negative rule set of a capture process, apply process, or messaging client, and creates the specified capture process, apply process, or messaging client if it does not exist
ADD_MESSAGE_PROPAGATION_RULE Procedure on page 106-38	Either adds a message rule to the positive rule set for a propagation, or adds a message rule to the negative rule set for a propagation, and creates the specified propagation if it does not exist
ADD_MESSAGE_RULE Procedure on page 106-42	Adds a message rule to either the positive or negative rule set of an apply process or messaging client, and creates the specified apply process or messaging client if it does not exist
ADD_SCHEMA_PROPAGATION_RULES Procedure on page 106-45	Either adds schema rules to the positive rule set for a propagation, or adds schema rules to the negative rule set for a propagation, and creates the specified propagation if it does not exist
ADD_SCHEMA_RULES Procedure on page 106-50	Adds schema rules to either the positive or negative rule set of a capture process, apply process, or messaging client, and creates the specified capture process, apply process, or messaging client if it does not exist
ADD_SUBSET_PROPAGATION_RULES Procedure on page 106-55	Adds subset rules to the positive rule set for a propagation, and creates the specified propagation if it does not exist
ADD_SUBSET_RULES Procedure on page 106-59	Adds subset rules to the positive rule set of a capture process, apply process, or messaging client, and creates the specified capture process, apply process, or messaging client if it does not exist
ADD_TABLE_PROPAGATION_RULES Procedure on page 106-64	Either adds table rules to the positive rule set for a propagation, or adds table rules to the negative rule set for a propagation, and creates the specified propagation if it does not exist
ADD_TABLE_RULES Procedure on page 106-69	Adds table rules to either the positive or negative rule set of a capture process, apply process, or messaging client, and creates the specified capture process, apply process, or messaging client if it does not exist

Table 106–2 (Cont.) DBMS_STREAMS_ADM Package Subprograms

Subprogram	Description
CLEANUP_INSTANTIATION_SETUP Procedure on page 106-74	Removes a Streams replication configuration that was set up by the PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures in this package
DELETE_COLUMN Procedure on page 106-78	Either adds or removes a declarative rule-based transformation which deletes a column from a row LCR that satisfies the specified rule
GET_SCN_MAPPING Procedure on page 106-80	Gets information about the system change number (SCN) values to use for Streams capture and apply processes in a Streams replication environment
MAINTAIN_GLOBAL Procedure on page 106-82	Configures a Streams environment that replicates changes at the database level between two databases
MAINTAIN_SCHEMAS Procedure on page 106-85	Configures a Streams environment that replicates changes to specified schemas between two databases
MAINTAIN_SIMPLE_TABLESPACE Procedure on page 106-89	Clones a simple tablespace from a source database at a destination database and uses Streams to maintain this tablespace at both databases. This procedure is deprecated.
MAINTAIN_SIMPLE_TTS Procedure on page 106-94	Clones a simple tablespace from a source database at a destination database and uses Streams to maintain this tablespace at both databases
MAINTAIN_TABLES Procedure on page 106-97	Configures a Streams environment that replicates changes to specified tables between two databases
MAINTAIN_TABLESPACES Procedure on page 106-101	Clones a set of tablespaces from a source database at a destination database and uses Streams to maintain these tablespaces at both databases. This procedure is deprecated.
MAINTAIN_TTS Procedure on page 106-108	Clones a set of tablespaces from a source database at a destination database and uses Streams to maintain these tablespaces at both databases
POST_INSTANTIATION_SETUP Procedure on page 106-111	Performs the actions required after instantiation to configure a Streams replication environment
PRE_INSTANTIATION_SETUP Procedure on page 106-115	Performs the actions required before instantiation to configure a Streams replication environment
PURGE_SOURCE_CATALOG Procedure on page 106-119	Removes all Streams data dictionary information at the local database for the specified object
RECOVER_OPERATION Procedure on page 106-120	Provides options for a Streams replication configuration operation that stopped because it encountered an error. This procedure either rolls forward the operation, rolls back the operation, or purges all of the metadata about the operation.

Table 106–2 (Cont.) DBMS_STREAMS_ADM Package Subprograms

Subprogram	Description
REMOVE_QUEUE Procedure on page 106-122	Removes the specified ANYDATA queue
REMOVE_RULE Procedure on page 106-123	Removes the specified rule or all rules from the rule set associated with the specified capture process, apply process, or propagation
REMOVE_STREAMS_CONFIGURATION Procedure on page 106-125	Removes the Streams configuration at the local database
RENAME_COLUMN Procedure on page 106-127	Either adds or removes a declarative rule-based transformation which renames a column in a row LCR that satisfies the specified rule
RENAME_SCHEMA Procedure on page 106-130	Either adds or removes a declarative rule-based transformation which renames a schema in a row LCR that satisfies the specified rule
RENAME_TABLE Procedure on page 106-132	Either adds or removes a declarative rule-based transformation which renames a table in a row LCR that satisfies the specified rule
SET_MESSAGE_NOTIFICATION Procedure on page 106-134	Sets a notification for messages that can be dequeued by a specified Streams messaging client from a specified queue
SET_RULE_TRANSFORM_FUNCTION Procedure on page 106-138	Sets or removes the transformation function name for a rule-based transformation
SET_UP_QUEUE Procedure on page 106-141	Creates a queue table and a queue for use with the capture, propagate, and apply functionality of Streams

Note: All subprograms commit unless specified otherwise.

ADD_COLUMN Procedure

This procedure either adds or removes a declarative rule-based transformation which adds a column to a row logical change record (row LCR) that satisfies the specified rule.

For the transformation to be performed when the specified rule evaluates to `TRUE`, the rule must be in the positive rule set of a Streams client. Streams clients include capture processes, propagations, apply processes, and messaging clients.

This procedure is overloaded. The `column_value` and `column_function` parameters are mutually exclusive.

Note:

- `ADD_COLUMN` transformations cannot add columns of the following datatypes: `BLOB`, `CLOB`, `NCLOB`, `BFILE`, `LONG`, `LONG RAW`, `ROWID`, and user-defined types (including object types, `REF`s, varrays, nested tables, and Oracle-supplied types).
- Declarative transformations can transform row LCRs only. These row LCRs can be captured row LCRs or user-enqueued row LCRs. Therefore, a DML rule must be specified when you run this procedure. If a DDL is specified, then the procedure raises an error.

See Also: *Oracle Streams Concepts and Administration* for more information about declarative rule-based transformations

Syntax

```
DBMS_STREAMS_ADM.ADD_COLUMN(
   rule_name     IN  VARCHAR2,
   table_name    IN  VARCHAR2,
   column_name   IN  VARCHAR2,
   column_value  IN  ANYDATA,
   value_type    IN  VARCHAR2     DEFAULT 'NEW',
   step_number   IN  NUMBER       DEFAULT 0,
   operation     IN  VARCHAR2     DEFAULT 'ADD');

DBMS_STREAMS_ADM.ADD_COLUMN(
   rule_name        IN  VARCHAR2,
   table_name       IN  VARCHAR2,
   column_name      IN  VARCHAR2,
   column_function  IN  VARCHAR2,
   value_type       IN  VARCHAR2     DEFAULT 'NEW',
   step_number      IN  NUMBER       DEFAULT 0,
   operation        IN  VARCHAR2     DEFAULT 'ADD');
```

Parameters

Table 106–3 ADD_COLUMN Procedure Parameters

Parameter	Description
rule_name	The name of the rule, specified as [*schema_name*.]*rule_name*. If NULL, then the procedure raises an error. For example, to specify a rule in the hr schema named employees12, enter hr.employees12. If the schema is not specified, then the current user is the default.
table_name	The name of the table to which the column is added in the row LCR, specified as [*schema_name*.]*object_name*. For example, hr.employees. If the schema is not specified, then the current user is the default.
column_name	The name of the column added to each row LCR that satisfies the rule.
column_value	The value of the added column. The column type is determined by the type specified in the ANYDATA object. This parameter cannot be specified if the column_function parameter is specified.
column_function	Either the 'SYSDATE' or the 'SYSTIMESTAMP' SQL function. The 'SYSDATE' SQL function places the current date and time set for the operating system on which the database resides. The datatype of the returned value is DATE, and the format returned depends on the value of the NLS_DATE_FORMAT initialization parameter. The 'SYSTIMESTAMP' SQL function returns the system date, including fractional seconds and time zone, of the system on which the database resides. The return type is TIMESTAMP WITH TIME ZONE. The function executes when the rule evaluates to TRUE. This parameter cannot be specified if the column_value parameter is specified.
value_type	Specify 'NEW' to add the column to the new values in the row LCR. Specify 'OLD' to add the column to the old values in the row LCR.
step_number	The order of execution of the transformation. See Also: *Oracle Streams Concepts and Administration* for more information about transformation ordering
operation	Specify 'ADD' to add the transformation to the rule. Specify 'REMOVE' to remove the transformation from the rule.

Usage Notes

When 'REMOVE' is specified for the operation parameter, all of the add column declarative rule-based transformations for the specified rule are removed that match the specified table_name, column_name, and step_number parameters. Nulls specified for these parameters act as wildcards. The following table lists the behavior of the ADD_COLUMN procedures when one or more of these parameters is NULL:

table_name	column_name	step_number	Result
NULL	NULL	NULL	Remove all add column transformations for the specified rule.
NULL	NULL	non-NULL	Remove all add column transformations with the specified step_number for the specified rule.

table_name	column_name	step_number	Result
NULL	non-NULL	non-NULL	Remove all add column transformations with the specified column_name and step_number for the specified rule.
non-NULL	NULL	non-NULL	Remove all add column transformations with the specified table_name and step_number for the specified rule.
NULL	non-NULL	NULL	Remove all add column transformations with the specified column_name for the specified rule.
non-NULL	non-NULL	NULL	Remove all add column transformations with the specified table_name and column_name for the specified rule.
non-NULL	NULL	NULL	Remove all add column transformations with the specified table_name for the specified rule.
non-NULL	non-NULL	non-NULL	Remove all add column transformations with the specified table_name, column_name, and step_number for the specified rule.

ADD_GLOBAL_PROPAGATION_RULES Procedure

This procedure either adds global rules to the positive rule set for a propagation, or adds global rules to the negative rule set for a propagation, and creates the specified propagation if it does not exist.

This procedure is overloaded. One version of this procedure contains two `OUT` parameters, and the other does not.

Syntax

```
DBMS_STREAMS_ADM.ADD_GLOBAL_PROPAGATION_RULES(
   streams_name            IN   VARCHAR2  DEFAULT NULL,
   source_queue_name       IN   VARCHAR2,
   destination_queue_name  IN   VARCHAR2,
   include_dml             IN   BOOLEAN   DEFAULT TRUE,
   include_ddl             IN   BOOLEAN   DEFAULT FALSE,
   include_tagged_lcr      IN   BOOLEAN   DEFAULT FALSE,
   source_database         IN   VARCHAR2  DEFAULT NULL,
   dml_rule_name           OUT  VARCHAR2,
   ddl_rule_name           OUT  VARCHAR2,
   inclusion_rule          IN   BOOLEAN   DEFAULT TRUE,
   and_condition           IN   VARCHAR2  DEFAULT NULL,
   queue_to_queue          IN   BOOLEAN   DEFAULT NULL);

DBMS_STREAMS_ADM.ADD_GLOBAL_PROPAGATION_RULES(
   streams_name            IN   VARCHAR2  DEFAULT NULL,
   source_queue_name       IN   VARCHAR2,
   destination_queue_name  IN   VARCHAR2,
   include_dml             IN   BOOLEAN   DEFAULT TRUE,
   include_ddl             IN   BOOLEAN   DEFAULT FALSE,
   include_tagged_lcr      IN   BOOLEAN   DEFAULT FALSE,
   source_database         IN   VARCHAR2  DEFAULT NULL,
   inclusion_rule          IN   BOOLEAN   DEFAULT TRUE,
   and_condition           IN   VARCHAR2  DEFAULT NULL,
   queue_to_queue          IN   BOOLEAN   DEFAULT NULL);
```

Parameters

Table 106–4 ADD_GLOBAL_PROPAGATION_RULES Procedure Parameters

Parameter	Description
`streams_name`	The name of the propagation. Do not specify an owner. If the specified propagation does not exist, then the procedure creates it automatically. If `NULL` and a propagation exists for the same source queue and destination queue (including database link), then the procedure uses this propagation. If `NULL` and no propagation exists for the same source queue and destination queue (including database link), then the procedure creates a propagation automatically with a system-generated name.

Table 106–4 (Cont.) ADD_GLOBAL_PROPAGATION_RULES Procedure Parameters

Parameter	Description
source_queue_name	The name of the source queue, specified as [*schema_name*.]*queue_name*. The current database must contain the source queue, and the queue must be ANYDATA type. For example, to specify a source queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter. If the schema is not specified, then the current user is the default.
destination_queue_name	The name of the destination queue, including a database link, specified as [*schema_name*.]*queue_name*[*@dblink_name*], if the destination queue is in a remote database. The queue must be ANYDATA type. For example, to specify a destination queue named streams_queue in the strmadmin schema and use a database link named dbs2.net, enter strmadmin.streams_queue@dbs2.net for this parameter. If the schema is not specified, then the current user is the default. If the database link is omitted, then the procedure uses the global name of the current database, and the source queue and destination queue must be in the same database. **Note:** Connection qualifiers are not allowed.
include_dml	If TRUE, then the procedure creates a rule for DML changes. If FALSE, then the procedure does not create a DML rule. NULL is not permitted.
include_ddl	If TRUE, then the procedure creates a rule for DDL changes. If FALSE, then the procedure does not create a DDL rule. NULL is not permitted.

Table 106–4 (Cont.) ADD_GLOBAL_PROPAGATION_RULES Procedure Parameters

Parameter	Description
include_tagged_lcr	If TRUE, then the procedure does not add a condition regarding Streams tags to the generated rules. Therefore, these rules can evaluate to TRUE regardless of whether a logical change record (LCR) has a non-NULL tag. If the rules are added to the positive rule set for the propagation, then an LCR is always considered for propagation, regardless of whether it has a non-NULL tag. If the rules are added to a positive rule set, then setting this parameter to TRUE is appropriate for a full (for example, standby) copy of a database. If the rules are added to the negative rule set for the propagation, then whether an LCR is discarded does not depend on the LCR's tag. If FALSE, then the procedure adds a condition to each generated rule that causes the rule to evaluate to TRUE only if an LCR has a NULL Streams tag. If the rules are added to the positive rule set for the propagation, then an LCR is considered for propagation only when the LCR contains a NULL tag. If the rules are added to a positive rule set, then setting this parameter to FALSE might be appropriate in update-anywhere configurations to avoid sending a change back to its source database. If the rules are added to the negative rule set for the propagation, then an LCR can be discarded only if it has a NULL tag. In most cases, specify TRUE for this parameter if the inclusion_rule parameter is set to FALSE. **See Also:** *Oracle Streams Replication Administrator's Guide* for more information about tags
source_database	The global name of the source database. The source database is where the changes originated. If NULL, then the procedure does not add a condition regarding the source database to the generated rules. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically. Oracle recommends that you specify a source database for propagation rules.
dml_rule_name	If include_dml is TRUE, then this parameter contains the DML rule name. If include_dml is FALSE, then this parameter contains a NULL.
ddl_rule_name	If include_ddl is TRUE, then this parameter contains the DDL rule name. If include_ddl is FALSE, then this parameter contains a NULL.
inclusion_rule	If inclusion_rule is TRUE, then the procedure adds the rules to the positive rule set for the propagation. If inclusion_rule is FALSE, then the procedure adds the rules to the negative rule set for the propagation. In either case, the system creates the rule set if it does not exist.

Table 106–4 (Cont.) ADD_GLOBAL_PROPAGATION_RULES Procedure Parameters

Parameter	Description
and_condition	If non-NULL, appends the specified condition to the system-generated rule condition using an AND clause in the following way: (*system_condition*) AND (*and_condition*) The variable in the specified condition must be :lcr. For example, to specify that the global rules generated by the procedure evaluate to TRUE only if the Streams tag is the hexadecimal equivalent of '02', specify the following condition: :lcr.get_tag() = HEXTORAW(''02'') The :lcr in the specified condition is converted to :dml or :ddl, depending on the rule that is being generated. If you are specifying an LCR member subprogram that is dependent on the LCR type (row or DDL), then make sure the procedure only generates the appropriate rule. Specifically, if you specify an LCR member subprogram that is valid only for row LCRs, then specify TRUE for the include_dml parameter and FALSE for the include_ddl parameter. If you specify an LCR member subprogram that is valid only for DDL LCRs, then specify FALSE for the include_dml parameter and TRUE for the include_ddl parameter. **See Also:** Chapter 188, "Logical Change Record TYPEs"
queue_to_queue	If TRUE, then the propagation is a queue to queue propagation. A queue-to-queue propagation always has its own propagation job and uses a service for automatic failover when the destination queue is a buffered queue in a Real Application Clusters (RAC) database. If FALSE or NULL, then the propagation is a queue-to-dblink propagation. A queue-to-dblink propagation can share a propagation job with other propagations that use the same database link and does not support automatic failover in a RAC environment. The procedure cannot change the queue to queue property of an exiting propagation. If the specified propagation exists, then the procedure behaves in the following way for each setting: ■ If TRUE and the specified propagation is not a queue to queue propagation, then the procedure raises an error. ■ If FALSE and the specified propagation is a queue to queue propagation, then the procedure raises an error. ■ If NULL, then the procedure does not change the queue to queue property of the propagation. **See Also:** *Oracle Streams Concepts and Administration* for more information about queue-to-queue propagations

Usage Notes

This procedure configures propagation using the current user. Only one propagation is allowed between a particular source queue and destination queue.

This procedure creates DML and DDL rules automatically based on include_dml and include_ddl parameter values, respectively. Each rule has a system-generated

rule name that consists of the database name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the database name plus the sequence number is too long, then the database name is truncated. A propagation uses the rules for filtering.

See Also:

- "Operational Notes" on page 106-7 and "Propagation Rules for LCRs" on page 106-9 for more information about the rules created by this procedure
- "Propagation User" on page 106-5

Examples

The following is an example of a global rule condition created for DML changes:

```
(:dml.is_null_tag() = 'Y' and :dml.get_source_database_name() = 'DBS1.NET' )
```

ADD_GLOBAL_RULES Procedure

This procedure adds rules to a rule set of one of the following types of Streams clients:

- Capture process rules for capturing changes to an entire database when the `streams_type` parameter is set to `capture`. See "Capture Process Rules for Changes in the Redo Log" on page 106-7 for more information about these rules.
- Apply process rules for applying all logical change records (LCRs) in a queue when the `streams_type` parameter is set to `apply`. The rules can specify that the LCRs must be from a particular source database. See "Apply Process Rules for LCRs" on page 106-9 for more information about these rules.
- Messaging client rules for dequeuing all user-enqueued LCRs from a queue when the `streams_type` parameter is set to `dequeue`. The rules can specify that the LCRs must be from a particular source database. See "Messaging Client Rules for LCRs" on page 106-11 for more information about these rules.

This procedure creates the specified capture process, apply process, or messaging client if it does not exist.

This procedure is overloaded. One version of this procedure contains two `OUT` parameters, and the other does not.

Caution: If you add global rules to the positive rule set for a capture process, then make sure you add rules to the negative capture process rule set to exclude database objects that are not support by Streams. Query the `DBA_STREAMS_UNSUPPORTED` data dictionary view to determine which database objects are not supported by Streams. If unsupported database objects are not excluded, then capture errors will result.

Note: Currently, messaging clients cannot dequeue buffered messages.

Syntax

```
DBMS_STREAMS_ADM.ADD_GLOBAL_RULES(
   streams_type        IN   VARCHAR2,
   streams_name        IN   VARCHAR2  DEFAULT NULL,
   queue_name          IN   VARCHAR2  DEFAULT 'streams_queue',
   include_dml         IN   BOOLEAN   DEFAULT TRUE,
   include_ddl         IN   BOOLEAN   DEFAULT FALSE,
   include_tagged_lcr  IN   BOOLEAN   DEFAULT FALSE,
   source_database     IN   VARCHAR2  DEFAULT NULL,
   dml_rule_name       OUT  VARCHAR2,
   ddl_rule_name       OUT  VARCHAR2,
   inclusion_rule      IN   BOOLEAN   DEFAULT TRUE,
   and_condition       IN   VARCHAR2  DEFAULT NULL);

DBMS_STREAMS_ADM.ADD_GLOBAL_RULES(
   streams_type        IN   VARCHAR2,
   streams_name        IN   VARCHAR2  DEFAULT NULL,
   queue_name          IN   VARCHAR2  DEFAULT 'streams_queue',
   include_dml         IN   BOOLEAN   DEFAULT TRUE,
   include_ddl         IN   BOOLEAN   DEFAULT FALSE,
```

```
  include_tagged_lcr  IN   BOOLEAN   DEFAULT FALSE,
  source_database     IN   VARCHAR2  DEFAULT NULL,
  inclusion_rule      IN   BOOLEAN   DEFAULT TRUE,
  and_condition       IN   VARCHAR2  DEFAULT NULL);
```

Parameters

Table 106–5 ADD_GLOBAL_RULES Procedure Parameters

Parameter	Description
streams_type	The type of Streams client: ■ Specify capture for a capture process. ■ Specify apply for an apply process. ■ Specify dequeue for a messaging client.
streams_name	The name of the capture process, apply process, or messaging client. Do not specify an owner. If the specified Streams client does not exist, then the procedure creates it automatically. If NULL, if streams_type is capture or dequeue, and if one relevant capture process or messaging client for the queue exists, then the relevant Streams client is used. If no relevant Streams client exists for the queue, then a Streams client is created automatically with a system-generated name. If NULL and multiple Streams clients of the specified streams_type for the queue exist, then the procedure raises an error. If NULL, if streams_type is apply, and if one relevant apply process exists, then the procedure uses the relevant apply process. The relevant apply process is identified in one of the following ways: ■ If one existing apply process has the source database specified in source_database and uses the queue specified in queue_name, then the procedure uses this apply process. ■ If source_database is NULL and one existing apply process is using the queue specified in queue_name, then the procedure uses this apply process. If NULL and no relevant apply process exists, then the procedure creates an apply process automatically with a system-generated name. If NULL and multiple relevant apply processes exist, then the procedure raises an error. An apply process and a messaging client cannot have the same name.
queue_name	The name of the local queue, specified as [*schema_name*.]*queue_name*. The current database must contain the queue, and the queue must be ANYDATA type. For example, to specify a queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter. If the schema is not specified, then the current user is the default. For capture process rules, this is the queue into which a capture process enqueues LCRs. For apply process rules, this is the queue from which an apply process dequeues messages. For messaging client rules, this is the queue from which a messaging client dequeues messages.

Table 106–5 (Cont.) ADD_GLOBAL_RULES Procedure Parameters

Parameter	Description
include_dml	If TRUE, then the procedure creates a rule for DML changes. If FALSE, then the procedure does not create a DML rule. NULL is not permitted.
include_ddl	If TRUE, then the procedure creates a rule for DDL changes. If FALSE, then the procedure does not create a DDL rule. NULL is not permitted.
include_tagged_lcr	If TRUE, then the procedure does not add a condition regarding Streams tags to the generated rules. Therefore, these rules can evaluate to TRUE regardless of whether a redo entry or LCR has a non-NULL tag. If the rules are added to the positive rule set for the process, then a redo entry is always considered for capture, and an LCR is always considered for apply, regardless of whether the redo entry or LCR has a non-NULL tag. If the rules are added to a positive rule set, then setting this parameter to TRUE is appropriate for a full (for example, standby) copy of a database. If the rules are added to the negative rule set for the process, then whether a redo entry or LCR is discarded does not depend on the tag. If FALSE, then the procedure adds a condition to each generated rule that causes the rule to evaluate to TRUE only if a redo entry or LCR has a NULL Streams tag. If the rules are added to the positive rule set for the process, then a redo entry is considered for capture, and an LCR is considered for apply, only when the redo entry or LCR contains a NULL tag. If the rules are added to a positive rule set, then setting this parameter to FALSE might be appropriate in update-anywhere configurations to avoid sending a change back to its source database. If the rules are added to the negative rule set for the process, then a redo entry or LCR can be discarded only if it has a NULL tag. In most cases, specify TRUE for this parameter if the inclusion_rule parameter is set to FALSE. **See Also:** *Oracle Streams Replication Administrator's Guide* for more information about tags
source_database	The global name of the source database. If NULL, then the procedure does not add a condition regarding the source database to the generated rules. For capture process rules, specify NULL or the global name of the local database if you are creating a capture process locally at the source database. If you are adding rules to a downstream capture process rule set at a downstream database, then specify the source database of the changes that will be captured. For apply process rules, specify the source database of the changes that will be applied by the apply process. The source database is the database where the changes originated. If an apply process applies captured messages, then the apply process can apply messages from only one capture process at one source database. For messaging client rules, specify NULL if you do not want the rules created by this procedure to have a condition for the source database. Specify a source database if you want the rules created by this procedure to have a condition for the source database. The source database is part of the information in an LCR, and user-constructed LCRs might or might not have this information. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically.

Table 106–5 (Cont.) ADD_GLOBAL_RULES Procedure Parameters

Parameter	Description
dml_rule_name	If include_dml is TRUE, then this parameter contains the DML rule name. If include_dml is FALSE, then this parameter contains a NULL.
ddl_rule_name	If include_ddl is TRUE, then this parameter contains the DDL rule name. If include_ddl is FALSE, then this parameter contains a NULL.
inclusion_rule	If inclusion_rule is TRUE, then the procedure adds the rules to the positive rule set for the Streams client. If inclusion_rule is FALSE, then the procedure adds the rules to the negative rule set for the Streams client. In either case, the system creates the rule set if it does not exist.
and_condition	If non-NULL, appends the specified condition to the system-generated rule condition using an AND clause in the following way: (*system_condition*) AND (*and_condition*) The variable in the specified condition must be :lcr. For example, to specify that the global rules generated by the procedure evaluate to TRUE only if the Streams tag is the hexadecimal equivalent of '02', specify the following condition: :lcr.get_tag() = HEXTORAW(''02'') The :lcr in the specified condition is converted to :dml or :ddl, depending on the rule that is being generated. If you are specifying an LCR member subprogram that is dependent on the LCR type (row or DDL), then make sure this procedure only generates the appropriate rule. Specifically, if you specify an LCR member subprogram that is valid only for row LCRs, then specify TRUE for the include_dml parameter and FALSE for the include_ddl parameter. If you specify an LCR member subprogram that is valid only for DDL LCRs, then specify FALSE for the include_dml parameter and TRUE for the include_ddl parameter. **See Also:** Chapter 188, "Logical Change Record TYPEs"

Usage Notes

This procedure creates DML and DDL rules automatically based on include_dml and include_ddl parameter values, respectively. Each rule has a system-generated rule name that consists of the database name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the database name plus the sequence number is too long, then the database name is truncated. A capture process, apply process, or messaging client uses the rules created for filtering.

See Also:

- "Operational Notes" on page 106-7
- "Security Model" on page 106-5

Examples

The following is an example of a global rule condition created for DML changes:

```
(:dml.is_null_tag() = 'Y' and :dml.get_source_database_name() = 'DBS1.NET' )
```

ADD_MESSAGE_PROPAGATION_RULE Procedure

This procedure adds a message rule to the positive rule set for a propagation, or adds a message rule to the negative rule set for a propagation, and creates the specified propagation if it does not exist.

This procedure is overloaded. One version of this procedure contains the `OUT` parameter `rule_name`, and the other does not.

Syntax

```
DBMS_STREAMS_ADM.ADD_MESSAGE_PROPAGATION_RULE(
   message_type            IN   VARCHAR2,
   rule_condition          IN   VARCHAR2,
   streams_name            IN   VARCHAR2  DEFAULT NULL,
   source_queue_name       IN   VARCHAR2,
   destination_queue_name  IN   VARCHAR2,
   inclusion_rule          IN   BOOLEAN   DEFAULT TRUE,
   rule_name               OUT  VARCHAR2,
   queue_to_queue          IN   BOOLEAN   DEFAULT NULL);

DBMS_STREAMS_ADM.ADD_MESSAGE_PROPAGATION_RULE(
   message_type            IN   VARCHAR2,
   rule_condition          IN   VARCHAR2,
   streams_name            IN   VARCHAR2  DEFAULT NULL,
   source_queue_name       IN   VARCHAR2,
   destination_queue_name  IN   VARCHAR2,
   inclusion_rule          IN   BOOLEAN   DEFAULT TRUE,
   queue_to_queue          IN   BOOLEAN   DEFAULT NULL);
```

Parameters

Table 106–6 ADD_MESSAGE_PROPAGATION_RULE Procedure Parameters

Parameter	Description
`message_type`	The type of the message. The type can be an Oracle built-in type, such as `VARCHAR2` or `NUMBER`, or it can be a user-defined type. If the type is not an Oracle built-in type, then it is specified as `[`*`schema_name`*`.]`*`type_name`*. If the schema is not specified, then the current user is the default. For example, to specify `VARCHAR2`, enter `VARCHAR2(`*`n`*`)`, where *n* is the size specification. To specify a type named `usr_msg` in the `strmadmin` schema, enter `strmadmin.usr_msg` for this parameter. The following datatypes require a size specification: `VARCHAR2`, `NVARCHAR2`, and `RAW`. **See Also:** *Oracle Database SQL Reference* for more information about datatypes
`rule_condition`	The rule condition for this message type. The rule variable name specified in the rule condition must be the following: `:msg` **See Also:** "Examples" on page 106-40

Table 106–6 (Cont.) ADD_MESSAGE_PROPAGATION_RULE Procedure Parameters

Parameter	Description
streams_name	The name of the propagation. Do not specify an owner. If the specified propagation does not exist, then the procedure creates it automatically. If NULL and a propagation exists for the same source queue and destination queue (including database link), then the procedure uses this propagation. If NULL and no propagation exists for the same source queue and destination queue (including database link), then the procedure creates a propagation automatically with a system-generated name.
source_queue_name	The name of the source queue, specified as [*schema_name.*]*queue_name*. The current database must contain the source queue, and the queue must be ANYDATA type. For example, to specify a source queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter. If the schema is not specified, then the current user is the default.
destination_queue_name	The name of the destination queue, including a database link, specified as [*schema_name.*]*queue_name*[*@dblink_name*], if the destination queue is in a remote database. The queue must be ANYDATA type. For example, to specify a destination queue named streams_queue in the strmadmin schema and use a database link named dbs2.net, enter strmadmin.streams_queue@dbs2.net for this parameter. If the schema is not specified, then the current user is the default. If the database link is omitted, then the procedure uses the global name of the current database, and the source queue and destination queue must be in the same database. **Note:** Connection qualifiers are not allowed.
inclusion_rule	If inclusion_rule is TRUE, then the procedure adds the rule to the positive rule set for the propagation. If inclusion_rule is FALSE, then the procedure adds the rule to the negative rule set for the propagation. In either case, the system creates the rule set if it does not exist.
rule_name	Contains the rule name

Table 106–6 (Cont.) ADD_MESSAGE_PROPAGATION_RULE Procedure Parameters

Parameter	Description
queue_to_queue	If TRUE, then the propagation is a queue to queue propagation. A queue-to-queue propagation always has its own propagation job and uses a service for automatic failover when the destination queue is a buffered queue in a Real Application Clusters (RAC) database. If FALSE or NULL, then the propagation is a queue-to-dblink propagation. A queue-to-dblink propagation can share a propagation job with other propagations that use the same database link and does not support automatic failover in a RAC environment. This procedure cannot change the queue to queue property of an exiting propagation. If the specified propagation exists, then the procedure behaves in the following way for each setting: ▪ If TRUE and the specified propagation is not a queue to queue propagation, then the procedure raises an error. ▪ If FALSE and the specified propagation is a queue to queue propagation, then the procedure raises an error. ▪ If NULL, then the procedure does not change the queue to queue property of the propagation. **See Also:** *Oracle Streams Concepts and Administration* for more information about queue-to-queue propagations

Usage Notes

This procedure configures propagation using the current user. Only one propagation is allowed between a particular source queue and destination queue.

When you use this procedure to create a rule set for a message rule, the new rule set does not have an evaluation context. If no evaluation context exists for the specified message type, then this procedure creates a new evaluation context and associates it with the new rule. The evaluation context also has a system-generated name. If you create new rules that use an existing message type, then the new rules use the existing evaluation context for the message type.

See Also:

- "Operational Notes" on page 106-7 and "Propagation Rules for User Messages" on page 106-9 for more information about the rules created by this procedure
- "Propagation User" on page 106-5

Examples

Suppose the message type is VARCHAR2(128). Given this type, the following rule condition can be specified:

```
':msg = ''HQ'''
```

This rule condition evaluates to TRUE if a user-enqueued message of type VARCHAR2(128) has HQ for its value.

Suppose the message type is usr_msg, and that this type has the following attributes: source_dbname, owner, name, and message. Given this type, the following rule condition can be specified:

```
':msg.source_dbname = ''DBS1.NET'' AND ' || ':msg.owner = ''HR'' AND  ' ||
':msg.name = ''EMPLOYEES'''
```

This rule condition evaluates to `TRUE` if a user-enqueued message of type `usr_msg` has `DBS1.NET` for its `source_dbname` attribute, `HR` for its `owner` attribute, and `EMPLOYEES` for its `name` attribute.

Note: The quotation marks in the preceding examples are all single quotation marks.

ADD_MESSAGE_RULE Procedure

This procedure adds a message rule to a rule set of one of the following types of Streams clients:

- Apply process rule for dequeuing user-enqueued messages of a specific message type from a queue when the `streams_type` parameter is set to `apply`. See "Apply Process Rules for User Messages" on page 106-10 for more information about such rules.
- Messaging client rule dequeuing user-enqueued messages of a specific message type from a queue when the `streams_type` parameter is set to `dequeue`. See "Messaging Client Rules for User Messages" on page 106-11 for more information about such rules.

This procedure also creates the specified Streams client if it does not exist.

This procedure is overloaded. One version of this procedure contains the `OUT` parameter `rule_name`, and the other does not.

Note: Currently, messaging clients cannot dequeue buffered messages.

Syntax

```
DBMS_STREAMS_ADM.ADD_MESSAGE_RULE(
   message_type    IN   VARCHAR2,
   rule_condition  IN   VARCHAR2,
   streams_type    IN   VARCHAR2,
   streams_name    IN   VARCHAR2  DEFAULT NULL,
   queue_name      IN   VARCHAR2  DEFAULT 'streams_queue',
   inclusion_rule  IN   BOOLEAN   DEFAULT TRUE,
   rule_name       OUT  VARCHAR2);

DBMS_STREAMS_ADM.ADD_MESSAGE_RULE(
   message_type    IN   VARCHAR2,
   rule_condition  IN   VARCHAR2,
   streams_type    IN   VARCHAR2,
   streams_name    IN   VARCHAR2  DEFAULT NULL,
   queue_name      IN   VARCHAR2  DEFAULT 'streams_queue',
   inclusion_rule  IN   BOOLEAN   DEFAULT TRUE);
```

Parameters

Table 106–7 ADD_MESSAGE_RULE Procedure Parameters

Parameter	Description
message_type	The type of the message. The type can be an Oracle built-in type, such as VARCHAR2 or NUMBER, or it can be a user-defined type. If the type is not an Oracle built-in type, then it is specified as [*schema_name*.]*type_name*. If the schema is not specified, then the current user is the default. For example, to specify VARCHAR2, enter VARCHAR2(*n*), where *n* is the size specification. To specify a type named usr_msg in the strmadmin schema, enter strmadmin.usr_msg for this parameter. The following datatypes require a size specification: VARCHAR2, NVARCHAR2, and RAW. **See Also:** *Oracle Database SQL Reference* for more information about datatypes
rule_condition	The rule condition for this message type. The rule variable name specified in the rule condition must be the following: :msg **See Also:** "Examples" on page 106-44
streams_type	The type of message consumer, either apply for apply process or dequeue for messaging client
streams_name	The name of the Streams apply process or messaging client. If the specified streams_type is apply, then specify the name of the apply process. Do not specify an owner. If the specified apply process does not exist, then the procedure creates it automatically with a system-generated name. If the specified streams_type is dequeue, then specify the messaging client. For example, if the user strmadmin is the messaging client, then specify strmadmin. If NULL and a relevant apply process or messaging client for the queue exists, then the procedure uses the relevant apply process or messaging client. If NULL and multiple relevant apply processes or messaging clients for the queue exist, then the procedure raises an error. If NULL and no Streams client of the specified streams_type exists for the queue, then the procedure creates an apply process or messaging client automatically with a system-generated name. An apply process and a messaging client cannot have the same name.
queue_name	The name of the local queue from which messages will be dequeued, specified as [*schema_name*.]*queue_name*. The current database must contain the queue, and the queue must be ANYDATA type. For example, to specify a queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter. If the schema is not specified, then the current user is the default.
inclusion_rule	If inclusion_rule is TRUE, then the procedure adds the rule to the positive rule set for the apply process or messaging client. If inclusion_rule is FALSE, then the procedure adds the rule to the negative rule set for the apply process or messaging client. In either case, the system creates the rule set if it does not exist.

Table 106–7 (Cont.) ADD_MESSAGE_RULE Procedure Parameters

Parameter	Description
rule_name	Contains the rule name

Usage Notes

If an apply process rule is added, then this procedure creates the apply process if it does not exist. An apply process created by this procedure can apply only user-enqueued messages, and dequeued messages are sent to the message handler for the apply process. If a messaging client rule is added, then this procedure creates a messaging client if it does not exist.

When you use this procedure to create a rule set for a message rule, the new rule set does not have an evaluation context. If no evaluation context exists for the specified message type, then this procedure creates a new evaluation context and associates it with the new rule. The evaluation context also has a system-generated name. If you create new rules that use an existing message type, then the new rules use the existing evaluation context for the message type.

See Also:

- "Operational Notes" on page 106-7
- "Security Model" on page 106-5
- ALTER_APPLY Procedure on page 15-4 for more information about setting a message handler for an apply process

Examples

Suppose the message type is VARCHAR2(128). Given this type, the following rule condition can be specified:

```
':msg = ''HQ'''
```

This rule condition evaluates to TRUE if a user-enqueued message of type VARCHAR2(128) has HQ for its value.

Suppose the message type is usr_msg, and that this type has the following attributes: source_dbname, owner, name, and message. Given this type, the following rule condition can be specified:

```
':msg.source_dbname = ''DBS1.NET'' AND ' || ':msg.owner = ''HR'' AND  ' ||
':msg.name = ''EMPLOYEES'''
```

This rule condition evaluates to TRUE if a user-enqueued message of type usr_msg has DBS1.NET for its source_dbname attribute, HR for its owner attribute, and EMPLOYEES for its name attribute.

Note: The quotation marks in the preceding examples are all single quotation marks.

ADD_SCHEMA_PROPAGATION_RULES Procedure

This procedure either adds schema rules to the positive rule set for a propagation, or adds schema rules to the negative rule set for a propagation, and creates the specified propagation if it does not exist.

This procedure is overloaded. One version of this procedure contains two `OUT` parameters, and the other does not.

Syntax

```
DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES(
   schema_name             IN   VARCHAR2,
   streams_name            IN   VARCHAR2  DEFAULT NULL,
   source_queue_name       IN   VARCHAR2,
   destination_queue_name  IN   VARCHAR2,
   include_dml             IN   BOOLEAN   DEFAULT TRUE,
   include_ddl             IN   BOOLEAN   DEFAULT FALSE,
   include_tagged_lcr      IN   BOOLEAN   DEFAULT FALSE,
   source_database         IN   VARCHAR2  DEFAULT NULL,
   dml_rule_name           OUT  VARCHAR2,
   ddl_rule_name           OUT  VARCHAR2,
   inclusion_rule          IN   BOOLEAN   DEFAULT TRUE,
   and_condition           IN   VARCHAR2  DEFAULT NULL,
   queue_to_queue          IN   BOOLEAN   DEFAULT NULL);

DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES(
   schema_name             IN   VARCHAR2,
   streams_name            IN   VARCHAR2  DEFAULT NULL,
   source_queue_name       IN   VARCHAR2,
   destination_queue_name  IN   VARCHAR2,
   include_dml             IN   BOOLEAN   DEFAULT TRUE,
   include_ddl             IN   BOOLEAN   DEFAULT FALSE,
   include_tagged_lcr      IN   BOOLEAN   DEFAULT FALSE,
   source_database         IN   VARCHAR2  DEFAULT NULL,
   inclusion_rule          IN   BOOLEAN   DEFAULT TRUE,
   and_condition           IN   VARCHAR2  DEFAULT NULL,
   queue_to_queue          IN   BOOLEAN   DEFAULT NULL);
```

Parameters

Table 106–8 ADD_SCHEMA_PROPAGATION_RULES Procedure Parameters

Parameter	Description
`schema_name`	The name of the schema. For example, `hr`.
`streams_name`	The name of the propagation. Do not specify an owner. If the specified propagation does not exist, then the procedure creates it automatically. If `NULL` and a propagation exists for the same source queue and destination queue (including database link), then the procedure uses this propagation. If `NULL` and no propagation exists for the same source queue and destination queue (including database link), then the procedure creates a propagation automatically with a system-generated name.

Table 106–8 (Cont.) ADD_SCHEMA_PROPAGATION_RULES Procedure Parameters

Parameter	Description
source_queue_name	The name of the source queue, specified as [*schema_name.*]*queue_name*. The current database must contain the source queue, and the queue must be ANYDATA type. For example, to specify a source queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter. If the schema is not specified, then the current user is the default.
destination_queue_name	The name of the destination queue, including a database link, specified as [*schema_name.*]*queue_name*[*@dblink_name*], if the destination queue is in a remote database. The queue must be ANYDATA type. For example, to specify a destination queue named streams_queue in the strmadmin schema and use a database link named dbs2.net, enter strmadmin.streams_queue@dbs2.net for this parameter. If the schema is not specified, then the current user is the default. If the database link is omitted, then the procedure uses the global name of the current database, and the source queue and destination queue must be in the same database. **Note:** Connection qualifiers are not allowed.
include_dml	If TRUE, then the procedure creates a rule for DML changes. If FALSE, then the procedure does not create a DML rule. NULL is not permitted.
include_ddl	If TRUE, then the procedure creates a rule for DDL changes. If FALSE, then the procedure does not create a DDL rule. NULL is not permitted.

Table 106–8 (Cont.) ADD_SCHEMA_PROPAGATION_RULES Procedure Parameters

Parameter	Description
include_tagged_lcr	If TRUE, then the procedure does not add a condition regarding Streams tags to the generated rules. Therefore, these rules can evaluate to TRUE regardless of whether a logical change record (LCR) has a non-NULL tag. If the rules are added to the positive rule set for the propagation, then an LCR is always considered for propagation, regardless of whether it has a non-NULL tag. If the rules are added to a positive rule set, then setting this parameter to TRUE is appropriate for a full (for example, standby) copy of a database. If the rules are added to the negative rule set for the propagation, then whether an LCR is discarded does not depend on the LCR's tag. If FALSE, then the procedure adds a condition to each generated rule that causes the rule to evaluate to TRUE only if an LCR has a NULL Streams tag. If the rules are added to the positive rule set for the propagation, then an LCR is considered for propagation only when the LCR contains a NULL tag. If the rules are added to a positive rule set, then setting this parameter to FALSE might be appropriate in update-anywhere configurations to avoid sending a change back to its source database. If the rules are added to the negative rule set for the propagation, then an LCR can be discarded only if it has a NULL tag. In most cases, specify TRUE for this parameter if the inclusion_rule parameter is set to FALSE. **See Also:** *Oracle Streams Replication Administrator's Guide* for more information about tags
source_database	The global name of the source database. The source database is where the change originated. If NULL, then the procedure does not add a condition regarding the source database to the generated rules. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically. Oracle recommends that you specify a source database for propagation rules.
dml_rule_name	If include_dml is TRUE, then this parameter contains the DML rule name. If include_dml is FALSE, then this parameter contains a NULL.
ddl_rule_name	If include_ddl is TRUE, then this parameter contains the DDL rule name. If include_ddl is FALSE, then this parameter contains a NULL.
inclusion_rule	If inclusion_rule is TRUE, then the procedure adds the rules to the positive rule set for the propagation. If inclusion_rule is FALSE, then the procedure adds the rules to the negative rule set for the propagation. In either case, the system creates the rule set if it does not exist.

Table 106–8 (Cont.) ADD_SCHEMA_PROPAGATION_RULES Procedure Parameters

Parameter	Description
and_condition	If non-NULL, appends the specified condition to the system-generated rule condition using an AND clause in the following way: (*system_condition*) AND (*and_condition*) The variable in the specified condition must be :lcr. For example, to specify that the schema rules generated by the procedure evaluate to TRUE only if the Streams tag is the hexadecimal equivalent of '02', specify the following condition: :lcr.get_tag() = HEXTORAW(''02'') The :lcr in the specified condition is converted to :dml or :ddl, depending on the rule that is being generated. If you are specifying an LCR member subprogram that is dependent on the LCR type (row or DDL), then make sure this procedure only generates the appropriate rule. Specifically, if you specify an LCR member subprogram that is valid only for row LCRs, then specify TRUE for the include_dml parameter and FALSE for the include_ddl parameter. If you specify an LCR member subprogram that is valid only for DDL LCRs, then specify FALSE for the include_dml parameter and TRUE for the include_ddl parameter. **See Also:** Chapter 188, "Logical Change Record TYPEs"
queue_to_queue	If TRUE, then the propagation is a queue to queue propagation. A queue-to-queue propagation always has its own propagation job and uses a service for automatic failover when the destination queue is a buffered queue in a Real Application Clusters (RAC) database. If FALSE or NULL, then the propagation is a queue-to-dblink propagation. A queue-to-dblink propagation can share a propagation job with other propagations that use the same database link and does not support automatic failover in a RAC environment. This procedure cannot change the queue to queue property of an exiting propagation. If the specified propagation exists, then the procedure behaves in the following way for each setting: ■ If TRUE and the specified propagation is not a queue to queue propagation, then the procedure raises an error. ■ If FALSE and the specified propagation is a queue to queue propagation, then the procedure raises an error. ■ If NULL, then the procedure does not change the queue to queue property of the propagation. **See Also:** *Oracle Streams Concepts and Administration* for more information about queue-to-queue propagations

Usage Notes

This procedure configures propagation using the current user. Only one propagation is allowed between a particular source queue and destination queue.

This procedure creates DML and DDL rules automatically based on include_dml and include_ddl parameter values, respectively. Each rule has a system-generated

rule name that consists of the schema name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the schema name plus the sequence number is too long, then the schema name is truncated. A propagation uses the rules created for filtering.

See Also:

- "Operational Notes" on page 106-7 and "Propagation Rules for LCRs" on page 106-9 for more information about the rules created by this procedure
- "Propagation User" on page 106-5

Examples

The following is an example of a schema rule condition created for DML changes:

```
((:dml.get_object_owner() = 'HR') and :dml.is_null_tag() = 'Y'
and :dml.get_source_database_name() = 'DBS1.NET' )
```

ADD_SCHEMA_RULES Procedure

This procedures adds rules to a rule set of one of the following types of Streams clients:

- Capture process rules for capturing changes to a specified schema when the `streams_type` parameter is set to `capture`. See "Capture Process Rules for Changes in the Redo Log" on page 106-7 for more information about these rules.
- Apply process rules for applying logical change records (LCRs) in a queue that contain changes to a specified schema when the `streams_type` parameter is set to `apply`. The rules can specify that the LCRs must be from a particular source database. See "Apply Process Rules for LCRs" on page 106-9 for more information about these rules.
- Messaging client rules for dequeuing user-enqueued LCRs from a queue that contain changes to a specified schema when the `streams_type` parameter is set to `dequeue`. The rules can specify that the LCRs must be from a particular source database. See "Messaging Client Rules for LCRs" on page 106-11 for more information about these rules.

This procedure creates the specified capture process, apply process, or messaging client if it does not exist.

This procedure is overloaded. One version of this procedure contains two `OUT` parameters, and the other does not.

Caution: If you add schema rules to the positive rule set for a capture process, then make sure you add rules to the negative capture process rule set to exclude database objects in the schema that are not support by Streams. Query the `DBA_STREAMS_UNSUPPORTED` data dictionary view to determine which database objects are not supported by Streams. If unsupported database objects are not excluded, then capture errors will result.

Note: Currently, messaging clients cannot dequeue buffered messages.

Syntax

```
DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
   schema_name          IN   VARCHAR2,
   streams_type         IN   VARCHAR2,
   streams_name         IN   VARCHAR2  DEFAULT NULL,
   queue_name           IN   VARCHAR2  DEFAULT 'streams_queue',
   include_dml          IN   BOOLEAN   DEFAULT TRUE,
   include_ddl          IN   BOOLEAN   DEFAULT FALSE,
   include_tagged_lcr   IN   BOOLEAN   DEFAULT FALSE,
   source_database      IN   VARCHAR2  DEFAULT NULL,
   dml_rule_name        OUT  VARCHAR2,
   ddl_rule_name        OUT  VARCHAR2,
   inclusion_rule       IN   BOOLEAN   DEFAULT TRUE,
   and_condition        IN   VARCHAR2  DEFAULT NULL);

DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
   schema_name          IN   VARCHAR2,
```

```
  streams_type        IN   VARCHAR2,
  streams_name        IN   VARCHAR2  DEFAULT NULL,
  queue_name          IN   VARCHAR2  DEFAULT 'streams_queue',
  include_dml         IN   BOOLEAN   DEFAULT TRUE,
  include_ddl         IN   BOOLEAN   DEFAULT FALSE,
  include_tagged_lcr  IN   BOOLEAN   DEFAULT FALSE,
  source_database     IN   VARCHAR2  DEFAULT NULL,
  inclusion_rule      IN   BOOLEAN   DEFAULT TRUE,
  and_condition       IN   VARCHAR2  DEFAULT NULL);
```

Parameters

Table 106–9 ADD_SCHEMA_RULES Procedure Parameters

Parameter	Description
schema_name	The name of the schema. For example, hr. You can specify a schema that does not yet exist, because Streams does not validate the existence of the schema.
streams_type	The type of Streams client: ■ Specify capture for a capture process. ■ Specify apply for an apply process. ■ Specify dequeue for a messaging client.
streams_name	The name of the capture process, apply process, or messaging client. Do not specify an owner. If the specified Streams client does not exist, then the procedure creates it automatically. If NULL, if streams_type is capture or dequeue, and if one relevant capture process or messaging client for the queue exists, then the procedure uses the relevant Streams client. If no relevant Streams client exists for the queue, then the procedure creates a Streams client automatically with a system-generated name. If NULL and multiple Streams clients of the specified streams_type for the queue exist, then the procedure raises an error. If NULL, if streams_type is apply, and if one relevant apply process exists, then the procedure uses the relevant apply process. The relevant apply process is identified in one of the following ways: ■ If one existing apply process has the source database specified in source_database and uses the queue specified in queue_name, then the procedure uses this apply process. ■ If source_database is NULL and one existing apply process is using the queue specified in queue_name, then the procedure uses this apply process. If NULL and no relevant apply process exists, then the procedure creates an apply process automatically with a system-generated name. If NULL and multiple relevant apply processes exist, then the procedure raises an error. An apply process and a messaging client cannot have the same name.

Table 106–9 (Cont.) ADD_SCHEMA_RULES Procedure Parameters

Parameter	Description
queue_name	The name of the local queue, specified as [*schema_name.*]*queue_name*. The current database must contain the queue, and the queue must be ANYDATA type. For example, to specify a queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter. If the schema is not specified, then the current user is the default. For capture process rules, this is the queue into which a capture process enqueues LCRs. For apply process rules, this is the queue from which an apply process dequeues messages. For messaging client rules, this is the queue from which a messaging client dequeues messages.
include_dml	If TRUE, then the procedure creates a rule for DML changes. If FALSE, then the procedure does not create a DML rule. NULL is not permitted.
include_ddl	If TRUE, then the procedure creates a rule for DDL changes. If FALSE, then the procedure does not create a DDL rule. NULL is not permitted.
include_tagged_lcr	If TRUE, then the procedure does not add a condition regarding Streams tags to the generated rules. Therefore, these rules can evaluate to TRUE regardless of whether a redo entry or LCR has a non-NULL tag. If the rules are added to the positive rule set for the process, then a redo entry is always considered for capture, and an LCR is always considered for apply, regardless of whether the redo entry or LCR has a non-NULL tag. If the rules are added to a positive rule set, then setting this parameter to TRUE is appropriate for a full (for example, standby) copy of a database. If the rules are added to the negative rule set for the process, then whether a redo entry or LCR is discarded does not depend on the tag. If FALSE, then the procedure adds a condition to each generated rule that causes the rule to evaluate to TRUE only if a redo entry or LCR has a NULL Streams tag. If the rules are added to the positive rule set for the process, then a redo entry is considered for capture, and an LCR is considered for apply, only when the redo entry or LCR contains a NULL tag. If the rules are added to a positive rule set, then setting this parameter to FALSE might be appropriate in update-anywhere configurations to avoid sending a change back to its source database. If the rules are added to the negative rule set for the process, then a redo entry or LCR can be discarded only if it has a NULL tag. In most cases, specify TRUE for this parameter if the inclusion_rule parameter is set to FALSE. **See Also:** *Oracle Streams Replication Administrator's Guide* for more information about tags

Table 106–9 (Cont.) ADD_SCHEMA_RULES Procedure Parameters

Parameter	Description
source_database	The global name of the source database. If NULL, then the procedure does not add a condition regarding the source database to the generated rules. For capture process rules, specify NULL or the global name of the local database if you are creating a capture process locally at the source database. If you are adding rules to a downstream capture process rule set at a downstream database, then specify the source database of the changes that will be captured. For apply process rules, specify the source database of the changes that will be applied by the apply process. The source database is the database where the changes originated. If an apply process applies captured messages, then the apply process can apply messages from only one capture process at one source database. For messaging client rules, specify NULL if you do not want the rules created by this procedure to have a condition for the source database. Specify a source database if you want the rules created by this procedure to have a condition for the source database. The source database is part of the information in an LCR, and user-constructed LCRs might or might not have this information. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically.
dml_rule_name	If include_dml is TRUE, then this parameter contains the DML rule name. If include_dml is FALSE, then this parameter contains a NULL.
ddl_rule_name	If include_ddl is TRUE, then this parameter contains the DDL rule name. If include_ddl is FALSE, then this parameter contains a NULL.
inclusion_rule	If inclusion_rule is TRUE, then the procedure adds the rules to the positive rule set for the Streams client. If inclusion_rule is FALSE, then the procedure adds the rules to the negative rule set for the Streams client. In either case, the system creates the rule set if it does not exist.

Table 106–9 (Cont.) ADD_SCHEMA_RULES Procedure Parameters

Parameter	Description
and_condition	If non-NULL, appends the specified condition to the system-generated rule condition using an AND clause in the following way: (*system_condition*) AND (*and_condition*) The variable in the specified condition must be :lcr. For example, to specify that the schema rules generated by the procedure evaluate to TRUE only if the Streams tag is the hexadecimal equivalent of '02', specify the following condition: :lcr.get_tag() = HEXTORAW(''02'') The :lcr in the specified condition is converted to :dml or :ddl, depending on the rule that is being generated. If you are specifying an LCR member subprogram that is dependent on the LCR type (row or DDL), then make sure this procedure only generates the appropriate rule. Specifically, if you specify an LCR member subprogram that is valid only for row LCRs, then specify TRUE for the include_dml parameter and FALSE for the include_ddl parameter. If you specify an LCR member subprogram that is valid only for DDL LCRs, then specify FALSE for the include_dml parameter and TRUE for the include_ddl parameter. **See Also:** Chapter 188, "Logical Change Record TYPEs"

Usage Notes

This procedure creates DML and DDL rules automatically based on include_dml and include_ddl parameter values, respectively. Each rule has a system-generated rule name that consists of the schema name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the schema name plus the sequence number is too long, then the schema name is truncated. A capture process, apply process, or messaging client uses the rules created for filtering.

See Also:

- "Operational Notes" on page 106-7
- "Security Model" on page 106-5

Examples

The following is an example of a schema rule condition created for DML changes:

```
((:dml.get_object_owner() = 'HR') and :dml.is_null_tag() = 'Y'
and :dml.get_source_database_name() = 'DBS1.NET' )
```

ADD_SUBSET_PROPAGATION_RULES Procedure

This procedures adds propagation rules that propagate the logical change records (LCRs) related to a subset of the rows in the specified table in a source queue to a destination queue, and creates the specified propagation if it does not exist.

This procedure is overloaded. One version of this procedure contains three `OUT` parameters, and the other does not.

Syntax

```
DBMS_STREAMS_ADM.ADD_SUBSET_PROPAGATION_RULES(
   table_name                IN   VARCHAR2,
   dml_condition             IN   VARCHAR2,
   streams_name              IN   VARCHAR2  DEFAULT NULL,
   source_queue_name         IN   VARCHAR2,
   destination_queue_name    IN   VARCHAR2,
   include_tagged_lcr        IN   BOOLEAN   DEFAULT FALSE,
   source_database           IN   VARCHAR2  DEFAULT NULL,
   insert_rule_name          OUT  VARCHAR2,
   update_rule_name          OUT  VARCHAR2,
   delete_rule_name          OUT  VARCHAR2,
   queue_to_queue            IN   BOOLEAN   DEFAULT NULL);

DBMS_STREAMS_ADM.ADD_SUBSET_PROPAGATION_RULES(
   table_name                IN   VARCHAR2,
   dml_condition             IN   VARCHAR2,
   streams_name              IN   VARCHAR2  DEFAULT NULL,
   source_queue_name         IN   VARCHAR2,
   destination_queue_name    IN   VARCHAR2,
   include_tagged_lcr        IN   BOOLEAN   DEFAULT FALSE,
   source_database           IN   VARCHAR2  DEFAULT NULL,
   queue_to_queue            IN   BOOLEAN   DEFAULT NULL);
```

Parameters

Table 106–10 ADD_SUBSET_PROPAGATION_RULES Procedure Parameters

Parameter	Description
`table_name`	The name of the table specified as `[`*`schema_name`*`.]`*`object_name`*. For example, `hr.employees`. If the schema is not specified, then the current user is the default. The specified table must exist in the same database as the propagation. Also, the specified table cannot have any LOB, `LONG`, or `LONG RAW` columns currently or in the future.
`dml_condition`	The subset condition. Specify this condition similar to the way you specify conditions in a `WHERE` clause in SQL. For example, to specify rows in the `hr.employees` table where the `salary` is greater than `4000` and the `job_id` is `SA_MAN`, enter the following as the condition: `' salary > 4000 and job_id = ''SA_MAN'' '` **Note:** The quotation marks in the preceding example are all single quotation marks.

Table 106–10 (Cont.) ADD_SUBSET_PROPAGATION_RULES Procedure Parameters

Parameter	Description
streams_name	The name of the propagation. Do not specify an owner. If the specified propagation does not exist, then the procedure creates it automatically. If NULL and a propagation exists for the same source queue and destination queue (including database link), then the procedure uses this propagation. If NULL and no propagation exists for the same source queue and destination queue (including database link), then the procedure creates a propagation automatically with a system-generated name.
source_queue_name	The name of the source queue, specified as [*schema_name*.]*queue_name*. The current database must contain the source queue, and the queue must be ANYDATA type. For example, to specify a source queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter. If the schema is not specified, then the current user is the default.
destination_queue_name	The name of the destination queue, including a database link, specified as [*schema_name*.]*queue_name*[*@dblink_name*], if the destination queue is in a remote database. The queue must be ANYDATA type. For example, to specify a destination queue named streams_queue in the strmadmin schema and use a database link named dbs2.net, enter strmadmin.streams_queue@dbs2.net for this parameter. If the schema is not specified, then the current user is the default. If the database link is omitted, then the procedure uses the global name of the current database, and the source queue and destination queue must be in the same database. **Note:** Connection qualifiers are not allowed.
include_tagged_lcr	If TRUE, then an LCR is always considered for propagation, regardless of whether it has a non-NULL tag. This setting is appropriate for a full (for example, standby) copy of a database. If FALSE, then an LCR is considered for propagation only when the LCR contains a NULL tag. A setting of FALSE is often specified in update-anywhere configurations to avoid sending a change back to its source database. **See Also:** *Oracle Streams Replication Administrator's Guide* for more information about tags
source_database	The global name of the source database. The source database is where the change originated. If NULL, then the procedure does not add a condition regarding the source database to the generated rules. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically. Oracle recommends that you specify a source database for propagation rules.

Table 106–10 (Cont.) ADD_SUBSET_PROPAGATION_RULES Procedure Parameters

Parameter	Description
insert_rule_name	Contains the system-generated INSERT rule name. This rule handles inserts, as well as updates that must be converted into inserts.
update_rule_name	Contains the system-generated UPDATE rule name. This rule handles updates that remain updates.
delete_rule_name	Contains the system-generated DELETE rule name. This rule handles deletes, as well as updates that must be converted into deletes
queue_to_queue	If TRUE, then the propagation is a queue to queue propagation. A queue-to-queue propagation always has its own propagation job and uses a service for automatic failover when the destination queue is a buffered queue in a Real Application Clusters (RAC) database. If FALSE or NULL, then the propagation is a queue-to-dblink propagation. A queue-to-dblink propagation can share a propagation job with other propagations that use the same database link and does not support automatic failover in a RAC environment. This procedure cannot change the queue to queue property of an exiting propagation. If the specified propagation exists, then the procedure behaves in the following way for each setting: ■ If TRUE and the specified propagation is not a queue to queue propagation, then the procedure raises an error. ■ If FALSE and the specified propagation is a queue to queue propagation, then the procedure raises an error. ■ If NULL, then the procedure does not change the queue to queue property of the propagation. **See Also:** *Oracle Streams Concepts and Administration* for more information about queue-to-queue propagations

Usage Notes

This procedure configures propagation using the current user. Only one propagation is allowed between a particular source queue and destination queue.

Running this procedure generates three rules for the specified propagation: one for INSERT statements, one for UPDATE statements, and one for DELETE statements. For INSERT and DELETE statements, only row LCRs that satisfy the condition specified for the dml_condition parameter are propagated. For UPDATE statements, the following variations are possible:

- If both the new and old values in a row LCR satisfy the specified dml_condition, then the row LCR is propagated without any changes.
- If neither the new or old values in a row LCR satisfy the specified dml_condition, then the row LCR is not propagated.
- If the old values for a row LCR satisfy the specified dml_condition, but the new values do not, then the update row LCR is converted into a delete row LCR.
- If the new values for a row LCR satisfy the specified dml_condition, but the old values do not, then the update row LCR is converted to an insert row LCR.

When an update is converted into an insert or a delete, it is called row migration.

A propagation uses the rules created for filtering. If the propagation does not have a positive rule set, then the procedure creates a positive rule set automatically, and the rules for propagating changes to the table are added to the positive rule set. A subset rule can be added to positive rule set only, not to a negative rule set. Other rules in an existing positive rule set for the propagation are not affected. Additional rules can be added using either the `DBMS_STREAMS_ADM` package or the `DBMS_RULE_ADM` package.

Rules for `INSERT`, `UPDATE`, and `DELETE` statements are created automatically when you run this procedure, and these rules are given a system-generated rule name. Each rule has a system-generated rule name that consists of the table name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the table name plus the sequence number is too long, then the table name is truncated. The `ADD_SUBSET_RULES` procedure is overloaded, and the system-generated rule names for `INSERT`, `UPDATE`, and `DELETE` statements are returned.

When you create propagation subset rules for a table, you should create an unconditional supplemental log group at the source database with all the columns in the table. Supplemental logging is required if an update must be converted to an insert. The propagation rule must have all the column values to be able to perform this conversion correctly.

Attention: Subset rules should only reside in positive rule sets. You should not add subset rules to negative rule sets. Doing so might have unpredictable results because row migration would not be performed on LCRs that are not discarded by the negative rule set.

See Also:

- "Operational Notes" on page 106-7 and "Propagation Rules for LCRs" on page 106-9 for more information about the rules created by this procedure
- "Propagation User" on page 106-5

Examples

The following is an example of a rule condition created for filtering a row LCR containing an update operation when the `dml_condition` is `region_id = 2`, the `table_name` is `hr.regions`, and the `source_database` is `dbs1.net`:

```
:dml.get_object_owner()='HR' AND :dml.get_object_name()='REGIONS'
AND :dml.is_null_tag()='Y' AND :dml.get_source_database_name()='DBS1.NET'
AND :dml.get_command_type()='UPDATE'
AND (:dml.get_value('NEW','"REGION_ID"') IS NOT NULL)
AND (:dml.get_value('OLD','"REGION_ID"') IS NOT NULL)
AND (:dml.get_value('OLD','"REGION_ID"').AccessNumber()=2)
AND (:dml.get_value('NEW','"REGION_ID"').AccessNumber()=2)
```

ADD_SUBSET_RULES Procedure

This procedure adds rules to a rule set of one of the following types of Streams clients:

- Capture process rules for capturing changes to a subset of rows in a specified table when the `streams_type` parameter is set to `capture`. See "Capture Process Rules for Changes in the Redo Log" on page 106-7 for more information about these rules.
- Apply process rules for applying logical change records (LCRs) in a queue that contain changes to a subset of rows in a specified table when the `streams_type` parameter is set to `apply`. The rules can specify that the LCRs must be from a particular source database. See "Apply Process Rules for LCRs" on page 106-9 for more information about these rules.
- Messaging client rules for dequeuing user-enqueued LCRs from a queue that contain changes to a subset of rows in a specified table when the `streams_type` parameter is set to `dequeue`. The rules can specify that the LCRs must be from a particular source database. See "Messaging Client Rules for LCRs" on page 106-11 for more information about these rules.

This procedure creates the specified capture process, apply process, or messaging client if it does not exist.

This procedure is overloaded. One version of this procedure contains three `OUT` parameters, and the other does not.

Note: Currently, messaging clients cannot dequeue buffered messages.

Syntax

```
DBMS_STREAMS_ADM.ADD_SUBSET_RULES(
   table_name          IN   VARCHAR2,
   dml_condition       IN   VARCHAR2,
   streams_type        IN   VARCHAR2 DEFAULT 'apply',
   streams_name        IN   VARCHAR2 DEFAULT NULL,
   queue_name          IN   VARCHAR2 DEFAULT 'streams_queue',
   include_tagged_lcr  IN   BOOLEAN  DEFAULT FALSE,
   source_database     IN   VARCHAR2 DEFAULT NULL,
   insert_rule_name    OUT  VARCHAR2,
   update_rule_name    OUT  VARCHAR2,
   delete_rule_name    OUT  VARCHAR2);

DBMS_STREAMS_ADM.ADD_SUBSET_RULES(
   table_name          IN   VARCHAR2,
   dml_condition       IN   VARCHAR2,
   streams_type        IN   VARCHAR2 DEFAULT 'apply',
   streams_name        IN   VARCHAR2 DEFAULT NULL,
   queue_name          IN   VARCHAR2 DEFAULT 'streams_queue',
   include_tagged_lcr  IN   BOOLEAN  DEFAULT FALSE,
   source_database     IN   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 106–11 ADD_SUBSET_RULES Procedure Parameters

Parameter	Description
table_name	The name of the table specified as [*schema_name*.]*object_name*. For example, hr.employees. If the schema is not specified, then the current user is the default. The specified table must exist in the same database as the capture process, apply process, or messaging client. Also, the specified table cannot have any LOB, LONG, or LONG RAW columns currently or in the future.
dml_condition	The subset condition. Specify this condition similar to the way you specify conditions in a WHERE clause in SQL. For example, to specify rows in the hr.employees table where the salary is greater than 4000 and the job_id is SA_MAN, enter the following as the condition: `' salary > 4000 and job_id = ''SA_MAN'' '` **Note:** The quotation marks in the preceding example are all single quotation marks.
streams_type	The type of Streams client: ■ Specify capture for a capture process. ■ Specify apply for an apply process. ■ Specify dequeue for a messaging client.
streams_name	The name of the capture process, apply process, or messaging client. Do not specify an owner. If the specified Streams client does not exist, then the procedure creates it automatically. If NULL, if streams_type is capture or dequeue, and if one relevant capture process or messaging client for the queue exists, then the procedure uses the relevant Streams client. If no relevant Streams client exists for the queue, then the procedure creates a Streams client automatically with a system-generated name. If NULL and multiple Streams clients of the specified streams_type for the queue exist, then the procedure raises an error. If NULL, if streams_type is apply, and if one relevant apply process exists, then the procedure uses the relevant apply process. The relevant apply process is identified in one of the following ways: ■ If one existing apply process has the source database specified in source_database and uses the queue specified in queue_name, then the procedure uses this apply process. ■ If source_database is NULL and one existing apply process is using the queue specified in queue_name, then the procedure uses this apply process. If NULL and no relevant apply process exists, then the procedure creates an apply process automatically with a system-generated name. If NULL and multiple relevant apply processes exist, then the procedure raises an error. An apply process and a messaging client cannot have the same name.

Table 106–11 (Cont.) ADD_SUBSET_RULES Procedure Parameters

Parameter	Description
queue_name	The name of the local queue, specified as [*schema_name*.]*queue_name*. The current database must contain the queue, and the queue must be ANYDATA type. For example, to specify a queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter. If the schema is not specified, then the current user is the default. For capture process rules, this is the queue into which a capture process enqueues LCRs. For apply process rules, this is the queue from which an apply process dequeues messages. For messaging client rules, this is the queue from which a messaging client dequeues messages.
include_tagged_lcr	If TRUE, then a redo entry is always considered for capture and an LCR is always considered for apply or dequeue, regardless of whether redo entry or LCR has a non-NULL tag. This setting is appropriate for a full (for example, standby) copy of a database. If FALSE, then a redo entry is considered for capture and an LCR is considered for apply or dequeue only when the redo entry or the LCR contains a NULL tag. A setting of FALSE is often specified in update-anywhere configurations to avoid sending a change back to its source database. **See Also:** *Oracle Streams Replication Administrator's Guide* for more information about tags
source_database	The global name of the source database. If NULL, then the procedure does not add a condition regarding the source database to the generated rules. For capture process rules, specify NULL or the global name of the local database if you are creating a capture process locally at the source database. If you are adding rules to a downstream capture process rule set at a downstream database, then specify the source database of the changes that will be captured. For apply process rules, specify the source database of the changes that will be applied by the apply process. The source database is the database where the changes originated. If an apply process applies captured messages, then the apply process can apply messages from only one capture process at one source database. For messaging client rules, specify NULL if you do not want the rules created by this procedure to have a condition for the source database. Specify a source database if you want the rules created by this procedure to have a condition for the source database. The source database is part of the information in an LCR, and user-constructed LCRs might or might not have this information. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically.
insert_rule_name	Contains the system-generated INSERT rule name. This rule handles inserts, as well as updates that must be converted into inserts.
update_rule_name	Contains the system-generated UPDATE rule name. This rule handles updates that remain updates.

Table 106–11 (Cont.) ADD_SUBSET_RULES Procedure Parameters

Parameter	Description
delete_rule_name	Contains the system-generated DELETE rule name. This rule handles deletes, as well as updates that must be converted into deletes

Usage Notes

Running this procedure generates three rules for the specified capture process, apply process, or messaging client: one for INSERT statements, one for UPDATE statements, and one for DELETE statements. For INSERT and DELETE statements, only DML changes that satisfy the condition specified for the dml_condition parameter are captured, applied, or dequeued. For UPDATE statements, the following variations are possible:

- If both the new and old values in a DML change satisfy the specified dml_condition, then the DML change is captured, applied, or dequeued without any changes.
- If neither the new or old values in a DML change satisfy the specified dml_condition, then the DML change is not captured, applied, or dequeued.
- If the old values for a DML change satisfy the specified dml_condition, but the new values do not, then the DML change is converted into a delete.
- If the new values for a DML change satisfy the specified dml_condition, but the old values do not, then the DML change is converted to an insert.

When an update is converted into an insert or a delete, it is called row migration.

A capture process, apply process, or messaging client uses the rules created for filtering. If the Streams client does not have a positive rule set, then this procedure creates a positive rule set automatically, and adds the rules for the table to the positive rule set. A subset rule can be added to positive rule set only, not to a negative rule set. Other rules in an existing rule set for the process are not affected. Additional rules can be added using either the DBMS_STREAMS_ADM package or the DBMS_RULE_ADM package.

Rules for INSERT, UPDATE, and DELETE statements are created automatically when you run this procedure, and these rules are given a system-generated rule name. Each rule has a system-generated rule name that consists of the table name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the table name plus the sequence number is too long, then the table name is truncated. The ADD_SUBSET_RULES procedure is overloaded, and the system-generated rule names for INSERT, UPDATE, and DELETE statements are returned.

Attention: Subset rules should only reside in positive rule sets. You should not add subset rules to negative rule sets. Doing so might have unpredictable results because row migration would not be performed on LCRs that are not discarded by the negative rule set.

See Also:

- "Operational Notes" on page 106-7
- "Security Model" on page 106-5

Examples

The following is an example of a rule condition created for filtering DML changes containing an update operation when the `dml_condition` is `region_id = 2`, the `table_name` is `hr.regions`, and the `source_database` is `dbs1.net`:

```
:dml.get_object_owner()='HR' AND :dml.get_object_name()='REGIONS'
AND :dml.is_null_tag()='Y' AND :dml.get_source_database_name()='DBS1.NET'
AND :dml.get_command_type()='UPDATE'
AND (:dml.get_value('NEW','"REGION_ID"') IS NOT NULL)
AND (:dml.get_value('OLD','"REGION_ID"') IS NOT NULL)
AND (:dml.get_value('OLD','"REGION_ID"').AccessNumber()=2)
AND (:dml.get_value('NEW','"REGION_ID"').AccessNumber()=2)
```

ADD_TABLE_PROPAGATION_RULES Procedure

This procedures adds table rules to the positive rule set for a propagation, or adds table rules to the negative rule set for a propagation, and creates the specified propagation if it does not exist.

This procedure is overloaded. One version of this procedure contains two `OUT` parameters, and the other does not.

Syntax

```
DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(
   table_name              IN   VARCHAR2,
   streams_name            IN   VARCHAR2  DEFAULT NULL,
   source_queue_name       IN   VARCHAR2,
   destination_queue_name  IN   VARCHAR2,
   include_dml             IN   BOOLEAN   DEFAULT TRUE,
   include_ddl             IN   BOOLEAN   DEFAULT FALSE,
   include_tagged_lcr      IN   BOOLEAN   DEFAULT FALSE,
   source_database         IN   VARCHAR2  DEFAULT NULL,
   dml_rule_name           OUT  VARCHAR2,
   ddl_rule_name           OUT  VARCHAR2,
   inclusion_rule          IN   BOOLEAN   DEFAULT TRUE,
   and_condition           IN   VARCHAR2  DEFAULT NULL,
   queue_to_queue          IN   BOOLEAN   DEFAULT NULL);

DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(
   table_name              IN   VARCHAR2,
   streams_name            IN   VARCHAR2  DEFAULT NULL,
   source_queue_name       IN   VARCHAR2,
   destination_queue_name  IN   VARCHAR2,
   include_dml             IN   BOOLEAN   DEFAULT TRUE,
   include_ddl             IN   BOOLEAN   DEFAULT FALSE,
   include_tagged_lcr      IN   BOOLEAN   DEFAULT FALSE,
   source_database         IN   VARCHAR2  DEFAULT NULL,
   inclusion_rule          IN   BOOLEAN   DEFAULT TRUE,
   and_condition           IN   VARCHAR2  DEFAULT NULL,
   queue_to_queue          IN   BOOLEAN   DEFAULT NULL);
```

Parameters

Table 106–12 ADD_TABLE_PROPAGATION_RULES Procedure Parameters

Parameter	Description
`table_name`	The name of the table specified as `[`*`schema_name`*`.]`*`table_name`*. For example, `hr.employees`. If the schema is not specified, then the current user is the default.
`streams_name`	The name of the propagation. Do not specify an owner. If the specified propagation does not exist, then the procedure creates it automatically. If `NULL` and a propagation exists for the same source queue and destination queue (including database link), then the procedure uses this propagation. If `NULL` and no propagation exists for the same source queue and destination queue (including database link), then the procedure creates a propagation automatically with a system-generated name.

Table 106–12 (Cont.) ADD_TABLE_PROPAGATION_RULES Procedure Parameters

Parameter	Description
source_queue_name	The name of the source queue, specified as [*schema_name*.]*queue_name*. The current database must contain the source queue, and the queue must be ANYDATA type. For example, to specify a source queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter. If the schema is not specified, then the current user is the default.
destination_queue_name	The name of the destination queue, including a database link, specified as [*schema_name*.]*queue_name*[*@dblink_name*], if the destination queue is in a remote database. The queue must be ANYDATA type. For example, to specify a destination queue named streams_queue in the strmadmin schema and use a database link named dbs2.net, enter strmadmin.streams_queue@dbs2.net for this parameter. If the schema is not specified, then the current user is the default. If the database link is omitted, then the procedure uses the global name of the current database, and the source queue and destination queue must be in the same database. **Note:** Connection qualifiers are not allowed.
include_dml	If TRUE, then the procedure creates a rule for DML changes. If FALSE, then the procedure does not create a DML rule. NULL is not permitted.
include_ddl	If TRUE, then the procedure creates a rule for DDL changes. If FALSE, then the procedure does not create a DDL rule. NULL is not permitted.

Table 106–12 (Cont.) ADD_TABLE_PROPAGATION_RULES Procedure Parameters

Parameter	Description
include_tagged_lcr	If TRUE, then the procedure does not add a condition regarding Streams tags to the generated rules. Therefore, these rules can evaluate to TRUE regardless of whether a logical change record (LCR) has a non-NULL tag. If the rules are added to the positive rule set for the propagation, then an LCR is always considered for propagation, regardless of whether it has a non-NULL tag. If the rules are added to a positive rule set, then setting this parameter to TRUE is appropriate for a full (for example, standby) copy of a database. If the rules are added to the negative rule set for the propagation, then whether an LCR is discarded does not depend on the LCR's tag. If FALSE, then the procedure adds a condition to each generated rule that causes the rule to evaluate to TRUE only if an LCR has a NULL Streams tag. If the rules are added to the positive rule set for the propagation, then an LCR is considered for propagation only when the LCR contains a NULL tag. If the rules are added to a positive rule set, then setting this parameter to FALSE might be appropriate in update-anywhere configurations to avoid sending a change back to its source database. If the rules are added to the negative rule set for the propagation, then an LCR can be discarded only if it has a NULL tag. In most cases, specify TRUE for this parameter if the inclusion_rule parameter is set to FALSE. **See Also:** *Oracle Streams Replication Administrator's Guide* for more information about tags
source_database	The global name of the source database. The source database is where the change originated. If NULL, then the procedure does not add a condition regarding the source database to the generated rules. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically. Oracle recommends that you specify a source database for propagation rules.
dml_rule_name	If include_dml is TRUE, then this parameter contains the DML rule name. If include_dml is FALSE, then this parameter contains a NULL.
ddl_rule_name	If include_ddl is TRUE, then this parameter contains the DDL rule name. If include_ddl is FALSE, then this parameter contains a NULL.
inclusion_rule	If inclusion_rule is TRUE, then the procedure adds the rules to the positive rule set for the propagation. If inclusion_rule is FALSE, then the procedure adds the rules to the negative rule set for the propagation. In either case, the system creates the rule set if it does not exist.

Table 106–12 (Cont.) ADD_TABLE_PROPAGATION_RULES Procedure Parameters

Parameter	Description
and_condition	If non-NULL, appends the specified condition to the system-generated rule condition using an AND clause in the following way: (*system_condition*) AND (*and_condition*) The variable in the specified condition must be :lcr. For example, to specify that the table rules generated by the procedure evaluate to TRUE only if the Streams tag is the hexadecimal equivalent of '02', specify the following condition: :lcr.get_tag() = HEXTORAW(''02'') The :lcr in the specified condition is converted to :dml or :ddl, depending on the rule that is being generated. If you are specifying an LCR member subprogram that is dependent on the LCR type (row or DDL), then make sure this procedure only generates the appropriate rule. Specifically, if you specify an LCR member subprogram that is valid only for row LCRs, then specify TRUE for the include_dml parameter and FALSE for the include_ddl parameter. If you specify an LCR member subprogram that is valid only for DDL LCRs, then specify FALSE for the include_dml parameter and TRUE for the include_ddl parameter. **See Also:** Chapter 188, "Logical Change Record TYPEs"
queue_to_queue	If TRUE, then the propagation is a queue to queue propagation. A queue-to-queue propagation always has its own propagation job and uses a service for automatic failover when the destination queue is a buffered queue in a Real Application Clusters (RAC) database. If FALSE or NULL, then the propagation is a queue-to-dblink propagation. A queue-to-dblink propagation can share a propagation job with other propagations that use the same database link and does not support automatic failover in a RAC environment. This procedure cannot change the queue to queue property of an exiting propagation. If the specified propagation exists, then the procedure behaves in the following way for each setting: ■ If TRUE and the specified propagation is not a queue to queue propagation, then the procedure raises an error. ■ If FALSE and the specified propagation is a queue to queue propagation, then the procedure raises an error. ■ If NULL, then the procedure does not change the queue to queue property of the propagation. **See Also:** *Oracle Streams Concepts and Administration* for more information about queue-to-queue propagations

Usage Notes

This procedure configures propagation using the current user. Only one propagation is allowed between a particular source queue and destination queue.

This procedure creates DML and DDL rules automatically based on include_dml and include_ddl parameter values, respectively. Each rule has a system-generated

rule name that consists of the table name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the table name plus the sequence number is too long, then the table name is truncated. A propagation uses the rules created for filtering.

See Also:

- "Operational Notes" on page 106-7 and "Propagation Rules for LCRs" on page 106-9 for more information about the rules created by this procedure
- "Security Model" on page 106-5

Examples

The following is an example of a table rule condition created for filtering DML statements:

```
(((:dml.get_object_owner() = 'HR' and :dml.get_object_name() = 'LOCATIONS'))
and :dml.is_null_tag() = 'Y' and :dml.get_source_database_name() = 'DBS1.NET' )
```

ADD_TABLE_RULES Procedure

This procedure adds rules to a rule set of one of the following types of Streams clients:

- Capture process rules for capturing changes to a specified table when the `streams_type` parameter is set to `capture`. See "Capture Process Rules for Changes in the Redo Log" on page 106-7 for more information about these rules.
- Apply process rules for applying logical change records (LCRs) in a queue that contain changes to a specified table when the `streams_type` parameter is set to `apply`. The rules can specify that the LCRs must be from a particular source database. See "Apply Process Rules for LCRs" on page 106-9 for more information about these rules.
- Messaging client rules for dequeuing user-enqueued LCRs from a queue that contain changes to a specified table when the `streams_type` parameter is set to `dequeue`. The rules can specify that the LCRs must be from a particular source database. See "Messaging Client Rules for LCRs" on page 106-11 for more information about these rules.

This procedure creates the specified capture process, apply process, or messaging client if it does not exist.

This procedure is overloaded. One version of this procedure contains two `OUT` parameters, and the other does not.

Note: Currently, messaging clients cannot dequeue buffered messages.

Syntax

```
DBMS_STREAMS_ADM.ADD_TABLE_RULES(
   table_name          IN   VARCHAR2,
   streams_type        IN   VARCHAR2,
   streams_name        IN   VARCHAR2 DEFAULT NULL,
   queue_name          IN   VARCHAR2 DEFAULT 'streams_queue',
   include_dml         IN   BOOLEAN  DEFAULT TRUE,
   include_ddl         IN   BOOLEAN  DEFAULT FALSE,
   include_tagged_lcr  IN   BOOLEAN  DEFAULT FALSE,
   source_database     IN   VARCHAR2 DEFAULT NULL,
   dml_rule_name       OUT  VARCHAR2,
   ddl_rule_name       OUT  VARCHAR2,
   inclusion_rule      IN   BOOLEAN   DEFAULT TRUE,
   and_condition       IN   VARCHAR2  DEFAULT NULL);

DBMS_STREAMS_ADM.ADD_TABLE_RULES(
   table_name          IN   VARCHAR2,
   streams_type        IN   VARCHAR2,
   streams_name        IN   VARCHAR2 DEFAULT NULL,
   queue_name          IN   VARCHAR2 DEFAULT 'streams_queue',
   include_dml         IN   BOOLEAN  DEFAULT TRUE,
   include_ddl         IN   BOOLEAN  DEFAULT FALSE,
   include_tagged_lcr  IN   BOOLEAN  DEFAULT FALSE,
   source_database     IN   VARCHAR2 DEFAULT NULL,
   inclusion_rule      IN   BOOLEAN   DEFAULT TRUE,
   and_condition       IN   VARCHAR2  DEFAULT NULL);
```

Parameters

Table 106–13 ADD_TABLE_RULES Procedure Parameters

Parameter	Description
table_name	The name of the table specified as [*schema_name.*]*object_name*. For example, hr.employees. If the schema is not specified, then the current user is the default. You can specify a table that does not yet exist, because Streams does not validate the existence of the table.
streams_type	The type of Streams client: ■ Specify capture for a capture process. ■ Specify apply for an apply process. ■ Specify dequeue for a messaging client.
streams_name	The name of the capture process, apply process, or messaging client. Do not specify an owner. If the specified Streams client does not exist, then the procedure creates it automatically. If NULL, if streams_type is capture or dequeue, and if one relevant capture process or messaging client for the queue exists, then the procedure uses the relevant Streams client. If no relevant Streams client exists for the queue, then the procedure creates a Streams client automatically with a system-generated name. If NULL and multiple Streams clients of the specified streams_type for the queue exist, then the procedure raises an error. If NULL, if streams_type is apply, and if one relevant apply process exists, then the procedure uses the relevant apply process. The relevant apply process is identified in one of the following ways: ■ If one existing apply process has the source database specified in source_database and uses the queue specified in queue_name, then the procedure uses this apply process. ■ If source_database is NULL and one existing apply process is using the queue specified in queue_name, then the procedure uses this apply process. If NULL and no relevant apply process exists, then the procedure creates an apply process automatically with a system-generated name. If NULL and multiple relevant apply processes exist, then the procedure raises an error. An apply process and a messaging client cannot have the same name.
queue_name	The name of the local queue, specified as [*schema_name.*]*queue_name*. The current database must contain the queue, and the queue must be ANYDATA type. For example, to specify a queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter. If the schema is not specified, then the current user is the default. For capture process rules, this is the queue into which a capture process enqueues LCRs. For apply process rules, this is the queue from which an apply process dequeues messages. For messaging client rules, this is the queue from which a messaging client dequeues messages.

Table 106–13 (Cont.) ADD_TABLE_RULES Procedure Parameters

Parameter	Description
include_dml	If TRUE, then the procedure creates a DML rule for DML changes. If FALSE, then the procedure does not create a DML rule. NULL is not permitted.
include_ddl	If TRUE, then the procedure creates a DDL rule for DDL changes. If FALSE, then the procedure does not create a DDL rule. NULL is not permitted.
include_tagged_lcr	If TRUE, then the procedure does not add a condition regarding Streams tags to the generated rules. Therefore, these rules can evaluate to TRUE regardless of whether a redo entry or LCR has a non-NULL tag. If the rules are added to the positive rule set for the process, then a redo entry is always considered for capture, and an LCR is always considered for apply, regardless of whether the redo entry or LCR has a non-NULL tag. If the rules are added to a positive rule set, then setting this parameter to TRUE is appropriate for a full (for example, standby) copy of a database. If the rules are added to the negative rule set for the process, then whether a redo entry or LCR is discarded does not depend on the tag. If FALSE, then the procedure adds a condition to each generated rule that causes the rule to evaluate to TRUE only if a redo entry or LCR has a NULL Streams tag. If the rules are added to the positive rule set for the process, then a redo entry is considered for capture, and an LCR is considered for apply, only when the redo entry or LCR contains a NULL tag. If the rules are added to a positive rule set, then setting this parameter to FALSE might be appropriate in update-anywhere configurations to avoid sending a change back to its source database. If the rules are added to the negative rule set for the process, then a redo entry or LCR can be discarded only if it has a NULL tag. In most cases, specify TRUE for this parameter if the inclusion_rule parameter is set to FALSE. **See Also:** *Oracle Streams Replication Administrator's Guide* for more information about tags

Table 106–13 (Cont.) ADD_TABLE_RULES Procedure Parameters

Parameter	Description
source_database	The global name of the source database. If NULL, then the procedure does not add a condition regarding the source database to the generated rules. For capture process rules, specify NULL or the global name of the local database if you are creating a capture process locally at the source database. If you are adding rules to a downstream capture process rule set at a downstream database, then specify the source database of the changes that will be captured. For apply process rules, specify the source database of the changes that will be applied by the apply process. The source database is the database where the changes originated. If an apply process applies captured messages, then the apply process can apply messages from only one capture process at one source database. For messaging client rules, specify NULL if you do not want the rules created by this procedure to have a condition for the source database. Specify a source database if you want the rules created by this procedure to have a condition for the source database. The source database is part of the information in an LCR, and user-constructed LCRs might or might not have this information. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically.
dml_rule_name	If include_dml is TRUE, then this parameter contains the DML rule name. If include_dml is FALSE, then this parameter contains a NULL.
ddl_rule_name	If include_ddl is TRUE, then this parameter contains the DDL rule name. If include_ddl is FALSE, then this parameter contains a NULL.
inclusion_rule	If inclusion_rule is TRUE, then the procedure adds the rules to the positive rule set for the Streams client. If inclusion_rule is FALSE, then the procedure adds the rules to the negative rule set for the Streams client. In either case, the system creates the rule set if it does not exist.

Table 106–13 (Cont.) ADD_TABLE_RULES Procedure Parameters

Parameter	Description
and_condition	If non-NULL, appends the specified condition to the system-generated rule condition using an AND clause in the following way: (*system_condition*) AND (*and_condition*) The variable in the specified condition must be :lcr. For example, to specify that the table rules generated by the procedure evaluate to TRUE only if the Streams tag is the hexadecimal equivalent of '02', specify the following condition: :lcr.get_tag() = HEXTORAW(''02'') The :lcr in the specified condition is converted to :dml or :ddl, depending on the rule that is being generated. If you are specifying an LCR member subprogram that is dependent on the LCR type (row or DDL), then make sure this procedure only generates the appropriate rule. Specifically, if you specify an LCR member subprogram that is valid only for row LCRs, then specify TRUE for the include_dml parameter and FALSE for the include_ddl parameter. If you specify an LCR member subprogram that is valid only for DDL LCRs, then specify FALSE for the include_dml parameter and TRUE for the include_ddl parameter. **See Also:** Chapter 188, "Logical Change Record TYPEs"

Usage Notes

This procedure creates DML and DDL rules automatically based on include_dml and include_ddl parameter values, respectively. Each rule has a system-generated rule name that consists of the table name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the table name plus the sequence number is too long, then the table name is truncated. A capture process, apply process, or messaging client uses the rules created for filtering.

See Also:

- "Operational Notes" on page 106-7
- "Security Model" on page 106-5

Examples

The following is an example of a table rule condition created for DML changes:

```
(((:dml.get_object_owner() = 'HR' and :dml.get_object_name() = 'LOCATIONS'))
and :dml.is_null_tag() = 'Y' and :dml.get_source_database_name() = 'DBS1.NET' )
```

CLEANUP_INSTANTIATION_SETUP Procedure

This procedure removes a Streams replication configuration that was set up by the PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures in this package. This procedure either remove the configuration directly, or it can generate a script that removes the configuration.

Run this procedure at the capture database. The capture database is the database that captures changes made to the source database.

Attention: When the CLEANUP_INSTANTIATION_SETUP procedure is run, the parameter values must match the parameter values specified when the corresponding PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures were run, except for the values of the following parameters: perform_actions, script_name, and script_directory_object.

See Also:

- "PRE_INSTANTIATION_SETUP Procedure" on page 106-115
- "POST_INSTANTIATION_SETUP Procedure" on page 106-111
- "Procedures That Configure a Streams Replication Environment" on page 106-11 for more information about this procedure

Syntax

```
DBMS_STREAMS_ADM.CLEANUP_INSTANTIATION_SETUP(
   maintain_mode           IN VARCHAR2,
   tablespace_names        IN DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET,
   source_database         IN VARCHAR2,
   destination_database    IN VARCHAR2,
   perform_actions         IN BOOLEAN   DEFAULT TRUE,
   script_name             IN VARCHAR2  DEFAULT NULL,
   script_directory_object IN VARCHAR2  DEFAULT NULL,
   capture_name            IN VARCHAR2  DEFAULT NULL,
   capture_queue_table     IN VARCHAR2  DEFAULT NULL,
   capture_queue_name      IN VARCHAR2  DEFAULT NULL,
   capture_queue_user      IN VARCHAR2  DEFAULT NULL,
   propagation_name        IN VARCHAR2  DEFAULT NULL,
   apply_name              IN VARCHAR2  DEFAULT NULL,
   apply_queue_table       IN VARCHAR2  DEFAULT NULL,
   apply_queue_name        IN VARCHAR2  DEFAULT NULL,
   apply_queue_user        IN VARCHAR2  DEFAULT NULL,
   bi_directional          IN BOOLEAN   DEFAULT FALSE,
   change_global_name      IN BOOLEAN   DEFAULT FALSE);
```

Parameters

Table 106–14 CLEANUP_INSTANTIATION_SETUP Procedure Parameters

Parameter	Description
maintain_mode	Specify one of the following: ▪ GLOBAL to clean up the Streams configuration that maintained the entire database in both the source and destination databases ▪ TRANSPORTABLE TABLESPACES to cleanup the Streams configuration that maintained a set of tablespaces at both the source and destination database
tablespace_names	If maintain_mode is set to TRANSPORTABLE TABLESPACES, then specify the local tablespace set to be cloned at the destination database and maintained by Streams. The tablespaces in the tablespace set must exist at the source database, but these tablespaces must not exist at the destination database. A directory object must exist for each directory that contains the datafiles for the tablespace set. The user who invokes this procedure must have READ privilege on these directory objects. If maintain_mode is set to GLOBAL, then specify NULL. **See Also:** TABLESPACE_SET Type on page 109-4
source_database	The global name of the source database. If NULL, then the procedure uses the global name of the local database.
destination_database	The global name of the destination database. A database link from the local database to the destination database with the same name as the global name of the destination database must exist and must be accessible to the user who runs the procedure. If NULL, then the procedure raises an error.
perform_actions	If TRUE, then this procedure performs the necessary actions to clean up the Streams configuration directly. If FALSE, then the procedure does not perform the necessary actions to clean up the Streams configuration directly. Specify FALSE when this procedure is generating a script that you can edit and then run. The procedure raises an error if you specify FALSE and either of the following parameters is NULL: ▪ script_name ▪ script_directory_object

Table 106–14 (Cont.) CLEANUP_INSTANTIATION_SETUP Procedure Parameters

Parameter	Description
script_name	If non-NULL and the perform_actions parameter is FALSE, then specify the name of the script generated by this procedure. The script contains all of the statements used to clean up the Streams configuration. If a file with the specified script name exists in the specified directory for the script_directory_object parameter, then the statements are appended to the existing file. If non-NULL and the perform_actions parameter is TRUE, then this procedure generates the specified script and performs the actions to clean up the Streams configuration directly. If NULL and the perform_actions parameter is TRUE, then this procedure directly performs the actions to clean up the Streams configuration without generating a script. If NULL and the perform_actions parameter is FALSE, then the procedure raises an error.
script_directory_object	The directory object for the directory on the local computer system into which the generated script is placed. If the script_name parameter is NULL, then this parameter is ignored, and this procedure does not generate a script. If NULL and the script_name parameter is non-NULL, then the procedure raises an error.
capture_name	The name of the capture processes configured to capture changes in the Streams configuration. Do not specify an owner. If NULL, then the procedure automatically identifies the capture processes with system-generated names created by the PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures.
capture_queue_table	The name of the queue table for each queue used by a capture process, specified as [*schema_name*.]*queue_table_name*. For example, strmadmin.streams_queue_table. If the schema is not specified, then the current user is the default. If NULL, then the procedure automatically identifies the capture queue tables with system-generated names created by the PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures.
capture_queue_name	The name of each queue used by a capture process, specified as [*schema_name*.]*queue_name*. For example, strmadmin.streams_queue. If the schema is not specified, then the queue table owner is the default. The queue owner automatically has privileges to perform all queue operations on the queue. If NULL, then the procedure automatically identifies the capture queues with system-generated names created by the PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures.
capture_queue_user	The name of the user who has ENQUEUE and DEQUEUE privileges for the queue at the source database. This user is a secure queue user of the queue.

Table 106–14 (Cont.) CLEANUP_INSTANTIATION_SETUP Procedure Parameters

Parameter	Description
propagation_name	The name of the propagations configured to propagate changes in the Streams configuration. Do not specify an owner. If NULL, then the procedure automatically identifies the propagations with system-generated names created by the PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures.
apply_name	The name of the apply processes configured to apply changes in the Streams configuration. Do not specify an owner. If NULL, then the procedure automatically identifies the apply processes with system-generated names created by the PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures.
apply_queue_table	The name of the queue table for each queue used by an apply process, specified as [*schema_name*.]*queue_table_name*. For example, strmadmin.streams_queue_table. If the schema is not specified, then the current user is the default. If NULL, then the procedure automatically identifies the apply queue tables with system-generated names created by the PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures.
apply_queue_name	The name of each queue used by an apply process, specified as [*schema_name*.]*queue_name*. For example, strmadmin.streams_queue. If the schema is not specified, then the queue table owner is the default. The queue owner automatically has privileges to perform all queue operations on the queue. If NULL, then the procedure automatically identifies the apply queues with system-generated names created by the PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures.
apply_queue_user	The name of the user who has ENQUEUE and DEQUEUE privileges for the queue at the destination database. This user is a secure queue user of the queue.
bi_directional	Specify TRUE if the Streams replication configuration is bi-directional between the database specified in source_database and the database specified in destination_database. Specify FALSE if the Streams replication configuration is one way replication from the current database to the database specified in destination_database.
change_global_name	If TRUE, then the procedure changes the global name of the database specified in destination_database to match the global name of the current database. If FALSE, then the procedure does not change the global name of the database specified in destination_database.

DELETE_COLUMN Procedure

This procedure either adds or removes a declarative rule-based transformation which deletes a column from a row logical change record (LCR) that satisfies the specified rule.

For the transformation to be performed when the specified rule evaluates to `TRUE`, the rule must be in the positive rule set of a Streams client. Streams clients include capture processes, propagations, apply processes, and messaging clients.

Note:

- The `DELETE_COLUMN` procedure supports the same datatypes supported by Streams capture processes.
- Declarative transformations can transform row LCRs only. These row LCRs can be captured row LCRs or user-enqueued row LCRs. Therefore, a DML rule must be specified when you run this procedure. If a DDL is specified, then the procedure raises an error.

See Also: *Oracle Streams Concepts and Administration* for more information about declarative rule-based transformations and about the datatypes supported by Streams capture processes

Syntax

```
DBMS_STREAMS_ADM.DELETE_COLUMN(
   rule_name     IN  VARCHAR2,
   table_name    IN  VARCHAR2,
   column_name   IN  VARCHAR2,
   value_type    IN  VARCHAR2   DEFAULT '*',
   step_number   IN  NUMBER     DEFAULT 0,
   operation     IN  VARCHAR2   DEFAULT 'ADD');
```

Parameters

Table 106–15 DELETE_COLUMN Procedure Parameters

Parameter	Description
`rule_name`	The name of the rule, specified as `[`*`schema_name`*`.]`*`rule_name`*. If `NULL`, then the procedure raises an error. For example, to specify a rule in the `hr` schema named `employees12`, enter `hr.employees12`. If the schema is not specified, then the current user is the default.
`table_name`	The name of the table from which the column is deleted in the row LCR, specified as `[`*`schema_name`*`.]`*`object_name`*. For example, `hr.employees`. If the schema is not specified, then the current user is the default.
`column_name`	The name of the column deleted from each row LCR that satisfies the rule.

Table 106–15 (Cont.) DELETE_COLUMN Procedure Parameters

Parameter	Description
value_type	Specify 'NEW' to delete the column from the new values in the row LCR. Specify 'OLD' to delete the column from the old values in the row LCR. Specify '*' to delete the column from both the old and new values in the row LCR.
step_number	The order of execution of the transformation. See Also: *Oracle Streams Concepts and Administration* for more information about transformation ordering
operation	Specify 'ADD' to add the transformation to the rule. Specify 'REMOVE' to remove the transformation from the rule.

Usage Notes

When 'REMOVE' is specified for the operation parameter, all of the delete column declarative rule-based transformations for the specified rule are removed that match the specified table_name, column_name, and step_number parameters. Nulls specified for these parameters act as wildcards. The following table lists the behavior of the DELETE_COLUMN procedure when one or more of these parameters is NULL:

table_name	column_name	step_number	Result
NULL	NULL	NULL	Remove all delete column transformations for the specified rule.
NULL	NULL	non-NULL	Remove all delete column transformations with the specified step_number for the specified rule.
NULL	non-NULL	non-NULL	Remove all delete column transformations with the specified column_name and step_number for the specified rule.
non-NULL	NULL	non-NULL	Remove all delete column transformations with the specified table_name and step_number for the specified rule.
NULL	non-NULL	NULL	Remove all delete column transformations with the specified column_name for the specified rule.
non-NULL	non-NULL	NULL	Remove all delete column transformations with the specified table_name and column_name for the specified rule.
non-NULL	NULL	NULL	Remove all delete column transformations with the specified table_name for the specified rule.
non-NULL	non-NULL	non-NULL	Remove all delete column transformations with the specified table_name, column_name, and step_number for the specified rule.

GET_SCN_MAPPING Procedure

This procedure gets information about the system change number (SCN) values to use for Streams capture and apply processes in a Streams replication environment. This information can be used for the following purposes:

- To recover transactions after point-in-time recovery is performed on a source database in a multiple source Streams environment
- To run flashback queries for the corresponding SCN at a source database and destination database in a Streams single source replication environment

> **See Also:** *Oracle Streams Replication Administrator's Guide* for information about point-in-time recovery and flashback queries in a Streams replication environment

Syntax

```
DBMS_STREAMS_ADM.GET_SCN_MAPPING(
   apply_name             IN  VARCHAR2,
   src_pit_scn            IN  NUMBER,
   dest_instantiation_scn OUT NUMBER,
   dest_start_scn         OUT NUMBER,
   dest_skip_txn_ids      OUT DBMS_UTILITY.NAME_ARRAY);
```

Parameters

Table 106–16 GET_SCN_MAPPING Procedure Parameters

Parameter	Description
`apply_name`	Name of the apply process which applies logical change records (LCRs) from the source database. The procedure raises an error if the specified apply process does not exist.
`src_pit_scn`	The SCN at the source database. For point-in-time recovery, specify the point-in-time recovery SCN at the source database. If the specified SCN is greater than the source commit SCN of the last applied transaction, then `NULL` is returned for both `dest_start_scn` and `dest_instantiation_scn`. In this case, no values can be returned for these parameters because the corresponding transaction has not been applied at the destination database yet.
`dest_instantiation_scn`	The SCN at the destination database that corresponds to the specified `src_pit_scn` at the source database. For point-in-time recovery, use this value for the instantiation SCNs at the source database during recovery.
`dest_start_scn`	For point in time recovery, the SCN to use for the `start_scn` parameter for the recovery capture process.

Table 106–16 (Cont.) GET_SCN_MAPPING Procedure Parameters

Parameter	Description
dest_skip_txn_ids	Transaction IDs of transactions that were skipped at the dest_instantiation_scn because the apply process was applying nondependent transactions out of order. For point in time recovery, these transaction IDs should be ignored by the recovery apply process. This parameter is relevant only if the commit_serialization for the apply process that applied these transactions was set to none, and the transactions were applied out of order.

MAINTAIN_GLOBAL Procedure

This procedure configures a Streams environment that replicates changes at the database level between two databases. This procedure can either configure the environment directly, or it can generate a script that configures the environment.

Run this procedure at the capture database. The capture database is the database that captures changes made to the source database.

Note:

- This procedure automatically excludes database objects that are not supported by Streams from the replication environment by adding rules to the negative rule set of each capture and apply process. Query the `DBA_STREAMS_UNSUPPORTED` data dictionary view to determine which database objects are not supported by Streams. If unsupported database objects are not excluded, then capture errors will result.
- If the `bi_directional` parameter is set to `TRUE`, then do not allow data manipulation language (DML) or data definition language (DDL) changes to the destination database while the `MAINTAIN_GLOBAL` procedure, or the script generated by the procedure, is running. This restriction does not apply to the source database.
- A capture process never captures changes in the `SYS`, `SYSTEM`, or `CTXSYS` schemas. This procedure does not configure replication for these schemas.

See Also: "Procedures That Configure a Streams Replication Environment" on page 106-11 for more information about this procedure

Syntax

```
DBMS_STREAMS_ADM.MAINTAIN_GLOBAL(
   source_directory_object      IN VARCHAR2,
   destination_directory_object IN VARCHAR2,
   source_database              IN VARCHAR2,
   destination_database         IN VARCHAR2,
   perform_actions              IN BOOLEAN   DEFAULT TRUE,
   script_name                  IN VARCHAR2  DEFAULT NULL,
   script_directory_object      IN VARCHAR2  DEFAULT NULL,
   dump_file_name               IN VARCHAR2  DEFAULT NULL,
   capture_name                 IN VARCHAR2  DEFAULT NULL,
   capture_queue_table          IN VARCHAR2  DEFAULT NULL,
   capture_queue_name           IN VARCHAR2  DEFAULT NULL,
   capture_queue_user           IN VARCHAR2  DEFAULT NULL,
   propagation_name             IN VARCHAR2  DEFAULT NULL,
   apply_name                   IN VARCHAR2  DEFAULT NULL,
   apply_queue_table            IN VARCHAR2  DEFAULT NULL,
   apply_queue_name             IN VARCHAR2  DEFAULT NULL,
   apply_queue_user             IN VARCHAR2  DEFAULT NULL,
   log_file                     IN VARCHAR2  DEFAULT NULL,
   bi_directional               IN BOOLEAN   DEFAULT FALSE,
   include_ddl                  IN BOOLEAN   DEFAULT FALSE,
```

```
  instantiation                IN INTEGER   DEFAULT
                                            DBMS_STREAMS_ADM.INSTANTIATION_FULL);
```

Parameters

See Also: "Common Parameters for the Configuration Procedures" on page 106-18 for descriptions of the procedure parameters

Table 106–17 MAINTAIN_GLOBAL Procedure Parameters

Parameter	Description
source_directory_object	The directory object for the directory on the computer system running the source database into which the generated Data Pump export dump file is placed. This file remains in this directory after the procedure completes. This parameter is ignored if instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_FULL_NETWORK or DBMS_STREAMS_ADM.INSTANTIATION_NONE. In this case, specify NULL for the source_directory_object parameter. If NULL and instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_FULL, then the procedure raises an error.
destination_directory_object	The directory object for the directory on the computer system running the destination database into which the generated Data Pump export dump file is transferred. This parameter is ignored if instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_FULL_NETWORK or DBMS_STREAMS_ADM.INSTANTIATION_NONE. In these cases, specify NULL for the destination_directory_object parameter. If NULL and instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_FULL, then the procedure raises an error.
dump_file_name	The name of the Data Pump export dump file. If a file with the specified file name exists in the specified directory for the source_directory_object or destination_directory_object parameter, then the procedure raises an error. This parameter is ignored if instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_FULL_NETWORK or DBMS_STREAMS_ADM.INSTANTIATION_NONE. If NULL and instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_FULL, then the export dump file name is generated by the system. In this case, the export dump file name is expat*nn*.dmp, where *nn* is a sequence number. The sequence number is incremented to produce an export dump file with a unique name in the source directory.

Table 106–17 (Cont.) MAINTAIN_GLOBAL Procedure Parameters

Parameter	Description
log_file	The name of the Data Pump export log file. This log file is placed in the same directory as the Data Pump export dump file. This parameter is ignored if instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_FULL_NETWORK or DBMS_STREAMS_ADM.INSTANTIATION_NONE. If NULL and instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_FULL, then the log file name is the same name as the export dump file name with an extension of .clg.
instantiation	Specify whether to perform instantiation and, if instantiation is performed, the type of instantiation: DBMS_STREAMS_ADM.INSTANTIATION_FULL performs a full Data Pump export at the source database and a Data Pump import of the export dump file at the destination database. The instantiation SCN is set for the shared database objects during import. If the instantiation parameter is set to this value, then the user who runs this procedure must have EXECUTE privilege on the DBMS_FILE_TRANSFER package. DBMS_STREAMS_ADM.INSTANTIATION_FULL_NETWORK performs a full network Data Pump import. A network import means that Data Pump performs the import without using an export dump file. The instantiation SCN is set for the shared database objects during import. If the instantiation parameter is set to this value, then a database link from the destination database to the source database with the same name as the global name of the source database must exist and must be accessible to the user who runs the procedure. DBMS_STREAMS_ADM.INSTANTIATION_NONE does not perform an instantiation. This setting is valid only if the perform_actions parameter is set to FALSE, and the procedure generates a configuration script. In this case, the configuration script does not perform an instantiation and does not set the instantiation SCN for each shared database object. Instead, you must perform the instantiation and ensure that instantiation SCN values are set properly. If you use the RMAN DUPLICATE or CONVERT DATABASE command for database instantiation, then the destination database cannot be the capture database. If this parameter is set to DBMS_STREAMS_ADM.INSTANTIATION_FULL or DBMS_STREAMS_ADM.INSTANTIATION_FULL_NETWORK, then the database objects being instantiated must exist at the source database, but these database objects must not exist at the destination database.

MAINTAIN_SCHEMAS Procedure

This procedure configures a Streams environment that replicates changes to specified schemas between two databases. This procedure can either configure the environment directly, or it can generate a script that configures the environment.

Run this procedure at the capture database. The capture database is the database that captures changes made to the source database.

This procedure is overloaded. One `schema_names` parameter is type `VARCHAR2` and the other `schema_name` parameters is type `DBMS_UTILITY.UNCL_ARRAY`. These parameters enable you to enter the list of schemas in different ways and are mutually exclusive.

Note:

- This procedure automatically excludes database objects that are not supported by Streams in the schemas from the replication environment by adding rules to the negative rule set of each capture and apply process. Query the `DBA_STREAMS_UNSUPPORTED` data dictionary view to determine which database objects are not supported by Streams. If unsupported database objects are not excluded, then capture errors will result.
- If the `bi_directional` parameter is set to `TRUE`, then do not allow data manipulation language (DML) or data definition language (DDL) changes to the shared database objects at the destination database while the `MAINTAIN_SCHEMAS` procedure, or the script generated by the procedure, is running. This restriction does not apply to the source database.

See Also: "Procedures That Configure a Streams Replication Environment" on page 106-11 for more information about this procedure

Syntax

```
DBMS_STREAMS_ADM.MAINTAIN_SCHEMAS(
   schema_names                 IN VARCHAR2,
   source_directory_object      IN VARCHAR2,
   destination_directory_object IN VARCHAR2,
   source_database              IN VARCHAR2,
   destination_database         IN VARCHAR2,
   perform_actions              IN BOOLEAN   DEFAULT TRUE,
   script_name                  IN VARCHAR2  DEFAULT NULL,
   script_directory_object      IN VARCHAR2  DEFAULT NULL,
   dump_file_name               IN VARCHAR2  DEFAULT NULL,
   capture_name                 IN VARCHAR2  DEFAULT NULL,
   capture_queue_table          IN VARCHAR2  DEFAULT NULL,
   capture_queue_name           IN VARCHAR2  DEFAULT NULL,
   capture_queue_user           IN VARCHAR2  DEFAULT NULL,
   propagation_name             IN VARCHAR2  DEFAULT NULL,
   apply_name                   IN VARCHAR2  DEFAULT NULL,
   apply_queue_table            IN VARCHAR2  DEFAULT NULL,
   apply_queue_name             IN VARCHAR2  DEFAULT NULL,
   apply_queue_user             IN VARCHAR2  DEFAULT NULL,
   log_file                     IN VARCHAR2  DEFAULT NULL,
```

```
   bi_directional               IN BOOLEAN   DEFAULT FALSE,
   include_ddl                  IN BOOLEAN   DEFAULT FALSE,
   instantiation                IN INTEGER   DEFAULT
                                             DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA);

DBMS_STREAMS_ADM.MAINTAIN_SCHEMAS(
   schema_names                 IN DBMS_UTILITY.UNCL_ARRAY,
   source_directory_object      IN VARCHAR2,
   destination_directory_object IN VARCHAR2,
   source_database              IN VARCHAR2,
   destination_database         IN VARCHAR2,
   perform_actions              IN BOOLEAN   DEFAULT TRUE,
   script_name                  IN VARCHAR2  DEFAULT NULL,
   script_directory_object      IN VARCHAR2  DEFAULT NULL,
   dump_file_name               IN VARCHAR2  DEFAULT NULL,
   capture_name                 IN VARCHAR2  DEFAULT NULL,
   capture_queue_table          IN VARCHAR2  DEFAULT NULL,
   capture_queue_name           IN VARCHAR2  DEFAULT NULL,
   capture_queue_user           IN VARCHAR2  DEFAULT NULL,
   propagation_name             IN VARCHAR2  DEFAULT NULL,
   apply_name                   IN VARCHAR2  DEFAULT NULL,
   apply_queue_table            IN VARCHAR2  DEFAULT NULL,
   apply_queue_name             IN VARCHAR2  DEFAULT NULL,
   apply_queue_user             IN VARCHAR2  DEFAULT NULL,
   log_file                     IN VARCHAR2  DEFAULT NULL,
   bi_directional               IN BOOLEAN   DEFAULT FALSE,
   include_ddl                  IN BOOLEAN   DEFAULT FALSE,
   instantiation                IN INTEGER   DEFAULT
                                             DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA);
```

Parameters

See Also: "Common Parameters for the Configuration Procedures" on page 106-18 for descriptions of the procedure parameters that are not in Table 106–18

Table 106–18 MAINTAIN_SCHEMAS Procedure Parameters

Parameter	Description
schema_names	The schemas to be configured for replication and maintained by Streams after configuration. The schemas can be specified in the following ways: ▪ Comma-delimited list of type VARCHAR2 ▪ A PL/SQL index-by table of type DBMS_UTILITY.UNCL_ARRAY, where each element is the name of a schema. The first schema should be in position 1. The last position must be NULL. If a specified schema does not exist at the source database, then the procedure raises an error. If this parameter is set to NULL, then the procedure raises an error.

Table 106–18 (Cont.) MAINTAIN_SCHEMAS Procedure Parameters

Parameter	Description
source_directory_object	The directory object for the directory on the computer system running the source database into which the generated Data Pump export dump file is placed. This file remains in this directory after the procedure completes. This parameter is ignored if instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA_NETWORK or DBMS_STREAMS_ADM.INSTANTIATION_NONE. In this case, specify NULL for the source_directory_object parameter. If NULL and instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA, then the procedure raises an error.
destination_directory_object	The directory object for the directory on the computer system running the destination database into which the generated Data Pump export dump file is transferred. This parameter is ignored if instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA_NETWORK or DBMS_STREAMS_ADM.INSTANTIATION_NONE. In this case, specify NULL for the destination_directory_object parameter. If NULL and instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA, then the procedure raises an error.
dump_file_name	The name of the Data Pump export dump file. If a file with the specified file name exists in the specified directory for the source_directory_object or destination_directory_object parameter, then the procedure raises an error. This parameter is ignored if instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA_NETWORK or DBMS_STREAMS_ADM.INSTANTIATION_NONE. If NULL and instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA, then the export dump file name is generated by the system. In this case, the export dump file name is expat*nn*.dmp, where *nn* is a sequence number. The sequence number is incremented to produce an export dump file with a unique name in the source directory.
capture_queue_user	The name of the user who requires ENQUEUE and DEQUEUE privileges for the queue at the source database. This user also is configured as a secure queue user of the queue. The queue user cannot grant these privileges to other users because they are not granted with the GRANT option. If NULL, then the procedure does not grant any privileges. You can also grant queue privileges to the appropriate users using the DBMS_AQADM package.

Table 106–18 (Cont.) MAINTAIN_SCHEMAS Procedure Parameters

Parameter	Description
log_file	The name of the Data Pump export log file. This log file is placed in the same directory as the Data Pump export dump file. This parameter is ignored if instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA_NETWORK or DBMS_STREAMS_ADM.INSTANTIATION_NONE. If NULL and instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA, then the log file name is the same name as the export dump file name with an extension of .clg.
instantiation	Specify whether to perform instantiation and, if instantiation is performed, the type of instantiation: DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA performs a full Data Pump export at the source database and a Data Pump import of the export dump file at the destination database. The instantiation SCN is set for the shared database objects during import. If the instantiation parameter is set to this value, then the user who runs this procedure must have EXECUTE privilege on the DBMS_FILE_TRANSFER package. DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA_NETWORK performs a full network Data Pump import. A network import means that Data Pump performs the import without using an export dump file. The instantiation SCN is set for the shared database objects during import. If the instantiation parameter is set to this value, then a database link from the destination database to the source database with the same name as the global name of the source database must exist and must be accessible to the user who runs the procedure. DBMS_STREAMS_ADM.INSTANTIATION_NONE does not perform an instantiation. This setting is valid only if the perform_actions parameter is set to FALSE, and the procedure generates a configuration script. In this case, the configuration script does not perform an instantiation and does not set the instantiation SCN for each shared database object. Instead, you must perform the instantiation and ensure that instantiation SCN values are set properly. If this parameter is set to DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA or DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA_NETWORK, then the database objects being instantiated must exist at the source database, but these database objects must not exist at the destination database.

MAINTAIN_SIMPLE_TABLESPACE Procedure

This procedure clones a simple tablespace from a source database at a destination database and uses Streams to maintain this tablespace at both databases. This procedure can either perform these actions directly, or it can generate a script that performs these actions. Run this procedure at the source database.

Note: This procedure is deprecated. It is replaced by the `MAINTAIN_SIMPLE_TTS` procedure.

See Also:

- "Deprecated Subprograms" on page 106-4
- MAINTAIN_SIMPLE_TTS Procedure on page 106-94

Syntax

```
DBMS_STREAMS_ADM.MAINTAIN_SIMPLE_TABLESPACE(
   tablespace_name              IN VARCHAR2,
   source_directory_object      IN VARCHAR2,
   destination_directory_object IN VARCHAR2,
   destination_database         IN VARCHAR2,
   setup_streams                IN BOOLEAN   DEFAULT TRUE,
   script_name                  IN VARCHAR2  DEFAULT NULL,
   script_directory_object      IN VARCHAR2  DEFAULT NULL,
   bi_directional               IN BOOLEAN   DEFAULT FALSE);
```

Parameters

Table 106–19 MAINTAIN_SIMPLE_TABLESPACE Procedure Parameters

Parameter	Description
`tablespace_name`	The local simple tablespace to be cloned at the destination database and maintained by Streams. The tablespace must exist at the source database, but it must not exist at the destination database. A directory object must exist for the directory that contains the datafile for the tablespace. The user who invokes this procedure must have `READ` privilege on this directory object. If `NULL`, then the procedure raises an error.
`source_directory_object`	The directory object for the directory on the computer system running the source database into which the generated Data Pump export dump file and the datafile for the cloned tablespace are placed. These files remain in this directory after the procedure completes. If `NULL`, then the procedure raises an error.
`destination_directory_object`	The directory object for the directory on the computer system running the destination database into which the generated Data Pump export dump file and the datafile for the cloned tablespace are transferred. If `NULL`, then the procedure raises an error.

Table 106–19 (Cont.) MAINTAIN_SIMPLE_TABLESPACE Procedure Parameters

Parameter	Description
destination_database	The global name of the destination database. A database link from the source database to the destination database with the same name as the global name of the destination database must exist. If NULL, then the procedure raises an error.
setup_streams	If TRUE, then the procedure performs the necessary actions to maintain the tablespace directly. If FALSE, then the procedure does not perform the necessary actions to maintain the tablespace directly. Specify FALSE when this procedure is generating a script that you can edit and then run. The procedure raises an error if you specify FALSE and either of the following parameters is NULL: ▪ script_name ▪ script_directory_object
script_name	If non-NULL and the setup_streams parameter is FALSE, then specify the name of the script generated by this procedure. The script contains all of the statements used to maintain the specified tablespace. If a file with the specified script name exists in the specified directory for the script_directory_object parameter, then the procedure appends the statements to the existing file. If non-NULL and the setup_streams parameter is TRUE, then this procedure generates the specified script and performs the actions to maintain the specified tablespace directly. If NULL and the setup_streams parameter is TRUE, then this procedure does not generate a script and performs the actions to maintain the specified tablespace directly. If NULL and the setup_streams parameter is FALSE, then the procedure raises an error.
script_directory_object	The directory object for the directory on the local computer system into which the generated script is placed. If the script_name parameter is NULL, then the procedure ignores this parameter and does not generate a script. If NULL and the script_name parameter is non-NULL, then the procedure raises an error.

Table 106–19 (Cont.) MAINTAIN_SIMPLE_TABLESPACE Procedure Parameters

Parameter	Description
bi_directional	Specify TRUE to configure bi-directional replication between the current database and the database specified in destination_database. Both databases are configured as source and destination databases, a capture and apply process is configured at both databases, and propagations are configured between the databases to propagate messages. Specify FALSE to configure one way replication from the current database to the database specified in destination_database. A capture process is configured at the current database, a propagation is configured to propagate messages from the current database to the destination database, and an apply process is configured at the destination database.

Usage Notes

The specified tablespace must be a simple tablespace. A simple tablespace is a single, self-contained tablespace that uses only one datafile. A self-contained tablespace has no references from the tablespace pointing outside of the tablespace. For example, if an index in the tablespace is for a table in a different tablespace, then the tablespace is not self-contained. This procedure cannot be used for a non simple tablespace or a set of tablespaces.

DDL Changes Not Maintained

This procedure does not configure the Streams environment to maintain DDL changes to the tablespace nor to the database objects in the tablespace. For example, the Streams environment is not configured to replicate ALTER TABLESPACE statements on the tablespace, nor is it configured to replicate ALTER TABLE statements on tables in the tablespace. You can configure the Streams environment to maintain DDL changes manually or modify generated scripts to achieve this.

Additional Documentation for this Procedure

The documentation in the following sections applies to the MAINTAIN_SIMPLE_TABLESPACE procedure:

- Single Source and Bi-Directional Configurations on page 106-13
- Change Cycling on page 106-15
- Automatic Platform Conversion on page 106-16

Requirements for Running this Procedure

Meet the following requirements when run the MAINTAIN_SIMPLE_TABLESPACE procedure:

- Run the procedure at the source database.
- Both databases must be open during configuration. If the procedure is generating a script only, then the database specified in the destination_database parameter does not need to be open when you run the procedure, but both databases must be open when you run the generated script.
- The user who runs this procedure should be granted DBA role. This user must have the necessary privileges to complete the following actions:

  - Create ANYDATA queues, capture processes, propagations, and apply processes.
  - Specify supplemental logging
  - Run subprograms in the DBMS_STREAMS_ADM and DBMS_AQADM packages.
  - Access the database specified in the destination_database parameter through a database link. This database link should have the same name as the global name of the destination database.
  - Run subprograms in the DBMS_STREAMS_TABLESPACES_ADM package
  - The necessary privileges to run the CLONE_SIMPLE_TABLESPACE procedure in the DBMS_STREAMS_TABLESPACES_ADM package at the source database. See CLONE_SIMPLE_TABLESPACE Procedure on page 109-14 for the list of required privileges.
  - The necessary privileges to run the ATTACH_SIMPLE_TABLESPACE procedure in the DBMS_STREAMS_TABLESPACES_ADM package at the destination database. See ATTACH_SIMPLE_TABLESPACE Procedure on page 109-7 for the list of required privileges.

  To ensure that the user who runs this procedure has the necessary privileges, you should configure a Streams administrator at each database, and each database link should be should be created in the Streams administrator's schema.

- If the bi_directional parameter is set to TRUE, then the corresponding user at the destination database must be able to use a database link to access the source database. This database link should have the same name as the global name of the source database.
- Each specified directory object must be created using the SQL statement CREATE DIRECTORY, and the user who invokes this procedure must have READ and WRITE privilege on each one.
- The databases configured by this procedure must be Oracle Database 10*g* Release 2 databases when this procedure is run under the following conditions:
  - The procedure is run at an Oracle Database 10*g* Release 2 database.
  - The setup_streams parameter is set to TRUE to configure the Streams replication environment directly.
- The databases configured by this procedure must be Oracle Database 10*g* Release 1 or later databases when this procedure is run under the following conditions:
  - The procedure is run at an Oracle Database 10*g* Release 2 database.
  - The setup_streams parameter is set to FALSE in this procedure, and the replication environment is configured with a generated script.

  If the script configures an Oracle Database 10*g* Release 1 database, then the script must be modified so that it does not configure features that are available only in Oracle Database 10*g* Release 2, such as queue-to-queue propagation.
- If the procedure is run at an Oracle Database 10*g* Release 1 database, then the databases configured by the procedure must be Oracle Database 10*g* Release 1 or later databases.

See Also: *Oracle Streams Concepts and Administration* for information about configuring a Streams administrator

Default Values for Parameters Excluded From the MAINTAIN_SIMPLE_TABLESPACE Procedure

This procedure uses the default values for the parameters in the MAINTAIN_TABLESPACES procedure that do not exist in the MAINTAIN_SIMPLE_TABLESPACE procedure. For example, this procedure creates a capture process at the source database named capture, because that is the default value for the capture_name parameter in the MAINTAIN_TABLESPACES procedure.

See Also: MAINTAIN_TABLESPACES Procedure on page 106-101

Configuration Progress and Recoverability

When this procedure is run with the setup_streams parameter set to TRUE, metadata about its configuration actions is recorded in the following data dictionary views: DBA_RECOVERABLE_SCRIPT, DBA_RECOVERABLE_SCRIPT_PARAMS, DBA_RECOVERABLE_SCRIPT_BLOCKS, and DBA_RECOVERABLE_SCRIPT_ERRORS. If the procedure stops because it encounters an error, then you can use the RECOVER_OPERATION procedure to complete the configuration after you correct the conditions that caused the error.

Note: When this procedure is run with the setup_streams parameter set to FALSE, these views are not populated. Also, the views are not populated when a script generated by this procedure is run.

See Also: "RECOVER_OPERATION Procedure" on page 106-120

MAINTAIN_SIMPLE_TTS Procedure

This procedure clones a simple tablespace from a source database at a destination database and uses Streams to maintain this tablespace at both databases. This procedure can either perform these actions directly, or it can generate a script that performs these actions.

Run this procedure at the capture database. The capture database is the database that captures changes made to the source database.

Note:

- This procedure automatically excludes database objects that are not supported by Streams in the tablespace from the replication environment by adding rules to the negative rule set of each capture and apply process. Query the `DBA_STREAMS_UNSUPPORTED` data dictionary view to determine which database objects are not supported by Streams. If unsupported database objects are not excluded, then capture errors will result.
- This procedure replaces the deprecated `MAINTAIN_SIMPLE_TABLESPACE` procedure.

See Also: "Procedures That Configure a Streams Replication Environment" on page 106-11 for more information about this procedure

Syntax

```
DBMS_STREAMS_ADM.MAINTAIN_SIMPLE_TTS(
   tablespace_name              IN VARCHAR2,
   source_directory_object      IN VARCHAR2,
   destination_directory_object IN VARCHAR2,
   source_database              IN VARCHAR2,
   destination_database         IN VARCHAR2,
   perform_actions              IN BOOLEAN   DEFAULT TRUE,
   script_name                  IN VARCHAR2  DEFAULT NULL,
   script_directory_object      IN VARCHAR2  DEFAULT NULL,
   bi_directional               IN BOOLEAN   DEFAULT FALSE);
```

Parameters

See Also: "Common Parameters for the Configuration Procedures" on page 106-18 for descriptions of the procedure parameters that are not in Table 106–20

Table 106–20 MAINTAIN_SIMPLE_TTS Procedure Parameters

Parameter	Description
tablespace_name	The local simple tablespace to be cloned at the destination database and maintained by Streams. The tablespace must exist at the source database, but it must not exist at the destination database. A directory object must exist for the directory that contains the datafile for the tablespace. The user who invokes this procedure must have READ privilege on this directory object. If NULL, then the procedure raises an error.
source_directory_object	The directory object for the directory on the computer system running the source database into which the generated Data Pump export dump file and the datafile for the cloned tablespace are placed. These files remain in this directory after the procedure completes. If NULL, then the procedure raises an error.
destination_directory_object	The directory object for the directory on the computer system running the destination database into which the generated Data Pump export dump file and the datafile for the cloned tablespace are transferred. If NULL, then the procedure raises an error.

Usage Notes

The specified tablespace must be a simple tablespace. A simple tablespace is a single, self-contained tablespace that uses only one datafile. A self-contained tablespace has no references from the tablespace pointing outside of the tablespace. For example, if an index in the tablespace is for a table in a different tablespace, then the tablespace is not self-contained. This procedure cannot be used for a non simple tablespace or a set of tablespaces.

DDL Changes Not Maintained

This procedure does not configure the Streams environment to maintain DDL changes to the tablespace nor to the database objects in the tablespace. For example, the Streams environment is not configured to replicate ALTER TABLESPACE statements on the tablespace, nor is it configured to replicate ALTER TABLE statements on tables in the tablespace. You can configure the Streams environment to maintain DDL changes manually or modify generated scripts to achieve this.

Additional Privileges Required by the MAINTAIN_SIMPLE_TTS Procedure

In addition to the required privileges described in "Requirements for Running These Procedures" on page 106-17, the user who runs the MAINTAIN_SIMPLE_TTS procedure must have the necessary privileges to complete the following actions:

- Run subprograms in the DBMS_STREAMS_TABLESPACES_ADM package
- The necessary privileges to run the CLONE_SIMPLE_TABLESPACE procedure in the DBMS_STREAMS_TABLESPACES_ADM package at the source database. See CLONE_SIMPLE_TABLESPACE Procedure on page 109-14 for the list of required privileges.
- The necessary privileges to run the ATTACH_SIMPLE_TABLESPACE procedure in the DBMS_STREAMS_TABLESPACES_ADM package at the destination database. See

ATTACH_SIMPLE_TABLESPACE Procedure on page 109-7 for the list of required privileges.

Default Values for Parameters Excluded From the MAINTAIN_SIMPLE_TTS Procedure

This procedure uses the default values for the parameters in the `MAINTAIN_TTS` procedure that do not exist in the `MAINTAIN_SIMPLE_TTS` procedure. For example, this procedure automatically generates the capture process name, because `NULL` is the default value for the `capture_name` parameter in the `MAINTAIN_TTS` procedure, and the procedure generates the capture process name when `NULL` is specified for `capture_name`.

See Also: MAINTAIN_TTS Procedure on page 106-108

MAINTAIN_TABLES Procedure

This procedure configures a Streams environment that replicates changes to specified tables between two databases. This procedure can either configure the environment directly, or it can generate a script that configures the environment.

Run this procedure at the capture database. The capture database is the database that captures changes made to the source database.

This procedure is overloaded. One `table_names` parameter is type `VARCHAR2` and the other `table_name` parameters is type `DBMS_UTILITY.UNCL_ARRAY`. These parameters enable you to enter the list of tables in different ways and are mutually exclusive.

Note: If the `bi_directional` parameter is set to `TRUE`, then do not allow data manipulation language (DML) or data definition language (DDL) changes to the shared database objects at the destination database while the `MAINTAIN_TABLES` procedure, or the script generated by the procedure, is running. This restriction does not apply to the source database.

See Also: "Procedures That Configure a Streams Replication Environment" on page 106-11 for more information about this procedure

Syntax

```
DBMS_STREAMS_ADM.MAINTAIN_TABLES(
   table_names                  IN VARCHAR2,
   source_directory_object      IN VARCHAR2,
   destination_directory_object IN VARCHAR2,
   source_database              IN VARCHAR2,
   destination_database         IN VARCHAR2,
   perform_actions              IN BOOLEAN   DEFAULT TRUE,
   script_name                  IN VARCHAR2  DEFAULT NULL,
   script_directory_object      IN VARCHAR2  DEFAULT NULL,
   dump_file_name               IN VARCHAR2  DEFAULT NULL,
   capture_name                 IN VARCHAR2  DEFAULT NULL,
   capture_queue_table          IN VARCHAR2  DEFAULT NULL,
   capture_queue_name           IN VARCHAR2  DEFAULT NULL,
   capture_queue_user           IN VARCHAR2  DEFAULT NULL,
   propagation_name             IN VARCHAR2  DEFAULT NULL,
   apply_name                   IN VARCHAR2  DEFAULT NULL,
   apply_queue_table            IN VARCHAR2  DEFAULT NULL,
   apply_queue_name             IN VARCHAR2  DEFAULT NULL,
   apply_queue_user             IN VARCHAR2  DEFAULT NULL,
   log_file                     IN VARCHAR2  DEFAULT NULL,
   bi_directional               IN BOOLEAN   DEFAULT FALSE,
   include_ddl                  IN BOOLEAN   DEFAULT FALSE,
   instantiation                IN INTEGER   DEFAULT
                                             DBMS_STREAMS_ADM.INSTANTIATION_TABLE);

DBMS_STREAMS_ADM.MAINTAIN_TABLES(
   table_names                  IN DBMS_UTILITY.UNCL_ARRAY,
   source_directory_object      IN VARCHAR2,
   destination_directory_object IN VARCHAR2,
   source_database              IN VARCHAR2,
```

```
destination_database         IN VARCHAR2,
perform_actions              IN BOOLEAN   DEFAULT TRUE,
script_name                  IN VARCHAR2  DEFAULT NULL,
script_directory_object      IN VARCHAR2  DEFAULT NULL,
dump_file_name               IN VARCHAR2  DEFAULT NULL,
capture_name                 IN VARCHAR2  DEFAULT NULL,
capture_queue_table          IN VARCHAR2  DEFAULT NULL,
capture_queue_name           IN VARCHAR2  DEFAULT NULL,
capture_queue_user           IN VARCHAR2  DEFAULT NULL,
propagation_name             IN VARCHAR2  DEFAULT NULL,
apply_name                   IN VARCHAR2  DEFAULT NULL,
apply_queue_table            IN VARCHAR2  DEFAULT NULL,
apply_queue_name             IN VARCHAR2  DEFAULT NULL,
apply_queue_user             IN VARCHAR2  DEFAULT NULL,
log_file                     IN VARCHAR2  DEFAULT NULL,
bi_directional               IN BOOLEAN   DEFAULT FALSE,
include_ddl                  IN BOOLEAN   DEFAULT FALSE,
instantiation                IN INTEGER   DEFAULT
                                          DBMS_STREAMS_ADM.INSTANTIATION_TABLE);
```

Parameters

See Also: "Common Parameters for the Configuration Procedures" on page 106-18 for descriptions of the procedure parameters that are not in Table 106–21

Table 106–21 MAINTAIN_TABLES Procedure Parameters

Parameter	Description
`table_names`	The tables to be configured for replication and maintained by Streams after configuration. The tables can be specified in the following ways: ▪ Comma-delimited list of type `VARCHAR2` ▪ A PL/SQL index-by table of type `DBMS_UTILITY.UNCL_ARRAY`, where each element is the name of a table. The first table should be in position 1. The last position must be `NULL`. Each table should be specified as `[`*`schema_name`*`.]`*`table_name`*. For example, `hr.employees`. If the schema is not specified, then the current user is the default. If a specified table does not exist at the source database, then the procedure raises an error. If this parameter is set to `NULL`, then the procedure raises an error.

Table 106–21 (Cont.) MAINTAIN_TABLES Procedure Parameters

Parameter	Description
source_directory_object	The directory object for the directory on the computer system running the source database into which the generated Data Pump export dump file is placed. This file remain in this directory after the procedure completes. This parameter is ignored if instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_TABLE_NETWORK or DBMS_STREAMS_ADM.INSTANTIATION_NONE. In this case, specify NULL for the source_directory_object parameter. If NULL and instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_TABLE, then the procedure raises an error.
destination_directory_object	The directory object for the directory on the computer system running the destination database into which the generated Data Pump export dump file is transferred. This parameter is ignored if instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_TABLE_NETWORK or DBMS_STREAMS_ADM.INSTANTIATION_NONE. In this case, specify NULL for the destination_directory_object parameter. If NULL and instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_TABLE, then the procedure raises an error.
dump_file_name	The name of the Data Pump export dump file. If a file with the specified file name exists in the specified directory for the source_directory_object or destination_directory_object parameter, then the procedure raises an error. This parameter is ignored if instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_TABLE_NETWORK or DBMS_STREAMS_ADM.INSTANTIATION_NONE. If NULL and instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_TABLE, then the export dump file name is generated by the system. In this case, the export dump file name is expat*nn*.dmp, where *nn* is a sequence number. The sequence number is incremented to produce an export dump file with a unique name in the source directory.
capture_queue_user	The name of the user who requires ENQUEUE and DEQUEUE privileges for the queue at the source database. This user also is configured as a secure queue user of the queue. The queue user cannot grant these privileges to other users because they are not granted with the GRANT option. If NULL, then the procedure does not grant any privileges. You can also grant queue privileges to the appropriate users using the DBMS_AQADM package.

Table 106–21 (Cont.) MAINTAIN_TABLES Procedure Parameters

Parameter	Description
log_file	The name of the Data Pump export log file. This log file is placed in the same directory as the Data Pump export dump file. This parameter is ignored if instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_TABLE_NETWORK or DBMS_STREAMS_ADM.INSTANTIATION_NONE. If NULL and instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_TABLE, then the log file name is the same name as the export dump file name with an extension of .clg.
instantiation	Specify whether to perform instantiation and, if instantiation is performed, the type of instantiation: DBMS_STREAMS_ADM.INSTANTIATION_TABLE performs a full Data Pump export at the source database and a Data Pump import of the export dump file at the destination database. The instantiation SCN is set for the shared database objects during import. If the instantiation parameter is set to this value, then the user who runs this procedure must have EXECUTE privilege on the DBMS_FILE_TRANSFER package. DBMS_STREAMS_ADM.INSTANTIATION_TABLE_NETWORK performs a full network Data Pump import. A network import means that Data Pump performs the import without using an export dump file. The instantiation SCN is set for the shared database objects during import. If the instantiation parameter is set to this value, then a database link from the destination database to the source database with the same name as the global name of the source database must exist and must be accessible to the user who runs the procedure. DBMS_STREAMS_ADM.INSTANTIATION_NONE does not perform an instantiation. This setting is valid only if the perform_actions parameter is set to FALSE, and the procedure generates a configuration script. In this case, the configuration script does not perform an instantiation and does not set the instantiation SCN for each shared database object. Instead, you must perform the instantiation and ensure that instantiation SCN values are set properly. If this parameter is set to DBMS_STREAMS_ADM.INSTANTIATION_TABLE or DBMS_STREAMS_ADM.INSTANTIATION_TABLE_NETWORK, then the database objects being instantiated must exist at the source database, but these database objects must not exist at the destination database.

MAINTAIN_TABLESPACES Procedure

This procedure clones a set of tablespaces from a source database at a destination database and uses Streams to maintain these tablespaces at both databases. This procedure can either perform these actions directly, or it can generate a script that performs these actions. Run this procedure at the source database.

Note: This procedure is deprecated. It is replaced by the `MAINTAIN_TTS` procedure.

See Also:

- "Deprecated Subprograms" on page 106-4
- MAINTAIN_TTS Procedure on page 106-108

Syntax

```
DBMS_STREAMS_ADM.MAINTAIN_TABLESPACES(
   tablespace_names             IN DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET,
   source_directory_object      IN VARCHAR2,
   destination_directory_object IN VARCHAR2,
   destination_database         IN VARCHAR2,
   setup_streams                IN BOOLEAN   DEFAULT TRUE,
   script_name                  IN VARCHAR2  DEFAULT NULL,
   script_directory_object      IN VARCHAR2  DEFAULT NULL,
   dump_file_name               IN VARCHAR2  DEFAULT NULL,
   source_queue_table           IN VARCHAR2  DEFAULT 'streams_queue_table',
   source_queue_name            IN VARCHAR2  DEFAULT 'streams_queue',
   source_queue_user            IN VARCHAR2  DEFAULT NULL,
   destination_queue_table      IN VARCHAR2  DEFAULT 'streams_queue_table',
   destination_queue_name       IN VARCHAR2  DEFAULT 'streams_queue',
   destination_queue_user       IN VARCHAR2  DEFAULT NULL,
   capture_name                 IN VARCHAR2  DEFAULT 'capture',
   propagation_name             IN VARCHAR2  DEFAULT NULL,
   apply_name                   IN VARCHAR2  DEFAULT NULL,
   log_file                     IN VARCHAR2  DEFAULT NULL,
   bi_directional               IN BOOLEAN   DEFAULT FALSE,
   include_ddl                  IN BOOLEAN   DEFAULT FALSE);
```

Parameters

Table 106–22 MAINTAIN_TABLESPACES Procedure Parameters

Parameter	Description
`tablespace_names`	The local tablespace set to be cloned at the destination database and maintained by Streams. The tablespaces in the tablespace set must exist at the source database, but these tablespaces must not exist at the destination database. A directory object must exist for each directory that contains the datafiles for the tablespace set. The user who invokes this procedure must have `READ` privilege on these directory objects. If `NULL`, then the procedure raises an error. **See Also:** TABLESPACE_SET Type on page 109-4

Table 106–22 (Cont.) MAINTAIN_TABLESPACES Procedure Parameters

Parameter	Description
source_directory_object	The directory object for the directory on the computer system running the source database into which the generated Data Pump export dump file and the datafiles that comprise the cloned tablespace set are placed. These files remain in this directory after the procedure completes. If NULL, then the procedure raises an error.
destination_directory_object	The directory object for the directory on the computer system running the destination database into which the generated Data Pump export dump file and the datafiles that comprise the cloned tablespace set are transferred. If NULL, then the procedure raises an error.
destination_database	The global name of the destination database. A database link from the source database to the destination database with the same name as the global name of the destination database must exist and must be accessible to the user who runs the procedure. If NULL, then the procedure raises an error.
setup_streams	If TRUE, then the procedure performs the necessary actions to maintain the tablespaces directly. If FALSE, then the procedure does not perform the necessary actions to maintain the tablespaces directly. Specify FALSE when this procedure is generating a script that you can edit and then run. The procedure raises an error if you specify FALSE and either of the following parameters is NULL: ■ script_name ■ script_directory_object
script_name	If non-NULL and the setup_streams parameter is FALSE, then specify the name of the script generated by this procedure. The script contains all of the statements used to maintain the specified tablespace set. If a file with the specified script name exists in the specified directory for the script_directory_object parameter, then the procedure appends the statements to the existing file. If non-NULL and the setup_streams parameter is TRUE, then this procedure generates the specified script and performs the actions to maintain the specified tablespace directly. If NULL and the setup_streams parameter is TRUE, then this procedure does not generate a script and performs the actions to maintain the specified tablespace set directly. If NULL and the setup_streams parameter is FALSE, then the procedure raises an error.

Table 106–22 (Cont.) MAINTAIN_TABLESPACES Procedure Parameters

Parameter	Description
script_directory_object	The directory object for the directory on the local computer system into which the generated script is placed. If the script_name parameter is NULL, then the procedure ignores this parameter and does not generate a script. If NULL and the script_name parameter is non-NULL, then the procedure raises an error.
dump_file_name	The name of the Data Pump export dump file that contains the specified tablespace set. If a file with the specified file name exists in the specified directory for the source_directory_object or destination_directory_object parameter, then the procedure raises an error. If NULL, then the export dump file name is generated by the system. In this case, the export dump file name is expat*nn*.dmp, where *nn* is a sequence number. The sequence number is incremented to produce an export dump file with a unique name in the source directory.
source_queue_table	The name of the queue table for the queue at the source database, specified as [*schema_name*.]*queue_table_name*. For example, strmadmin.streams_queue_table. If the schema is not specified, then the current user is the default.
source_queue_name	The name of the queue at the source database that will function as the ANYDATA queue, specified as [*schema_name*.]*queue_name*. For example, strmadmin.streams_queue. If the schema is not specified, then the queue table owner is the default. The queue owner automatically has privileges to perform all queue operations on the queue.
source_queue_user	The name of the user who requires ENQUEUE and DEQUEUE privileges for the queue at the source database. This user also is configured as a secure queue user of the queue. The queue user cannot grant these privileges to other users because they are not granted with the GRANT option. If NULL, then the procedure does not grant any privileges. You can also grant queue privileges to the appropriate users using the DBMS_AQADM package.
destination_queue_table	The name of the queue table for the queue at the destination database, specified as [*schema_name*.]*queue_table_name*. For example, strmadmin.streams_queue_table. If the schema is not specified, then the current user is the default.
destination_queue_name	The name of the queue at the destination database that will function as the ANYDATA queue, specified as [*schema_name*.]*queue_name*. For example, strmadmin.streams_queue. If the schema is not specified, then the queue table owner is the default. The queue owner automatically has privileges to perform all queue operations on the queue.

Table 106–22 (Cont.) MAINTAIN_TABLESPACES Procedure Parameters

Parameter	Description
destination_queue_user	The name of the user who requires ENQUEUE and DEQUEUE privileges for the queue at the destination database. This user also is configured as a secure queue user of the queue. The queue user cannot grant these privileges to other users because they are not granted with the GRANT option. If NULL, then the procedure does not grant any privileges. You can also grant queue privileges to the appropriate users using the DBMS_AQADM package.
capture_name	The name of each capture process configured to capture changes to the database objects in the tablespace set. Do not specify an owner. If the specified name matches the name of an existing capture process, then the procedure uses the existing capture process and adds the rules for capturing changes to the database objects in the tablespace set to the positive capture process rule set. **Note:** The capture process name cannot be altered after the capture process is created.
propagation_name	The name of each propagation configured to propagate changes to the database objects in the tablespace set. Do not specify an owner. If the specified name matches the name of an existing propagation, then the procedure uses the existing propagation and adds the rules for propagating changes to the database objects in the tablespace set to the positive propagation rule set. If NULL, then the system generates a name for each propagation it creates. **Note:** The propagation name cannot be altered after the propagation is created.
apply_name	The name of each apply process configured to apply changes to the database objects in the tablespace set. Do not specify an owner. If the specified name matches the name of an existing apply process, then the procedure uses the existing apply process and adds the rules for applying changes to the database objects in the tablespace set to the positive apply process rule set. The specified name must not match the name of an existing messaging client at the destination database. If NULL, then the system generates a name for each apply process it creates. **Note:** The apply process name cannot be altered after the apply process is created.
log_file	The name of the Data Pump export log file. This log file is placed in the same directory as the Data Pump export dump file. If NULL, then the log file name is the same name as the export dump file name with an extension of .clg.

Table 106–22 (Cont.) MAINTAIN_TABLESPACES Procedure Parameters

Parameter	Description
bi_directional	Specify TRUE to configure bi-directional replication between the current database and the database specified in destination_database. Both databases are configured as source and destination databases, a capture and apply process is configured at both databases, and propagations are configured between the databases to propagate messages. Specify FALSE to configure one way replication from the current database to the database specified in destination_database. A capture process is configured at the current database, a propagation is configured to propagate messages from the current database to the destination database, and an apply process is configured at the destination database.
include_ddl	Specify TRUE to configure a Streams replication environment that maintains both DML and DDL changes. Specify FALSE to configure a Streams replication environment that maintains DML changes only. When this parameter is set to FALSE, DDL changes, such as ALTER TABLE, will not be replicated.

Usage Notes

The specified set of tablespaces must be self-contained. In this context "self-contained" means that there are no references from inside the set of tablespaces pointing outside of the set of tablespaces. For example, if a partitioned table is partially contained in the set of tablespaces, then the set of tablespaces is not self-contained.

> **See Also:** *Oracle Database Administrator's Guide* for more information about self-contained tablespace sets

Additional Documentation for this Procedure

The documentation in the following sections applies to the MAINTAIN_TABLESPACES procedure:

- Single Source and Bi-Directional Configurations on page 106-13
- Change Cycling on page 106-15
- Automatic Platform Conversion on page 106-16

Requirements for Running this Procedure

Meet the following requirements when run the MAINTAIN_TABLESPACES procedure:

- Run the procedure at the source database.
- Both databases must be open during configuration. If the procedure is generating a script only, then the database specified in the destination_database parameter does not need to be open when you run the procedure, but both databases must be open when you run the generated script.
- The user who runs this procedure should be granted DBA role. This user must have the necessary privileges to complete the following actions:
  - Create ANYDATA queues, capture processes, propagations, and apply processes.

  - Specify supplemental logging
  - Run subprograms in the DBMS_STREAMS_ADM and DBMS_AQADM packages.
  - Access the database specified in the destination_database parameter through a database link. This database link should have the same name as the global name of the destination database.
  - Run subprograms in the DBMS_STREAMS_TABLESPACES_ADM package
  - The necessary privileges to run the CLONE_TABLESPACES procedure in the DBMS_STREAMS_TABLESPACES_ADM package at the source database. See CLONE_TABLESPACES Procedure on page 109-16 for the list of required privileges.
  - The necessary privileges to run the ATTACH_TABLESPACES procedure in the DBMS_STREAMS_TABLESPACES_ADM package at the destination database. See ATTACH_TABLESPACES Procedure on page 109-9 for the list of required privileges.

  To ensure that the user who runs this procedure has the necessary privileges, you should configure a Streams administrator at each database, and each database link should be should be created in the Streams administrator's schema.

- If the bi_directional parameter is set to TRUE, then the corresponding user at the destination database must be able to use a database link to access the source database. This database link should have the same name as the global name of the source database.
- Each specified directory object must be created using the SQL statement CREATE DIRECTORY, and the user who invokes this procedure must have READ and WRITE privilege on each one.
- The databases configured by this procedure must be Oracle Database 10*g* Release 2 databases when this procedure is run under the following conditions:
  - The procedure is run at an Oracle Database 10*g* Release 2 database.
  - The setup_streams parameter is set to TRUE to configure the Streams replication environment directly.
- The databases configured by this procedure must be Oracle Database 10*g* Release 1 or later databases when this procedure is run under the following conditions:
  - The procedure is run at an Oracle Database 10*g* Release 2 database.
  - The setup_streams parameter is set to FALSE in this procedure, and the replication environment is configured with a generated script.

  If the script configures an Oracle Database 10*g* Release 1 database, then the script must be modified so that it does not configure features that are available only in Oracle Database 10*g* Release 2, such as queue-to-queue propagation.
- If the procedure is run at an Oracle Database 10*g* Release 1 database, then the databases configured by the procedure must be Oracle Database 10*g* Release 1 or later databases.

See Also: *Oracle Streams Concepts and Administration* for information about configuring a Streams administrator

Configuration Progress and Recoverability

When this procedure is run with the setup_streams parameter set to TRUE, metadata about its configuration actions is recorded in the following data dictionary

views: DBA_RECOVERABLE_SCRIPT, DBA_RECOVERABLE_SCRIPT_PARAMS, DBA_RECOVERABLE_SCRIPT_BLOCKS, and DBA_RECOVERABLE_SCRIPT_ERRORS. If the procedure stops because it encounters an error, then you can use the RECOVER_OPERATION procedure to complete the configuration after you correct the conditions that caused the error.

Note: When this procedure is run with the setup_streams parameter set to FALSE, these views are not populated. Also, the views are not populated when a script generated by this procedure is run.

See Also: "RECOVER_OPERATION Procedure" on page 106-120

MAINTAIN_TTS Procedure

This procedure clones a set of tablespaces from a source database at a destination database and uses Streams to maintain these tablespaces at both databases. This procedure can either perform these actions directly, or it can generate a script that performs these actions.

Run this procedure at the capture database. The capture database is the database that captures changes made to the source database.

Note:

- This procedure automatically excludes database objects that are not supported by Streams in the tablespaces from the replication environment by adding rules to the negative rule set of each capture and apply process. Query the `DBA_STREAMS_UNSUPPORTED` data dictionary view to determine which database objects are not supported by Streams. If unsupported database objects are not excluded, then capture errors will result.
- This procedure replaces the deprecated `MAINTAIN_TABLESPACES` procedure.

See Also: "Procedures That Configure a Streams Replication Environment" on page 106-11 for more information about this procedure

Syntax

```
DBMS_STREAMS_ADM.MAINTAIN_TTS(
   tablespace_names             IN DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET,
   source_directory_object      IN VARCHAR2,
   destination_directory_object IN VARCHAR2,
   source_database              IN VARCHAR2,
   destination_database         IN VARCHAR2,
   perform_actions              IN BOOLEAN   DEFAULT TRUE,
   script_name                  IN VARCHAR2  DEFAULT NULL,
   script_directory_object      IN VARCHAR2  DEFAULT NULL,
   dump_file_name               IN VARCHAR2  DEFAULT NULL,
   capture_name                 IN VARCHAR2  DEFAULT NULL,
   capture_queue_table          IN VARCHAR2  DEFAULT NULL,
   capture_queue_name           IN VARCHAR2  DEFAULT NULL,
   capture_queue_user           IN VARCHAR2  DEFAULT NULL,
   propagation_name             IN VARCHAR2  DEFAULT NULL,
   apply_name                   IN VARCHAR2  DEFAULT NULL,
   apply_queue_table            IN VARCHAR2  DEFAULT NULL,
   apply_queue_name             IN VARCHAR2  DEFAULT NULL,
   apply_queue_user             IN VARCHAR2  DEFAULT NULL,
   log_file                     IN VARCHAR2  DEFAULT NULL,
   bi_directional               IN BOOLEAN   DEFAULT FALSE,
   include_ddl                  IN BOOLEAN   DEFAULT FALSE);
```

Parameters

See Also: "Common Parameters for the Configuration Procedures" on page 106-18 for descriptions of the procedure parameters that are not in Table 106–23

Table 106–23 MAINTAIN_TTS Procedure Parameters

Parameter	Description
tablespace_names	The local tablespace set to be cloned at the destination database and maintained by Streams. The tablespaces in the tablespace set must exist at the source database, but these tablespaces must not exist at the destination database. A directory object must exist for each directory that contains the datafiles for the tablespace set. The user who invokes this procedure must have READ privilege on these directory objects. If NULL, then the procedure raises an error. **See Also:** TABLESPACE_SET Type on page 109-4
source_directory_object	The directory object for the directory on the computer system running the source database into which the generated Data Pump export dump file and the datafiles that comprise the cloned tablespace set are placed. These files remain in this directory after the procedure completes. If NULL, then the procedure raises an error.
destination_directory_object	The directory object for the directory on the computer system running the destination database into which the generated Data Pump export dump file and the datafiles that comprise the cloned tablespace set are transferred. If NULL, then the procedure raises an error.
dump_file_name	The name of the Data Pump export dump file that contains the specified tablespace set. If a file with the specified file name exists in the specified directory for the source_directory_object or destination_directory_object parameter, then the procedure raises an error. If NULL, then the export dump file name is generated by the system. In this case, the export dump file name is expat*nn*.dmp, where *nn* is a sequence number. The sequence number is incremented to produce an export dump file with a unique name in the source directory.
log_file	The name of the Data Pump export log file. This log file is placed in the same directory as the Data Pump export dump file. If NULL, then the log file name is the same name as the export dump file name with an extension of .clg.

Usage Notes

The specified set of tablespaces must be self-contained. In this context "self-contained" means that there are no references from inside the set of tablespaces pointing outside

of the set of tablespaces. For example, if a partitioned table is partially contained in the set of tablespaces, then the set of tablespaces is not self-contained.

> **See Also:** *Oracle Database Administrator's Guide* for more information about self-contained tablespace sets

Additional Privileges Required by the MAINTAIN_TTS Procedure

In addition to the required privileges described in "Requirements for Running These Procedures" on page 106-17, the user who runs the `MAINTAIN_TTS` procedure must have the necessary privileges to complete the following actions:

- Run subprograms in the `DBMS_STREAMS_TABLESPACES_ADM` package
- The necessary privileges to run the `CLONE_TABLESPACES` procedure in the `DBMS_STREAMS_TABLESPACES_ADM` package at the source database. See CLONE_TABLESPACES Procedure on page 109-16 for the list of required privileges.
- The necessary privileges to run the `ATTACH_TABLESPACES` procedure in the `DBMS_STREAMS_TABLESPACES_ADM` package at the destination database. See ATTACH_TABLESPACES Procedure on page 109-9 for the list of required privileges.

POST_INSTANTIATION_SETUP Procedure

This procedure performs the actions required after instantiation to configure a Streams replication environment.

Run this procedure at the capture database. The capture database is the database that captures changes made to the source database.

To complete the Streams replication configuration, follow these steps:

1. Run the PRE_INSTANTIATION_SETUP procedure at the source database.
2. Perform any necessary instantiation actions.
3. Run the POST_INSTANTIATION_SETUP procedure at the source database.

Typically, the Streams replication environment configured using these steps serves one of the following purposes:

- Replicates changes to shared database objects to keep the database objects synchronized at different databases.
- Replicates changes to database objects during a database maintenance operation, such migrating a database to a different platform. In this case, use the CLEANUP_INSTANTIATION_SETUP procedure to remove the replication environment after the maintenance operation is complete.

Attention: When the POST_INSTANTIATION_SETUP procedure is run, the parameter values must match the parameter values specified when the corresponding PRE_INSTANTIATION_SETUP procedure was run, except for the values of the following parameters: perform_actions, script_name, script_directory_object, and start_processes.

Note: A capture process never captures changes in the SYS, SYSTEM, or CTXSYS schemas. This procedure does not configure replication for these schemas.

See Also:

- "PRE_INSTANTIATION_SETUP Procedure" on page 106-115
- "CLEANUP_INSTANTIATION_SETUP Procedure" on page 106-74
- "Procedures That Configure a Streams Replication Environment" on page 106-11 for more information about this procedure
- *Oracle Streams Replication Administrator's Guide* for information about setting up a Streams replication environment
- *Oracle Streams Concepts and Administration* for information about completing database maintenance operations

Syntax

```
DBMS_STREAMS_ADM.POST_INSTANTIATION_SETUP(
   maintain_mode           IN VARCHAR2,
```

```
  tablespace_names        IN DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET,
  source_database         IN VARCHAR2,
  destination_database    IN VARCHAR2,
  perform_actions         IN BOOLEAN          DEFAULT TRUE,
  script_name             IN VARCHAR2         DEFAULT NULL,
  script_directory_object IN VARCHAR2         DEFAULT NULL,
  capture_name            IN VARCHAR2         DEFAULT NULL,
  capture_queue_table     IN VARCHAR2         DEFAULT NULL,
  capture_queue_name      IN VARCHAR2         DEFAULT NULL,
  capture_queue_user      IN VARCHAR2         DEFAULT NULL,
  propagation_name        IN VARCHAR2         DEFAULT NULL,
  apply_name              IN VARCHAR2         DEFAULT NULL,
  apply_queue_table       IN VARCHAR2         DEFAULT NULL,
  apply_queue_name        IN VARCHAR2         DEFAULT NULL,
  apply_queue_user        IN VARCHAR2         DEFAULT NULL,
  bi_directional          IN BOOLEAN          DEFAULT FALSE,
  include_ddl             IN BOOLEAN          DEFAULT FALSE,
  start_processes         IN BOOLEAN          DEFAULT FALSE,
  instantiation_scn       IN NUMBER           DEFAULT NULL,
  exclude_schemas         IN VARCHAR2         DEFAULT NULL,
  exclude_flags           IN BINARY_INTEGER   DEFAULT NULL);
```

Parameters

See Also: "Common Parameters for the Configuration Procedures" on page 106-18 for descriptions of the procedure parameters that are not in Table 106–24

Table 106–24 POST_INSTANTIATION_SETUP Procedure Parameters

Parameter	Description
`maintain_mode`	Specify one of the following: ▪ `GLOBAL` to maintain the entire database by configuring replication between the local database and the database specified in the `destination_database` parameter ▪ `TRANSPORTABLE TABLESPACES` to maintain a set of tablespaces by configuring replication between the local database and the database specified in the `destination_database` parameter
`tablespace_names`	If `maintain_mode` is set to `TRANSPORTABLE TABLESPACES`, then specify the local tablespace set to be cloned at the destination database and maintained by Streams. The tablespaces in the tablespace set must exist at the source database, but these tablespaces must not exist at the destination database. Also, a directory object must exist for each directory that contains the datafiles for the tablespace set. The user who invokes this procedure must have `READ` privilege on these directory objects. If `maintain_mode` is set to `GLOBAL`, then specify `NULL`. **See Also:** TABLESPACE_SET Type on page 109-4

Table 106–24 (Cont.) POST_INSTANTIATION_SETUP Procedure Parameters

Parameter	Description
start_processes	If TRUE, then the procedure starts each capture process and apply process. Any disabled capture or apply process created by the PRE_INSTANTITAION_SETUP procedure also is started. If FALSE, then the procedure does not start any capture processes or apply processes.
instantiation_scn	Specify the instantiation SCN for the database objects at the destination database if the instantiation SCN was not set during instantiation. The instantiation SCN is not set automatically during RMAN instantiations, but the correct instantiation SCN value should be determined during an RMAN instantiation. See the *Oracle Streams Replication Administrator's Guide* for more information. Specify NULL if the instantiation SCN was set for the database objects at the destination database during instantiation. The instantiation SCN can be set during export/import instantiations.
exclude_schemas	A comma-delimited list of schemas to exclude from the Streams configuration. Schema rules are added to the negative rule sets of each capture process to exclude these schemas. Specify an asterisk (*) to exclude all of the schemas in the database. If NULL, then the procedure does not exclude any schemas in the database. This parameter is valid only if the MAINTAIN_MODE parameter is set to GLOBAL. If the MAINTAIN_MODE parameter is set to TRANSPORTABLE TABLESPACES, then the procedure ignores this parameter.
exclude_flags	Specify what is excluded from the replication configuration in the schemas specified by the exclude_schemas parameter. This parameter works the same way in the PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures. See "Usage Notes" on page 106-117 for the PRE_INSTANTIATION_SETUP procedure for more information.

Usage Notes

The following sections contain usage notes for this procedure.

Self-Contained Tablespace Sets

If the maintain_mode parameter is set to TRANSPORTABLE TABLESPACES, then the specified set of tablespaces must be self-contained. In this context "self-contained" means that there are no references from inside the set of tablespaces pointing outside of the set of tablespaces. For example, if a partitioned table is partially contained in the set of tablespaces, then the set of tablespaces is not self-contained.

> **See Also:** *Oracle Database Administrator's Guide* for more information about self-contained tablespace sets

Destination Database Renamed During RMAN Database Instantiation

If the maintain_mode parameter is set to GLOBAL, then database instantiation is required before running the POST_INSTANTIATION_SETUP procedure. If the RMAN

DUPLICATE or RMAN CONVERT DATABASE command is used for database instantiation, then the global name of the destination database can be renamed to the global name of the source database during instantiation. In this case, before you run the POST_INSTANTIATION_SETUP procedure, complete the following steps:

1. Rename the global name of the destination database back to the name specified in the destination_database parameter.
2. At the destination database, drop and recreate any loopback database links that existed on the source and were cloned on the destination database. For example, suppose the source database dbs1.net has a database link that refers to itself. Suppose the destination database is dbs2.net. At the destination database, drop and recreate this database link as a loopback database link that refers to itself (dbs2.net).
3. At the destination database, drop any database links that were cloned from the source database and are from the source database to the destination database. For example, if the source database is dbs1.net and the destination database is dbs2.net, then drop any database links on the destination database that are from dbs1.net to dbs2.net.
4. Create a database link from the destination database to the source database with the same name as the global name of the source database. The database link must be accessible to the Streams administrator at the destination database.

   This database link is required because the POST_INSTANTIATION_SETUP procedure runs the SET_GLOBAL_INSTANTIATION_SCN procedure in the DBMS_APPLY_ADM package at the destination database, and the SET_GLOBAL_INSTANTIATION_SCN procedure requires the database link. The instantiation SCN is set to the value specified in the instantiation_scn parameter of the POST_INSTANTIATION_SETUP procedure.

Note: When the RMAN DUPLICATE or CONVERT DATABASE command is used for database instantiation, the destination database cannot be the capture database.

Streams Components Removed From the Destination Database

If the maintain_mode parameter is set to GLOBAL, then database instantiation is required before running the POST_INSTANTIATION_SETUP procedure. During database instantiation, Streams components created by the PRE_INSTANTIATION_SETUP procedure, such as Streams clients and queues, can be copied from the source database to the destination database. The POST_INSTANTIATION_SETUP procedure removes the Stream components created by the PRE_INSTANTIATION_SETUP procedure from the destination database.

In some cases, rule sets and rules created by the PRE_INSTANTIATION_SETUP procedure might not be removed from the destination database. The POST_INSTANTIATION_SETUP procedure does not associate these rule sets and rules with any Stream clients in the destination database. Optionally, you can remove these rule sets and rules from the destination database after the POST_INSTANTIATION_SETUP procedure, or the script generated by the procedure, completes.

Note: The POST_INSTANTIATION_SETUP procedure only removes Streams components that were created by the PRE_INSTANTIATION_SETUP procedure. It does not remove Streams components that were created in a different way.

PRE_INSTANTIATION_SETUP Procedure

This procedure performs the actions required before instantiation to configure a Streams replication environment.

Run this procedure at the capture database. The capture database is the database that captures changes made to the source database.

To complete the Streams replication configuration, follow these steps:

1. Run the `PRE_INSTANTIATION_SETUP` procedure at the database that will be the source database in the Stream replication environment.
2. Perform any necessary instantiation actions.
3. Run the `POST_INSTANTIATION_SETUP` procedure at the source database.

Typically, the Streams replication environment configured using these steps serves one of the following purposes:

- Replicates changes to shared database objects to keep the database objects synchronized at different databases.
- Replicates changes to database objects during a database maintenance operation, such migrating a database to a different platform. In this case, use the `CLEANUP_INSTANTIATION_SETUP` procedure to remove the replication environment after the maintenance operation is complete.

Note:

- A capture process never captures changes in the `SYS`, `SYSTEM`, or `CTXSYS` schemas. This procedure does not configure replication for these schemas.
- When the RMAN `DUPLICATE` or `CONVERT DATABASE` command is used for database instantiation, the destination database cannot be the capture database.

See Also:

- "POST_INSTANTIATION_SETUP Procedure" on page 106-111
- "CLEANUP_INSTANTIATION_SETUP Procedure" on page 106-74
- "Procedures That Configure a Streams Replication Environment" on page 106-11 for more information about this procedure
- *Oracle Streams Replication Administrator's Guide* for information about setting up a Streams replication environment
- *Oracle Streams Concepts and Administration* for information about completing database maintenance operations

Syntax

```
DBMS_STREAMS_ADM.PRE_INSTANTIATION_SETUP(
   maintain_mode           IN VARCHAR2,
   tablespace_names        IN DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET,
   source_database         IN VARCHAR2,
   destination_database    IN VARCHAR2,
```

```
  perform_actions         IN BOOLEAN         DEFAULT TRUE,
  script_name             IN VARCHAR2        DEFAULT NULL,
  script_directory_object IN VARCHAR2        DEFAULT NULL,
  capture_name            IN VARCHAR2        DEFAULT NULL,
  capture_queue_table     IN VARCHAR2        DEFAULT NULL,
  capture_queue_name      IN VARCHAR2        DEFAULT NULL,
  capture_queue_user      IN VARCHAR2        DEFAULT NULL,
  propagation_name        IN VARCHAR2        DEFAULT NULL,
  apply_name              IN VARCHAR2        DEFAULT NULL,
  apply_queue_table       IN VARCHAR2        DEFAULT NULL,
  apply_queue_name        IN VARCHAR2        DEFAULT NULL,
  apply_queue_user        IN VARCHAR2        DEFAULT NULL,
  bi_directional          IN BOOLEAN         DEFAULT FALSE,
  include_ddl             IN BOOLEAN         DEFAULT FALSE,
  start_processes         IN BOOLEAN         DEFAULT FALSE,
  exclude_schemas         IN VARCHAR2        DEFAULT NULL,
  exclude_flags           IN BINARY_INTEGER  DEFAULT NULL);
```

Parameters

See Also: "Common Parameters for the Configuration Procedures" on page 106-18 for descriptions of the procedure parameters that are not in Table 106–25

Table 106–25 PRE_INSTANTIATION_SETUP Procedure Parameters

Parameter	Description
maintain_mode	Specify one of the following: ■ GLOBAL to maintain the entire database by configuring replication between the local database and the database specified in the destination_database parameter ■ TRANSPORTABLE TABLESPACES to maintain a set of tablespaces by configuring replication between the local database and the database specified in the destination_database parameter
tablespace_names	If maintain_mode is set to TRANSPORTABLE TABLESPACES, then specify the local tablespace set to be cloned at the destination database and maintained by Streams. The tablespaces in the tablespace set must exist at the source database, but these tablespaces must not exist at the destination database. Also, a directory object must exist for each directory that contains the datafiles for the tablespace set. The user who invokes this procedure must have READ privilege on these directory objects. If maintain_mode is set to GLOBAL, then specify NULL. **See Also:** TABLESPACE_SET Type on page 109-4
capture_queue_table	The name of the queue table for each queue used by a capture process, specified as [*schema_name.*]*queue_table_name*. For example, strmadmin.streams_queue_table. If the schema is not specified, then the current user is the default. If NULL, then the system generates a name for the queue table of each queue used by a capture process, and the current user is the owner of each queue table.

Table 106–25 (Cont.) PRE_INSTANTIATION_SETUP Procedure Parameters

Parameter	Description
start_processes	If TRUE, then the procedure starts each capture process and apply process. If FALSE, then the procedure does not start any capture processes or apply processes.
exclude_schemas	A comma-delimited list of schemas to exclude from the Streams configuration. Schema rules are added to the negative rule sets of each capture process to exclude these schemas. Specify an asterisk (*) to exclude all of the schemas in the database. If NULL, then the procedure does not exclude any schemas in the database. This parameter is valid only if the MAINTAIN_MODE parameter is set to GLOBAL. If the MAINTAIN_MODE parameter is set to TRANSPORTABLE TABLESPACES, then the procedure ignores this parameter.
exclude_flags	Specify what to exclude from the replication configuration in the schemas specified by the exclude_schemas parameter. See "Usage Notes" on page 106-117 for more information.

Usage Notes

The following sections contain usage notes for this procedure.

Self-Contained Tablespace Sets

If the maintain_mode parameter is set to TRANSPORTABLE TABLESPACES, then the specified set of tablespaces must be self-contained. In this context "self-contained" means that there are no references from inside the set of tablespaces pointing outside of the set of tablespaces. For example, if a partitioned table is partially contained in the set of tablespaces, then the set of tablespaces is not self-contained.

> **See Also:** *Oracle Database Administrator's Guide* for more information about self-contained tablespace sets

The exclude_flags Parameter

Specify one of the following values:

- DBMS_STREAMS_ADM.EXCLUDE_FLAGS_FULL to exclude changes to the schemas and all of the database objects in the schemas
- DBMS_STREAMS_ADM.EXCLUDE_FLAGS_UNSUPPORTED to exclude changes to the database objects that are not supported by Streams in the schemas

If both of these values are specified, then the procedure raises an error.

In addition to DBMS_STREAMS_ADM.EXCLUDE_FLAGS_FULL or DBMS_STREAMS_ADM.EXCLUDE_FLAGS_UNSUPPORTED, specify one or both of the following values:

- DBMS_STREAMS_ADM.EXCLUDE_FLAGS_DML to exclude data manipulation language (DML) changes made to the excluded database objects
- DBMS_STREAMS_ADM.EXCLUDE_FLAGS_FULL to exclude data definition language (DDL) changes made to the excluded database objects

Use the plus sign (+) to specify more than one of these values. For example, to maintain DML changes to the tables in a schemas specified by the exclude_schemas

parameter but exclude DDL changes to these schemas and the database objects in these schemas, specify the following for this parameter:

```
DBMS_STREAMS_ADM.EXCLUDE_FLAGS_FULL +
DBMS_STREAMS_ADM.EXCLUDE_FLAGS_DDL
```

To exclude DML and DDL changes made to unsupported database objects in the schemas specified by the `exclude_schemas` parameter, specify the following for this parameter:

```
DBMS_STREAMS_ADM.EXCLUDE_FLAGS_UNSUPPORTED +
DBMS_STREAMS_ADM.EXCLUDE_FLAGS_DML +
DBMS_STREAMS_ADM.EXCLUDE_FLAGS_DDL
```

Rules for the excluded database objects are added to the negative rule set of each capture process. Therefore, changes to the excluded database objects will not be captured and replicated.

This parameter is valid only if the `maintain_mode` parameter is set to `GLOBAL` and the `exclude_schemas` parameter is set to a non-`NULL` value. If the `maintain_mode` parameter is set to `GLOBAL` and the `exclude_schemas` parameter is set to a `NULL`, then the procedure ignores this parameter. If the `maintain_mode` parameter is set to `TRANSPORTABLE TABLESPACES`, then this the procedure ignores this parameter and excludes any database objects in the specified tablespace set that are not supported by Streams from the Streams configuration automatically.

Also, if schemas are specified in the `exclude_schemas` parameter, but the `exclude_flags` parameter is set to `NULL`, then the procedure does not add any rules to the negative rule set of any capture process, and the procedure includes the schemas specified in the `exclude_schemas` parameter in the replication environment.

PURGE_SOURCE_CATALOG Procedure

This procedure removes all Streams data dictionary information at the local database for the specified object. You can use this procedure to remove Streams metadata that is not needed currently and will not be needed in the future.

Syntax

```
DBMS_STREAMS_ADM.PURGE_SOURCE_CATALOG(
   source_database     IN  VARCHAR2,
   source_object_name  IN  VARCHAR2,
   source_object_type  IN  VARCHAR2);
```

Parameters

Table 106–26 PURGE_SOURCE_CATALOG Procedure Parameters

Parameter	Description
source_database	The global name of the source database containing the object. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically.
source_object_name	The name of the object specified as [*schema_name*.]*object_name*. For example, hr.employees. If the schema is not specified, then the current user is the default.
source_object_type	Type of the object. Currently, TABLE is the only possible object type.

Usage Notes

The global name of the source database containing the object must be specified for the source_database parameter. If the current database is not the source database for the object, then the procedure removes data dictionary information about the object from the current database, not the source database.

For example, suppose changes to the hr.employees table at the dbs1.net source database are being applied to the hr.employees table at the dbs2.net destination database. Also, suppose hr.employees at dbs2.net is not a source at all. In this case, specifying dbs2.net as the source_database for this table results in an error. However, specifying dbs1.net as the source_database for this table while running the PURGE_SOURCE_CATALOG procedure at the dbs2.net database removes data dictionary information about the table at dbs2.net.

Do not run this procedure at a database if either of the following conditions are true:

- Logical change records (LCRs) captured by the capture process for the object are or might be applied locally without reinstantiating the object.
- LCRs captured by the capture process for the object are or might be forwarded by the database without reinstantiating the object.

Note: These conditions do not apply to LCRs that were not created by the capture process. That is, these conditions do not apply to user-created LCRs.

RECOVER_OPERATION Procedure

This procedure provides options for a Streams replication configuration operation that stopped because it encountered an error. This procedure either rolls forward the operation, rolls back the operation, or purges all of the metadata about the operation.

This procedure only can perform these actions for replication configuration operations being done by one of the following procedures:

- MAINTAIN_GLOBAL Procedure
- MAINTAIN_SCHEMAS Procedure
- MAINTAIN_SIMPLE_TABLESPACE Procedure
- MAINTAIN_SIMPLE_TTS Procedure
- MAINTAIN_TABLES Procedure
- MAINTAIN_TABLESPACES Procedure
- MAINTAIN_TTS Procedure
- PRE_INSTANTIATION_SETUP Procedure
- POST_INSTANTIATION_SETUP Procedure

When these procedures configure the replication environment directly (not by generating a script), information about the configuration actions is stored in the following data dictionary views when the procedure is running:

- DBA_RECOVERABLE_SCRIPT
- DBA_RECOVERABLE_SCRIPT_PARAMS
- DBA_RECOVERABLE_SCRIPT_BLOCKS
- DBA_RECOVERABLE_SCRIPT_ERRORS

The data dictionary views are populated at the database where the replication configuration procedure is run. When one of these procedures completes successfully, metadata about the configuration operation is purged from these views. However, when one of these procedures encounters an error and stop, metadata about the configuration operation remains in these views. In this case, you can either roll forward, roll back, or purge the metadata about the operation using the RECOVER_OPERATION procedure. If you choose to roll forward the operation, then correct conditions that caused the errors reported in DBA_RECOVERABLE_SCRIPT_ERRORS before proceeding.

Run the RECOVER_OPERATION procedure at the database where the replication configuration procedure was run.

> **Note:** To run the RECOVER_OPERATION procedure, both databases must be Oracle Database 10*g* Release 2 databases.

Syntax

```
DBMS_STREAMS_ADM.RECOVER_OPERATION(
   script_id       IN  RAW,
   operation_mode  IN  VARCHAR2  DEFAULT 'FORWARD');
```

Parameters

Table 106–27 RECOVER_OPERATION Procedure Parameters

Parameter	Description
script_id	The operation id of the procedure invocation that is being rolled forward, rolled back, or purged. Query the SCRIPT_ID column of the DBA_RECOVERABLE_SCRIPT data dictionary view to determine the operation id.
operation_mode	If FORWARD, then the procedure rolls forward the operation. Specify FORWARD to try to complete the operation. If ROLLBACK, then the procedure rolls back all of the actions performed in the operation. If the rollback is successful, then the procedure purges all of the metadata about the operation. If PURGE, then the procedure purges all of the metadata about the operation without rolling the operation back.

REMOVE_QUEUE Procedure

This procedure removes the specified ANYDATA queue.

Specifically, this procedure performs the following actions:

1. Waits until all current enqueue and dequeue transactions commit.
2. Stops the queue, which means that no further enqueues into the queue or dequeues from the queue are allowed.
3. Drops the queue.
4. If the drop_unused_queue_table parameter is set to TRUE, then drops the queue table if it is empty and no other queues are using it.
5. If the cascade parameter is set to TRUE, then drops all of the Streams clients that are using the queue.

Note: The specified queue must be a ANYDATA queue.

Syntax

```
DBMS_STREAMS_ADM.REMOVE_QUEUE(
   queue_name                IN  VARCHAR2,
   cascade                   IN  BOOLEAN  DEFAULT FALSE,
   drop_unused_queue_table   IN  BOOLEAN  DEFAULT TRUE);
```

Parameters

Table 106–28 REMOVE_QUEUE Procedure Parameters

Parameter	Description
queue_name	The name of the queue to remove, specified as [*schema_name*.]*queue_name*. For example, strmadmin.streams_queue. If the schema is not specified, then the current user is the default.
cascade	If TRUE, then the procedure drops any Streams clients that use the queue. If FALSE, then the procedure raises an error if there are any Streams clients that use the queue. Before you run this procedure with the cascade parameter set to FALSE, make sure no Streams clients are using the queue currently.
drop_unused_queue_table	If TRUE and the queue table for the queue is empty, then the procedure drops the queue table. The queue table is not dropped if it contains any messages or if it is used by another queue. If FALSE, then the procedure does not drop the queue table.

REMOVE_RULE Procedure

This procedure removes the specified rule or all rules from the rule set associated with the specified capture process, apply process, propagation, or messaging client.

If this procedure results in an empty positive rule set for a messaging client, then the procedure drops the messaging client automatically.

> **Note:** If a rule was automatically created by the system, and you want to drop the rule, then you should use this procedure to remove the rule instead of the `DBMS_RULE_ADM.DROP_RULE` procedure. If you use the `DBMS_RULE_ADM.DROP_RULE` procedure, then some metadata about the rule might remain.

Syntax

```
DBMS_STREAMS_ADM.REMOVE_RULE(
   rule_name         IN  VARCHAR2,
   streams_type      IN  VARCHAR2,
   streams_name      IN  VARCHAR2,
   drop_unused_rule  IN  BOOLEAN  DEFAULT TRUE,
   inclusion_rule    IN  BOOLEAN  DEFAULT TRUE);
```

Parameters

Table 106–29 REMOVE_RULE Procedure Parameters

Parameter	Description
`rule_name`	The name of the rule to remove, specified as `[`*`schema_name`*`.]`*`rule_name`*. If `NULL`, then the procedure removes all rules from the specified capture process, apply process, propagation, or messaging client rule set. For example, to specify a rule in the `hr` schema named `prop_rule1`, enter `hr.prop_rule1`. If the schema is not specified, then the current user is the default.
`streams_type`	The type of Streams client, either `capture` for a capture process, `apply` for an apply process, `propagation` for a propagation, or `dequeue` for a messaging client
`streams_name`	The name of the Streams client, which can be a capture process, apply process, propagation, or messaging client. Do not specify an owner. If the specified Streams client does not exist, but there is metadata in the data dictionary that associates the rule with this client, then the procedure removes the metadata. If the specified Streams client does not exist, and there is no metadata in the data dictionary that associates the rule with this client, then the procedure raises an error.
`drop_unused_rule`	If `TRUE` and the rule is not in any rule set, then the procedure drops the rule from the database. If `TRUE` and the rule exists in any rule set, then the procedure does not drop the rule from the database. If `FALSE`, then the procedure does not drop the rule from the database.

Table 106–29 (Cont.) REMOVE_RULE Procedure Parameters

Parameter	Description
`inclusion_rule`	If `inclusion_rule` is `TRUE`, then the procedure removes the rule from the positive rule set for the Streams client. If `inclusion_rule` is `FALSE`, then the procedure removes the rule from the negative rule set for the Streams client.

REMOVE_STREAMS_CONFIGURATION Procedure

This procedure removes the Streams configuration at the local database.

Syntax

```
DBMS_STREAMS_ADM.REMOVE_STREAMS_CONFIGURATION;
```

Usage Notes

Specifically, this procedure performs the following actions at the local database:

- Drops all capture processes
- If any tables have been prepared for instantiation, then aborts preparation for instantiation for the table using the ABORT_TABLE_INSTANTIATION procedure in the DBMS_CAPTURE_ADM package
- If any schemas have been prepared for instantiation, then aborts preparation for instantiation for the schema using the ABORT_SCHEMA_INSTANTIATION procedure in the DBMS_CAPTURE_ADM package
- If the database has been prepared for instantiation, then aborts preparation for instantiation for the database using the ABORT_GLOBAL_INSTANTIATION procedure in the DBMS_CAPTURE_ADM package
- Drops propagations that were created using either the DBMS_STREAMS_ADM package or the DBMS_PROPAGATION_ADM package. Before a propagation is dropped, its propagation job is disabled. Does not drop propagations that were created using the DBMS_AQADM package.
- Disables all propagation jobs used by propagations
- Drops all apply processes. If there are apply errors in the error queue for an apply process, then this procedure deletes these apply errors before it drops the apply process.
- Removes specifications for DDL handlers used by apply processes, but does not delete the PL/SQL procedures used by these handlers
- Removes specifications for message handlers used by apply processes, but does not delete the PL/SQL procedures used by these handlers
- Removes specifications for precommit handlers used by apply processes, but does not delete the PL/SQL procedures used by these handlers
- Removes the instantiation SCN and ignore SCN for each apply object and schema and for the entire database
- Removes messaging clients
- Unsets message notification specifications that were set using the SET_MESSAGE_NOTIFICATION procedure in the DBMS_STREAMS_ADM package
- Removes specifications for DML handlers and error handlers, but does not delete the PL/SQL procedures used by these handlers
- Removes update conflict handlers
- Removes specifications for substitute key columns for apply tables
- Drops rules that were created using the DBMS_STREAMS_ADM package. Does not drop rules that were created using the DBMS_RULE_ADM package.

This procedure stops capture processes and apply processes before it drops them.

Attention: Running this procedure is dangerous. You should run this procedure only if you are sure you want to remove the entire Streams configuration at a database.

Note:

- Running this procedure repeatedly does not cause errors. If the procedure fails to complete, then you can run it again.
- This procedure commits multiple times.

See Also:

- STOP_CAPTURE Procedure on page 20-27 in the `DBMS_CAPTURE_ADM` package
- STOP_APPLY Procedure on page 15-55 in the `DBMS_APPLY_ADM` package
- REMOVE_RULE Procedure on page 106-123 in the `DBMS_STREAMS_ADM` package

RENAME_COLUMN Procedure

This procedure either adds or removes a declarative rule-based transformation which renames a column in a row logical change record (LCR) that satisfies the specified rule.

For the transformation to be performed when the specified rule evaluates to `TRUE`, the rule must be in the positive rule set of a Streams client. Streams clients include capture processes, propagations, apply processes, and messaging clients.

Note:

- The `RENAME_COLUMN` procedure supports the same datatypes supported by Streams capture processes.
- Declarative transformations can transform row LCRs only. These row LCRs can be captured row LCRs or user-enqueued row LCRs. Therefore, a DML rule must be specified when you run this procedure. If a DDL is specified, then the procedure raises an error.

See Also: *Oracle Streams Concepts and Administration* for more information about declarative rule-based transformations and about the datatypes supported by Streams capture processes

Syntax

```
DBMS_STREAMS_ADM.RENAME_COLUMN(
   rule_name         IN  VARCHAR2,
   table_name        IN  VARCHAR2,
   from_column_name  IN  VARCHAR2,
   to_column_name    IN  VARCHAR2,
   value_type        IN  VARCHAR2  DEFAULT '*',
   step_number       IN  NUMBER    DEFAULT 0,
   operation         IN  VARCHAR2  DEFAULT 'ADD');
```

Parameters

Table 106–30 RENAME_COLUMN Procedure Parameters

Parameter	Description
`rule_name`	The name of the rule, specified as `[`*`schema_name`*`.]`*`rule_name`*. If `NULL`, then the procedure raises an error. For example, to specify a rule in the `hr` schema named `employees12`, enter `hr.employees12`. If the schema is not specified, then the current user is the default.
`table_name`	The name of the table in which the column is renamed in the row LCR, specified as `[`*`schema_name`*`.]`*`object_name`*. For example, `hr.employees`. If the schema is not specified, then the current user is the default.
`from_column_name`	The name of the column to be renamed in each row LCR that satisfies the rule.
`to_column_name`	The new name of the column in each row LCR that satisfies the rule.

Table 106–30 (Cont.) RENAME_COLUMN Procedure Parameters

Parameter	Description
value_type	Specify 'NEW' to rename the column in the new values in the row LCR. Specify 'OLD' to rename the column in the old values in the row LCR. Specify '*' to rename the column in both the old and new values in the row LCR.
step_number	The order of execution of the transformation. See Also: *Oracle Streams Concepts and Administration* for more information about transformation ordering
operation	Specify 'ADD' to add the transformation to the rule. Specify 'REMOVE' to remove the transformation from the rule.

Usage Notes

When 'REMOVE' is specified for the operation parameter, all of the rename column declarative rule-based transformations for the specified rule are removed that match the specified table_name, column_name, and step_number parameters. Nulls specified for these parameters act as wildcards. The following table lists the behavior of the RENAME_COLUMN procedure when one or more of these parameters is NULL:

table_name	from_column_name	to_column_name	step_number	Result
NULL	NULL	NULL	NULL	Remove all rename column transformations for the specified rule.
NULL	NULL	NULL	non-NULL	Remove all rename column transformations with the specified step_number for the specified rule.
NULL	NULL	non-NULL	non-NULL	Remove all rename column transformations with the specified to_column_name and step_number for the specified rule.
NULL	non-NULL	non-NULL	non-NULL	Remove all rename column transformations with the specified table_name and step_number for the specified rule.
NULL	NULL	non-NULL	NULL	Remove all rename column transformations with the specified column_name for the specified rule.
non-NULL	NULL	non-NULL	NULL	Remove all rename column transformations with the specified table_name and column_name for the specified rule.
NULL	non-NULL	NULL	NULL	Remove all rename column transformations with the specified table_name for the specified rule.

table_name	from_column_name	to_column_name	step_number	Result
NULL	non-NULL	non-NULL	NULL	Remove all rename column transformations with the specified table_name, column_name, and step_number for the specified rule.
non-NULL	NULL	non-NULL	NULL	Remove all rename column transformations with the specified table_name, column_name, and step_number for the specified rule.
non-NULL	non-NULL	non-NULL	NULL	Remove all rename column transformations with the specified table_name, column_name, and step_number for the specified rule.
non-NULL	non-NULL	non-NULL	non-NULL	Remove all rename column transformations with the specified table_name, column_name, and step_number for the specified rule.

RENAME_SCHEMA Procedure

This procedure either adds or removes a declarative rule-based transformation which renames a schema in a row logical change record (LCR) that satisfies the specified rule.

For the transformation to be performed when the specified rule evaluates to TRUE, the rule must be in the positive rule set of a Streams client. Streams clients include capture processes, propagations, apply processes, and messaging clients.

Note: Declarative transformations can transform row LCRs only. These row LCRs can be captured row LCRs or user-enqueued row LCRs. Therefore, a DML rule must be specified when you run this procedure. If a DDL is specified, then the procedure raises an error.

See Also: *Oracle Streams Concepts and Administration* for more information about declarative rule-based transformations

Syntax

```
DBMS_STREAMS_ADM.RENAME_SCHEMA(
   rule_name         IN  VARCHAR2,
   from_schema_name  IN  VARCHAR2,
   to_schema_name    IN  VARCHAR2,
   step_number       IN  NUMBER    DEFAULT 0,
   operation         IN  VARCHAR2  DEFAULT 'ADD');
```

Parameters

Table 106–31 RENAME_SCHEMA Procedure Parameters

Parameter	Description
rule_name	The name of the rule, specified as [*schema_name.*]*rule_name*. If NULL, then the procedure raises an error.
	For example, to specify a rule in the hr schema named employees12, enter hr.employees12. If the schema is not specified, then the current user is the default.
from_schema_name	The name of the schema to be renamed in each row LCR that satisfies the rule.
to_schema_name	The new name of the schema in each row LCR that satisfies the rule.
step_number	The order of execution of the transformation.
	See Also: *Oracle Streams Concepts and Administration* for more information about transformation ordering
operation	Specify 'ADD' to add the transformation to the rule.
	Specify 'REMOVE' to remove the transformation from the rule.

Usage Notes

When 'REMOVE' is specified for the operation parameter, all of the rename schema declarative rule-based transformations for the specified rule are removed that match the specified from_schema_name, to_schema_name, and step_number parameters. Nulls specified for these parameters act as wildcards. The following table lists the behavior of the RENAME_SCHEMA procedure when one or more of these parameters is NULL:

from_schema_name	to_schema_name	step_number	Result
NULL	NULL	NULL	Remove all rename schema transformations for the specified rule.
NULL	NULL	non-NULL	Remove all rename schema transformations with the specified step_number for the specified rule.
NULL	non-NULL	non-NULL	Remove all rename schema transformations with the specified to_schema_name and step_number for the specified rule.
non-NULL	NULL	non-NULL	Remove all rename schema transformations with the specified from_schema_name and step_number for the specified rule.
NULL	non-NULL	NULL	Remove all rename schema transformations with the specified to_schema_name for the specified rule.
non-NULL	non-NULL	NULL	Remove all rename schema transformations with the specified from_schema_name and to_schema_name for the specified rule.
non-NULL	NULL	NULL	Remove all rename schema transformations with the specified from_schema_name for the specified rule.
non-NULL	non-NULL	non-NULL	Remove all rename schema transformations with the specified from_schema_name, to_schema_name, and step_number for the specified rule.

RENAME_TABLE Procedure

This procedure either adds or removes a declarative rule-based transformation which renames a table in a row logical change record (row LCR) that satisfies the specified rule.

For the transformation to be performed when the specified rule evaluates to `TRUE`, the rule must be in the positive rule set of a Streams client. Streams clients include capture processes, propagations, apply processes, and messaging clients.

Note: Declarative transformations can transform row LCRs only. These row LCRs can be captured row LCRs or user-enqueued row LCRs. Therefore, a DML rule must be specified when you run this procedure. If a DDL is specified, then the procedure raises an error.

See Also: *Oracle Streams Concepts and Administration* for more information about declarative rule-based transformations

Syntax

```
DBMS_STREAMS_ADM.RENAME_TABLE(
   rule_name        IN  VARCHAR2,
   from_table_name  IN  VARCHAR2,
   to_table_name    IN  VARCHAR2,
   step_number      IN  NUMBER    DEFAULT 0,
   operation        IN  VARCHAR2  DEFAULT 'ADD');
```

Parameters

Table 106–32 RENAME_TABLE Procedure Parameters

Parameter	Description
`rule_name`	The name of the rule, specified as `[`*`schema_name`*`.]`*`rule_name`*. If `NULL`, then the procedure raises an error. For example, to specify a rule in the `hr` schema named `employees12`, enter `hr.employees12`. If the schema is not specified, then the current user is the default.
`from_table_name`	The name of the table to be renamed in each row LCR that satisfies the rule, specified as `[`*`schema_name`*`.]`*`object_name`*. For example, `hr.employees`. If the schema is not specified, then the current user is the default.
`to_table_name`	The new name of the table in each row LCR that satisfies the rule, specified as `[`*`schema_name`*`.]`*`object_name`*. For example, `humres.staff`. The transformation can rename the table only, the schema only, or the table and the schema. If the schema is not specified, then the current user is the default.
`step_number`	The order of execution of the transformation. See Also: *Oracle Streams Concepts and Administration* for more information about transformation ordering
`operation`	Specify `'ADD'` to add the transformation to the rule. Specify `'REMOVE'` to remove the transformation from the rule.

Usage Notes

When 'REMOVE' is specified for the operation parameter, all of the rename table declarative rule-based transformations for the specified rule are removed that match the specified from_table_name, to_table_name, and step_number parameters. Nulls specified for these parameters act as wildcards. The following table lists the behavior of the RENAME_TABLE procedure when one or more of these parameters is NULL:

from_table_name	to_table_name	step_number	Result
NULL	NULL	NULL	Remove all rename table transformations for the specified rule.
NULL	NULL	non-NULL	Remove all rename table transformations with the specified step_number for the specified rule.
NULL	non-NULL	non-NULL	Remove all rename table transformations with the specified to_table_name and step_number for the specified rule.
non-NULL	NULL	non-NULL	Remove all rename table transformations with the specified from_table_name and step_number for the specified rule.
NULL	non-NULL	NULL	Remove all rename table transformations with the specified to_table_name for the specified rule.
non-NULL	non-NULL	NULL	Remove all rename table transformations with the specified from_table_name and to_table_name for the specified rule.
non-NULL	NULL	NULL	Remove all rename table transformations with the specified from_table_name for the specified rule.
non-NULL	non-NULL	non-NULL	Remove all rename table transformations with the specified from_table_name, to_table_name, and step_number for the specified rule.

SET_MESSAGE_NOTIFICATION Procedure

This procedure sets a notification for messages that can be dequeued by a specified Streams messaging client from a specified queue. A notification is sent when a message is enqueued into the specified queue and the specified messaging client can dequeue the message because the message satisfies its rule sets.

Note: Currently, messaging clients cannot dequeue buffered messages.

Syntax

```
DBMS_STREAMS_ADM.SET_MESSAGE_NOTIFICATION(
   streams_name          IN  VARCHAR2,
   notification_action   IN  VARCHAR2,
   notification_type     IN  VARCHAR2     DEFAULT 'PROCEDURE',
   notification_context  IN  ANYDATA      DEFAULT NULL,
   include_notification  IN  BOOLEAN      DEFAULT TRUE,
   queue_name            IN  VARCHAR2     DEFAULT 'streams_queue');
```

Parameters

Table 106–33 SET_MESSAGE_NOTIFICATION Procedure Parameters

Parameter	Description
`streams_name`	The name of the Streams messaging client. Do not specify an owner. For example, if the user `strmadmin` is the messaging client, then specify `strmadmin`.

`notification_action`	The action to be performed on message notification. Specify one of the following:
	
	▪ For URL notifications, specify a URL without the prefix `http://`.
	
	For example, to specify the URL `http://www.company.com:8080`, enter the following:
	
	`www.company.com:8080`
	
	▪ For email notifications, specify an email address.
	
	For example, to specify an the email address `xyz@company.com`, enter the following:
	
	`xyz@company.com`
	
	▪ For PL/SQL procedure notifications, specify an existing user-defined PL/SQL procedure in the form `[`*`schema_name`*`.]`*`procedure_name`*. If the *`schema_name`* is not specified, then the user who invokes the `SET_MESSAGE_NOTIFICATION` procedure is the default. The procedure must be a `PLSQLCALLBACK` data structure.
	
	For example, to specify a procedure named `notify_orders` in the `oe` schema, enter the following:
	
	`oe.notify_orders`
	
	See Also: Examples on page 106-136 for more information about message notification procedures

Table 106–33 (Cont.) SET_MESSAGE_NOTIFICATION Procedure Parameters

Parameter	Description
notification_type	The type of notification. Specify one of the following: ■ HTTP if you specified a URL for notification_action ■ MAIL if you specified an email address for notification_action ■ PROCEDURE if you specified a user-defined procedure for notification_action The type must match the specification for the notification_action parameter.
notification_context	The context of the notification. The context must be specified using RAW datatype information. For example, to specify the hexidecimal equivalent of 'FF', enter the following: ANYDATA.ConvertRaw(HEXTORAW('FF')) The notification context is passed the PL/SQL procedure in procedure notifications and is not relevant for mail or HTTP notifications.
include_notification	If TRUE, then the procedure adds this notification for the specified streams_name and queue_name. That is, specifying TRUE turns on the notification for the streams_name and queue_name. If FALSE, then the procedure removes this notification for the specified streams_name and queue_name. That is, specifying FALSE turns off the notification for the streams_name and queue_name. If you specify FALSE, then this procedure ignores any specified values for the notification_action or notification_context parameters.
queue_name	The name of a local ANYDATA queue, specified as [*schema_name*.]*queue_name*. The current database must contain the queue. The specified queue must be a ANYDATA queue. For example, to specify a queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter. If the schema is not specified, then the current user is the default.

Usage Notes

You can specify one of the following types of notifications:

- An email address to which message notifications are sent. When a relevant message is enqueued into the queue, an email with the message properties is mailed to the specified email address.
- A PL/SQL procedure to be invoked on a notification. When a relevant message is enqueued into the queue, the specified PL/SQL procedure is invoked with the message properties. This PL/SQL procedure can dequeue the message.
- An HTTP URL to which the notification is posted. When a relevant message is enqueued into the queue, a notification with the message properties is posted to the specified URL specified.

A client does not need to be connected to the database to receive a notification.

If you register for email notifications, then you should use the DBMS_AQELM package to set the host name and port name for the SMTP server that will be used by the

database to send email notifications. If required, then you should set the send-from email address, which is set by the database as the `sent from` field. You need a Java-enabled database to use this feature.

If you register for HTTP notifications, you might want to use the `DBMS_AQELM` package to set the host name and port number for the proxy server and a list of no-proxy domains that will be used by the database to post HTTP notifications.

Each notification is an `AQXmlNotification`, which includes of the following:

- `notification_options`, which includes the following:
  - `destination` - The destination queue from which the message was dequeued
  - `consumer_name` - The name of the messaging client that dequeued the message
- `message_set` - The set of message properties

See Also:

- The documentation for the `DBMS_AQELM` package for more information on email notifications and HTTP notifications
- *Oracle Streams Concepts and Administration* for more information about setting message notifications
- *Oracle Streams Advanced Queuing User's Guide and Reference* and *Oracle XML DB Developer's Guide* for more information about message notifications and XML
- *Oracle Streams Concepts and Administration* for more information about how rules are used in Streams

Examples

If you use a message notification procedure, then this PL/SQL procedure must have the following signature:

```
PROCEDURE procedure_name(
  context  IN  ANYDATA,
  reginfo  IN  SYS.AQ$_REG_INFO,
  descr    IN  SYS.AQ$_DESCRIPTOR);
```

Here, *`procedure_name`* stands for the name of the procedure. The procedure is a `PLSQLCALLBACK` data structure that specifies the user-defined PL/SQL procedure to be invoked on message notification.

The following is a simple example of a notification procedure that dequeues a message of type `oe.user_msg` using the message identifier and consumer name sent by the notification. To complete the example, first create the type:

```
CREATE TYPE oe.user_msg AS OBJECT(
  object_name    VARCHAR2(30),
  object_owner   VARCHAR2(30),
  message        VARCHAR2(50));
/
```

Next, create the procedure:

```
CREATE OR REPLACE PROCEDURE oe.notification_dequeue(
  context  ANYDATA,
  reginfo  SYS.AQ$_REG_INFO,
```

```
  descr    SYS.AQ$_DESCRIPTOR)
AS
  dequeue_options     DBMS_AQ.DEQUEUE_OPTIONS_T;
  message_properties  DBMS_AQ.MESSAGE_PROPERTIES_T;
  message_handle      RAW(16);
  message             ANYDATA;
  oe_message          oe.user_msg;
  rc                  PLS_INTEGER;
BEGIN
  -- Get the message identifier and consumer name from the descriptor
  dequeue_options.msgid := descr.msg_id;
  dequeue_options.consumer_name := descr.consumer_name;
  -- Dequeue the message
  DBMS_AQ.DEQUEUE(
    queue_name         => descr.queue_name,
    dequeue_options    => dequeue_options,
    message_properties => message_properties,
    payload            => message,
    msgid              => message_handle);
  rc := message.getobject(oe_message);
  COMMIT;
END;
/
```

See Also: *Oracle Database PL/SQL Packages and Types Reference* for more information about `PLSQLCALLBACK` data structures

SET_RULE_TRANSFORM_FUNCTION Procedure

This procedure sets or removes the transformation function name for a custom rule-based transformation.

Syntax

```
DBMS_STREAMS_ADM.SET_RULE_TRANSFORM_FUNCTION(
   rule_name           IN  VARCHAR2,
   transform_function  IN  VARCHAR2);
```

Parameters

Table 106–34 SET_RULE_TRANSFORM_FUNCTION Procedure Parameters

Parameter	Description
`rule_name`	The name of the rule whose rule-based transformation function you are setting or removing, specified as `[`*`schema_name`*`.]`*`rule_name`*.
	For example, to specify a rule in the `hr` schema named `prop_rule1`, enter `hr.prop_rule1`. If the schema is not specified, then the current user is the default.
`transform_function`	Either the name of the transformation function to be used in the rule-based transformation for the rule or `NULL`.
	If you specify a transformation function name, then specify an existing function in one of the following forms: ▪ `[`*`schema_name`*`.]`*`function_name`* ▪ `[`*`schema_name`*`.]`*`package_name.function_name`* If the function is in a package, then you must specify the *`package_name`*. For example, to specify a function in the `transform_pkg` package in the `hr` schema named `executive_to_management`, enter `hr.transform_pkg.executive_to_management`. An error is returned if the specified procedure does not exist. If the *`schema_name`* is not specified, then the user who invokes the rule-based transformation function is the default.
	If you specify `NULL`, then the `SET_RULE_TRANSFORM_FUNCTION` procedure removes the current custom rule-based transformation from the rule.

Usage Notes

The following sections contain usage notes for this procedure:

- Transformation Function Signature
- Rule Action Context
- User Who Calls the Transformation Function
- Function Verification

Transformation Function Signature

A custom rule-based transformation function always operates on one message, but it can return one message or many messages. A custom rule-based transformation

function that returns one message is a one-to-one transformation function. A one-to-one transformation function must have the following signature:

```
FUNCTION user_function (
   parameter_name   IN  ANYDATA)
RETURN ANYDATA;
```

Here, *user_function* stands for the name of the function and *parameter_name* stands for the name of the parameter passed to the function. The parameter passed to the function is an ANYDATA encapsulation of a message, and the function must return an ANYDATA encapsulation of a message.

A custom rule-based transformation function that can return more than one message is a one-to-many transformation function. A one-to-many transformation function must have the following signature:

```
FUNCTION user_function (
   parameter_name  IN  ANYDATA)
RETURN STREAMS$_ANYDATA_ARRAY;
```

Here, *user_function* stands for the name of the function and *parameter_name* stands for the name of the parameter passed to the function. The parameter passed to the function is an ANYDATA encapsulation of a message, and the function must return an array that contains zero or more ANYDATA encapsulations of a message. If the array contains zero ANYDATA encapsulations of a message, then the original message is discarded.

The STREAMS$_ANYDATA_ARRAY type is an Oracle-supplied type that has the following definition:

```
CREATE OR REPLACE TYPE SYS.STREAMS$_ANYDATA_ARRAY
   AS VARRAY(2147483647) of ANYDATA
/
```

The following restrictions apply to custom rule-based transformations that use one-to-many functions:

- Rules that are associated with one-to-many functions are supported for Streams capture processes only. These rules must not be added to rule sets used by other Streams clients, including propagations, apply processes, and messaging clients.
- One-to-many functions only can operate on row logical change records (row LCRs). They cannot operate on DDL LCRs.
- Row LCRs returned by a one-to-many function cannot contain piecewise LOB, LONG, or LONG RAW operations.
- The one-to-many function must return row LCRs in the correct order. The order of row LCRs in the array (starting from index 1) is the order that the row LCRs will be executed in the transaction.

When an apply process dequeues row LCRs that are the result of a transformation by a one-to-many function, the apply process uses the instantiation SCN of the LCR passed to the one-to-many function for all of row LCRs.

Note:

- An error is raised if a one-to-one or one-to-many transformation function returns `NULL`.
- Only one custom rule-based transformation can be specified for a particular rule. You cannot specify both a one-to-one and a one-to-many transformation function for the same rule.
- For any LCR constructed and returned by a custom rule-based transformation, the `source_database_name`, `transaction_id`, and `scn` parameter values must match the values in the original LCR. Oracle automatically specifies the values in the original LCR for these parameters, even if an attempt is made to construct LCRs with different values.

Rule Action Context

This procedure modifies the specified rule's action context to specify the transformation. A rule's action context is optional information associated with a rule that is interpreted by the client of the rules engine after the rule evaluates to `TRUE` for a message. The client of the rules engine can be a user-created application or an internal feature of Oracle, such as Streams. The Streams clients include capture processes, propagations, apply processes, and messaging clients. The information in an action context is an object of type `SYS.RE$NV_LIST`, which consists of a list of name-value pairs.

A custom rule-based transformation in Streams always consists of the following name-value pair in an action context:

- If the function is a one-to-one transformation function, then the name is `STREAMS$_TRANSFORM_FUNCTION`. If the function is a one-to-many transformation function, then the name is `STREAMS$_ARRAY_TRANS_FUNCTION`.
- The value is a `ANYDATA` instance containing a PL/SQL function name specified as a `VARCHAR2`. This function performs the transformation.

User Who Calls the Transformation Function

The user that calls the transformation function must have `EXECUTE` privilege on the function. The following list describes which user calls the transformation function:

- If a transformation is specified for a rule used by a capture process, then the user who calls the transformation function is the capture user for the capture process.
- If a transformation is specified for a rule used by a propagation, then the user who calls the transformation function is the owner of the source queue for the propagation.
- If a transformation is specified on a rule used by an apply process, then the user who calls the transformation function is the apply user for the apply process.
- If a transformation is specified on a rule used by a messaging client, then the user who calls the transformation function is the user who invokes the messaging client.

Function Verification

This procedure does not verify that the specified transformation function exists. If the function does not exist, then an error is raised when a Streams client tries to invoke the transformation function.

SET_UP_QUEUE Procedure

This procedure creates a queue table and a ANYDATA queue.

Syntax

```
DBMS_STREAMS_ADM.SET_UP_QUEUE(
   queue_table     IN  VARCHAR2  DEFAULT 'streams_queue_table',
   storage_clause  IN  VARCHAR2  DEFAULT NULL,
   queue_name      IN  VARCHAR2  DEFAULT 'streams_queue',
   queue_user      IN  VARCHAR2  DEFAULT NULL,
   comment         IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 106–35 SET_UP_QUEUE Procedure Parameters

Parameter	Description
queue_table	The name of the queue table specified as [*schema_name.*]*queue_table_name*. For example, strmadmin.streams_queue_table. If the schema is not specified, then the current user is the default. If the queue table owner is not specified, then the procedure specifies the user who runs this procedure automatically as the queue table owner.
storage_clause	The storage clause for queue table The storage parameter is included in the CREATE TABLE statement when the queue table is created. You can specify any valid table storage clause. If a tablespace is not specified here, then the procedure creates the queue table and all its related objects in the default user tablespace of the user who runs this procedure. If a tablespace is specified here, then the procedure creates the queue table and all its related objects in the tablespace specified in the storage clause. If NULL, then the procedure uses the storage characteristics of the tablespace in which the queue table is created. **See Also:** *Oracle Database SQL Reference* for more information about storage clauses
queue_name	The name of the queue that will function as the ANYDATA queue, specified as [*schema_name.*]*queue_name*. For example, strmadmin.streams_queue. If the schema is not specified, then the procedure uses the queue table owner. The owner of the queue table must also be the owner of the queue. The queue owner automatically has privileges to perform all queue operations on the queue. If the schema is not specified for this parameter, and the queue table owner is not specified in queue_table, then the current user is the default.

Table 106–35 (Cont.) SET_UP_QUEUE Procedure Parameters

Parameter	Description
queue_user	The name of the user who requires ENQUEUE and DEQUEUE privileges for the queue. This user also is configured as a secure queue user of the queue. The queue user cannot grant these privileges to other users because they are not granted with the GRANT option. If NULL, then the procedure does not grant any privileges. You can also grant queue privileges to the appropriate users using the DBMS_AQADM package.
comment	The comment for the queue

Usage Notes

Set up includes the following actions:

- If the specified queue table does not exist, then this procedure runs the CREATE_QUEUE_TABLE procedure in the DBMS_AQADM package to create the queue table with the specified storage clause. If this procedure creates the queue table, then it creates a multiple consumer ANYDATA queue that is both a secure queue and a transactional queue.

  Also, if the database is Oracle Database 10*g* release 2 or later, the sort_list setting in CREATE_QUEUE_TABLE is set to commit_time. If the database is a release prior to Oracle Database 10*g* release 2, the sort_list setting in CREATE_QUEUE_TABLE is set to enq_time.

- If the specified queue table already exists, then the queue uses the properties of the existing queue table.
- If the specified queue name does not exist, then this procedure runs the CREATE_QUEUE procedure in the DBMS_AQADM package to create the queue.
- This procedure starts the queue.
- If a queue user is specified, then this procedure configures this user as a secure queue user of the queue and grants ENQUEUE and DEQUEUE privileges on the queue to the specified queue user.

  To configure the queue user as a secure queue user, this procedure creates an Advanced Queuing agent with the same name as the user name, if one does not already exist. If an agent with this name already exists and is associated with the queue user only, then it is used. SET_UP_QUEUE then runs the ENABLE_DB_ACCESS procedure in the DBMS_AQADM package, specifying the agent and the user.

Note:

- To enqueue messages into and dequeue messages from a queue, a queue user must have `EXECUTE` privilege on the `DBMS_STREAMS_MESSAGING` package or the `DBMS_AQ` package. The `SET_UP_QUEUE` procedure does not grant this privilege.
- If the agent that `SET_UP_QUEUE` tries to create already exists and is associated with a user other than the user specified by `queue_user`, then the procedure raises an error. In this case, rename or remove the existing agent, and retry `SET_UP_QUEUE`.
- Queue names and queue table names can be a maximum of 24 bytes.

See Also: *Oracle Streams Concepts and Administration* for more information about secure queue users

107

DBMS_STREAMS_AUTH

The DBMS_STREAMS_AUTH package, one of a set of Streams packages, provides subprograms for granting privileges to Streams administrators and revoking privileges from Streams administrators.

> **See Also:** *Oracle Streams Concepts and Administration* for more information about this package and Streams administrators

This chapter contains the following topic:

Summary of DBMS_STREAMS_AUTH Subprograms

Table 107–1 DBMS_STREAMS_AUTH Package Subprograms

Subprogram	Description
GRANT_ADMIN_PRIVILEGE Procedure on page 107-3	Either grants the privileges needed by a user to be a Streams administrator directly, or generates a script that can be used to grant these privileges
GRANT_REMOTE_ADMIN_ACCESS Procedure on page 107-5	Enables a remote Streams administrator to perform administrative actions at the local database by connecting to the grantee using a database link
REVOKE_ADMIN_PRIVILEGE Procedure on page 107-6	Either revokes Streams administrator privileges from a user directly, or generates a script that can be used to revoke these privileges
REVOKE_REMOTE_ADMIN_ACCESS Procedure on page 107-8	Disables a remote Streams administrator from performing administrative actions by connecting to the grantee using a database link

Note: All subprograms commit unless specified otherwise.

GRANT_ADMIN_PRIVILEGE Procedure

This procedure either grants the privileges needed by a user to be a Streams administrator directly, or generates a script that can be used to grant these privileges.

Syntax

```
DBMS_STREAMS_AUTH.GRANT_ADMIN_PRIVILEGE(
   grantee           IN  VARCHAR2,
   grant_privileges  IN  BOOLEAN   DEFAULT TRUE,
   file_name         IN  VARCHAR2  DEFAULT NULL,
   directory_name    IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 107–2 GRANT_ADMIN_PRIVILEGE Procedure Parameters

Parameter	Description
grantee	The user to whom privileges are granted
grant_privileges	If TRUE, then the procedure grants the privileges to the specified grantee directly, and adds the grantee to the DBA_STREAMS_ADMINISTRATOR data dictionary view with YES for both the LOCAL_PRIVILEGES column and the ACCESS_FROM_REMOTE column. If the user already has an entry in this data dictionary view, then the procedure does not make another entry, and no error is raised. If TRUE and any of the grant statements fail, then the procedure raises an error. If FALSE, then the procedure does not grant the privileges to the specified grantee directly, and does not add the grantee to the DBA_STREAMS_ADMINISTRATOR data dictionary view. You specify FALSE when the procedure is generating a file that you will edit and then run. If you specify FALSE and either the file_name or directory_name parameter is NULL, then the procedure raises an error.
file_name	The name of the file generated by the procedure. The file contains all of the statements that grant the privileges. If a file with the specified file name exists in the specified directory name, then the grant statements are appended to the existing file. If NULL, then the procedure does not generate a file.
directory_name	The directory into which the generated file is placed. The specified directory must be a directory object created using the SQL statement CREATE DIRECTORY. If you specify a directory, then the user who invokes the procedure must have WRITE privilege on the directory object. If the file_name parameter is NULL, then this parameter is ignored, and the procedure does not generate a file. If NULL and the file_name parameter is non-NULL, then the procedure raises an error.

Usage Notes

The user who runs the procedure must be an administrative user who can grant privileges to other users.

Specifically, the procedure grants the following privileges to the specified user:

- The RESTRICTED SESSION system privilege

- EXECUTE on the following packages:
  - DBMS_APPLY_ADM
  - DBMS_AQ
  - DBMS_AQADM
  - DBMS_AQIN
  - DBMS_AQELM
  - DBMS_CAPTURE_ADM
  - DBMS_FLASHBACK
  - DBMS_PROPAGATION_ADM
  - DBMS_RULE_ADM
  - DBMS_STREAMS_ADM
  - DBMS_STREAMS_MESSAGING
  - DBMS_TRANSFORM
- Privileges to enqueue messages into and dequeue messages from any queue
- Privileges to manage any queue
- Privileges to create, alter, and execute any of the following types of objects in the user's own schema and in other schemas:
  - Evaluation contexts
  - Rule sets
  - Rules

  In addition, the grantee has the ability to grant these privileges to other users.
- SELECT privilege on data dictionary views related to Streams
- The ability to allow a remote Streams administrator to perform administrative actions through a database link by connecting to the grantee. This ability is enabled by running the GRANT_REMOTE_ADMIN_ACCESS procedure in this package.

Note:

- To view all of the statements run by the procedure in detail, you can use the procedure to generate a script and then view the script in a text editor.
- This procedure does not grant any roles to the grantee.
- This procedure grants only the privileges necessary to configure and administer a Streams environment. You can grant more privileges to the grantee if necessary.

See Also:

- "GRANT_REMOTE_ADMIN_ACCESS Procedure" on page 107-5
- *Oracle Streams Concepts and Administration* for more information about configuring a Streams administrator

GRANT_REMOTE_ADMIN_ACCESS Procedure

This procedure enables a remote Streams administrator to perform administrative actions at the local database by connecting to the grantee using a database link.

Syntax

```
DBMS_STREAMS_AUTH.GRANT_REMOTE_ADMIN_ACCESS(
   grantee  IN  VARCHAR2);
```

Parameters

Table 107–3 GRANT_REMOTE_ADMIN_ACCESS Procedure Parameter

Parameter	Description
grantee	The user who allows remote access. The procedure adds the grantee to the DBA_STREAMS_ADMINISTRATOR data dictionary view with YES for the ACCESS_FROM_REMOTE column. If the user already has an entry in this data dictionary view, then the procedure does not make another entry. Instead, it updates the ACCESS_FROM_REMOTE column to YES.

Usage Notes

Typically, you run the procedure and specify a grantee at a local source database if a downstream capture process captures changes originating at the local source database. The Streams administrator at a downstream capture database administers the source database using this connection. You can also run the procedure at a database running an apply process so that a remote Streams administrator can set instantiation SCNs at the local database.

Note: The GRANT_ADMIN_PRIVILEGE procedure runs this procedure.

See Also: "GRANT_ADMIN_PRIVILEGE Procedure" on page 107-3

REVOKE_ADMIN_PRIVILEGE Procedure

This procedure either revokes Streams administrator privileges from a user directly, or generates a script that can be used to revoke these privileges.

Syntax

```
DBMS_STREAMS_AUTH.REVOKE_ADMIN_PRIVILEGE(
   grantee            IN  VARCHAR2,
   revoke_privileges  IN  BOOLEAN   DEFAULT TRUE,
   file_name          IN  VARCHAR2  DEFAULT NULL,
   directory_name     IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 107–4 REVOKE_ADMIN_PRIVILEGE Procedure Parameters

Parameter	Description
grantee	The user from whom privileges are revoked
revoke_privileges	If TRUE, then the procedure revokes the privileges from the specified user directly, and removes the user from the DBA_STREAMS_ADMINISTRATOR data dictionary view. If the user does not have a record in this data dictionary view, then the procedure does not remove a record from the view, and no error is raised. If TRUE and any of the revoke statements fail, then the procedure raises an error. A revoke statement will fail if the user is not granted the privilege that is being revoked. If FALSE, then the procedure does not revoke the privileges to the specified user directly, and does not remove the user from the DBA_STREAMS_ADMINISTRATOR data dictionary view. You specify FALSE when the procedure is generating a file that you will edit and then run. If you specify FALSE and either the file_name or directory_name parameter is NULL, then the procedure does not raise an error.
file_name	The name of the file generated by this procedure. The file contains all of the statements that revoke the privileges. If a file with the specified file name exists in the specified directory name, then the revoke statements are appended to the existing file. If NULL, then the procedure does not generate a file.
directory_name	The directory into which the generated file is placed. The specified directory must be a directory object created using the SQL statement CREATE DIRECTORY. If you specify a directory, then the user who invokes the procedure must have WRITE privilege on the directory object. If the file_name parameter is NULL, then this parameter is ignored, and the procedure does not generate a file. If NULL and the file_name parameter is non-NULL, then the procedure raises an error.

Usage Notes

The user who runs this procedure must be an administrative user who can revoke privileges from other users. Specifically, this procedure revokes the privileges granted by running the GRANT_ADMIN_PRIVILEGE procedure in this package.

Note: To view all of the statements run by this procedure in detail, you can use the procedure to generate a script and then view the script in a text editor.

See Also: "GRANT_ADMIN_PRIVILEGE Procedure" on page 107-3

REVOKE_REMOTE_ADMIN_ACCESS Procedure

This procedure disables a remote Streams administrator from performing administrative actions by connecting to the grantee using a database link.

Note: The `REVOKE_ADMIN_PRIVILEGE` procedure runs this procedure.

See Also: "REVOKE_ADMIN_PRIVILEGE Procedure" on page 107-6

Syntax

```
DBMS_STREAMS_AUTH.REVOKE_REMOTE_ADMIN_ACCESS(
   grantee  IN  VARCHAR2);
```

Parameters

Table 107–5 REVOKE_REMOTE_ADMIN_ACCESS Procedure Parameter

Parameter	Description
`grantee`	The user for whom access from a remote Streams administrator is disabled. If a row for the grantee exists in the `DBA_STREAMS_ADMINISTRATOR` data dictionary view, then the procedure updates the `ACCESS_FROM_REMOTE` column for the grantee to `NO`. If, after this update, both the `LOCAL_PRIVILEGES` column and the `ACCESS_FROM_REMOTE` column are `NO` for the grantee, then the procedure removes the grantee from the view. If no row for the grantee exists in the `DBA_STREAMS_ADMINISTRATOR` data dictionary view, then the procedure does not update the view and does not raise an error.

108

DBMS_STREAMS_MESSAGING

The DBMS_STREAMS_MESSAGING package, one of a set of Streams packages, provides interfaces to enqueue messages into and dequeue messages from a ANYDATA queue.

Note: Currently, messaging clients cannot dequeue buffered messages. In addition, the DBMS_STREAMS_MESSAGING package cannot be used to enqueue messages into or dequeue messages from a buffered queue.

See Also:

- *Oracle Streams Concepts and Administration* for more information about Streams and for an example that uses the procedures in this package
- *Oracle Streams Advanced Queuing User's Guide and Reference* for more information about queues and messaging

This chapter contains the following topic:

Summary of DBMS_STREAMS_MESSAGING Subprograms

Table 108–1 DBMS_STREAMS_MESSAGING Package Subprograms

Subprogram	Description
DEQUEUE Procedure on page 108-3	Uses the specified Streams messaging client to dequeue a message from the specified queue
ENQUEUE Procedure on page 108-5	The current user enqueues a message into the specified queue

Note: The subprograms in this package do not commit.

DEQUEUE Procedure

This procedure uses the specified Streams messaging client to dequeue a message from the specified queue.

This procedure is overloaded. One version of this procedure contains the `msgid OUT` parameter, and the other does not.

Syntax

```
DBMS_STREAMS_MESSAGING.DEQUEUE(
   queue_name    IN   VARCHAR2,
   streams_name  IN   VARCHAR2,
   payload       OUT  ANYDATA,
   dequeue_mode  IN   VARCHAR2        DEFAULT 'REMOVE',
   navigation    IN   VARCHAR2        DEFAULT 'NEXT MESSAGE',
   wait          IN   BINARY_INTEGER  DEFAULT FOREVER,
   msgid         OUT  RAW);

DBMS_STREAMS_MESSAGING.DEQUEUE(
   queue_name    IN   VARCHAR2,
   streams_name  IN   VARCHAR2,
   payload       OUT  ANYDATA,
   dequeue_mode  IN   VARCHAR2        DEFAULT 'REMOVE',
   navigation    IN   VARCHAR2        DEFAULT 'NEXT MESSAGE',
   wait          IN   BINARY_INTEGER  DEFAULT FOREVER);
```

Parameters

Table 108–2 DEQUEUE Procedure Parameters

Parameter	Description
`queue_name`	The name of the local queue from which messages will be dequeued, specified as `[`*`schema_name`*`.]`*`queue_name`*. The current database must contain the queue, and the queue must be a secure queue of `ANYDATA` type. For example, to specify a queue named `streams_queue` in the `strmadmin` schema, enter `strmadmin.streams_queue` for this parameter. If the schema is not specified, then the current user is the default.
`streams_name`	The name of the Streams messaging client. For example, if the user `strmadmin` is the messaging client, then specify `strmadmin`. If `NULL` and a relevant messaging client for the queue exists, then the procedure uses the relevant messaging client. If `NULL` and multiple relevant messaging clients for the queue exist, then the procedure raises an error.
`payload`	The payload that is dequeued
`dequeue_mode`	Specify one of the following settings: `REMOVE`: Read the message and delete it. This setting is the default. The message can be retained in the queue table based on the retention properties. `LOCKED`: Read and obtain a write lock on the message. The lock lasts for the duration of the transaction. This setting is equivalent to a select for update statement. `BROWSE`: Read the message without acquiring any lock on the message. This specification is equivalent to a select statement.

Table 108–2 (Cont.) DEQUEUE Procedure Parameters

Parameter	Description
navigation	Specifies the position of the message that will be retrieved. First, the position is determined. Second, the search criterion is applied. Finally, the message is retrieved. Specify one of the following settings: NEXT MESSAGE: Retrieve the next message that is available and matches the search criteria. If the previous message belongs to a message group, then retrieve the next available message that matches the search criteria and belongs to the message group. This setting is the default. NEXT TRANSACTION: Skip the remainder of the current message group (if any) and retrieve the first message of the next message group. This setting can only be used if message grouping is enabled for the current queue. FIRST MESSAGE: Retrieves the first message which is available and matches the search criteria. This setting resets the position to the beginning of the queue. **Note:** Each message group contains the messages in a single transaction. **See Also:** *Oracle Streams Advanced Queuing User's Guide and Reference* for more information about dequeue options
wait	Either FOREVER or NO_WAIT If FOREVER, then the dequeue call is blocked without a time out until a message is available in the queue. If NO_WAIT, then a wait time of zero seconds is used. In this case, the dequeue will return immediately even if there are no messages in the queue.
msgid	Specifies the message identifier of the message that is dequeued

Exceptions

Table 108–3 DEQUEUE Procedure Exceptions

Exception	Description
ENDOFCURTRANS	Dequeue has reached the end of the messages in the current transaction. Specify this exception in the following way: SYS.DBMS_STREAMS_MESSAGING.ENDOFCURTRANS Every dequeue procedure should include an exception handler that handles this exception.
NOMOREMSGS	There are no more messages in the queue for the dequeue operation. Specify this exception in the following way: SYS.DBMS_STREAMS_MESSAGING.NOMOREMSGS A dequeue procedure that specifies NO_WAIT for the wait parameter should include an exception handler that handles this exception.

ENQUEUE Procedure

This procedure enables the current user to enqueue a message into the specified queue.

This procedure is overloaded. One version of this procedure contains the `msgid OUT` parameter, and the other does not.

Syntax

```
DBMS_STREAMS_MESSAGING.ENQUEUE(
   queue_name  IN   VARCHAR2,
   payload     IN   ANYDATA,
   msgid       OUT  RAW);

DBMS_STREAMS_MESSAGING.ENQUEUE(
   queue_name  IN   VARCHAR2,
   payload     IN   ANYDATA);
```

Parameters

Table 108–4 ENQUEUE Procedure Parameters

Parameter	Description
`queue_name`	The name of the local queue into which messages will be enqueued, specified as `[`*`schema_name`*`.]`*`queue_name`*. The current database must contain the queue, and the queue must be a secure queue of `ANYDATA` type. For example, to specify a queue named `streams_queue` in the `strmadmin` schema, enter `strmadmin.streams_queue` for this parameter. If the schema is not specified, then the current user is the default.
`payload`	The payload that is enqueued
`msgid`	Specifies the message identifier of the message that is enqueued

Usage Notes

To successfully enqueue messages into a queue, the current user must be mapped to a unique Advanced Queuing agent with the same name as the current user. You can run the `DBMS_STREAMS_ADM.SET_UP_QUEUE` procedure and specify a user as the queue user to grant the necessary privileges to the user to perform enqueues. The Advanced Queuing agent is created automatically when you run `SET_UP_QUEUE` and specify a queue user.

See Also: SET_UP_QUEUE Procedure on page 106-141

109

DBMS_STREAMS_TABLESPACE_ADM

The DBMS_STREAMS_TABLESPACE_ADM package, one of a set of Streams packages, provides administrative interfaces for copying tablespaces between databases and moving tablespaces from one database to another. This package uses transportable tablespaces, Data Pump, the DBMS_FILE_TRANSFER package, and the DBMS_FILE_GROUP package.

> **See Also:** *Oracle Streams Concepts and Administration* and *Oracle Streams Replication Administrator's Guide* for more information about this package and Streams

This chapter contains the following topics:

Using DBMS_STREAMS_TABLESPACE_ADM

This section contains topics which relate to using the `DBMS_STREAMS_TABLESPACE_ADM` package.

Overview

Either a simple tablespace or a self-contained tablespace set must be specified in each procedure in this package.

A **self-contained tablespace** has no references from the tablespace pointing outside of the tablespace. For example, if an index in the tablespace is for a table in a different tablespace, then the tablespace is not self-contained. A **simple tablespace** is a self-contained tablespace that uses only one datafile.

A simple tablespace must be specified in the following procedures:

- ATTACH_SIMPLE_TABLESPACE Procedure
- CLONE_SIMPLE_TABLESPACE Procedure
- DETACH_SIMPLE_TABLESPACE Procedure
- PULL_SIMPLE_TABLESPACE Procedure

A **self-contained tablespace set** has no references from inside the set of tablespaces pointing outside of the set of tablespaces. For example, if a partitioned table is partially contained in the set of tablespaces, then the set of tablespaces is not self-contained.

A self-contained tablespace set must be specified in the following procedures:

- ATTACH_TABLESPACES Procedure
- CLONE_TABLESPACES Procedure
- DETACH_TABLESPACES Procedure
- PULL_TABLESPACES Procedure

To determine whether a set of tablespaces is self-contained, use the `TRANSPORT_SET_CHECK` procedure in the Oracle supplied package `DBMS_TTS`.

> **See Also:** *Oracle Database Administrator's Guide* for more information about self-contained tablespaces and tablespace sets

Types

This package contains the PL/SQL types listed in Table 109–1.

Table 109–1 DBMS_STREAMS_TABLESPACE_ADM Types

Type	Description
DIRECTORY_OBJECT_SET Type on page 109-4	Contains the names of one or more directory objects
FILE Type on page 109-4	Contains the directory object associated with a directory and the name of the file in the directory
FILE_SET Type on page 109-4	Contains one or more files
TABLESPACE_SET Type on page 109-4	Contains the names of one or more tablespaces

DIRECTORY_OBJECT_SET Type

Contains the names of one or more directory objects. Each name must be a directory object created using the SQL statement `CREATE DIRECTORY`.

Syntax

```
TYPE DIRECTORY_OBJECT_SET IS TABLE OF VARCHAR2(32)
  INDEX BY BINARY_INTEGER;
```

FILE Type

Contains the directory object associated with a directory and the name of the file in the directory.

Syntax

```
TYPE FILE IS RECORD(
   directory_object VARCHAR2(32),
   file_name        VARCHAR2(4000));
```

Attributes

Table 109–2 FILE Attributes

Attribute	Description
`directory_object`	The name of a directory object. You must specify the name of a directory object created using the SQL statement `CREATE DIRECTORY`.
`file_name`	The name of the file in the corresponding directory associated with the directory object

FILE_SET Type

Contains one or more files.

Syntax

```
TYPE FILE_SET IS TABLE OF FILE
   INDEX BY BINARY_INTEGER;
```

TABLESPACE_SET Type

Contains the names of one or more tablespaces.

Syntax

```
TYPE TABLESPACE_SET IS TABLE OF VARCHAR2(32)
   INDEX BY BINARY_INTEGER;
```

Summary of DBMS_STREAMS_TABLESPACE_ADM Subprograms

Table 109–3 DBMS_STREAMS_TABLESPACE_ADM Package Subprograms

Subprogram	Description
ATTACH_SIMPLE_TABLESPACE Procedure on page 109-7	Uses Data Pump to import a simple tablespace previously exported using the DBMS_STREAMS_TABLESPACE_ADM package or Data Pump export
ATTACH_TABLESPACES Procedure on page 109-9	Uses Data Pump to import a self-contained tablespace set previously exported using the DBMS_STREAMS_TABLESPACE_ADM package, Data Pump export, or the Recovery Manager (RMAN) TRANSPORT TABLESPACE command
CLONE_SIMPLE_TABLESPACE Procedure on page 109-14	Clones a simple tablespace. The tablespace can later be attached to a database.
CLONE_TABLESPACES Procedure on page 109-16	Clones a set of self-contained tablespaces. The tablespaces can later be attached to a database.
DETACH_SIMPLE_TABLESPACE Procedure on page 109-20	Detaches a simple tablespace. The tablespace can later be attached to a database.
DETACH_TABLESPACES Procedure on page 109-22	Detaches a set of self-contained tablespaces. The tablespaces can later be attached to a database.
PULL_SIMPLE_TABLESPACE Procedure on page 109-26	Copies a simple tablespace from a remote database and attaches it to the current database
PULL_TABLESPACES Procedure on page 109-28	Copies a set of self-contained tablespaces from a remote database and attaches the tablespaces to the current database

Note: All subprograms commit unless specified otherwise.

ATTACH_SIMPLE_TABLESPACE Procedure

This procedure uses Data Pump to import a simple tablespace previously exported using the DBMS_STREAMS_TABLESPACE_ADM package, Data Pump export, or the Recovery Manager (RMAN) TRANSPORT TABLESPACE command.

Syntax

```
DBMS_STREAMS_TABLESPACE_ADM.ATTACH_SIMPLE_TABLESPACE(
   directory_object      IN  VARCHAR2,
   tablespace_file_name  IN  VARCHAR2,
   converted_file_name   IN  VARCHAR2  DEFAULT NULL,
   datafile_platform     IN  VARCHAR2  DEFAULT NULL,
   tablespace_name       OUT VARCHAR2);
```

Parameters

Table 109–4 ATTACH_SIMPLE_TABLESPACE Procedure Parameters

Parameter	Description
directory_object	The directory that contains the Data Pump dump file and the datafile for the tablespace. You must specify the name of a directory object created using the SQL statement CREATE DIRECTORY. The name of the Data Pump export dump file must be the same as the datafile name for the tablespace, except with a .dmp extension. If the converted_file_name is non-NULL, specify the dump file produced by the export database, not the file name after conversion. The Data Pump import log file is written to this directory. The name of the log file is the same as the datafile name for the tablespace, except with an .alg extension. If a file already exists with the same name as the log file in the directory, then the procedure overwrites the file. If NULL, then the procedure raises an error.
tablespace_file_name	The name of the datafile for the tablespace being imported. If NULL, then the procedure raises an error.
converted_file_name	If the datafile_platform parameter is non-NULL and is not the same as the platform of the local import database, then specify a file name for the converted datafile. The datafile is converted to the platform of the local import database and copied to the new file name. The existing datafile is not modified nor deleted. If non-NULL and the datafile_platform parameter is NULL, then the procedure ignores this parameter. If non-NULL and the datafile_platform parameter specifies the same platform as the local import database, then the procedure ignores this parameter. If NULL and the datafile_platform parameter is non-NULL, then the procedure raises an error.

Table 109–4 (Cont.) ATTACH_SIMPLE_TABLESPACE Procedure Parameters

Parameter	Description
datafile_platform	Specify NULL if the platform is the same for the export database and the current import database. Specify the platform for the export database if the platform is different for the export database and the import database. You can determine the platform of a database by querying the PLATFORM_NAME column in the V$DATABASE dynamic performance view. The V$TRANSPORTABLE_PLATFORM dynamic performance view lists all platforms that support cross-platform transportable tablespaces.
tablespace_name	Contains the name of the attached tablespace. The attached tablespace is read-only. Use an ALTER TABLESPACE statement to make the tablespace read/write if necessary.

Usage Notes

To run this procedure, a user must meet the following requirements:

- Have IMP_FULL_DATABASE role
- Have READ and WRITE privilege on the directory object that contains the Data Pump export dump file and the datafiles for the tablespaces in the set, specified by the directory_object parameter

Automatic Storage Management (ASM) directories cannot be used with this procedure.

See Also: Overview on page 109-3

ATTACH_TABLESPACES Procedure

This procedure uses Data Pump to import a self-contained tablespace set previously exported using the DBMS_STREAMS_TABLESPACE_ADM package, Data Pump export, or the Recovery Manager (RMAN) TRANSPORT TABLESPACE command.

This procedure is overloaded and consists of the following versions:

- One version of the procedure uses a Data Pump job name in the datapump_job_name parameter. This job performs the Data Pump import to complete the attach operation. In addition, if the platform at the export database is different than the local database platform, then this procedure optionally can create datafiles for the tablespace set that can be used with the local platform.
- The other version of the procedure uses a file group that can consist of multiple versions of the tablespace set in a tablespace repository. A tablespace repository is a collection of tablespace sets in a file group repository. When this version of the procedure is run, a Data Pump import is performed. This version of the procedure uses the files in a file group version and can copy the export dump file, export log file, and the datafiles that comprise the tablespace set into the specified directories. The file group and version are specified using the file_group_name and version_name parameters, respectively. This version of the procedure does not require a datafiles platform specification if the platform at the export database is different than the local database platform. Instead, the tablespace set is migrated automatically to the correct platform when it is attached.

Syntax

```
DBMS_STREAMS_TABLESPACE_ADM.ATTACH_TABLESPACES(
   datapump_job_name          IN OUT VARCHAR2,
   dump_file                  IN     FILE,
   tablespace_files           IN     FILE_SET,
   converted_files            IN     FILE_SET,
   datafiles_platform         IN     VARCHAR2  DEFAULT NULL,
   log_file                   IN     FILE      DEFAULT NULL,
   tablespace_names           OUT    TABLESPACE_SET);

DBMS_STREAMS_TABLESPACE_ADM.ATTACH_TABLESPACES(
   file_group_name            IN   VARCHAR2,
   version_name               IN   VARCHAR2  DEFAULT NULL,
   datafiles_directory_object IN   VARCHAR2  DEFAULT NULL,
   logfile_directory_object   IN   VARCHAR2  DEFAULT NULL,
   repository_db_link         IN   VARCHAR2  DEFAULT NULL,
   tablespace_names           OUT  TABLESPACE_SET);
```

Parameters

Table 109–5 ATTACH_TABLESPACES Procedure Parameters

Parameter	Description
data_pump_job_name	The Data Pump job name. Specify a Data Pump job name if you want to adhere to naming conventions or if you want to track the job more easily. If NULL, then the system generates a Data Pump job name.

Table 109–5 (Cont.) ATTACH_TABLESPACES Procedure Parameters

Parameter	Description
dump_file	The file name of the Data Pump dump file to import. If NULL or if a file attribute is NULL, then the procedure raises an error.
tablespace_files	The file set that contains the datafiles for the tablespace set being imported. If NULL, then the procedure raises an error.
converted_files	If the datafiles_platform parameter is non-NULL and is not the same as the platform for the local import database, then specify a file set with the names of the converted datafiles. The datafiles are converted to the platform of the local import database and copied to the new file names. In this case, the number of files in the specified file set must match the number of files in the file set specified for the tablespace_files parameter. The existing datafiles are not modified nor deleted. If non-NULL and the datafiles_platform parameter is NULL, then the procedure ignores this parameter. If non-NULL and the datafiles_platform parameter specifies the same platform as the local import database, then the procedure ignores this parameter. If NULL and the datafiles_platform parameter is non-NULL, then the procedure raises an error.
datafiles_platform	Specify NULL if the platform is the same for the export database and the current import database. Specify the platform for the export database if the platform is different for the export database and the import database. You can determine the platform of a database by querying the PLATFORM_NAME column in the V$DATABASE dynamic performance view. The V$TRANSPORTABLE_PLATFORM dynamic performance view lists all platforms that support cross-platform transportable tablespaces.
log_file	Specify the log file name for the Data Pump import. If NULL or if at least one file parameter is NULL, then the system generates a log file name with the extension .alg and places it in the Data Pump export dump file directory. If a file already exists with the same name as the log file in the directory, then the procedure overwrites the file.
file_group_name	The name of the file group, specified as *[schema_name.]file_group_name*. For example, if the schema is hq_dba and the file group name is sales, then specify hq_dba.sales. If the schema is not specified, then the current user is the default.
version_name	The name of the file group version to attach. If NULL, then the procedure uses the version with the latest creation time for the file group.

Table 109–5 (Cont.) ATTACH_TABLESPACES Procedure Parameters

Parameter	Description
datafiles_directory_object	The directory object into which the datafiles and Data Pump export dump file are copied. The files are copied from the tablespace repository directories to this directory. If non-NULL, the attached tablespaces use the files in specified directory. However, the file group version specified in the version_name parameter consists of the files in the original directory, not in the directory specified by this datafiles_directory_object parameter. If NULL, then the procedure does not copy the datafiles and dump file.
logfile_directory_object	The directory object into which the Data Pump import log file is placed. The system generates a log file name with the extension .alg. If NULL, then the procedure places the import log file in the same directory as the dump file.
repository_db_link	If the file group is in a different database, then specify the name of the database link to the database that contains the file group. The database link must be accessible to the user who runs the procedure. If this parameter is non-NULL, then meet the following requirements: ▪ Each directory object that contains files in the version being attached must exist on both databases. ▪ The corresponding directory objects must have the same names on both databases. If NULL, then the procedure does not use a database link, and the procedure uses the file group in the local database.
tablespace_names	Contains the names of the attached tablespaces. The attached tablespaces are read-only. Use ALTER TABLESPACE statements to make the tablespaces read/write if necessary.

Usage Notes

The following sections contain usage notes for this procedure:

- User Requirements
- Procedures Used to Clone or Detach a Tablespace Set
- When the Attach Database Is Different Than the Clone or Detach Database
- Automatic Storage Management Directories

See Also:

- Overview on page 109-3
- *Oracle Streams Concepts and Administration*

User Requirements

To run either version of this procedure, a user must meet the following requirements:

- Have IMP_FULL_DATABASE role

- Have READ and WRITE privilege on the directory objects that contain the Data Pump export dump file and the datafiles for the tablespaces in the set, specified by the dump_file and tablespace_files parameters, or by the datafiles_directory_object parameter
- Have WRITE privilege on the directory object that will hold the Data Pump import log file, specified by the log_file parameter or logfile_directory_object parameter if it is non-NULL

If the Data Pump job version of the procedure is run, then the user must have WRITE privilege on the directory objects that will hold the converted datafiles for the tablespaces in the set if platform conversion is necessary. These directory objects are specified by the converted_files parameter if it is non-NULL.

If the file group version of the procedure is run, then the user must have the necessary privileges to manage the file group.

Procedures Used to Clone or Detach a Tablespace Set

After a tablespace set is cloned or detached using the CLONE_TABLESPACES or DETACH_TABLEPSACES procedure, respectively, the tablespace set can be attached to a database using the ATTACH_TABLESPACES procedure. If the Data Pump job version of the CLONE_TABLESPACES or DETACH_TABLEPSACES procedure was used, then use the Data Pump job version of the ATTACH_TABLESPACES procedure. If the file group version of the CLONE_TABLESPACES or DETACH_TABLEPSACES procedure was used, then use the file group version of the ATTACH_TABLESPACES procedure.

See Also:

- CLONE_TABLESPACES Procedure on page 109-16
- DETACH_TABLESPACES Procedure on page 109-22

When the Attach Database Is Different Than the Clone or Detach Database

You can attach a tablespace set to a different database than the database from which the tablespace set was cloned or detached. The two databases might or might not share a file system. If the two databases do not share a file system, then you must transfer the dump file and datafiles to the remote system using the DBMS_FILE_TRANSFER package, FTP, or some other method. You can attach the tablespace set in one of the following ways depending on the version of the ATTACH_TABLESPACES procedure you use:

- If you use the Data Pump job version of the procedure, then specify the relevant files on the file system. The directory object names can be different in the databases.
- If you use the file group version of the procedure, then you can use the repository_db_link parameter to specify the database where tablespace repository resides. The directory objects for the files must exist and must match in the databases.

See Also:

- CLONE_TABLESPACES Procedure on page 109-16
- DETACH_TABLESPACES Procedure on page 109-22
- Chapter 41, "DBMS_FILE_GROUP" for more information about file groups

Automatic Storage Management Directories

Automatic Storage Management (ASM) directories can be specified for the directory objects that store data files and export dump files, but ASM directories cannot be specified for directory objects that store log files.

> **See Also:** *Oracle Database Utilities* for information about specifying ASM directories for directory objects

CLONE_SIMPLE_TABLESPACE Procedure

This procedure clones a simple tablespace. The specified tablespace must be online.

Specifically, this procedure performs the following actions:

1. Makes the specified tablespace read-only if it is not read-only
2. Uses Data Pump to export the metadata for the tablespace and places the dump file in the specified directory
3. Places the datafile for the specified tablespace in the specified directory
4. If this procedure made the tablespace read-only, then makes the tablespace read/write

In addition, this procedure optionally can create a datafile for the tablespace that can be used with a platform that is different than the local database platform.

Syntax

```
DBMS_STREAMS_TABLESPACE_ADM.CLONE_SIMPLE_TABLESPACE(
   tablespace_name      IN  VARCHAR2,
   directory_object     IN  VARCHAR2,
   destination_platform IN  VARCHAR2  DEFAULT NULL,
   tablespace_file_name OUT VARCHAR2);
```

Parameters

Table 109–6 CLONE_SIMPLE_TABLESPACE Procedure Parameters

Parameter	Description
`tablespace_name`	The tablespace to be cloned. If `NULL`, then the procedure raises an error.
`directory_object`	The directory where the Data Pump export dump file, the Data Pump export log file, and the datafile for the tablespace are placed. You must specify the name of a directory object created using the SQL statement `CREATE DIRECTORY`. The name of the Data Pump export dump file is the same as the datafile name for the tablespace, except with a `.dmp` extension. If a file already exists with the same name as the dump file in the directory, then the procedure raises an error. The name of the log file is the same as the datafile name for the tablespace, except with a `.clg` extension. If a file already exists with the same name as the log file in the directory, then the procedure overwrites the file. If `NULL`, then the procedure raises an error.
`destination_platform`	Specify `NULL` if the platform is the same for the current export database and the intended import database. Specify the platform for the intended import database if the platform is different for the export database and the import database. You can determine the platform of a database by querying the `PLATFORM_NAME` column in the `V$DATABASE` dynamic performance view. The `V$TRANSPORTABLE_PLATFORM` dynamic performance view lists all platforms that support cross-platform transportable tablespaces.

Table 109–6 (Cont.) CLONE_SIMPLE_TABLESPACE Procedure Parameters

Parameter	Description
tablespace_file_name	Contains the name of the cloned tablespace datafile. This datafile is placed in the directory specified by the parameter directory_object.

Usage Notes

To run this procedure, a user must meet the following requirements:

- Have EXP_FULL_DATABASE role
- Have access to at least one data dictionary view that contains information about the tablespaces. These views include DBA_TABLESPACES and USER_TABLESPACES.
- Have MANAGE TABLESPACE or ALTER TABLESPACE on a tablespace if the tablespace must be made read-only
- Have READ privilege on the directory object for the directory that contains the datafile for the tablespace. The name of this tablespace is specified by the tablespace_name parameter. If a directory object does not exist for this directory, then create the directory object and grant the necessary privileges before you run this procedure.
- Have READ and WRITE privilege on the directory object that will contain the Data Pump export dump file, specified by the directory_object parameter
- If the file group version of the procedure is run, then the user must have the necessary privileges to manage file group.

After cloning a tablespace using this procedure, you can add the tablespace to a different database using the ATTACH_SIMPLE_TABLESPACE procedure. If the database is a remote database and you want to use the ATTACH_SIMPLE_TABLESPACE procedure, then you can transfer the dump file and datafile to the remote system using the DBMS_FILE_TRANSFER package, FTP, or some other method.

Automatic Storage Management (ASM) directories cannot be used with this procedure.

See Also:

- Overview on page 109-3
- ATTACH_SIMPLE_TABLESPACE Procedure on page 109-7 and PULL_SIMPLE_TABLESPACE Procedure on page 109-26

CLONE_TABLESPACES Procedure

This procedure clones a set of self-contained tablespaces. All of the tablespaces in the specified tablespace set must be online.

Specifically, this procedure performs the following actions:

1. Makes any read/write tablespace in the specified tablespace set read-only
2. Uses Data Pump to export the metadata for the tablespaces in the tablespace set and places the dump file in the specified directory
3. Places the datafiles that comprise the specified tablespace set in the specified directory
4. If this procedure made a tablespace read-only, then makes the tablespace read/write

This procedure is overloaded and consists of the following versions:

- One version of the procedure uses a Data Pump job name in the `datapump_job_name` parameter. This job performs the Data Pump export. This version of the procedure completes the clone operation by placing the export dump file, export log file, and the datafiles that comprise the tablespace set in the specified directories, but the files are not added to a file group version. In addition, this version of the procedure optionally can create datafiles for the tablespace set that can be used with a platform that is different than the local database platform.
- The other version of the procedure uses a file group that can consist of multiple versions of the tablespace set in a tablespace repository. A tablespace repository is a collection of tablespace sets in a file group repository. When this version of the procedure is run, a Data Pump export is performed, and this version of the procedure completes the clone operation by placing the export dump file, export log file, and the datafiles that comprise the tablespace set in the appropriate file group version. The file group and version are specified using the `file_group_name` and `version_name` parameters, respectively. This version of the procedure does not require a destination platform specification if the destination platform is different. Instead, the tablespace set is migrated automatically to the correct platform when it is attached at the destination database using the file group version of the `ATTACH_TABLESPACES` procedure.

Syntax

```
DBMS_STREAMS_TABLESPACE_ADM.CLONE_TABLESPACES(
   datapump_job_name            IN OUT VARCHAR2,
   tablespace_names             IN     TABLESPACE_SET,
   dump_file                    IN     FILE,
   tablespace_directory_objects IN     DIRECTORY_OBJECT_SET,
   destination_platform         IN     VARCHAR2  DEFAULT NULL,
   log_file                     IN     FILE      DEFAULT NULL,
   tablespace_files             OUT    FILE_SET);

DBMS_STREAMS_TABLESPACE_ADM.CLONE_TABLESPACES(
   tablespace_names             IN  TABLESPACE_SET,
   tablespace_directory_object  IN  VARCHAR2  DEFAULT NULL,
   log_file_directory_object    IN  VARCHAR2  DEFAULT NULL,
   file_group_name              IN  VARCHAR2,
   version_name                 IN  VARCHAR2  DEFAULT NULL,
   repository_db_link           IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 109–7 CLONE_TABLESPACES Procedure Parameters

Parameter	Description
data_pump_job_name	The Data Pump job name. Specify a Data Pump job name if you want to adhere to naming conventions or if you want to track the job more easily. If NULL, then the system generates a Data Pump job name.
tablespace_names	The tablespace set to be cloned. If NULL, then the procedure raises an error.
dump_file	The file name of the Data Pump dump file that is exported. If NULL or if a file attribute is NULL, then the procedure raises an error. If the specified file already exists, then the procedure raises an error.
tablespace_directory_objects	The set of directory objects into which the datafiles for the tablespaces are copied. If more than one directory object is in the set, then the procedure copies a datafile to each directory object in the set in sequence. In this case, if the end of the directory object set is reached, then datafile copying starts again with the first directory object in the set. If NULL, then the procedure copies datafiles for the tablespace set to the dump file directory.
destination_platform	Specify NULL if the platform is the same for the current export database and the intended import database. Specify the platform for the intended import database if the platform is different for the export database and the import database. You can determine the platform of a database by querying the PLATFORM_NAME column in the V$DATABASE dynamic performance view. The V$TRANSPORTABLE_PLATFORM dynamic performance view lists all platforms that support cross-platform transportable tablespaces.
log_file	Specify the log file name for the Data Pump export. If NULL or if at least one file parameter is NULL, then the system generates a log file name with the extension .clg and places it in the dump file directory. If a file already exists with the same name as the log file in the directory, then the procedure overwrites the file.

Table 109–7 (Cont.) CLONE_TABLESPACES Procedure Parameters

Parameter	Description
tablespace_directory_object	The directory object into which the datafiles are copied and Data Pump export dump file is placed. The system generates a dump file name with the extension .dmp. If NULL, then the procedure copies the datafiles to and places the dump file in the default directory object for the version. If the version does not have a default directory object, then the procedure uses the default directory object for the file group. If NULL and no default directory object exists for the version or file group, then the procedure raises an error.
log_file_directory_object	The directory object into which the Data Pump export log file is placed. The system generates a log file name with the extension .clg. If NULL, then the procedure uses the directory object specified in tablespace_directory_object.
file_group_name	The name of the file group, specified as *[schema_name.]file_group_name*. For example, if the schema is hq_dba and the file group name is sales, then specify hq_dba.sales. If the schema is not specified, then the current user is the default. If the specified file group does not exist, then the procedure creates it.
version_name	The name of the version into which the cloned tablespace set is placed. The specified version name cannot be a positive integer. If the specified version does not exist, then the procedure creates it. If the specified version exists, then the procedure adds the tablespace set to the version. Only one Data Pump export dump file can exist in a version. If NULL, then the procedure creates a new version, and the version number can be used to manage the version.
repository_db_link	If the file group is in a remote database, then specify the name of the database link to the database that contains the file group. The database link must be accessible to the user who runs the procedure. If this parameter is non-NULL, then the directory object specified in tablespace_directory_object must exist on the local database and on the remote database. If tablespace_directory_object is NULL, then the default directory object must exist on both databases. The directory object must have the same name on each database and must correspond to the same directory on a shared file system. If NULL, then the procedure does not use a database link, and the procedure uses the file group in the local database.
tablespace_files	Contains the datafiles for the cloned tablespace set. These datafiles are placed in the directories specified by the directory objects in the parameter tablespace_directory_objects.

Usage Notes

To run either version of this procedure, a user must meet the following requirements:

- Have EXP_FULL_DATABASE role
- Have access to at least one data dictionary view that contains information about the tablespaces. These views include DBA_TABLESPACES and USER_TABLESPACES.
- Have MANAGE TABLESPACE or ALTER TABLESPACE on a tablespace if the tablespace must be made read-only
- Have READ privilege on the directory objects for the directories that contain the datafiles for the tablespace set. The names of these tablespaces are specified by the tablespace_names parameter. If a directory object does not exist for one or more of these directories, then create the directory objects and grant the necessary privileges before you run this procedure.
- Have READ and WRITE privilege on the directory object that will contain the Data Pump export dump file, specified by the dump_file parameter or the tablespace_directory_object parameter
- Have WRITE privilege on the directory objects that will contain the copied datafiles for the tablespaces in the set, specified by the tablespace_directory_objects parameter if non-NULL or the tablespace_directory_object parameter
- Have WRITE privilege on the directory object that will contain the Data Pump export log file, specified by the log_file parameter if non-NULL or the log_file_directory_object parameter if non-NULL

If the file group version of the procedure is run, then the user must have the necessary privileges to manage the file group.

Automatic Storage Management (ASM) directories can be specified for the directory objects that store data files and export dump files, but ASM directories cannot be specified for directory objects that store log files.

After cloning a tablespace set using this procedure, you can attach the tablespaces to a different database using the ATTACH_TABLESPACES procedure.

See Also:

- Overview on page 109-3
- ATTACH_TABLESPACES Procedure on page 109-9
- Chapter 41, "DBMS_FILE_GROUP" for more information about file groups
- *Oracle Streams Concepts and Administration*

DETACH_SIMPLE_TABLESPACE Procedure

This procedure detaches a simple tablespace. The specified tablespace must be online.

Specifically, this procedure performs the following actions:

1. Makes the specified tablespace read-only if it is not read-only
2. Uses Data Pump to export the metadata for the tablespace and places the dump file in the directory that contains the tablespace datafile
3. Drops the tablespace and its contents from the database

Syntax

```
DBMS_STREAMS_TABLESPACE_ADM.DETACH_SIMPLE_TABLESPACE(
   tablespace_name       IN  VARCHAR2,
   directory_object      OUT VARCHAR2,
   tablespace_file_name  OUT VARCHAR2);
```

Parameters

Table 109–8 DETACH_SIMPLE_TABLESPACE Procedure Parameters

Parameter	Description
data_pump_job_name	The Data Pump job name. Specify a Data Pump job name if you want to adhere to naming conventions or if you want to track the job more easily. If NULL, then the system generates a Data Pump job name.
directory_object	Contains the directory where the Data Pump export dump file and the Data Pump export log file are placed. The procedure uses the directory of the datafile for the tablespace. Therefore, make sure a directory object created using the SQL statement CREATE DIRECTORY exists for this directory. The name of the Data Pump export dump file is the same as the datafile name for the tablespace, except with a .dmp extension. If a file already exists with the same name as the dump file in the directory, then the procedure raises an error. The name of the log file is the same as the datafile name for the tablespace, except with a .dlg extension. If a file already exists with the same name as the log file in the directory, then the procedure overwrites the file.
tablespace_file_name	Contains the name of the detached tablespace datafile.

Usage Notes

To run this procedure, a user must meet the following requirements:

- Have EXP_FULL_DATABASE role
- Have access to at least one data dictionary view that contains information about the tablespaces. These views include DBA_TABLESPACES and USER_TABLESPACES.
- Have DROP TABLESPACE privilege
- Have MANAGE TABLESPACE or ALTER TABLESPACE on a tablespace if the tablespace must be made read-only

- Have READ and WRITE privilege on the directory object for the directory that contains the tablespace datafile. The name of this tablespace is specified by the tablespace_name parameter. If a directory object does not exist for this directory, then create the directory object and grant the necessary privileges before you run this procedure. This directory also will contain the Data Pump export dump file generated by this procedure.

After detaching a tablespace using this procedure, you can add the tablespace to a different database using the ATTACH_SIMPLE_TABLESPACE procedure. If the database is a remote database and you want to use the ATTACH_SIMPLE_TABLESPACE procedure, then you can transfer the dump file and datafile to the remote system using the DBMS_FILE_TRANSFER package, FTP, or some other method. You can use the two OUT parameters in this procedure to accomplish the attach or pull operation.

Automatic Storage Management (ASM) directories cannot be used with this procedure.

Note: Do not use the DETACH_SIMPLE_TABLESPACE procedure on a tablespace if the tablespace is using the Oracle-managed files feature. If you do, then the datafile for the tablespace is dropped automatically when the tablespace is dropped.

See Also:

- Overview on page 109-3
- ATTACH_SIMPLE_TABLESPACE Procedure on page 109-7 and PULL_SIMPLE_TABLESPACE Procedure on page 109-26
- *Oracle Database Administrator's Guide* for more information about the Oracle-managed files feature

DETACH_TABLESPACES Procedure

This procedure detaches a set of self-contained tablespaces. All of the tablespaces in the specified tablespace set must be online and any table partitions must not span tablespaces in the tablespace set.

Specifically, this procedure performs the following actions:

1. Makes any read/write tablespace in the specified tablespace set read-only
2. Uses Data Pump to export the metadata for the tablespace set and places the dump file in the specified directory
3. Drops the tablespaces in the specified tablespace set and their contents from the database

This procedure does not move or copy the datafiles that comprise the specified tablespace set.

This procedure is overloaded and consists of the following versions:

- One version of the procedure uses a Data Pump job name in the `datapump_job_name` parameter. This job performs the Data Pump export. This version of the procedure completes the detach operation by placing the export dump file and export log file in the specified directories, but the files are not added to a file group version.
- The other version of the procedure uses a file group that can consist of multiple versions of the tablespace set in a tablespace repository. A tablespace repository is a collection of tablespace sets in a file group repository. When this version of the procedure is run, a Data Pump export is performed, and this version of the procedure completes the detach operation by placing the export dump file and export log file in the appropriate file group version. The datafiles that comprise the tablespace set are not moved or copied, but they are referenced in the version that is detached. The file group and version are specified using the `file_group_name` and `version_name` parameters, respectively. Also, if the destination platform is different, then the tablespace set is migrated automatically to the correct platform when it is attached at the destination database using the file group version of the `ATTACH_TABLESPACES` procedure.

> **Note:** Do not use the `DETACH_TABLESPACES` procedure if any of the tablespaces in the tablespace set are using the Oracle-managed files feature. If you do, then the datafiles for these tablespaces are dropped automatically when the tablespaces are dropped.

Syntax

```
DBMS_STREAMS_TABLESPACE_ADM.DETACH_TABLESPACES(
   datapump_job_name         IN OUT VARCHAR2,
   tablespace_names          IN     TABLESPACE_SET,
   dump_file                 IN     FILE,
   log_file                  IN     FILE  DEFAULT NULL,
   tablespace_files          OUT    FILE_SET);

DBMS_STREAMS_TABLESPACE_ADM.DETACH_TABLESPACES(
   tablespace_names          IN TABLESPACE_SET,
   export_directory_object   IN VARCHAR2  DEFAULT NULL,
   log_file_directory_object IN VARCHAR2  DEFAULT NULL,
   file_group_name           IN VARCHAR2,
```

```
   version_name              IN VARCHAR2  DEFAULT NULL,
   repository_db_link        IN VARCHAR2  DEFAULT NULL);
```

Parameters

Table 109–9 DETACH_TABLESPACES Procedure Parameters

Parameter	Description
data_pump_job_name	The Data Pump job name. Specify a Data Pump job name if you want to adhere to naming conventions or if you want to track the job more easily. If NULL, then the system generates a Data Pump job name.
tablespace_names	The tablespace set to be detached. If NULL, then the procedure raises an error.
dump_file	The file name of the Data Pump dump file that is exported. If NULL or if a file attribute is NULL, then the procedure raises an error. If the specified file already exists, then the procedure raises an error.
log_file	Specify the log file name for the Data Pump export. If NULL or if at least one file parameter is NULL, then the system generates a log file name with the extension .dlg and places it in the dump file directory. If a file already exists with the same name as the log file in the directory, then the procedure overwrites the file.
tablespace_files	Contains the names of the datafiles for the detached tablespace set.
export_directory_object	The directory object into which the Data Pump export dump file is placed. The system generates a dump file name with the extension .dmp. If NULL, then procedure places the dump file in the default directory object for the version. If the version does not have a default directory object, then the procedure uses the default directory object for the file group. If NULL and no default directory object exists for the version or file group, then the procedure raises an error.
log_file_directory_object	The directory object into which the Data Pump export log file is placed. The system generates a log file name with the extension .dlg. If NULL, then the procedure places the export log file in the same directory as the export dump file.
file_group_name	The name of the file group, specified as *[schema_name.]file_group_name*. For example, if the schema is hq_dba and the file group name is sales, then specify hq_dba.sales. If the schema is not specified, then the current user is the default. If the specified file group does not exist, then the procedure creates it.

Table 109–9 (Cont.) DETACH_TABLESPACES Procedure Parameters

Parameter	Description
version_name	The name of the version into which the detached tablespace set is placed. The specified version name cannot be a positive integer. If the specified version does not exist, then the procedure creates it. If the specified version exists, then procedure adds the tablespace set to the version. Only one Data Pump export dump file can exist in a version. If NULL, then the procedure creates a new version, and the version number can be used to manage the version.
repository_db_link	If the file group is in a remote database, then specify the name of the database link to the database that contains the file group. The database link must be accessible to the user who runs the procedure. If this parameter is non-NULL, then the directory object specified in export_directory_object must exist on the local database and on the remote database. If export_directory_object is NULL, then the default directory object must exist on both databases. The directory object must have the same name on each database and must correspond to the same directory on a shared file system. If NULL, then the procedure does not use a database link, and the procedure uses the file group in the local database.

Usage Notes

To run this either version of this procedure, a user must meet the following requirements:

- Have EXP_FULL_DATABASE role
- Have access to at least one data dictionary view that contains information about the tablespaces. These views include DBA_TABLESPACES and USER_TABLESPACES.
- Have DROP TABLESPACE privilege
- Have MANAGE TABLESPACE or ALTER TABLESPACE on a tablespace if the tablespace must be made read-only
- Have READ privilege on the directory objects for the directories that contain the datafiles for the tablespace set. The names of these tablespaces are specified by the tablespace_names parameter. If a directory object does not exist for one or more of these directories, then create the directory objects and grant the necessary privileges before you run this procedure.
- Have READ and WRITE privilege on the directory object that will contain the Data Pump export dump file, specified by the dump_file parameter or the export_directory_object parameter
- Have WRITE privilege on the directory object that will contain the Data Pump export the log file, specified by the log_file parameter if non-NULL or by the log_file_directory_object parameter if non-NULL

If the file group version of the procedure is run, then the user must have the necessary privileges to manage the file group.

Automatic Storage Management (ASM) directories can be specified for the directory objects that store data files and export dump files, but ASM directories cannot be specified for directory objects that store log files.

After detaching a tablespace set using this procedure, you can attach the tablespaces to a different database using the `ATTACH_TABLESPACES` procedure.

See Also:

- Overview on page 109-3
- ATTACH_TABLESPACES Procedure on page 109-9
- Chapter 41, "DBMS_FILE_GROUP" for more information about file groups
- *Oracle Streams Concepts and Administration*
- *Oracle Database Administrator's Guide* for more information about the Oracle-managed files feature

PULL_SIMPLE_TABLESPACE Procedure

This procedure copies a simple tablespace from a remote database and attaches it to the current database. The specified tablespace at the remote database must be online.

Specifically, this procedure performs the following actions:

1. Makes the specified tablespace read-only at the remote database if it is not read-only
2. Uses Data Pump to export the metadata for the tablespace
3. Uses a database link and the `DBMS_FILE_TRANSFER` package to transfer the datafile for the tablespace and the log file for the Data Pump export to the current database
4. Places the datafile for the specified tablespace and the log file for the Data Pump export in the specified directory at the local database
5. If this procedure made the tablespace read-only, then makes the tablespace read/write
6. Uses Data Pump to import the metadata for the tablespace in the at the local database

In addition, this procedure optionally can create a datafile for the tablespace that can be used with the local platform, if the platform at the remote database is different than the local database platform.

Syntax

```
DBMS_STREAMS_TABLESPACE_ADM.PULL_SIMPLE_TABLESPACE(
   tablespace_name      IN VARCHAR2,
   database_link        IN VARCHAR2,
   directory_object     IN VARCHAR2  DEFAULT NULL,
   conversion_extension IN VARCHAR2  DEFAULT NULL);
```

Parameters

Table 109–10 PULL_SIMPLE_TABLESPACE Procedure Parameters

Parameter	Description
`tablespace_name`	The tablespace to be pulled. If `NULL`, then the procedure raises an error.
`database_link`	The name of the database link to the database that contains the tablespace to pull. The database link must be accessible to the user who runs the procedure. If `NULL`, then the procedure raises an error.
`directory_object`	The directory object to which the datafile for the tablespace is copied on the local database. You must specify the name of a directory object created using the SQL statement `CREATE DIRECTORY`. The Data Pump import log file is written to this directory. The name of the log file is the same as the datafile name for the tablespace, except with a `.plg` extension. If a file already exists with the same name as the log file in the directory, then the procedure overwrites the file. If `NULL`, then the procedure raises an error.

Table 109–10 (Cont.) PULL_SIMPLE_TABLESPACE Procedure Parameters

Parameter	Description
conversion_extension	Specify NULL if the platform is the same for the remote export database and the current import database. If the platform is different for the export database and the import database, then specify an extension for the tablespace datafile that is different than the extension for the tablespace datafile at the remote database. In this case, the procedure transfers the datafile to the import database and converts it to be compatible with the current import database platform automatically. After conversion is complete, the original datafile is deleted at the import database.

Usage Notes

To run this procedure, a user must meet the following requirements on the remote database:

- Have the EXP_FULL_DATABASE role
- Have EXECUTE privilege on the DBMS_STREAMS_TABLESPACE_ADM package
- Have access to at least one data dictionary view that contains information about the tablespaces. These views include DBA_TABLESPACES and USER_TABLESPACES.
- Have MANAGE TABLESPACE or ALTER TABLESPACE privilege on a tablespace if the tablespace must be made read-only
- Have READ privilege on the directory object for the directory that contains the datafile for the tablespace. The name of this tablespace is specified by the tablespace_name parameter. If a directory object does not exist for this directory, then create the directory object and grant the necessary privileges before you run this procedure.

To run this procedure, a user must meet the following requirements on the local database:

- Have the roles IMP_FULL_DATABASE and EXECUTE_CATALOG_ROLE
- Have WRITE privilege on the directory object that will contain the Data Pump export the log file, specified by the log_file parameter if non-NULL
- Have WRITE privilege on the directory object that will hold the datafile for the tablespace, specified by the directory_object parameter

Automatic Storage Management (ASM) directories cannot be used with this procedure.

See Also: Overview on page 109-3

PULL_TABLESPACES Procedure

This procedure copies a set of self-contained tablespaces from a remote database and attaches the tablespaces to the current database. All of the tablespaces in the specified tablespace set at the remote database must be online.

Specifically, this procedure performs the following actions:

1. Makes any read/write tablespace in the specified tablespace set at the remote database read-only
2. Uses Data Pump to export the metadata for the tablespaces in the tablespace set
3. Uses a database link and the `DBMS_FILE_TRANSFER` package to transfer the datafiles for the tablespace set and the log file for the Data Pump export to the current database
4. Places the datafiles that comprise the specified tablespace set in the specified directories at the local database
5. Places the log file for the Data Pump export in the specified directory at the local database
6. If this procedure made a tablespace read-only, then makes the tablespace read/write
7. Uses Data Pump to import the metadata for the tablespaces in the tablespace set at the local database

In addition, this procedure optionally can create datafiles for the tablespace set that can be used with the local platform, if the platform at the remote database is different than the local database platform.

Syntax

```
DBMS_STREAMS_TABLESPACE_ADM.PULL_TABLESPACES(
   datapump_job_name            IN OUT VARCHAR2,
   database_link                IN     VARCHAR2,
   tablespace_names             IN     TABLESPACE_SET,
   tablespace_directory_objects IN     DIRECTORY_OBJECT_SET,
   log_file                     IN     FILE,
   conversion_extension         IN     VARCHAR2  DEFAULT NULL);
```

Parameters

Table 109–11 PULL_TABLESPACES Procedure Parameters

Parameter	Description
`data_pump_job_name`	The Data Pump job name. Specify a Data Pump job name if you want to adhere to naming conventions or if you want to track the job more easily. If `NULL`, then the system generates a Data Pump job name.
`database_link`	The name of the database link to the database that contains the tablespace set to pull. The database link must be accessible to the user who runs the procedure. If `NULL`, then the procedure raises an error.
`tablespace_names`	The tablespace set to be pulled. If `NULL`, then the procedure raises an error.

Table 109–11 (Cont.) PULL_TABLESPACES Procedure Parameters

Parameter	Description
tablespace_directory_objects	The set of directory objects to which the datafiles for the tablespaces are copied. If more than one directory object is in the set, then the procedure copies a datafile to each directory object in the set in sequence. In this case, if the end of the directory object set is reached, then datafile copying starts again with the first directory object in the set. If NULL, then the procedure raises an error.
log_file	Specify the log file name for the Data Pump export. If NULL or if at least one file parameter is NULL, then the system generates a log file name with the extension .plg and places it in one of the datafile directories. If a file already exists with the same name as the log file in the directory, then the procedure overwrites the file.
conversion_extension	Specify NULL if the platform is the same for the remote export database and the current import database. If the platform is different for the export database and the import database, then specify an extension for the tablespace datafiles that is different than the extension for the tablespace datafiles at the remote database. In this case, the procedure transfers the datafiles to the import database and converts them to be compatible with the current import database platform automatically. After conversion is complete, the original datafiles are deleted at the import database.

Usage Notes

To run this procedure, a user must meet the following requirements on the remote database:

- Have the EXP_FULL_DATABASE role
- Have EXECUTE privilege on the DBMS_STREAMS_TABLESPACE_ADM package
- Have access to at least one data dictionary view that contains information about the tablespaces. These views include DBA_TABLESPACES and USER_TABLESPACES.
- Have MANAGE TABLESPACE or ALTER TABLESPACE privilege on a tablespace if the tablespace must be made read-only
- Have READ privilege on the directory objects for the directories that contain the datafiles for the tablespace set. The names of these tablespaces are specified by the tablespace_names parameter. If a directory object does not exist for one or more of these directories, then create the directory objects and grant the necessary privileges before you run this procedure.

To run this procedure, a user must meet the following requirements on the local database:

- Have the roles IMP_FULL_DATABASE and EXECUTE_CATALOG_ROLE
- Have WRITE privilege on the directory object that will contain the Data Pump export the log file, specified by the log_file parameter if non-NULL

- Have `WRITE` privilege on the directory objects that will hold the datafiles for the tablespaces in the set, specified by the `tablespace_directory_objects` parameter

Automatic Storage Management (ASM) directories can be specified for the directory objects that store data files and export dump files, but ASM directories cannot be specified for directory objects that store log files.

See Also: Overview on page 109-3

110

DBMS_TRACE

The DBMS_TRACE package contains the interface to trace PL/SQL functions, procedures, and exceptions.

This chapter contains the following topics:

Using DBMS_TRACE

Overview

DBMS_TRACE provides subprograms to start and stop PL/SQL tracing in a session. Oracle collects the trace data as the program executes and writes it to database tables.

A typical session involves:

- Starting PL/SQL tracing in session (DBMS_TRACE.SET_PLSQL_TRACE).
- Running an application to be traced.
- Stopping PL/SQL tracing in session (DBMS_TRACE.CLEAR_PLSQL_TRACE).

Security Model

This package must be created under `SYS`.

Constants

DBMS_TRACE uses these constants:

```
trace_all_calls          constant INTEGER := 1;
trace_enabled_calls      constant INTEGER := 2;
trace_all_exceptions     constant INTEGER := 4;
trace_enabled_exceptions constant INTEGER := 8;
trace_all_sql            constant INTEGER := 32;
trace_enabled_sql        constant INTEGER := 64;
trace_all_lines          constant INTEGER := 128;
trace_enabled_lines      constant INTEGER := 256;
trace_stop               constant INTEGER := 16384;
trace_pause              constant INTEGER := 4096;
trace_resume             constant INTEGER := 8192;
trace_limit              constant INTEGER := 16;
trace_major_version      constant BINARY_INTEGER := 1;
trace_minor_version      constant BINARY_INTEGER := 0;
```

Oracle recommends using the symbolic form for all these constants.

Restrictions

You cannot use PL/SQL tracing in a shared server environment.

Operational Notes

Controlling Data Volume

Profiling large applications may produce a large volume of data. You can control the volume of data collected by *enabling* specific program units for trace data collection.

You can enable a program unit by compiling it debug. This can be done in one of two ways:

```
alter session set plsql_debug=true;
create or replace ... /* create the library units - debug information will be
generated */
```

or:

```
/* recompile specific library unit with debug option */
alter [PROCEDURE | FUNCTION | PACKAGE BODY]  <libunit-name> compile debug;
```

Note: You cannot use the second method for anonymous blocks.

You can limit the amount of storage used in the database by retaining only the most recent 8,192 records (approximately) by including `TRACE_LIMIT` in the `TRACE_LEVEL` parameter of the `SET_PLSQL_TRACE` procedure.

Creating Database Tables to Collect DBMS_TRACE Output

You must create database tables into which the DBMS_TRACE package writes output. Otherwise, the data is not collected. To create these tables, run the script `TRACETAB.SQL`. The tables this script creates are owned by SYS.

Collecting Trace Data

The PL/SQL features you can trace are described in the script `DBMSPBT.SQL`. Some of the key tracing features are:

- Tracing Calls
- Tracing Exceptions
- Tracing SQL
- Tracing Lines

Additional features of DBMS_TRACE also allow pausing and resuming trace, and limiting the output.

Tracing Calls

Two levels of call tracing are available:

- Level 1: Trace all calls. This corresponds to the constant `trace_all_calls`.

- Level 2: Trace calls to enabled program units only. This corresponds to the constant `trace_enabled_calls`.

Enabling cannot be detected for remote procedure calls (RPCs); hence, RPCs are only traced with level 1.

Tracing Exceptions

Two levels of exception tracing are available:

- Level 1: Trace all exceptions. This corresponds to `trace_all_exceptions`.
- Level 2: Trace exceptions raised in enabled program units only. This corresponds to `trace_enabled_exceptions`.

Tracing SQL

Two levels of SQL tracing are available:

- Level 1: Trace all SQL. This corresponds to the constant `trace_all_sql`.
- Level 2: Trace SQL in enabled program units only. This corresponds to the constant `trace_enabled_sql`.

Tracing Lines

Two levels of line tracing are available:

- Level 1: Trace all lines. This corresponds to the constant `trace_all_lines`.
- Level 2: Trace lines in enabled program units only. This corresponds to the constant `trace_enabled_lines`.

When tracing lines, Oracle adds a record to the database each time the line number changes. This includes line number changes due to procedure calls and returns.

Note: For both all types of tracing, level 1 overrides level 2. For example, if both level 1 and level 2 are enabled, then level 1 takes precedence.

Collected Data

If tracing is requested only for enabled program units, and if the current program unit is not enabled, then no trace data is written.

When tracing calls, both the call and return are traced. The check for whether tracing is "enabled" passes if either the called routine or the calling routine is "enabled".

Call tracing will always output the program unit type, program unit name, and line number for both the caller and the callee. It will output the caller's stack depth. If the caller's unit is enabled, the calling procedure name will also be output. If the callee's unit is enabled, the called procedure name will be output

Exception tracing writes out the line number. Raising the exception shows information on whether the exception is user-defined or pre-defined. It also shows the exception number in the case of pre-defined exceptions. Both the place where the exceptions are raised and their handler is traced. The check for tracing being "enabled" is done independently for the place where the exception is raised and the place where the exception is handled.

All calls to `DBMS_TRACE.SET_PLSQL_TRACE` and `DBMS_TRACE.CLEAR_PLSQL_TRACE` place a special trace record in the database. Therefore, it is always possible to determine when trace settings were changed.

Trace Control

As well as determining which items are collected, you can pause and resume the trace process. No information is gathered between the time that tracing is paused and the time that it is resumed. The constants `TRACE_PAUSE` and `TRACE_RESUME` are used to accomplish this. Trace records are generated to indicate that the trace was paused/resumed.

It is also possible to retain only the last 8,192 trace events of a run by using the constant `TRACE_LIMIT`. This allows tracing to be turned on without filling up the database. When tracing stops, the last 8,192 records are saved. The limit is approximate, since it is not checked on every trace record. At least the requested number of trace records will be generated; up to 1,000 additional records may be generated.

Summary of DBMS_TRACE Subprograms

Table 110–1 DBMS_TRACE Package Subprograms

Subprogram	Description
CLEAR_PLSQL_TRACE Procedure on page 110-11	Stops trace data dumping in session
PLSQL_TRACE_VERSION Procedure on page 110-12	Gets the version number of the trace package
SET_PLSQL_TRACE Procedure on page 110-13	Starts tracing in the current session

CLEAR_PLSQL_TRACE Procedure

This procedure disables trace data collection.

Syntax

```
DBMS_TRACE.CLEAR_PLSQL_TRACE;
```

PLSQL_TRACE_VERSION Procedure

This procedure gets the version number of the trace package. It returns the major and minor version number of the DBMS_TRACE package.

Syntax

```
DBMS_TRACE.PLSQL_TRACE_VERSION (
   major OUT BINARY_INTEGER,
   minor OUT BINARY_INTEGER);
```

Parameters

Table 110–2 PLSQL_TRACE_VERSION Procedure Parameters

Parameter	Description
major	Major version number of DBMS_TRACE.
minor	Minor version number of DBMS_TRACE.

SET_PLSQL_TRACE Procedure

This procedure enables PL/SQL trace data collection.

Syntax

```
DBMS_TRACE.SET_PLSQL_TRACE (
   trace_level INTEGER);
```

Parameters

Table 110–3 SET_PLSQL_TRACE Procedure Parameters

Parameter	Description
`trace_level`	You must supply one or more of the constants as listed on page 110-5. By summing the constants, you can enable tracing of multiple PL/SQL language features simultaneously. The control constants "trace_pause", "trace_resume" and "trace_stop" should not be used in combination with other constants Also see "Collecting Trace Data" on page 110-7 for more information.

111

DBMS_TRANSACTION

The DBMS_TRANSACTION package provides access to SQL transaction statements from stored procedures.

> **See Also:** *Oracle Database SQL Reference*

This chapter contains the following topics:

Using DBMS_TRANSACTION

Security Model

This package runs with the privileges of calling user, rather than the package owner `SYS`.

Summary of DBMS_TRANSACTION Subprograms

Table 111–1 DBMS_TRANSACTION Package Subprograms

Subprogram	Description
ADVISE_COMMIT Procedure on page 111-6	Equivalent to the SQL statement: `ALTER SESSION ADVISE COMMIT`
ADVISE_NOTHING Procedure on page 111-7	Equivalent to the SQL statement: `ALTER SESSION ADVISE NOTHING`
ADVISE_ROLLBACK Procedure on page 111-8	Equivalent to the SQL statement: `ALTER SESSION ADVISE ROLLBACK`
BEGIN_DISCRETE_TRANSACTION Procedure on page 111-9	Sets "discrete transaction mode" for this transaction
COMMIT Procedure on page 111-10	Equivalent to the SQL statement: `COMMIT`
COMMIT_COMMENT Procedure on page 111-11	Equivalent to the SQL statement: `COMMIT COMMENT <text>`
COMMIT_FORCE Procedure on page 111-12	Equivalent to the SQL statement: `COMMIT FORCE <text>, <number>"`
LOCAL_TRANSACTION_ID Function on page 111-13	Returns the local (to instance) unique identifier for the current transaction
PURGE_LOST_DB_ENTRY Procedure on page 111-14	Enables removal of incomplete transactions from the local site when the remote database is destroyed or re-created before recovery completes
PURGE_MIXED Procedure on page 111-16	Deletes information about a given mixed outcome transaction
READ_ONLY Procedure on page 111-17	Equivalent to the SQL statement: `SET TRANSACTION READ ONLY`
READ_WRITE Procedure on page 111-18	equivalent to the SQL statement: `SET TRANSACTION READ WRITE`
ROLLBACK Procedure on page 111-19	Equivalent to the SQL statement: `ROLLBACK`
ROLLBACK_FORCE Procedure on page 111-20	Equivalent to the SQL statement: `ROLLBACK FORCE <text>`
ROLLBACK_SAVEPOINT Procedure on page 111-21	Equivalent to the SQL statement: `ROLLBACK TO SAVEPOINT <savepoint_name>`
SAVEPOINT Procedure on page 111-22	Equivalent to the SQL statement: `SAVEPOINT <savepoint_name>`
STEP_ID Function on page 111-23	Returns local (to local transaction) unique positive integer that orders the DML operations of a transaction

Table 111–1 (Cont.) DBMS_TRANSACTION Package Subprograms

Subprogram	Description
USE_ROLLBACK_SEGMENT Procedure on page 111-24	Equivalent to the SQL statement: `SET TRANSACTION USE ROLLBACK SEGMENT <rb_seg_name>`

ADVISE_COMMIT Procedure

This procedure is equivalent to the SQL statement:

```
ALTER SESSION ADVISE COMMIT
```

Syntax

```
DBMS_TRANSACTION.ADVISE_COMMIT;
```

ADVISE_NOTHING Procedure

This procedure is equivalent to the SQL statement:

```
ALTER SESSION ADVISE NOTHING
```

Syntax

```
DBMS_TRANSACTION.ADVISE_NOTHING;
```

ADVISE_ROLLBACK Procedure

This procedure is equivalent to the SQL statement:

```
ALTER SESSION ADVISE ROLLBACK
```

Syntax

```
DBMS_TRANSACTION.ADVISE_ROLLBACK;
```

BEGIN_DISCRETE_TRANSACTION Procedure

This procedure sets "discrete transaction mode" for this transaction.

Syntax

```
DBMS_TRANSACTION.BEGIN_DISCRETE_TRANSACTION;
```

Exceptions

Table 111–2 BEGIN_DISCRETE_TRANSACTION Procedure Exceptions

Exception	Description
ORA-08175	A transaction attempted an operation which cannot be performed as a discrete transaction. If this exception is encountered, then rollback and retry the transaction
ORA-08176	A transaction encountered data changed by an operation that does not generate rollback data: create index, direct load or discrete transaction. If this exception is encountered, then retry the operation that received the exception.

Examples

```
DISCRETE_TRANSACTION_FAILED exception;
    pragma exception_init(DISCRETE_TRANSACTION_FAILED, -8175);
CONSISTENT_READ_FAILURE exception;
    pragma exception_init(CONSISTENT_READ_FAILURE, -8176);
```

COMMIT Procedure

This procedure is equivalent to the SQL statement:

```
COMMIT
```

This procedure is included for completeness, the functionality being already implemented as part of PL/SQL.

Syntax

```
DBMS_TRANSACTION.COMMIT;
```

COMMIT_COMMENT Procedure

This procedure is equivalent to the SQL statement:

```
COMMIT COMMENT <text>
```

Syntax

```
DBMS_TRANSACTION.COMMIT_COMMENT (
   cmnt VARCHAR2);
```

Parameters

Table 111–3 COMMIT_COMMENT Procedure Parameters

Parameter	Description
cmnt	Comment to associate with this commit.

COMMIT_FORCE Procedure

This procedure is equivalent to the SQL statement:

```
COMMIT FORCE <text>, <number>"
```

Syntax

```
DBMS_TRANSACTION.COMMIT_FORCE (
   xid VARCHAR2,
   scn VARCHAR2 DEFAULT NULL);
```

Parameters

Table 111–4 COMMIT_FORCE Procedure Parameters

Parameter	Description
`xid`	Local or global transaction ID.
`scn`	System change number.

LOCAL_TRANSACTION_ID Function

This function returns the local (to instance) unique identifier for the current transaction. It returns null if there is no current transaction.

Syntax

```
DBMS_TRANSACTION.LOCAL_TRANSACTION_ID (
   create_transaction BOOLEAN := FALSE)
  RETURN VARCHAR2;
```

Parameters

Table 111–5 LOCAL_TRANSACTION_ID Function Parameters

Parameter	Description
`create_transaction`	If true, then start a transaction if one is not currently active.

PURGE_LOST_DB_ENTRY Procedure

When a failure occurs during commit processing, automatic recovery consistently resolves the results at all sites involved in the transaction. However, if the remote database is destroyed or re-created before recovery completes, then the entries used to control recovery in DBA_2PC_PENDING and associated tables are never removed, and recovery will periodically retry. Procedure PURGE_LOST_DB_ENTRY enables removal of such transactions from the local site.

Syntax

```
DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY (
   xid VARCHAR2);
```

Parameters

Table 111–6 PURGE_LOST_DB_ENTRY Procedure Parameters

Parameter	Description
xid	Must be set to the value of the LOCAL_TRAN_ID column in the DBA_2PC_PENDING table.

Usage Notes

WARNING: PURGE_LOST_DB_ENTRY should *only* be used when the other database is lost or has been re-created. Any other use may leave the other database in an unrecoverable or inconsistent state.

Before automatic recovery runs, the transaction may show up in DBA_2PC_PENDING as state "collecting", "committed", or "prepared". If the DBA has forced an in-doubt transaction to have a particular result by using "commit force" or "rollback force", then states "forced commit" or "forced rollback" may also appear. Automatic recovery normally deletes entries in any of these states. The only exception is when recovery finds a forced transaction which is in a state inconsistent with other sites in the transaction; in this case, the entry is left in the table and the MIXED column has the value 'yes'.

However, under certain conditions, it may not be possible for automatic recovery to run. For example, a remote database may have been permanently lost. Even if it is re-created, it gets a new database ID, so that recovery cannot identify it (a possible symptom is ORA-02062). In this case, the DBA may use the procedure PURGE_LOST_DB_ENTRY to clean up the entries in any state other than "prepared". The DBA does not need to be in any particular hurry to resolve these entries, because they are not holding any database resources.

The following table indicates what the various states indicate about the transaction and what the DBA actions should be:

Table 111–7 PURGE_LOST_DB_ENTRY Procedure States

State of Column	State of Global Transaction	State of Local Transaction	Normal DBA Action	Alternative DBA Action
Collecting	Rolled back	Rolled back	None	PURGE_LOST_DB_ENTRY (See Note 1)
Committed	Committed	Committed	None	PURGE_LOST_DB_ENTRY (See Note 1)
Prepared	Unknown	Prepared	None	FORCE COMMIT or ROLLBACK
Forced commit	Unknown	Committed	None	PURGE_LOST_DB_ENTRY (See Note 1)
Forced rollback	Unknown	Rolled back	None	PURGE_LOST_DB_ENTRY (See Note 1)
Forced commit (mixed)	Mixed	Committed	(See Note 2)	
Forced rollback (mixed)	Mixed	Rolled back	(See Note 2)	

NOTE 1: Use only if significant reconfiguration has occurred so that automatic recovery cannot resolve the transaction. Examples are total loss of the remote database, reconfiguration in software resulting in loss of two-phase commit capability, or loss of information from an external transaction coordinator such as a TP monitor.

NOTE 2: Examine and take any manual action to remove inconsistencies; then use the procedure PURGE_MIXED.

PURGE_MIXED Procedure

When in-doubt transactions are forced to commit or rollback (instead of letting automatic recovery resolve their outcomes), there is a possibility that a transaction can have a mixed outcome: Some sites commit, and others rollback. Such inconsistency cannot be resolved automatically by Oracle; however, Oracle flags entries in `DBA_2PC_PENDING` by setting the `MIXED` column to a value of 'yes'.

Oracle never automatically deletes information about a mixed outcome transaction. When the application or DBA is certain that all inconsistencies that might have arisen as a result of the mixed transaction have been resolved, this procedure can be used to delete the information about a given mixed outcome transaction.

Syntax

```
DBMS_TRANSACTION.PURGE_MIXED (
   xid VARCHAR2);
```

Parameters

Table 111–8 PURGE_MIXED Procedure Parameters

Parameter	Description
`xid`	Must be set to the value of the `LOCAL_TRAN_ID` column in the `DBA_2PC_PENDING` table.

READ_ONLY Procedure

This procedure is equivalent to the SQL statement:

```
SET TRANSACTION READ ONLY
```

Syntax

```
DBMS_TRANSACTION.READ_ONLY;
```

READ_WRITE Procedure

This procedure is equivalent to the SQL statement:

```
SET TRANSACTION READ WRITE
```

Syntax

```
DBMS_TRANSACTION.READ_WRITE;
```

ROLLBACK Procedure

This procedure is equivalent to the SQL statement:

```
ROLLBACK
```

This procedure is included for completeness, the functionality being already implemented as part of PL/SQL.

Syntax

```
DBMS_TRANSACTION.ROLLBACK;
```

ROLLBACK_FORCE Procedure

This procedure is equivalent to the SQL statement:

```
ROLLBACK FORCE <text>
```

Syntax

```
DBMS_TRANSACTION.ROLLBACK_FORCE (
   xid VARCHAR2);
```

Parameters

Table 111–9 ROLLBACK_FORCE Procedure Parameters

Parameter	Description
`xid`	Local or global transaction ID.

ROLLBACK_SAVEPOINT Procedure

This procedure is equivalent to the SQL statement:

```
ROLLBACK TO SAVEPOINT <savepoint_name>
```

This procedure is included for completeness, the functionality being already implemented as part of PL/SQL.

Syntax

```
DBMS_TRANSACTION.ROLLBACK_SAVEPOINT (
   savept VARCHAR2);
```

Parameters

Table 111–10 ROLLBACK_SAVEPOINT Procedure Parameters

Parameter	Description
savept	Savepoint identifier.

SAVEPOINT Procedure

This procedure is equivalent to the SQL statement:

```
SAVEPOINT <savepoint_name>
```

This procedure is included for completeness, the feature being already implemented as part of PL/SQL.

Syntax

```
DBMS_TRANSACTION.SAVEPOINT (
   savept VARCHAR2);
```

Parameters

Table 111–11 SAVEPOINT Procedure Parameters

Parameter	Description
savept	Savepoint identifier.

STEP_ID Function

This function returns local (to local transaction) unique positive integer that orders the DML operations of a transaction.

Syntax

```
DBMS_TRANSACTION.STEP_ID
   RETURN NUMBER;
```

USE_ROLLBACK_SEGMENT Procedure

This procedure is equivalent to the SQL statement:

```
SET TRANSACTION USE ROLLBACK SEGMENT <rb_seg_name>
```

Syntax

```
DBMS_TRANSACTION.USE_ROLLBACK_SEGMENT (
   rb_name VARCHAR2);
```

Parameters

Table 111–12 USE_ROLLBACK_SEGMENT Procedure Parameters

Parameter	Description
rb_name	Name of rollback segment to use.

112

DBMS_TRANSFORM

The DBMS_TRANSFORM package provides an interface to the message format transformation features of Oracle Advanced Queuing.

> **See Also:** *Oracle Streams Advanced Queuing User's Guide and Reference* for more on message format transformations.

This chapter contains the following topic:

Summary of DBMS_TRANSFORM Subprograms

Table 112–1 DBMS_TRANSFORM Package Subprograms

Subprograms	Description
CREATE_ TRANSFORMATION Procedure on page 112-3	Creates a transformation that maps an object of the source type to an object of the destination type
DROP_ TRANSFORMATION Procedure on page 112-5	Drops the given transformation
MODIFY_ TRANSFORMATION Procedure on page 112-6	Modifies an existing transformation

CREATE_TRANSFORMATION Procedure

This procedure creates a transformation that maps an object of the source type to an object of the target type. The transformation expression can be a SQL expression or a PL/SQL function. It must return an object of the target type.

Syntax

```
DBMS_TRANSFORM.CREATE_TRANSFORMATION (
    schema               VARCHAR2(30),
    name                 VARCHAR2(30),
    from_schema          VARCHAR2(30),
    from_type            VARCHAR2(30),
    to_schema            VARCHAR2(30),
    to_type              VARCHAR2(30),
    transformation       VARCHAR2(4000));
```

Parameters

Table 112–2 CREATE_TRANSFORMATION Procedure Parameters

Parameter	Description
schema	Specifies the schema of the transformation.
name	Specifies the name of the transformation.
from_schema	Specifies the schema of the source type.
from_type	Specifies the source type.
to_schema	Specifies the target type schema.
to_type	Specifies the target type.
transformation	Specifies the transformation expression, returning an object of the target type. The expression must be a function returning an object of the target type or a constructor expression for the target type. You can choose not to specify a transformation expression and instead specify transformations for attributes of the target type using MODIFY_TRANSFORMATION.

Usage Notes

- The transformation expression must be a SQL expression or a PL/SQL function returning the type of the specified attribute of the target type.
- To create, modify or drop transformations, a user must be granted execute privileges on DBMS_TRANSFORM. The user must also have execute privileges on the user defined types that are the source and destination types of the transformation. In addition, the user must also have execute privileges on any PLSQL function being used in the transformation function.
- The transformation cannot write database state (perform DML) or commit or rollback the current transaction.
- The transformation must be a SQL function with source type as input type, returning an object of the target type. It could also be a SQL expression of target type, referring to a source type. All references to the source type must be of the form *source.user_data*.

- Both source and target types must be non-scalar database types. A null transformation expression maps to a null target object.

For using the transformation at enqueue and dequeue time, the login user invoking the operation must have execute privileges on the PLSQL functions used by the transformation. For propagation, the owning schema of the queue must have these privileges.

DROP_TRANSFORMATION Procedure

This procedure drops the given transformation.

Syntax

```
DBMS_TRANSFORM.DROP_TRANSFORMATION (
  schema      VARCHAR2(30),
  name        VARCHAR2(30));
```

Parameters

Table 112–3 DROP_TRANSFORMATION Procedure Parameters

Parameter	Description
schema	Specifies the schema of the transformation.
name	Specifies the name of the transformation.

MODIFY_TRANSFORMATION Procedure

This procedure modifies the transformation expression for the given transformation.

Syntax

```
DBMS_TRANSFORM.MODIFY_TRANSFORMATION (
   schema             VARCHAR2(30),
   name               VARCHAR2(30),
   attribute_number   INTEGER,
   transformation     VARCHAR2(4000));
```

Parameters

Table 112–4 MODIFY_TRANSFORMATION Procedure Parameters

Parameter	Description
`schema`	Specifies the schema of the transformation.
`name`	Specifies the name of the transformation.
`attribute_number`	The attribute of the target type for which the new transformation expression is being specified. When specifying the new transformation as a single expression of the target type, specify a value of 0.
`transformation`	The transformation expression must be a SQL expression or a PL/SQL function returning the type of the specified attribute of the target type. If the attribute_number is 0, then the expression must be a PL/SQL function returning an object of the target type or a constructor expression for the target type.

Usage Notes

- If the new transformation is a single expression of the target type, it may be specified with an `attribute_number` of 0. The new transformation may also be specified for each attribute of the target type.
- You can use this procedure to define the transformation as a separate expression for each attribute of the target type. For large transformations, this representation may be more readable and allow the application of fine grain control over the transformation. If the transformation expression was left unspecified for some of the attributes of the target type, they are evaluated to null when the transformation is applied.

113

DBMS_TDB

The DBMS_TDB package reports whether a database can be transported between platforms using the RMAN `CONVERT DATABASE` command. It verifies that databases on the current host platform are of the same endian format as the destination platform, and that the state of the current database does not prevent transport of the database.

> **See Also:** *Oracle Database Backup and Recovery Advanced User's Guide* regarding database transport using `CONVERT DATABASE`

This chapter contains the following topics:

Using DBMS_TDB

This section contains topics which relate to using `DBMS_TDB`.

Overview

In many cases, Oracle supports transporting databases between platforms which have the same endian format. However, even when the endian formats are the same, a database must undergo a conversion process to move from one platform to another. There are also preconditions required for the process of transporting a database, such as having the database to be transported open read-only.

The `DBMS_TDB` package serves two purposes:

- Confirming that Oracle supports transporting a database from a given source platform to a given target platform
- Determining whether a database to be transported has been properly prepared for transport, and if not, identifying the condition that prevents database transport

The actual conversion is performed using the Recovery Manager `CONVERT DATABASE` command. For a complete discussion of the requirements for transporting a database, the process of converting a database for transport across platforms, and examples of the use of the DBMS_TDB subprograms in the conversion process, see*Oracle Database Backup and Recovery Advanced User's Guide*.

Security Model

Use of this package requires the DBA privilege.

Constants

The DBMS_TDB package defines several enumerated constants that should be used for specifying parameter values. Enumerated constants must be prefixed with the package name, for example, `DBMS_TDB.SKIP_NONE`.

The DBMS_TDB package uses the constants shown in Table 113–1.

Table 113–1 DBMS_TDB Constants

Name	Type	Value	Description
`SKIP_NONE`	`NUMBER`	`0`	Check all files when checking whether a database is ready for transport.
`SKIP_OFFLINE`	`NUMBER`	`2`	Skip files in offline tablespaces when checking whether a database is ready for transport.
`SKIP_READONLY`	`NUMBER`	`3`	Skip files in readonly tablespaces when checking whether a database is ready for transport.

Views

The DBMS_TDB package uses the following view listed in *Oracle Database Reference*:

- V$DB_TRANSPORTABLE_PLATFORM, which specifies which combinations of source and target platforms support database transport.

Operational Notes

- The subprograms in this package are useful both in determining whether the desired cross-platform database conversion is possible, and in checking whether your database is ready for conversion. See *Oracle Database Backup and Recovery Advanced User's Guide* for details on the different uses of these subprograms are used in the conversion process.
- The subprograms in this package return simple `TRUE` or `FALSE` results to indicate whether database transport is possible. Use the subprograms with `SERVEROUTPUT ON` for informative messages about why transport is not possible.

Summary of DBMS_TDB Subprograms

Table 113–2 DBMS_TDB Package Subprograms

Subprogram	Description
CHECK_DB Function on page 113-9	Checks whether a database can be transported to a target platform
CHECK_EXTERNAL Function on page 113-9	Checks whether a database has external tables, directory or BFILEs

CHECK_DB Function

This function checks whether a database can be transported to a target platform. It tests whether transport is supported at all for a given source and destination platform, and whether the database is currently in the correct state for transport.

You can specify whether to skip checking parts of the database that are read-only or offline, if you do not plan to transport them.

The function is overloaded. The different functionality of each form of syntax is presented along with the definition.

Syntax

```
DBMS_TDB.CHECK_DB (
     target_platform_name   IN VARCHAR2,
     skip_option            IN  NUMBER)
   RETURN BOOLEAN;

DBMS_TDB.CHECK_DB (
     target_platform_name   IN VARCHAR2)
   RETURN BOOLEAN;

DBMS_TDB.CHECK_DB
   RETURN BOOLEAN;
```

Parameters

Table 113–3 CHECK_DB Procedure Parameters

Parameter	Description
target_platform_name	The name of the destination platform, as it appears in V$DB_TRANSPORTABLE_PLATFORM
skip_option	Specifies which, if any, parts of the database to skip when checking whether the database can be transported. Supported values are listed in Table 113–1, " DBMS_TDB Constants" on page 113-5.

Return Values

If the database cannot be transported to the target platform or is not ready to be transported, returns FALSE. If the database is ready for transport, returns TRUE.

Usage Notes

- If SERVEROUTPUT is ON, the output will contain the reasons why the database cannot be transported and how to fix the problems. For details on possible reasons and fixes, see Table 113–4, " Reasons for CHECK_DB Function to Return FALSE" on page 113-9.

Table 113–4 Reasons for CHECK_DB Function to Return FALSE

Cause	Action
Unrecognized target platform name.	Check V$DB_TRANSPORTABLE_PLATFORM for recognized platform names
Target platform has a different endian format.	Conversion is not supported

Table 113–4 (Cont.) Reasons for CHECK_DB Function to Return FALSE

Cause	Action
Database is not open read-only.	Open database read-only and retry
There are active or in-doubt transactions in the database.	Open the database read-write. After the active transactions are rolled back, open the database read-only and retry. This can happen if users flashback the database and open it read only. The active transactions will be rolled back when the database is opened read-write.
Deferred transaction rollback needs to be done.	Open the database read-write and bring online the necessary tablespaces. Once the deferred transaction rollback is complete, open the database read-only and retry.
Database compatibility version is below 10.	Change the init.ora `COMPATIBLE` parameter to 10 or higher, open the database read-only and retry
Some tablespaces have not been open read-write with compatibility version is 10 or higher.	Change the init.ora `COMPATIBLE` parameter to 10 or higher. Then open the affected tablespaces read-write. Then shut down the database, open it read-only, and retry.

Examples

This example illustrates the use of `CHECK_DB` with a database that is open read-write:

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
       db_ready BOOLEAN;
     BEGIN
       db_ready := DBMS_TDB.CHECK_DB('Microsoft Windows IA (32-bit)');
     END;
     /

Database is not open READ ONLY. Please open database READ ONLY and retry.

PL/SQL procedure successfully completed.
```

CHECK_EXTERNAL Function

This function determines whether a database has external tables, directories or BFILEs.

Syntax

```
DBMS_TDB.CHECK_EXTERNAL
   RETURN BOOLEAN;
```

Return Values

If the database has external tables, directories, or BFILEs, return `TRUE`. Otherwise, return `FALSE`.

Usage Notes

- If `SERVEROUTPUT` is `ON`, the function will output the names of the external tables, directories and BFILEs in the database.
- The database must be open read-write.

Examples

This example illustrates the use of `CHECK_EXTERNAL` with a database that has several external tables, directories and BFILEs:

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
        external BOOLEAN;
     BEGIN
       external := DBMS_TDB.CHECK_EXTERNAL;
     END;
     /
The following external tables exist in the database:
SH.SALES_TRANSACTIONS_EXT
The following directories exist in the database:
SYS.MEDIA_DIR, SYS.DATA_FILE_DIR, SYS.LOG_FILE_DIR, SYS.DATA_PUMP_DIR
The following BFILEs exist in the database:
PM.PRINT_MEDIA

PL/SQL procedure successfully completed.
```

114

DBMS_TTS

The DBMS_TTS package checks if the transportable set is self-contained. All violations are inserted into a temporary table that can be selected from the view `TRANSPORT_SET_VIOLATIONS`.

See Also:

- *Oracle Database Administrator's Guide*
- *Oracle Database Upgrade Guide*

This chapter contains the following topics:

Using DBMS_TTS

Security Model

Only users having the `execute_catalog_role` can execute this procedure. This role is initially only assigned to user SYS.

Exceptions

```
ts_not_found  EXCEPTION;
PRAGMA exception_init(ts_not_found, -29304);
ts_not_found_num NUMBER := -29304;

invalid_ts_list  EXCEPTION;
PRAGMA exception_init(invalid_ts_list, -29346);
invalid_ts_list_num NUMBER := -29346;

sys_or_tmp_ts     EXCEPTION;
PRAGMA exception_init(sys_or_tmp_ts, -29351);
sys_or_tmp_ts_num NUMBER := -29351;
```

Operational Notes

With respect to transportable tablespaces, disabled and enabled referential integrity constraints are handled differently:

- A disabled referential integrity constraint does not violate the transportability rules and is dropped during the import phase.
- An enabled referential integrity constraint violates the transportability rules if it references a table in a tablespace outside the transportable set.

Summary of DBMS_TTS Subprograms

These two procedures are designed to be called by database administrators.

Table 114–1 DBMS_TTS Package Subprograms

Subprogram	Description
DOWNGRADE Procedure on page 114-7	Downgrades transportable tablespace related data
TRANSPORT_SET_CHECK Procedure on page 114-8	Checks if a set of tablespaces (to be transported) is self-contained

DOWNGRADE Procedure

This procedure downgrades transportable tablespace related data.

Syntax

```
DBMS_TTS.DOWNGRADE;
```

TRANSPORT_SET_CHECK Procedure

This procedure checks if a set of tablespaces (to be transported) is self-contained. After calling this procedure, the user may select from a view to see a list of violations, if there are any.

Syntax

```
DBMS_TTS.TRANSPORT_SET_CHECK (
   ts_list          IN CLOB,
   incl_constraints IN BOOLEAN DEFAULT FALSE,
   full_check       IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 114–2 TRANSPORT_SET_CHECK Procedure Parameters

Parameter	Description
ts_list	List of tablespace, separated by comma.
incl_constraints	TRUE if you want to count in referential integrity constraints when examining if the set of tablespaces is self-contained. (The incl_constraints parameter is a default so that TRANSPORT_SET_CHECK will work if it is called with only the ts_list argument.)
full_check	Indicates whether a full or partial dependency check is required. If TRUE, treats all IN and OUT pointers (dependencies) and captures them as violations if they are not self-contained in the transportable set. The parameter should be set to TRUE for TSPITR or if a strict version of transportable is desired. By default the parameter is set to false. It will only consider OUT pointers as violations.

Examples

If the view does not return any rows, then the set of tablespaces is self-contained. For example,

```
SQLPLUS> EXECUTE DBMS_TTS.TRANSPORT_SET_CHECK('foo,bar', TRUE);
SQLPLUS> SELECT * FROM TRANSPORT_SET_VIOLATIONS;
```

115

DBMS_TYPES

The DBMS_TYPES package consists of constants, which represent the built-in and user-defined types.

This chapter contains the following topics:

- Using DBMS_TYPES
  - Constants
  - Exceptions

Using DBMS_TYPES

Constants

The following table lists the constants in the DBMS_TYPES package.

Table 115–1 DBMS_TYPES Constants

Constant	Description
NO_DATA	Is only relevant if PieceWise is called, for a collection or anydataset. Denotes the end of collection/anydataset when all the elements have been accessed
SUCCESS	The operation succeeded
TYPECODE_BDOUBLE	A NUMBER type
TYPECODE_BFILE	A BFILE type
TYPECODE_BFLOAT	A NUMBER type
TYPECODE_BLOB	A BLOB type
TYPECODE_CFILE	A CFILE type
TYPECODE_CHAR	A CHAR type
TYPECODE_CLOB	A CLOB type
TYPECODE_DATE	A DATE type
TYPECODE_INTERVAL_DS	An INTERVAL_DS type
TYPECODE_INTERVAL_YM	A INTERVAL_YM type
TYPECODE_MLSLABEL	An MLSLABEL type
TYPECODE_NAMEDCOLLECTION	A named collection (VARRAY/nested table) type
TYPECODE_NUMBER	A NUMBER type
TYPECODE_OBJECT	An OBJECT type
TYPECODE_OPAQUE	An OPAQUE type
TYPECODE_RAW	A RAW type
TYPECODE_REF	A REF type
TYPECODE_TABLE	A nested table collection type
TYPECODE_TIMESTAMP	A TIMESTAMP type
TYPECODE_TIMESTAMP_LTZ	A TIMESTAMP_LTZ type
TYPECODE_TIMESTAMP_TZ	A TIMESTAMP_TZ type
TYPECODE_VARCHAR2	A VARCHAR2 type
TYPECODE_VARCHAR	A VARCHAR type
TYPECODE_VARRAY	A VARRAY collection type

Exceptions

- INVALID_PARAMETERS
- INCORRECT_USAGE
- TYPE_MISMATCH

116

DBMS_UTILITY

The DBMS_UTILITY package provides various utility subprograms.

This chapter contains the following topics:

Using DBMS_UTILITY

Security Model

DBMS_UTILITY runs with the privileges of the calling user for the NAME_RESOLVE Procedure, the COMPILE_SCHEMA Procedure, and the ANALYZE_SCHEMA Procedure. This is necessary so that the SQL works correctly.

The package does not run as SYS. The privileges are checked using DBMS_DDL.

Constants

The DBMS_UTILITY package uses the constants shown in Table 116–1, " DBMS_ UTILITY Constants".

Table 116–1 DBMS_UTILITY Constants

Name	Type	Value	Description
INV_ERROR_ON_ RESTRICTIONS	PLS_INTEGER	1	This constant is the only legal value for the p_option_flags parameter of the INVALIDATE subprogram

Types

dblink_array

```
TYPE dblink_array IS TABLE OF VARCHAR2(128) INDEX BY BINARY_INTEGER;
```

Lists of database links should be stored here.

index_table_type

```
TYPE index_table_type IS TABLE OF BINARY_INTEGER INDEX BY BINARY_INTEGER;
```

The order in which objects should be generated is returned here.

instance_record

```
  TYPE instance_record IS RECORD (
       inst_number   NUMBER,
       inst_name     VARCHAR2(60));
  TYPE instance_table IS TABLE OF instance_record INDEX BY BINARY_INTEGER;
```

The list of active instance number and instance name.

The starting index of `instance_table` is 1; `instance_table` is dense.

lname_array

```
TYPE lname_array IS TABLE OF VARCHAR2(4000) index by BINARY_INTEGER;
```

Lists of Long `NAME` should be stored here, it includes fully qualified attribute names.

name_array

```
TYPE name_array IS TABLE OF VARCHAR2(30) INDEX BY BINARY_INTEGER;
```

Lists of `NAME` should be stored here.

number_array

```
TYPE number_array IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;
```

The order in which objects should be generated is returned here for users.

uncl_array

```
TYPE uncl_array IS TABLE OF VARCHAR2(227) INDEX BY BINARY_INTEGER;
```

Lists of "USER"."NAME"."COLUMN"@LINK should be stored here.

Deprecated Subprograms

Obsolete with Oracle Database Release 10*g*:

- ANALYZE_DATABASE Procedure
- ANALYZE_SCHEMA Procedure

Exceptions

The following table lists the exceptions raised by DBMS_UTILITY.

Table 116–2 Exceptions Raised by DBMS_UTILITY

Exception	Error Code	Description
INV_NOT_EXIST_OR_NO_PRIV	-24237	Raised by the INVALIDATE subprogram when the object_id argument is NULL or invalid, or when the caller does not have CREATE privileges on the object being invalidated
INV_MALFORMED_SETTINGS	-24238	Raised by the INVALIDATE subprogram if a compiler setting is specified more than once in the p_plsql_object_settings parameter
INV_RESTRICTED_OBJECT	-24239	Raised by the INVALIDATE subprogram when different combinations of conditions pertaining to the p_object_id parameter are contravened

Summary of DBMS_UTILITY Subprograms

Table 116–3 DBMS_UTILITY Package Subprograms

Subprogram	Description
ACTIVE_INSTANCES Procedure on page 116-10	Returns the active instance
ANALYZE_DATABASE Procedure on page 116-11	Analyzes all the tables, clusters, and indexes in a database [see also Deprecated Subprograms]
ANALYZE_PART_OBJECT Procedure on page 116-12	Analyzes the given tables and indexes
ANALYZE_SCHEMA Procedure on page 116-13	Analyzes all the tables, clusters, and indexes in a schema [see also Deprecated Subprograms]
CANONICALIZE Procedure on page 116-14	Canonicalizes a given string
COMMA_TO_TABLE Procedures on page 116-15	Converts a comma-delimited list of names into a PL/SQL table of names
COMPILE_SCHEMA Procedure on page 116-16	Compiles all procedures, functions, packages, and triggers in the specified schema
CREATE_ALTER_TYPE_ERROR_TABLE Procedure on page 116-17	Creates an error table to be used in the `EXCEPTION` clause of the `ALTER TYPE` statement
CURRENT_INSTANCE Function on page 116-18	Returns the current connected instance number
DATA_BLOCK_ADDRESS_BLOCK Function on page 116-19	Gets the block number part of a data block address
DATA_BLOCK_ADDRESS_FILE Function on page 116-20	Gets the file number part of a data block address
DB_VERSION Procedure on page 116-21	Returns version information for the database
EXEC_DDL_STATEMENT Procedure on page 116-22	Executes the DDL statement in `parse_string`
FORMAT_CALL_STACK Function on page 116-23	Formats the current call stack
FORMAT_ERROR_BACKTRACE Function on page 116-24	Formats the backtrace from the point of the current error to the exception handler where the error has been caught
FORMAT_ERROR_STACK Function on page 116-27	Formats the current error stack
GET_CPU_TIME Function on page 116-28	Returns the current CPU time in 100th's of a second
GET_DEPENDENCY Procedure on page 116-29	Shows the dependencies on the object passed in.
GET_HASH_VALUE Function on page 116-30	Computes a hash value for the given string
GET_PARAMETER_VALUE Function on page 116-31	Gets the value of specified init.ora parameter
GET_TIME Function on page 116-33	Finds out the current time in 100th's of a second

Table 116–3 (Cont.) DBMS_UTILITY Package Subprograms

Subprogram	Description
INVALIDATE Procedure on page 116-34	Invalidates a database object and (optionally) modifies its PL/SQL compiler parameter settings
IS_CLUSTER_DATABASE Function on page 116-37	Finds out if this database is running in cluster database mode
MAKE_DATA_BLOCK_ ADDRESS Function on page 116-38	Creates a data block address given a file number and a block number
NAME_RESOLVE Procedure on page 116-39	Resolves the given name
NAME_RESOLVE Procedure on page 116-39	Calls the parser to parse the given name
PORT_STRING Function on page 116-42	Returns a string that uniquely identifies the version of Oracle and the operating system
TABLE_TO_COMMA Procedures on page 116-43	Converts a PL/SQL table of names into a comma-delimited list of names
VALIDATE Procedure on page 116-44	Converts a PL/SQL table of names into a comma-delimited list of names

ACTIVE_INSTANCES Procedure

This procedure returns the active instance.

Syntax

```
DBMS_UTILITY.ACTIVE_INSTANCES (
   instance_table   OUT INSTANCE_TABLE,
   instance_count   OUT NUMBER);
```

Parameters

Table 116–4 ACTIVE_INSTANCES Procedure Parameters

Procedure	Description
`instance_table`	Contains a list of the active instance numbers and names. When no instance is up, the list is empty.
`instance_count`	Number of active instances.

ANALYZE_DATABASE Procedure

> **Note:** This subprogram is obsolete with release Oracle Database Release 10*g*. It is retained in documentation for reasons of backward compatibility. For current functionality, see Chapter 103, "DBMS_STATS" on page 103-1.

This procedure runs the ANALYZE command on all the tables, clusters, and indexes in a database. Use this procedure to collect nonoptimizer statistics. For optimizer statistics, use the DBMS_STATS.GATHER_DATABASE_STATS procedure.

Syntax

```
DBMS_UTILITY.ANALYZE_DATABASE (
   method           VARCHAR2,
   estimate_rows    NUMBER   DEFAULT NULL,
   estimate_percent NUMBER   DEFAULT NULL,
   method_opt       VARCHAR2 DEFAULT NULL);
```

Parameters

Table 116–5 ANALYZE_DATABASE Procedure Parameters

Parameter	Description
method	One of ESTIMATE, COMPUTE or DELETE. If ESTIMATE, then either estimate_rows or estimate_percent must be nonzero.
estimate_rows	Number of rows to estimate.
estimate_percent	Percentage of rows to estimate. If estimate_rows is specified, then ignore this parameter.
method_opt	Method options of the following format: [FOR TABLE] [FOR ALL [INDEXED] COLUMNS] [SIZE n] [FOR ALL INDEXES]

Exceptions

Table 116–6 ANALYZE_DATABASE Procedure Exceptions

Exception	Description
ORA-20000	Insufficient privileges for some object in this database.

Usage Notes

Use this procedure to collect nonoptimizer statistics. For optimizer statistics, use the DBMS_STATS.GATHER_TABLE_STATS or DBMS_STATS.GATHER_INDEX_STATS procedure.

ANALYZE_PART_OBJECT Procedure

This procedure is equivalent to SQL:

```
"ANALYZE TABLE|INDEX [<schema>.]<object_name> PARTITION <pname> [<command_type>]
[<command_opt>] [<sample_clause>]
```

Syntax

```
DBMS_UTILITY.ANALYZE_PART_OBJECT (
   schema        IN VARCHAR2 DEFAULT NULL,
   object_name   IN VARCHAR2 DEFAULT NULL,
   object_type   IN CHAR     DEFAULT 'T',
   command_type  IN CHAR     DEFAULT 'E',
   command_opt   IN VARCHAR2 DEFAULT NULL,
   sample_clause IN VARCHAR2 DEFAULT 'sample 5 percent ');
```

Parameters

Table 116–7 ANALYZE_PART_OBJECT Procedure Parameters

Parameter	Description
`schema`	Schema of the `object_name`.
`object_name`	Name of object to be analyzed, must be partitioned.
`object_type`	Type of object, must be `T` (table) or `I` (index).
`command_type`	Must be `V` (validate structure)
`command_opt`	Other options for the command type.
	For `C`, `E` it can be `FOR` table, `FOR` all `LOCAL` indexes, `FOR` all columns or combination of some of the 'for' options of analyze statistics (table). For `V`, it can be `CASCADE` when `object_type` is T.
`sample_clause`	The sample clause to use when `command_type` is '`E`'.

Usage Notes

For each partition of the object, run in parallel using job queues.

ANALYZE_SCHEMA Procedure

> **Note:** This subprogram is obsolete with Oracle Database Release 10*g*. It is retained in documentation for reasons of backward compatibility. For current functionality, see Chapter 103, "DBMS_STATS" on page 103-1.

This procedure runs the `ANALYZE` command on all the tables, clusters, and indexes in a schema. Use this procedure to collect nonoptimizer statistics. For optimizer statistics, use the `DBMS_STATS.GATHER_SCHEMA_STATS` procedure.

Syntax

```
DBMS_UTILITY.ANALYZE_SCHEMA (
   schema           VARCHAR2,
   method           VARCHAR2,
   estimate_rows    NUMBER   DEFAULT NULL,
   estimate_percent NUMBER   DEFAULT NULL,
   method_opt       VARCHAR2 DEFAULT NULL);
```

Parameters

Table 116–8 ANALYZE_SCHEMA Procedure Parameters

Parameter	Description
`schema`	Name of the schema.
`method`	One of `ESTIMATE`, `COMPUTE` or `DELETE`. If `ESTIMATE`, then either `estimate_rows` or `estimate_percent` must be nonzero.
`estimate_rows`	Number of rows to estimate.
`estimate_percent`	Percentage of rows to estimate. If `estimate_rows` is specified, then ignore this parameter.

`method_opt`	Method options of the following format:
	`[ FOR TABLE ]`
	`[ FOR ALL [INDEXED] COLUMNS] [SIZE n]`
	`[ FOR ALL INDEXES ]`

Exceptions

Table 116–9 ANALYZE_SCHEMA Procedure Exceptions

Exception	Description
`ORA-20000`	Insufficient privileges for some object in this schema.

CANONICALIZE Procedure

This procedure canonicalizes the given string. The procedure handles a single reserved or key word (such as 'table'), and strips off white spaces for a single identifier so that ' table ' becomes TABLE.

Syntax

```
DBMS_UTILITY.CANONICALIZE(
   name           IN    VARCHAR2,
   canon_name     OUT   VARCHAR2,
   canon_len      IN    BINARY_INTEGER);
```

Parameters

Table 116–10 CANONICALIZE Procedure Parameters

Parameter	Description
name	The string to be canonicalized
canon_name	The canonicalized string
canon_len	The length of the string (in bytes) to canonicalize

Return Values

Returns the first canon_len bytes in canon_name.

Usage Notes

- If name is NULL, canon_name becomes NULL.
- If name is not a dotted name, and if name begins and ends with a double quote, remove both quotes. Alternatively, convert to upper case with NLS_UPPER. Note that this case does not include a name with special characters, such as a space, but is not doubly quoted.
- If name is a dotted name (such as a."b".c), for each component in the dotted name in the case in which the component begins and ends with a double quote, no transformation will be performed on this component. Alternatively, convert to upper case with NLS_UPPER and apply begin and end double quotes to the capitalized form of this component. In such a case, each canonicalized component will be concatenated together in the input position, separated by ".".
- Any other character after a[.b]* will be ignored.
- The procedure does not handle cases like 'A B.'

Examples

- a becomes A
- "a" becomes a
- "a".b becomes "a"."B"
- "a".b,c.f becomes "a"."B" with",c.f" ignored.

COMMA_TO_TABLE Procedures

These procedures converts a comma-delimited list of names into a PL/SQL table of names. The second version supports fully-qualified attribute names.

Syntax

```
DBMS_UTILITY.COMMA_TO_TABLE (
   list   IN  VARCHAR2,
   tablen OUT BINARY_INTEGER,
   tab    OUT uncl_array);

DBMS_UTILITY.COMMA_TO_TABLE (
   list   IN  VARCHAR2,
   tablen OUT BINARY_INTEGER,
   tab    OUT lname_array);
```

Parameters

Table 116–11 COMMA_TO_TABLE Procedure Parameters

Parameter	Description
`list`	Comma separated list of tables.
`tablen`	Number of tables in the PL/SQL table.
`tab`	PL/SQL table which contains list of table names.

Return Values

A PL/SQL table is returned, with values `1..n` and `n+1 is null`.

Usage Notes

The `list` must be a non-empty comma-delimited list: Anything other than a comma-delimited list is rejected. Commas inside double quotes do not count.

Entries in the comma-delimited list cannot include multibyte characters such as hyphens (-).

The values in `tab` are cut from the original list, with no transformations.

COMPILE_SCHEMA Procedure

This procedure compiles all procedures, functions, packages, and triggers in the specified schema.

Syntax

```
DBMS_UTILITY.COMPILE_SCHEMA (
   schema          VARCHAR2,
   compile_all     BOOLEAN DEFAULT TRUE,
   reuse_settings  BOOLEAN DEFAULT FALSE);
```

Parameters

Table 116–12 COMPILE_SCHEMA Procedure Parameters

Parameter	Description
`schema`	Name of the schema
`compile_all`	If `TRUE`, will compile everything within the schema regardless of whether it is VALID If FALSE, will compile only INVALID objects
`reuse_settings`	Indicates whether the session settings in the objects should be reused, or whether the current session settings should be adopted instead

Exceptions

Table 116–13 COMPILE_SCHEMA Procedure Exceptions

Exception	Description
`ORA-20000`	Insufficient privileges for some object in this schema
`ORA-20001`	Cannot recompile `SYS` objects
`ORA-20002`	Maximum iterations exceeded. Some objects may not have been recompiled.

Usage Notes

After calling this procedure, you should select from view `ALL_OBJECTS` for items with status of `INVALID` to see if all objects were successfully compiled.

To see the errors associated with `INVALID` objects, you may use the Enterprise Manager command:

```
SHOW ERRORS <type> <schema>.<name>
```

CREATE_ALTER_TYPE_ERROR_TABLE Procedure

This procedure creates an error table to be used in the `EXCEPTION` clause of the `ALTER TYPE` statement.

Syntax

```
DBMS_UTILITY.CREATE_ALTER_TYPE_ERROR_TABLE(
   schema_name      IN     VARCHAR2,
   table_name       IN     VARCHAR2);
```

Parameters

Table 116–14 CREATE_ALTER_TYPE_ERROR_TABLE Procedure Parameters

Parameter	Description
`schema_name`	The name of the schema.
`table_name`	The name of the table created.

Exceptions

An error is returned if the table already exists.

CURRENT_INSTANCE Function

This function returns the current connected instance number. It returns `NULL` when connected instance is down.

Syntax

```
DBMS_UTILITY.CURRENT_INSTANCE
   RETURN NUMBER;
```

DATA_BLOCK_ADDRESS_BLOCK Function

This function gets the block number part of a data block address.

Syntax

```
DBMS_UTILITY.DATA_BLOCK_ADDRESS_BLOCK (
   dba NUMBER)
  RETURN NUMBER;
```

Parameters

Table 116–15 DATA_BLOCK_ADDRESS_BLOCK Function Parameters

Parameter	Description
dba	Data block address.

Pragmas

```
pragma restrict_references(data_block_address_block, WNDS, RNDS, WNPS, RNPS);
```

Return Values

Block offset of the block.

Usage Notes

This function should not be used with datablocks which belong to bigfile tablespaces.

DATA_BLOCK_ADDRESS_FILE Function

This function gets the file number part of a data block address.

Syntax

```
DBMS_UTILITY.DATA_BLOCK_ADDRESS_FILE (
   dba NUMBER)
  RETURN NUMBER;
```

Parameters

Table 116–16 DATA_BLOCK_ADDRESS_FILE Function Parameters

Parameter	Description
dba	Data block address.

Pragmas

```
pragma restrict_references (data_block_address_file, WNDS, RNDS, WNPS, RNPS);
```

Return Values

File that contains the block.

Usage Notes

This function should not be used with datablocks which belong to bigfile tablespaces.

DB_VERSION Procedure

This procedure returns version information for the database.

Syntax

```
DBMS_UTILITY.DB_VERSION (
   version       OUT VARCHAR2,
   compatibility OUT VARCHAR2);
```

Parameters

Table 116–17 DB_VERSION Procedure Parameters

Parameter	Description
`version`	A string which represents the internal software version of the database (for example, 7.1.0.0.0). The length of this string is variable and is determined by the database version.
`compatibility`	The compatibility setting of the database determined by the "compatible" `init.ora` parameter. If the parameter is not specified in the `init.ora` file, then `NULL` is returned.

EXEC_DDL_STATEMENT Procedure

This procedure executes the DDL statement in `parse_string`.

Syntax

```
DBMS_UTILITY.EXEC_DDL_STATEMENT (
   parse_string IN VARCHAR2);
```

Parameters

Table 116–18 EXEC_DDL_STATEMENT Procedure Parameters

Parameter	Description
`parse_string`	DDL statement to be executed.

FORMAT_CALL_STACK Function

This function formats the current call stack. This can be used on any stored procedure or trigger to access the call stack. This can be useful for debugging.

Syntax

```
DBMS_UTILITY.FORMAT_CALL_STACK
  RETURN VARCHAR2;
```

Pragmas

```
pragma restrict_references(format_call_stack,WNDS);
```

Return Values

This returns the call stack, up to 2000 bytes.

FORMAT_ERROR_BACKTRACE Function

This procedure displays the call stack at the point where an exception was raised, even if the procedure is called from an exception handler in an outer scope. The output is similar to the output of the SQLERRM function, but not subject to the same size limitation.

Syntax

```
DBMS_UTILITY.FORMAT_ERROR_BACKTRACE
  RETURN VARCHAR2;
```

Return Values

The backtrace string. A NULL string is returned if no error is currently being handled.

Examples

```
CREATE OR REPLACE PROCEDURE Log_Errors ( i_buff in varchar2 ) IS
  g_start_pos integer := 1;
  g_end_pos  integer;

  FUNCTION Output_One_Line RETURN BOOLEAN IS
  BEGIN
    g_end_pos := Instr ( i_buff, Chr(10), g_start_pos );

    CASE g_end_pos > 0
      WHEN true THEN
        DBMS_OUTPUT.PUT_LINE ( Substr ( i_buff, g_start_pos,
g_end_pos-g_start_pos ) );
        g_start_pos := g_end_pos+1;
        RETURN TRUE;

      WHEN FALSE THEN
        DBMS_OUTPUT.PUT_LINE ( Substr ( i_buff, g_start_pos,
(Length(i_buff)-g_start_pos)+1 ) );
        RETURN FALSE;
    END CASE;
  END Output_One_Line;

BEGIN
  WHILE Output_One_Line() LOOP NULL;
  END LOOP;
END Log_Errors;
/

Set Doc Off
Set Feedback off
Set Echo Off

CREATE OR REPLACE PROCEDURE P0 IS
  e_01476 EXCEPTION; pragma exception_init ( e_01476, -1476 );
BEGIN
  RAISE e_01476;
END P0;
/
Show Errors

CREATE OR REPLACE PROCEDURE P1 IS
```

```
BEGIN
  P0();
END P1;
/
SHOW ERRORS

CREATE OR REPLACE PROCEDURE P2 IS
BEGIN
  P1();
END P2;
/
SHOW ERRORS

CREATE OR REPLACE PROCEDURE P3 IS
BEGIN
  P2();
END P3;
/
SHOW ERRORS

CREATE OR REPLACE PROCEDURE P4 IS
  BEGIN P3(); END P4;
/
CREATE OR REPLACE PROCEDURE P5 IS
  BEGIN P4(); END P5;
/
SHOW ERRORS

CREATE OR REPLACE PROCEDURE Top_Naive IS
BEGIN
  P5();
END Top_Naive;
/
SHOW ERRORS

CREATE OR REPLACE PROCEDURE Top_With_Logging IS
  -- NOTE: SqlErrm in principle gives the same info as Format_Error_Stack.
  -- But SqlErrm is subject to some length limits,
  -- while Format_Error_Stack is not.
BEGIN
  P5();
EXCEPTION
  WHEN OTHERS THEN
    Log_Errors ( 'Error_Stack...' || Chr(10) ||
      DBMS_UTILITY.FORMAT_ERROR_STACK() );
    Log_Errors ( 'Error_Backtrace...' || Chr(10) ||
      DBMS_UTILITY.FORMAT_ERROR_BACKTRACE() );
    DBMS_OUTPUT.PUT_LINE ( '----------' );
END Top_With_Logging;
/
SHOW ERRORS

--------------------------------------------------------------------------------

Set ServerOutput On
call Top_Naive()
  /*
  ERROR at line 1:
  ORA-01476: divisor is equal to zero
  ORA-06512: at "U.P0", line 4
```

```
  ORA-06512: at "U.P1", line 3
  ORA-06512: at "U.P2", line 3
  ORA-06512: at "U.P3", line 3
  ORA-06512: at "U.P4", line 2
  ORA-06512: at "U.P5", line 2
  ORA-06512: at "U.TOP_NAIVE", line 3
  */
  ;

Set ServerOutput On
call Top_With_Logging()
  /*
  Error_Stack...
  ORA-01476: divisor is equal to zero
  Error_Backtrace...
  ORA-06512: at "U.P0", line 4
  ORA-06512: at "U.P1", line 3
  ORA-06512: at "U.P2", line 3
  ORA-06512: at "U.P3", line 3
  ORA-06512: at "U.P4", line 2
  ORA-06512: at "U.P5", line 2
  ORA-06512: at "U.TOP_WITH_LOGGING", line 6
  ----------
  */
  ;

/*
  ORA-06512:
  Cause:
    Backtrace message as the stack is
    unwound by unhandled exceptions.
  Action:
    Fix the problem causing the exception
    or write an exception handler for this condition.
    Or you may need to contact your application administrator
    or database administrator.
*/
```

FORMAT_ERROR_STACK Function

This function formats the current error stack. This can be used in exception handlers to look at the full error stack.

Syntax

```
DBMS_UTILITY.FORMAT_ERROR_STACK
  RETURN VARCHAR2;
```

Return Values

This returns the error stack, up to 2000 bytes.

Return Values

See FORMAT_ERROR_BACKTRACE Function on page 116-24.

GET_CPU_TIME Function

This function returns the current CPU time in 100th's of a second. The returned CPU time is the number of 100th's of a second from some arbitrary epoch.

Syntax

```
DBMS_UTILITY.GET_CPU_TIME
 RETURN NUMBER;
```

Return Values

Time is the number of 100th's of a second from some arbitrary epoch.

GET_DEPENDENCY Procedure

This procedure shows the dependencies on the object passed in.

Syntax

```
DBMS_UTILITY.GET_DEPENDENCY
 type      IN     VARCHAR2,
 schema    IN     VARCHAR2,
 name      IN     VARCHAR2);
```

Parameters

Table 116–19 GET_DEPENDENCY Procedure Parameters

Parameter	Description
type	The type of the object, for example if the object is a table give the type as 'TABLE'.
schema	The schema name of the object.
name	The name of the object.

Usage Notes

This procedure uses the DBMS_OUTPUTpackage to display results, and so you must declare SET SERVEROUTPUT ON if you wish to view dependencies. Alternatively, any application that checks the DBMS_OUTPUT output buffers can invoke this subprogram and then retrieve the output by means of DBMS_OUTPUT subprograms such as GET_LINES.

GET_HASH_VALUE Function

This function computes a hash value for the given string.

Syntax

```
DBMS_UTILITY.GET_HASH_VALUE (
   name      VARCHAR2,
   base      NUMBER,
   hash_size NUMBER)
  RETURN NUMBER;
```

Parameters

Table 116–20 GET_HASH_VALUE Function Parameters

Parameter	Description
name	String to be hashed.
base	Base value for the returned hash value to start at.
hash_size	Desired size of the hash table.

Pragmas

```
pragma restrict_references(get_hash_value, WNDS, RNDS, WNPS, RNPS);
```

Return Values

A hash value based on the input string. For example, to get a hash value on a string where the hash value should be between 1000 and 3047, use 1000 as the base value and 2048 as the hash_size value. Using a power of 2 for the hash_size parameter works best.

GET_PARAMETER_VALUE Function

This function gets the value of specified `init.ora` parameter.

Syntax

```
DBMS_UTILITY.GET_PARAMETER_VALUE (
   parnam     IN        VARCHAR2,
   intval     IN OUT    BINARY_INTEGER,
   strval     IN OUT    VARCHAR2)
  RETURN BINARY_INTEGER;
```

Parameters

Table 116–21 GET_PARAMETER_VALUE Function Parameters

Parameter	Description
`parnam`	Parameter name.
`intval`	Value of an integer parameter or the value length of a string parameter.
`strval`	Value of a string parameter.

Return Values

Parameter type:

- 0 if parameter is an `INTEGER/BOOLEAN` parameter
- 1 if parameter is a string/file parameter

Usage Notes

When using `DBMS_UTILITY.GET_PARAMETER_VALUE`, only the first parameter setting of `/dir1` is returned when `init.ora` is set as follows:

```
utl_file_dir = /dir1
utl_file_dir = /dir2
```

However, the full comma-delimited string is returned if you are using:

```
utl_file_dir = /dir1, /dir2
```

Examples

```
DECLARE
  parnam VARCHAR2(256);
  intval BINARY_INTEGER;
  strval VARCHAR2(256);
  partyp BINARY_INTEGER;
BEGIN
  partyp := dbms_utility.get_parameter_value('max_dump_file_size',
                                             intval, strval);
  dbms_output.put('parameter value is: ');
  IF partyp = 1 THEN
    dbms_output.put_line(strval);
  ELSE
    dbms_output.put_line(intval);
  END IF;
  IF partyp = 1 THEN
```

```
    dbms_output.put('parameter value length is: ');
    dbms_output.put_line(intval);
  END IF;
  dbms_output.put('parameter type is: ');
  IF partyp = 1 THEN
    dbms_output.put_line('string');
  ELSE
    dbms_output.put_line('integer');
  END IF;
END;
```

GET_TIME Function

This function determines the current time in 100th's of a second. This subprogram is primarily used for determining elapsed time. The subprogram is called twice – at the beginning and end of some process – and then the first (earlier) number is subtracted from the second (later) number to determine the time elapsed.

Syntax

```
DBMS_UTILITY.GET_TIME
  RETURN NUMBER;
```

Return Values

Time is the number of 100th's of a second from the point in time at which the subprogram is invoked.

Usage Notes

Numbers are returned in the range -2147483648 to 2147483647 depending on platform and machine, and your application must take the sign of the number into account in determining the interval. For instance, in the case of two negative numbers, application logic must allow that the first (earlier) number will be larger than the second (later) number which is closer to zero. By the same token, your application should also allow that the first (earlier) number be negative and the second (later) number be positive.

INVALIDATE Procedure

This procedure invalidates a database object and (optionally) modifies its PL/SQL compiler parameter settings. It also invalidates any objects that (directly or indirectly) depend on the object being invalidated.

Syntax

```
DBMS_UTILITY.INVALIDATE (
   p_object_id              NUMBER,
   p_plsql_object_settings  VARCHAR2 DEFAULT NULL,
   p_option_flags           PLS_INTEGER DEFAULT 0);
```

Parameters

Table 116–22 INVALIDATE Procedure Parameters

Parameter	Description
p_object_id	ID number of object to be invalidated. This is the same as the value of the OBJECT_ID column from ALL_OBJECTS. If the object_id argument is NULL or invalid then the exception inv_not_exist_or_no_priv is raised. The caller of this procedure must have create privileges on the object being invalidated else the inv_not_exist_or_no_priv exception is raised.
p_plsql_object_ settings	This optional parameter is ignored if the object specified by p_ object_id is not a PL/SQL object. If no value is specified for this parameter then the PL/SQL compiler settings are left unchanged, that is, equivalent to REUSE SETTINGS. If a value is provided, it must specify the values of the PL/SQL compiler settings separated by one or more spaces. Each setting can be specified only once else inv_malformed_settings exception will be raised. The setting values are changed only for the object specified by p_object_id and do not affect dependent objects that may be invalidated. The setting names and values are case insensitive. If a setting is omitted and REUSE SETTINGS is specified, then if a value was specified for the compiler setting in an earlier compilation of this library unit, Oracle Database uses that earlier value. If a setting is omitted and REUSE SETTINGS was not specified or no value has been specified for the parameter in an earlier compilation, then the database will obtain the value for that setting from the session environment.
p_option_flags	This parameter is optional and defaults to zero (no flags). Option flags supported by invalidate. ▪ inv_error_on_restrictions (see Constants on page 116-4): The subprogram imposes various restrictions on the objects that can be invalidated. For example, the object specified by p_object_id cannot be a table. By default, invalidate quietly returns on these conditions (and does not raise an exception). If the caller sets this flag, the exception inv_restricted_object is raised.

Exceptions

Table 116–23 INVALIDATE Exceptions

Exception	Description
INV_NOT_EXIST_OR_NO_PRIV	Raised when the object_id argument is NULL or invalid, or when the caller does not have CREATE privileges on the object being invalidated
INV_MALFORMED_SETTINGS	Raised if a compiler setting is specified more than once in the p_plsql_object_settings parameter
INV_RESTRICTED_OBJECT	Raised when different combinations of conditions pertaining to the p_object_id parameter are contravened

Usage Notes

The object type (object_type column from ALL_OBJECTS) of the object specified by p_object_id must be a PROCEDURE, FUNCTION, PACKAGE, PACKAGE BODY, TRIGGER, TYPE, TYPE BODY, LIBRARY, VIEW, OPERATOR, SYNONYM, or JAVA CLASS. If the object is not one of these types and the flag inv_error_on_restrictions is specified in p_option_flags then the exception inv_restricted_object is raised, else no action is taken.

If the object specified by p_object_id is the package specification of STANDARD, DBMS_STANDARD, or specification or body of DBMS_UTILITY and the flag inv_error_on_restrictions is specified in p_option_flags then the exception inv_restricted_object is raised, else no action is taken.

If the object specified by p_object_id is an object type specification and there exist tables which depend on the type and the flag inv_error_on_restrictions is specified in p_option_flags then the exception inv_restricted_object is raised, else no action is taken.

Examples

Example 1

```
DBMS_UTILITY.INVALIDATE (1232, 'PLSQL_OPTIMIZE_LEVEL = 2 REUSE SETTINGS');
```

Assume that the object_id 1232 refers to the procedure remove_emp in the HR schema. Then the above call will mark the remove_emp procedure invalid and change it's PLSQL_OPTIMIZE_LEVEL compiler setting to 2. The values of other compiler settings will remain unchanged since REUSE SETTINGS is specified.

Objects that depend on hr.remove_emp will also get marked invalid. Their compiler parameters will not be changed.

Example 2

```
DBMS_UTILITY.INVALIDATE (40775, 'plsql_code_type = native');
```

Assume that the object_id 40775 refers to the type body leaf_category_typ in the OE schema. Then the above call will mark the type body invalid and change its PLSQL_CODE_TYPE compiler setting to NATIVE. The values of other compiler settings will be picked up from the current session environment since REUSE SETTINGS has not been specified.

Since no objects can depend on bodies, there are no cascaded invalidations.

Example 3

```
DBMS_UTILITY.INVALIDATE (40796);
```

Assume that the `object_id` 40796 refers to the view `oc_orders` in the `OE` schema. Then the above call will mark the `oc_orders` view invalid.

Objects that depend on `oe.oc_orders` will also get marked invalid.

IS_CLUSTER_DATABASE Function

This function finds out if this database is running in cluster database mode.

Syntax

```
DBMS_UTILITY.IS_CLUSTER_DATABASE
  RETURN BOOLEAN;
```

Return Values

This function returns `TRUE` if this instance was started in cluster database mode; `FALSE` otherwise.

MAKE_DATA_BLOCK_ADDRESS Function

This function creates a data block address given a file number and a block number. A data block address is the internal structure used to identify a block in the database. This function is useful when accessing certain fixed tables that contain data block addresses.

Syntax

```
DBMS_UTILITY.MAKE_DATA_BLOCK_ADDRESS (
   file  NUMBER,
   block NUMBER)
  RETURN NUMBER;
```

Parameters

Table 116–24 MAKE_DATA_BLOCK_ADDRESS Function Parameters

Parameter	Description
`file`	File that contains the block.
`block`	Offset of the block within the file in terms of block increments.

Pragmas

```
pragma restrict_references (make_data_block_address, WNDS, RNDS, WNPS, RNPS);
```

Return Values

Data block address.

NAME_RESOLVE Procedure

This procedure resolves the given name, including synonym translation and authorization checking as necessary.

Syntax

```
DBMS_UTILITY.NAME_RESOLVE (
   name          IN  VARCHAR2,
   context       IN  NUMBER,
   schema        OUT VARCHAR2,
   part1         OUT VARCHAR2,
   part2         OUT VARCHAR2,
   dblink        OUT VARCHAR2,
   part1_type    OUT NUMBER,
   object_number OUT NUMBER);
```

Parameters

Table 116–25 NAME_RESOLVE Procedure Parameters

Parameter	Description
name	Name of the object. This can be of the form [[a.]b.]c[@d], where a, b, c are SQL identifier and d is a dblink. No syntax checking is performed on the dblink. If a dblink is specified, or if the name resolves to something with a dblink, then object is not resolved, but the schema, part1, part2 and dblink OUT parameters are filled in. a, b and c may be delimited identifiers, and may contain Globalization Support (NLS) characters (single and multibyte).
context	Must be an integer between 0 and 8.
schema	Schema of the object: c. If no schema is specified in name, then the schema is determined by resolving the name.
part1	First part of the name. The type of this name is specified part1_type (synonym or package).
part2	If this is non-NULL, then this is a subprogram name. If part1 is non-NULL, then the subprogram is within the package indicated by part1. If part1 is NULL, then the subprogram is a top-level subprogram.
dblink	If this is non-NULL, then a database link was either specified as part of name or name was a synonym which resolved to something with a database link. In this case, if further name translation is desired, then you must call the DBMS_UTILITY.NAME_RESOLVE procedure on this remote node.
part1_type	Type of part1 is: ■ 5 - synonym ■ 7 - procedure (top level) ■ 8 - function (top level) ■ 9 - package
object_number	Object identifier

Exceptions

All errors are handled by raising exceptions. A wide variety of exceptions are possible, based on the various syntax error that are possible when specifying object names.

NAME_TOKENIZE Procedure

This procedure calls the parser to parse the given name as "a [. b [. c]][@ dblink]". It strips double quotes, or converts to uppercase if there are no quotes. It ignores comments of all sorts, and does no semantic analysis. Missing values are left as NULL.

Syntax

```
DBMS_UTILITY.NAME_TOKENIZE (
   name    IN  VARCHAR2,
   a       OUT VARCHAR2,
   b       OUT VARCHAR2,
   c       OUT VARCHAR2,
   dblink  OUT VARCHAR2,
   nextpos OUT BINARY_INTEGER);
```

Parameters

For each of a, b, c, dblink, tell where the following token starts in anext, bnext, cnext, dnext respectively.

PORT_STRING Function

This function returns a string that identifies the operating system and the `TWO TASK PROTOCOL` version of the database. For example, "`VAX/VMX-7.1.0.0`"

The maximum length is port-specific.

Syntax

```
DBMS_UTILITY.PORT_STRING
   RETURN VARCHAR2;
```

Pragmas

```
pragma restrict_references(port_string, WNDS, RNDS, WNPS, RNPS);
```

TABLE_TO_COMMA Procedures

These procedures converts a PL/SQL table of names into a comma-delimited list of names. This takes a PL/SQL table, `1..n`, terminated with `n+1 null`. The second version supports fully-qualified attribute names.

Syntax

```
DBMS_UTILITY.TABLE_TO_COMMA (
   tab    IN  UNCL_ARRAY,
   tablen OUT BINARY_INTEGER,
   list   OUT VARCHAR2);

DBMS_UTILITY.TABLE_TO_COMMA (
   tab    IN  lname_array,
   tablen OUT BINARY_INTEGER,
   list   OUT VARCHAR2);
```

Parameters

Table 116–26 TABLE_TO_COMMA Procedure Parameters

Parameter	Description
`tab`	PL/SQL table which contains list of table names.
`tablen`	Number of tables in the PL/SQL table.
`list`	Comma separated list of tables.

Return Values

A comma-delimited list and the number of elements found in the table.

VALIDATE Procedure

This procedure makes invalid database objects valid.

Syntax

```
DBMS_UTILITY.VALIDATE(
  object_id       NUMBER);
```

Parameters

Table 116–27 VALIDATE Procedure Parameters

Parameter	Description
`object_id`	The ID number of object to be validated. This is the same as the value of the `OBJECT_ID` column from `ALL_OBJECTS`.

Usage Notes

No errors are raised if the object does not exist or is already valid or is an object that cannot be validated.

117

DBMS_WARNING

The DBMS_WARNING package provides a way to manipulate the behavior of PL/SQL warning messages, in particular by reading and changing the setting of the PLSQL_WARNINGS initialization parameter to control what kinds of warnings are suppressed, displayed, or treated as errors. This package provides the interface to query, modify and delete current system or session settings.

This chapter contains the following topics:

Using DBMS_WARNING

- Security Model

Security Model

Note that for all the following interfaces, if value of the scope parameter is `SYSTEM`, then the user must have `ALTER SYSTEM` privilege.

Summary of DBMS_WARNING Subprograms

Table 117–1 DBMS_WARNING Package Subprograms

Subprogram	Description
ADD_WARNING_SETTING_CAT Procedure on page 117-5	Modifies the current `session` or `system` warning settings of the `warning_category` previously supplied
ADD_WARNING_SETTING_NUM Procedure on page 117-6	Modifies the current `session` or `system` warning settings of the or `warning_number` previously supplied
GET_CATEGORY Function on page 117-7	Returns the category name, given the message number
GET_WARNING_SETTING_CAT Function on page 117-8	Returns the specific warning category in the session
GET_WARNING_SETTING_NUM Function on page 117-9	Returns the specific warning number in the session
GET_WARNING_SETTING_STRING Function on page 117-10	Returns the entire warning string for the current session
SET_WARNING_SETTING_STRING Procedure on page 117-11	Replaces previous settings with the new value

ADD_WARNING_SETTING_CAT Procedure

You can modify the current session's or system's warning settings with the value supplied. The value will be added to the existing parameter setting if the value for the `warning_category` or `warning_value` has not been set, or override the existing value. The effect of calling this function is same as adding the qualifier (`ENABLE/DISABLE/ERROR`) on the category specified to the end of the current session or system setting.

Syntax

```
DBMS_WARNING.ADD_WARNING_SETTING_CAT (
   warning_category    IN    VARCHAR2,
   warning_value       IN    VARCHAR2,
   scope               IN    VARCAHR2);
```

Parameters

Table 117–2 ADD_WARNING_SETTING_CAT Procedure Parameters

Parameter	Description
`warning_category`	Name of the category. Allowed values are `ALL`, `INFORMATIONAL`, `SEVERE` and `PERFORMANCE`.
`warning_value`	Value for the category. Allowed values are `ENABLE`, `DISABLE`, and `ERROR`.
`scope`	Specifies if the changes are being performed in the session context or the system context. Allowed values are `SESSION` or `SYSTEM`.

ADD_WARNING_SETTING_NUM Procedure

You can modify the current `session` or `system` warning settings with the value supplied. If the value was already set, you will override the existing value. The effect of calling this function is same as adding the qualifier (`ENABLE` / `DISABLE`/ `ERROR`) on the category specified to the end of the current session or system setting.

Syntax

```
DBMS_WARNING.ADD_WARNING_SETTING_NUM (
   warning_number      IN    NUMBER,
   warning_value       IN    VARCHAR2,
   scope               IN    VARCAHR2);
```

Parameters

Table 117–3 ADD_WARNING_SETTING_NUM Procedure Parameters

Parameter	Description
`warning_number`	The warning number. Allowed values are all valid warning numbers.
`warning_value`	Value for the category. Allowed values are `ENABLE`, `DISABLE`, and `ERROR`.
`scope`	Specifies if the changes are being performed in the session context or the system context. Allowed values are `SESSION` or `SYSTEM`.

GET_CATEGORY Function

This function returns the category name, given the message number.

Syntax

```
DBMS_WARNING.GET_CATEGORY (
   warning_number  IN   pls_integer)
RETURN VARCHAR2;
```

Parameters

Table 117–4 GET_CATEGORY Function Parameters

Parameter	Description
warning_number	The warning message number.

GET_WARNING_SETTING_CAT Function

This function returns the specific warning category setting for the current session.

Syntax

```
DBMS_WARNING.GET_WARNING_SETTING_CAT (
   warning_category    IN    VARCHAR2)
RETURN warning_value;
```

Parameters

Table 117–5 GET_WARNING_SETTING_CAT Function Parameters

Parameter	Description
warning_category	Name of the category. Allowed values are all valid category names (ALL, INFORMATIONAL, SEVERE and PERFORMANCE).

GET_WARNING_SETTING_NUM Function

This function returns the specific warning number setting for the current session.

Syntax

```
DBMS_WARNING.GET_WARNING_SETTING_NUM (
   warning_number      IN    NUMBER)
RETURN warning_value;
```

Parameters

Table 117–6 GET_WARNING_SETTING_NUM Function Parameters

Parameter	Description
warning_number	Warning number. Allowed values are all valid warning numbers.

GET_WARNING_SETTING_STRING Function

This function returns the entire warning string for the current session.

Syntax

```
DBMS_WARNING.GET_WARNING_SETTING_STRING
 RETURN pls_integer;
```

Usage Notes

Use this function when you do not have `SELECT` privilege on `v$parameter` or `v$paramater2` fixed tables, or if you want to parse the warning string yourself and then modify and set the new value using `SET_WARNING_SETTING_STRING`.

SET_WARNING_SETTING_STRING Procedure

This procedureS replaces previous settings with the new value. The warning string may contain mix of category and warning numbers using the same syntax as used on the right hand side of '=' when issuing an `ALTER SESSION` or `SYSTEM SET PLSQL_WARNINGS` command. This will have same effect as `ALTER SESSION` OR `ALTER SYSTEM` command.

Syntax

```
DBMS_WARNING.SET_WARNING_SETTING_STRING (
   warning_value   IN   VARCHAR2,
   scope           IN   VARCHAR2);
```

Parameters

Table 117–7 SET_WARNING_SETTING_STRING Procedure Parameters

Parameter	Description
`warning_value`	The new string that will constitute the new value.
`scope`	This will specify if the changes are being done in the session context, or system context. Allowed values are `SESSION` or `SYSTEM`.

118

DBMS_WORKLOAD_REPOSITORY

The `DBMS_WORKLOAD_REPOSITORY` package lets you manage the Workload Repository, performing operations such as managing snapshots and baselines.

The chapter contains the following topic:

Using DBMS_WORKLOAD_REPOSITORY

This section contains topics which relate to using the DBMS_WORKLOAD_REPOSITORY package.

Examples

This example shows how to generate an AWR text report with the DBMS_WORKLOAD_REPOSITORY package for database identifier 1557521192, instance id 1, snapshot ids 5390 and 5391 and with default options.

```
-- make sure to set line size appropriately
-- set linesize 152
SELECT output FROM TABLE(
   DBMS_WORKLOAD_REPOSITORY.AWR_REPORT_TEXT(
     1557521192,  1, 5390, 5392) ) ;
```

You can call the DBMS_WORKLOAD_REPOSITORY packaged functions directly as in the example, but Oracle recommends you use the corresponding supplied SQL script (awrrpt.sql in this case) for the packaged function, which prompts the user for required information.

Summary of DBMS_WORKLOAD_REPOSITORY Subprograms

Table 118–1 DBMS_WORKLOAD_REPOSITORY Package Subprograms

Subprogram	Description
ASH_REPORT_HTML Function on page 118-5	Displays the ASH report in HTML
ASH_REPORT_TEXT Function on page 118-7	Displays the ASH report in text
AWR_DIFF_REPORT_HTML Function on page 118-9	Displays the AWR Diff-Diff report in HTML
AWR_DIFF_REPORT_TEXT Function on page 118-10	Displays the AWR Diff-Diff report in text
AWR_REPORT_HTML Function on page 118-11	Displays the AWR report in HTML
AWR_REPORT_TEXT Function on page 118-12	Displays the AWR report in text
AWR_SQL_REPORT_HTML Function on page 118-13	Displays the AWR SQL Report in HTML format
AWR_SQL_REPORT_TEXT Function on page 118-14	Displays the AWR SQL Report in text format
CREATE_BASELINE Function and Procedure on page 118-15	Creates a single baseline
CREATE_SNAPSHOT Function and Procedure on page 118-16	Creates a manual snapshot immediately
DROP_BASELINE Procedure on page 118-17	Drops a range of snapshots
DROP_SNAPSHOT_RANGE Procedure on page 118-18	Activates service
MODIFY_SNAPSHOT_ SETTINGS Procedures on page 118-19	Modifies the snapshot settings.

ASH_REPORT_HTML Function

This table function displays the ASH Spot report in HTML.

Syntax

```
DBMS_WORKLOAD_REPOSITORY.ASH_REPORT_HTML(
   l_dbid          IN NUMBER,
   l_inst_num      IN NUMBER,
   l_btime         IN DATE,
   l_etime         IN DATE,
   l_options       IN NUMBER    DEFAULT 0,
   l_slot_width    IN NUMBER    DEFAULT 0,
   l_sid           IN NUMBER    DEFAULT NULL,
   l_sql_id        IN VARCHAR2  DEFAULT NULL,
   l_wait_class    IN VARCHAR2  DEFAULT NULL,
   l_service_hash  IN NUMBER    DEFAULT NULL,
   l_module        IN VARCHAR2  DEFAULT NULL,
   l_action        IN VARCHAR2  DEFAULT NULL,
   l_client_id     IN VARCHAR2  DEFAULT NULL)
 RETURN awrrpt_html_type_table PIPELINED;
```

Parameters

Table 118–2 ASH_REPORT_HTML Parameters

Parameter	Description
l_dbid	The database identifier
l_inst_num	The instance number
l_btime	The 'begin time'
l_etime	The 'end time'
l_options	Report level (currently not used)
l_slot_width	Specifies (in seconds) how wide the slots used in the "Top Activity" section of the report should be. This argument is optional, and if it is not specified the time interval between l_btime and l_etime is appropriately split into not more than 10 slots.
l_sid	The session ID (see Usage Notes)
l_sql_id	The SQL ID (see Usage Notes)
l_wait_class	The wait class name (see Usage Notes)
l_service_hash	The service name hash (see Usage Notes)
l_module	The module name (see Usage Notes)
l_action	The action name (see Usage Notes)
l_client_id	The client ID for end-to-end backtracing (see Usage Notes)

Return Values

The output will be one column of VARCHAR2(500).

Usage Notes

- You can call the function directly but Oracle recommends you use the `ashrpt.sql` script which prompts users for the required information.
- The unspecified optional arguments are used to generate an ASH Reports that specify 'report targets' such as a SQL statement, or a session, or a particular Service/Module combination. These arguments are specified to restrict the ASH rows that would be used to generate the report. For example, to generate an ASH report on a particular SQL statement, such as SQL_ID 'abcdefghij123 ' pass that `sql_id` value to the `l_sql_id` argument:

  ```
  l_sql_id =>  'abcdefghij123'
  ```

  Any combination of those optional arguments can be passed in, and only rows in ASH that satisfy all of those 'report targets' will be used. If multiple 'report targets' are specified, `AND` conditional logic is used to connect them. For example, to generate an ASH report on `MODULE` "PAYROLL" and `ACTION` "PROCESS", use the following predicate:

  ```
  l_module =>  'PAYROLL', l_action =>  'PROCESS'
  ```

  Valid SQL wildcards can be used in all the arguments that are of type `VARCHAR2`.

Table 118–3 ASH_REPORT_HTML: Wildcards Allowed (or Not) in Arguments

Argument Name	Comment	Wildcard Allowed
`l_sid`	The session ID (for example, `V$SESSION.SID`)	No
`l_sql_id`	The SQL ID (for example, `V$SQL.SQL_ID`)	Yes
`l_wait_class`	The wait class name (for example, `V$EVENT_NAME.WAIT_CLASS`)	Yes
`l_service_hash`	The service name hash (for example, `V$ACTIVE_SERVICES.NAME_HASH`)	No
`l_module`	The module name (for example, `V$SESSION.MODULE`)	Yes
`l_action`	The action name (for example, `V$SESSION.ACTION`)	Yes
`l_client_id`	The client ID for end-to-end backtracing (for example, `V$SESSION.CLIENT_IDENTIFIER`)	Yes

ASH_REPORT_TEXT Function

This table function displays the ASH Spot report in text.

Syntax

```
DBMS_WORKLOAD_REPOSITORY.ASH_REPORT_TEXT(
   l_dbid          IN NUMBER,
   l_inst_num      IN NUMBER,
   l_btime         IN DATE,
   l_etime         IN DATE,
   l_options       IN NUMBER    DEFAULT 0,
   l_slot_width    IN NUMBER    DEFAULT 0,
   l_sid           IN NUMBER    DEFAULT NULL,
   l_sql_id        IN VARCHAR2  DEFAULT NULL,
   l_wait_class    IN VARCHAR2  DEFAULT NULL,
   l_service_hash  IN NUMBER    DEFAULT NULL,
   l_module        IN VARCHAR2  DEFAULT NULL,
   l_action        IN VARCHAR2  DEFAULT NULL,
   l_client_id     IN VARCHAR2  DEFAULT NULL)
 RETURN awrrpt_text_type_table PIPELINED;
```

Parameters

Table 118–4 ASH_REPORT_TEXT Parameters

Parameter	Description
l_dbid	The database identifier
l_inst_num	The instance number
l_btime	The 'begin time'
l_etime	The 'end time'
l_options	Report level (currently not used)
l_slot_width	Specifies (in seconds) how wide the slots used in the "Top Activity" section of the report should be. This argument is optional, and if it is not specified the time interval between l_btime and l_etime is appropriately split into not more than 10 slots.
l_sid	The session ID (see Usage Notes)
l_sql_id	The SQL ID (see Usage Notes)
l_wait_class	The wait class name (see Usage Notes)
l_service_hash	The service name hash (see Usage Notes)
l_module	The module name (see Usage Notes)
l_action	The action name (see Usage Notes)
l_client_id	The client ID for end-to-end backtracing (see Usage Notes)

Return Values

The output will be one column of VARCHAR2(80).

Usage Notes

- You can call the function directly but Oracle recommends you use the `ashrpt.sql` script which prompts users for the required information.
- The unspecified optional arguments are used to generate an ASH Reports that specify 'report targets' such as a SQL statement, or a session, or a particular Service/Module combination. These arguments are specified to restrict the ASH rows that would be used to generate the report. For example, to generate an ASH report on a particular SQL statement, such as SQL_ID 'abcdefghij123 ' pass that sql_id value to the `l_sql_id` argument:

  ```
  l_sql_id =>  'abcdefghij123'
  ```

  Any combination of those optional arguments can be passed in, and only rows in ASH that satisfy all of those 'report targets' will be used. If multiple 'report targets' are specified, `AND` conditional logic is used to connect them. For example, to generate an ASH report on `MODULE` "PAYROLL" and `ACTION` "PROCESS", use the following predicate:

  ```
  l_module =>  'PAYROLL', l_action =>  'PROCESS'
  ```

  Valid SQL wildcards can be used in all the arguments that are of type `VARCHAR2`.

Table 118–5 ASH_REPORT_TEXT: Wildcards Allowed (or Not) in Arguments

Argument Name	Comment	Wildcard Allowed
`l_sid`	The session ID (for example, `V$SESSION.SID`)	No
`l_sql_id`	The SQL ID (for example, `V$SQL.SQL_ID`)	Yes
`l_wait_class`	The wait class name (for example, `V$EVENT_NAME.WAIT_CLASS`)	Yes
`l_service_hash`	The service name hash (for example, `V$ACTIVE_SERVICES.NAME_HASH`)	No
`l_module`	The module name (for example, `V$SESSION.MODULE`)	Yes
`l_action`	The action name (for example, `V$SESSION.ACTION`)	Yes
`l_client_id`	The client ID for end-to-end backtracing (for example, `V$SESSION.CLIENT_IDENTIFIER`)	Yes

AWR_DIFF_REPORT_HTML Function

This table function displays the AWR Compare Periods report in HTML.

Syntax

```
DBMS_WORKLOAD_REPOSITORY.AWR_DIFF_REPORT_HTML(
   dbid1     IN NUMBER,
   inst_num1 IN NUMBER,
   bid1      IN NUMBER,
   eid1      IN NUMBER,
   dbid2     IN NUMBER,
   inst_num2 IN NUMBER,
   bid2      IN NUMBER,
   eid2      IN NUMBER)
  RETURN awrdrpt_text_type_table PIPELINED;
```

Parameters

Table 118–6 AWR_DIFF_REPORT_HTML Parameters

Parameter	Description
dbid1	1st database identifier
inst_num1	1st instance number
bid1	1st 'Begin Snapshot' ID
eid1	1st 'End Snapshot' ID
dbid2	2nd database identifier
inst_num2	2nd instance number
bid2	2nd 'Begin Snapshot' ID
eid2	2nd 'End Snapshot' ID

Return Values

The output will be one column of VARCHAR2(500).

Usage Notes

You can call the function directly but Oracle recommends you use the awrddrpt.sql script which prompts users for the required information.

AWR_DIFF_REPORT_TEXT Function

This table function displays the AWR Compare Periods report in text.

Syntax

```
DBMS_WORKLOAD_REPOSITORY.AWR_DIFF_REPORT_TEXT(
   dbid1     IN NUMBER,
   inst_num1 IN NUMBER,
   bid1      IN NUMBER,
   eid1      IN NUMBER,
   dbid2     IN NUMBER,
   inst_num2 IN NUMBER,
   bid2      IN NUMBER,
   eid2      IN NUMBER)
  RETURN awrdrpt_text_type_table PIPELINED;
```

Parameters

Table 118–7 AWR_DIFF_REPORT_TEXT Parameters

Parameter	Description
dbid1	1st database identifier
inst_num1	1st instance number
bid1	1st 'Begin Snapshot' ID
eid1	1st 'End Snapshot' ID
dbid2	2nd database identifier
inst_num2	2nd instance number
bid2	2nd 'Begin Snapshot' ID
eid2	2nd 'End Snapshot' ID

Return Values

The output will be one column of VARCHAR2(500).

Usage Notes

You can call the function directly but Oracle recommends you use the awrddrpt.sql script which prompts users for the required information.

AWR_REPORT_HTML Function

This table function displays the AWR report in HTML.

Syntax

```
DBMS_WORKLOAD_REPOSITORY.AWR_REPORT_HTML(
   l_dbid       IN    NUMBER,
   l_inst_num   IN    NUMBER,
   l_bid        IN    NUMBER,
   l_eid        IN    NUMBER,
   l_options    IN    NUMBER DEFAULT 0)
 RETURN awrrpt_text_type_table PIPELINED;
```

Parameters

Table 118–8 AWR_REPORT_HTML Parameters

Parameter	Description
l_dbid	The database identifier
l_inst_num	The instance number
l_bid	The 'Begin Snapshot' ID
l_eid	The 'End Snapshot' ID
l_options	A flag to specify to control the output of the report. Currently, Oracle supports one value: ■ l_options - 8. Displays the ADDM specific portions of the report. These sections include the Buffer Pool Advice, Shared Pool Advice, and PGA Target Advice.

Return Values

The output will be one column of VARCHAR2(150).

Usage Notes

You can call the function directly but Oracle recommends you use the awrrpt.sql script which prompts users for the required information.

AWR_REPORT_TEXT Function

This table function displays the AWR report in text.

Syntax

```
DBMS_WORKLOAD_REPOSITORY.AWR_REPORT_TEXT(
   l_dbid       IN    NUMBER,
   l_inst_num   IN    NUMBER,
   l_bid        IN    NUMBER,
   l_eid        IN    NUMBER,
   l_options    IN    NUMBER DEFAULT 0)
 RETURN awrrpt_text_type_table PIPELINED;
```

Parameters

Table 118–9 AWR_REPORT_TEXT Parameters

Parameter	Description
`l_dbid`	The database identifier
`l_insT_num`	The `instance number`
`l_bid`	The `'Begin Snapshot' ID`
`l_eid`	The `'End Snapshot' ID`
`l_options`	A flag to specify to control the output of the report. Currently, Oracle supports one value: ▪ `l_options - 8`. Displays the ADDM specific portions of the report. These sections include the Buffer Pool Advice, Shared Pool Advice, and PGA Target Advice.

Return Values

The output will be one column of `VARCHAR2(80)`.

Usage Notes

You can call the function directly but Oracle recommends you use the `awrrpt.sql` script which prompts users for the required information.

AWR_SQL_REPORT_HTML Function

This table function displays the AWR SQL Report in HTML format.

Syntax

```
DBMS_WORKLOAD_REPOSITORY.AWR_SQL_REPORT_HTML(
   l_dbid       IN    NUMBER,
   l_inst_num   IN    NUMBER,
   l_bid        IN    NUMBER,
   l_eid        IN    NUMBER,
   l_sqlid      IN    VARCHAR2,
   l_options    IN    NUMBER DEFAULT 0)
 RETURN awrrpt_html_type_table PIPELINED;
```

Parameters

Table 118–10 AWR_SQL_REPORT_HTML Parameters

Parameter	Description
l_dbid	The database identifier
l_inst_num	The instance number
l_bid	The 'Begin Snapshot' ID
l_eid	The 'End Snapshot' ID
l_sqlid	The SQL ID of statement to be analyzed
l_options	A flag to specify to control the output of the report. Currently, not used.

Return Values

The output will be one column of VARCHAR2(500).

Usage Notes

You can call the function directly but Oracle recommends you use the awrsqrpt.sql script which prompts users for the required information.

AWR_SQL_REPORT_TEXT Function

This table function displays the AWR SQL Report in text format.

Syntax

```
DBMS_WORKLOAD_REPOSITORY.AWR_SQL_REPORT_TEXT(
   l_dbid       IN    NUMBER,
   l_inst_num   IN    NUMBER,
   l_bid        IN    NUMBER,
   l_eid        IN    NUMBER,
   l_sqlid      IN    VARCHAR2,
   l_options    IN    NUMBER DEFAULT 0)
 RETURN awrrpt_text_type_table PIPELINED;
```

Parameters

Table 118–11 AWR_SQL_REPORT_TEXT Parameters

Parameter	Description
l_dbid	The database identifier
l_inst_num	The instance number
l_bid	The 'Begin Snapshot' ID
l_eid	The 'End Snapshot' ID
l_sqlid	The SQL ID of statement to be analyzed
l_options	A flag to specify to control the output of the report. Currently, not used.

Return Values

The output will be one column of VARCHAR2(120).

Usage Notes

You can call the function directly but Oracle recommends you use the awrsqrpt.sql script which prompts users for the required information.

CREATE_BASELINE Function and Procedure

This function and procedure creates a baseline.

Syntax

```
DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE(
   start_snap_id    IN  NUMBER,
   end_snap_id      IN  NUMBER,
   baseline_name    IN  VARCHAR2,
   dbid             IN  NUMBER DEFAULT NULL);

DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE(
   start_snap_id    IN  NUMBER,
   end_snap_id      IN  NUMBER,
   baseline_name    IN  VARCHAR2,
   dbid             IN  NUMBER DEFAULT NULL)
 RETURN NUMBER;
```

Parameters

Table 118–12 CREATE_BASELINE Parameters

Parameter	Description
start_snap_id	The start snapshot sequence number.'
end_snap_id	The end snapshot sequence number.
baseline_name	The name of baseline.
dbid	The database id (default to local DBID).

Examples

This example creates a baseline (named 'oltp_peakload_bl') between snapshots 105 and 107 for the local database:

```
EXECUTE DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE (start_snap_id => 105,
end_snap_id => 107,
baseline_name => 'oltp_peakload_bl');
```

If you query the DBA_HIST_BASELINE view after the Create Baseline action, you will see the newly created baseline in the Workload Repository.

CREATE_SNAPSHOT Function and Procedure

This function and procedure create snapshots.In the case of the function, the snapshot ID is returned.

Syntax

```
DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT(
   flush_level IN VARCHAR2 DEFAULT 'TYPICAL');

DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT(
   flush_level IN VARCHAR2 DEFAULT 'TYPICAL')
 RETURN NUMBER;
```

Parameters

Table 118–13 CREATE_SNAPSHOT Parameters

Parameter	Description
flush_level	The flush level for the snapshot is either 'TYPICAL' or 'ALL'

Examples

This example creates a manual snapshot at the TYPICAL level:

```
EXECUTE DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT();
```

If you query the DBA_HIST_SNAPSHOT view after the CREATE_SNAPSHOT action, you will see one more snapshot ID added to the Workload Repository.

DROP_BASELINE Procedure

This procedure drops a baseline.

Syntax

```
DBMS_WORKLOAD_REPOSITORY.DROP_BASELINE(
   baseline_name  IN  VARCHAR2,
   cascade        IN  BOOLEAN DEFAULT false,
   dbid           IN  NUMBER DEFAULT NULL);
```

Parameters

Table 118–14 DROP_BASELINE Parameters

Parameter	Description
baseline_name	The name of baseline.
cascade	If TRUE, the pair of snapshots associated with the baseline will also be dropped. Otherwise, only the baseline is removed.
dbid	The (optional) database id (default to local DBID).

Examples

This example drops the baseline 'oltp_peakload_bl' without dropping the underlying snapshots:

```
EXECUTE DBMS_WORKLOAD_REPOSITORY.DROP_BASELINE (
       baseline_name => 'oltp_peakload_bl');
```

If you query the DBA_HIST_BASELINE view after the DROP_BASELINE action, you will see the specified baseline definition is removed. You can query the DBA_HIST_SNAPSHOT view to find that the underlying snapshots are left intact.

DROP_SNAPSHOT_RANGE Procedure

This procedure drops a range of snapshots.

Syntax

```
DBMS_WORKLOAD_REPOSITORY.DROP_SNAPSHOT_RANGE(
   low_snap_id    IN  NUMBER,
   high_snap_id   IN  NUMBER
   dbid           IN  NUMBER DEFAULT NULL);
```

Parameters

Table 118–15 DROP_SNAPSHOT_RANGE Procedure Parameters

Parameter	Description
low_snap_id	The low snapshot id of snapshots to drop.
high_snap_id	The high snapshot id of snapshots to drop.
dbid	The database id (default to local DBID.

Examples

This example drops the range of snapshots between snapshot id 102 to 105 for the local database:

```
EXECUTE DBMS_WORKLOAD_REPOSITORY.DROP_SNAPSHOT_RANGE(102, 105);
```

If you query the dba_hist_snapshot view after the Drop Snapshot action, you will see that snapshots 102 to 105 are removed from the Workload Repository.

MODIFY_SNAPSHOT_SETTINGS Procedures

This procedure controls three aspects of snapshot generation.

- The INTERVAL setting affects how often snapshots are automatically captured.
- The RETENTION setting affects how long snapshots are retained in the Workload Repository.
- The number of SQL captured for each Top criteria. If the user manually specifies a value for Top N SQL, the AWR SQL collection will use the user-specified number for both automatic and manual snapshots.

There are two overloads. The first takes a NUMBER and the second takes a VARCHAR2 for the topnsql argument. The differences are described under the Parameters description.

Syntax

```
DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS(
   retention   IN  NUMBER    DEFAULT NULL,
   interval    IN  NUMBER    DEFAULT NULL,
   topnsql     IN  NUMBER    DEFAULT NULL,
   dbid        IN  NUMBER    DEFAULT NULL);

DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS(
   retention   IN  NUMBER    DEFAULT NULL,
   interval    IN  NUMBER    DEFAULT NULL,
   topnsql     IN  VARCHAR2,
   dbid        IN  NUMBER    DEFAULT NULL);
```

Parameters

Table 118–16 MODIFY_SNAPSHOT_SETTINGS Procedure Parameters

Parameter	Description
retention	The new retention time (in minutes). The specified value must be in the range of MIN_RETENTION (1 day) to MAX_RETENTION (100 years). If ZERO is specified, snapshots will be retained forever. A large system-defined value will be used as the retention setting. If NULL is specified, the old value for retention is preserved.
interval	The new interval setting between each snapshot, in units of minutes. The specified value must be in the range MIN_RETENTION (10 minutes) to MAX_RETENTION (1 year). If ZERO is specified, automatic and manual snapshots will be disabled. A large system-defined value will be used as the retention setting. If NULL is specified, the current value is preserved.

Table 118–16 (Cont.) MODIFY_SNAPSHOT_SETTINGS Procedure Parameters

Parameter	Description
topnsql	■ If NUMBER: Top N SQL size. The number of Top SQL to flush for each SQL criteria (Elapsed Time, CPU Time, Parse Calls, Shareable Memory, Version Count). The value for this setting will not be affected by the statistics/flush level and will override the system default behavior for the AWR SQL collection. The setting will have a minimum value of 30 and a maximum value of 100000000. Specifying NULL will keep the current setting. ■ If VARCHAR2: Users are allowed to specify the following values: (DEFAULT, MAXIMUM, N), where N is the number of Top SQL to flush for each SQL criteria. Specifying DEFAULT will revert the system back to the default behavior of Top 30 for statistics level TYPICAL and Top 100 for statistics level ALL. Specifying MAXIMUM will cause the system to capture the complete set of SQL in the cursor cache. Specifying the number N is equivalent to setting the Top N SQL with the NUMBER type. Specifying NULL for this argument will keep the current setting.
dbid	The database identifier in AWR for which to modify the snapshot settings. If NULL is specified, the local dbid will be used. Defaults to NULL.

Examples

This example changes the interval setting to one hour and the retention setting to two weeks for the local database:

```
EXECUTE DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS(
  interval  =>  60,
  retention =>  20160);
```

If you query the DBA_HIST_WR_CONTROL table after this procedure is executed, you will see the changes to these settings.

119

DBMS_WM

The `DBMS_WM` package provides an interface to Oracle Database Workspace Manager (often referred to as Workspace Manager).

Documentation of DBMS_WM

For a complete description of this package, see `DBMS_WM` in *Oracle Database Application Developer's Guide - Workspace Manager*.

120

DBMS_XDB

The DBMS_XDB package supports the following features:

- Resource Management subprograms which complement Resource Views
- The Access Control List (ACL)-based Security Mechanism
- Configuration Session Management
- Creation of the XDB username

See Also:

- *Oracle XML DB Developer's Guide*
- *Oracle Database New Features*

This chapter contains the following topics:

Using DBMS_XDB

This section contains topics which relate to using the `DBMS_XDB` package.

Overview

The DBMS_XDB package supports the following features:

- The Resource Management functionality providesLINK Procedure, EXISTSRESOURCE Function, LOCKRESOURCE Function, GETLOCKTOKEN Procedure, UNLOCKRESOURCE Function, CREATERESOURCE Functions, RENAMERESOURCE Procedure, DELETERESOURCE Procedure, GETRESOID Function, CREATEOIDPATH Function, REBUILDHIERARCHICALINDEX Procedure and CREATEFOLDER Function subprograms which complement Resource Views.
- The Access Control List (ACL)-based Security Mechanism can be used with in-hierarchy ACLs stored by the database or in-memory ACLs that may be stored outside the database. Some of these methods can be used for both Oracle resources and arbitrary database objects. Use CHECKPRIVILEGES Function, GETACLDOCUMENT Function, CHANGEPRIVILEGES Function and GETPRIVILEGES Function, LINK Procedure, LINK Procedure, LINK Procedure, LINK Procedure, LINK Procedure, LINK Procedure for Oracle Resources. ACLCHECKPRIVILEGES Function provides access to Oracle's ACL-based Security mechanism without storing objects in the Hierarchy.
- Configuration Session Management is supported by CFG_REFRESH Procedure, CFG_GET Function and CFG_UPDATE Procedure. methods.
- The XDB username is created during XDB installation. This user owns a set of default tables and packages. GETXDB_TABLESPACE Function and MOVEXDB_TABLESPACE Procedure enable movement of schemas to a specified tablespace, and support the default SYSAUX tablespace introduction

Constants

Table 120–1 Defined Constants for DBMS_XDB

Constant	Type	Value	Description
DELETE_RESOURCE	NUMBER	1	Deletes a resource; fails if the resource has children.
DELETE_RECURSIVE	NUMBER	2	Deletes a resource and its children, if any.
DELETE_FORCE	NUMBER	3	Deletes the resource, even if the object it contains is invalid.
DELETE_RECURSIVE_FORCE	NUMBER	4	Deletes a resource and its children, if any, even if the object it contains is invalid.

Summary of DBMS_XDB Subprograms

Table 120–2 DBMS_XDB Package Subprograms

Subprogram	Description
ACLCHECKPRIVILEGES Function on page 120-7	Checks access privileges granted to the current user by specified ACL document on a resource whose owner is specified by the 'owner' parameter.
APPENDRESOURCEMETADATA Procedure on page 120-8	Takes in user-defined metadata either as a `REF` to `XMLTYPE` or an `XMLTYPE` and adds it to the desired resource
CFG_GET Function on page 120-9	Retrieves the session's configuration information
CFG_REFRESH Procedure on page 120-10	Refreshes the session's configuration information to the latest configuration
CFG_UPDATE Procedure on page 120-11	Updates the configuration information
CHANGEPRIVILEGES Function on page 120-12	Adds the given ACE to the given resource's ACL
CHECKPRIVILEGES Function on page 120-13	Checks access privileges granted to the current user on the specified resource
CREATEFOLDER Function on page 120-14	Creates a new folder resource in the hierarchy
CREATEOIDPATH Function on page 120-15	Creates a virtual path to the resource based on object ID
CREATERESOURCE Functions on page 120-16	Creates a new resource
DELETERESOURCE Procedure on page 120-18	Deletes a resource from the hierarchy
DELETERESOURCEMETADATA Procedures on page 120-19	Deletes metadata from a resource (can be used for schema-based or nonschema-based metadata)
EXISTSRESOURCE Function on page 120-20	Determines if a resource is the hierarchy, based on its absolute path
GETACLDOCUMENT Function on page 120-21	Retrieves ACL document that protects resource given its path name
GETFTPPORT Function on page 120-22	Gets the value of the current FTP port
GETHTTPPORT Function on page 120-23	Gets the value of the current HTTP port
GETLOCKTOKEN Procedure on page 120-24	Returns that resource's lock token for the current user given a path to a resource
GETPRIVILEGES Function on page 120-25	Gets all privileges granted to the current user on the given resource
GETRESOID Function on page 120-26	Returns the object ID of the resource from its absolute path
GETXDB_TABLESPACE Function on page 120-27	Returns the current tablespace of the XDB (user)
LINK Procedure on page 120-28	Creates a link to an existing resource

Table 120–2 (Cont.) DBMS_XDB Package Subprograms

Subprogram	Description
LOCKRESOURCE Function on page 120-29	Gets a WebDAV-style lock on that resource given a path to that resource
MOVEXDB_TABLESPACE Procedure on page 120-30	Moves the XDB (user) to the specified tablespace
PURGERESOURCEMETADATA Procedure on page 120-31	Deletes all user metadata from a resource.
REBUILDHIERARCHICALINDEX Procedure on page 120-32	Rebuilds the hierarchical index after import or export operations
RENAMERESOURCE Procedure on page 120-33	Renames the XDB resource
SETACL Procedure on page 120-34	Sets the ACL on the given resource
SETFTPPORT Procedure on page 120-35	Sets the FTP port to a new value
SETHTTPPORT Procedure on page 120-36	Sets the HTTP port to a new value
UPDATERESOURCEMETADATA Procedures on page 120-37	Updates metadata for a resource
UNLOCKRESOURCE Function on page 120-39	Unlocks the resource given a lock token and resource path

ACLCHECKPRIVILEGES Function

This function checks access privileges granted to the current user by specified ACL document by the OWNER of the resource. Returns positive integer if all privileges are granted.

Syntax

```
DBMS_XDB.ACLCHECKPRIVILEGES(
   acl_path  IN  VARCHAR2,
   owner     IN  VARCHAR2,
   privs     IN  xmltype)
 RETURN PLS_INTEGER;
```

Parameters

Table 120–3 ACLCHECKPRIVILEGES Function Parameters

Parameter	Description
acl_path	Absolute path in the Hierarchy for ACL document
owner	Resource owner name; the pseudo user "DAV:owner" is replaced by this user during ACL privilege resolution
privs	An XMLType instance of the privilege element specifying the requested set of access privileges. See description for CHECKPRIVILEGES Function.

APPENDRESOURCEMETADATA Procedure

This procedure takes in user-defined metadata either as a REF to XMLTYPE or an XMLTYPE and adds it to the desired resource.

Syntax

```
DBMS_XDB.APPENDRESOURCEMETADATA (
 abspath   IN  VARCHAR2,
 metadata  IN  XMLTYPE);

DBMS_XDB.APPENDRESOURCEMETADATA (
 abspath   IN  VARCHAR2,
 metadata  IN  REF SYS.XMLTYPE);
```

Parameters

Table 120–4 APPENDRESOURCEMETADATA Procedure

Parameter	Description
abspath	Absolute path of the resource
metadata	Metadata can be schema based or nonschema-based. Schema-based metadata will be stored in its own table.

Usage Notes

- In the case in which a REF is passed in, the procedure stores the REF in the resource, and the metadata is stored in a separate table. In this case you are responsible for populating the RESID column for the metadata table. Note that theREF passed in must be unique. In other words, there must not be aREF with the same value in the resource metadata, as this would violate uniqueness of properties. An error will be thrown if users attempt to add a REF that already exists.
- In the case where the XMLTYPE is passed in, the data is parsed to determine if it is schema-based or not and stored accordingly.

CFG_GET Function

This function retrieves the session's configuration information as an `XMLType` instance.

Syntax

```
DBMS_XDB.CFG_GET
 RETURN SYS.XMLType;
```

CFG_REFRESH Procedure

This procedure refreshes the session's configuration information to the latest configuration.

Syntax

```
DBMS_XDB.CFG_REFRESH;
```

CFG_UPDATE Procedure

This procedure updates the configuration information and commits the change.

Syntax

```
DBMS_XDB.CFG_UPDATE(
   xdbconfig   IN  SYS.XMLTYPE);
```

Parameters

Table 120–5 CFG_UPDATE Procedure Parameters

Parameter	Description
xdbconfig	The new configuration data

CHANGEPRIVILEGES Function

This function adds the given ACE to the given resource's ACL.

Syntax

```
DBMS_XDB.CHANGEPRIVILEGES(
   res_path   IN    VARCHAR2,
   ace        IN    xmltype)
 RETURN PLS_INTEGER;
```

Parameters

Table 120–6 CHANGEPRIVILEGES Function Parameters

Parameter	Description
res_path	Path name of the resource for which privileges need to be changed
ace	An XMLType instance of the <ace> element which specifies the <principal>, the operation <grant> and the list of privileges

Return Values

A positive integer if the ACL was successfully modified.

Usage Notes

If no ACE with the same principal and the same operation (grant/deny) already exists in the ACL, the new ACE is added at the end of the ACL.

CHECKPRIVILEGES Function

This function checks access privileges granted to the current user on the specified resource.

Syntax

```
DBMS_XDB.CHECKPRIVILEGES(
   res_path   IN  VARCHAR2,
   privs      IN  xmltype)
 RETURN PLS_INTEGER;
```

Parameters

Table 120–7 CHECKPRIVILEGES Function Parameters

Parameter	Description
`res_path`	Absolute path in the Hierarchy for resource
`privs`	An `XMLType` instance of the privilege element specifying the requested set of access privileges

Return Values

A positive integer if all requested privileges granted.

CREATEFOLDER Function

This function creates a new folder resource in the hierarchy.

Syntax

```
DBMS_XDB.CREATEFOLDER(
   path   IN  VARCHAR2)
 RETURN BOOLEAN;
```

Parameters

Table 120–8 CREATEFOLDER Function Parameters

Parameter	Description
`path`	Path name for the new folder

Return Values

`TRUE` if operation successful; `FALSE`, otherwise.

Usage Notes

The given path name's parent folder must already exist in the hierarchy: if `'/folder1/folder2'` is passed as the path parameter, then `'/folder1'` must already exist.

CREATEOIDPATH Function

This function creates a virtual path to the resource based on object ID.

Syntax

```
DBMS_XDB.CREATEOIDPATH(
   oid    IN   RAW)
 RETURN VARCHAR2;
```

Parameters

Table 120–9 CREATEOIDPATH Function Parameters

Parameter	Description
oid	Object ID of the resource

CREATERESOURCE Functions

The functions create a new resource. The description of the overload options precede each version of the syntax

Syntax

Creates a new resource with the given string as its contents:

```
DBMS_XDB.CREATERESOURCE(
   path    IN  VARCHAR2,
   data    IN  VARCHAR2)
 RETURN BOOLEAN;
```

Creates a new resource with the given `XMLType` data as its contents:

```
DBMS_XDB.CREATERESOURCE(
   path    IN  VARCHAR2,
   data    IN  SYS.XMLTYPE)
 RETURN BOOLEAN;
```

Given a `REF` to an existing `XMLType` row, creates a resource whose contents point to that row. That row should not already exist inside another resource:

```
DBMS_XDB.CREATERESOURCE(
   path     IN  VARCHAR2,
   datarow  IN  REF SYS.XMLTYPE)
 RETURN BOOLEAN;
```

Creates a resource with the given `BLOB` as its contents, and specifies character set of the source `BLOB`:

```
DBMS_XDB.CREATERESOURCE(
   path    IN  VARCHAR2,
   data    IN  BLOB,
   csid    IN  NUMBER :=0)
 RETURN BOOLEAN;
```

Creates a resource with the given `BFILE` as its contents, and specifies character set of the source `BFILE`:

```
DBMS_XDB.CREATERESOURCE(
   path    IN  VARCHAR2,
   data    IN  BFILE,
   csid    IN  NUMBER :=0)
 RETURN BOOLEAN;
```

Creates a resource with the given `CLOB` as its contents:

```
DBMS_XDB.CREATERESOURCE(
   path    IN  VARCHAR2,
   data    IN  CLOB)
 RETURN BOOLEAN;
```

Parameters

Table 120–10 CREATERESOURCE Function Parameters

Parameter	Description
`path`	Path name of the resource to create. The path name's parent folder must already exist in the hierarchy. In other words, if `/foo/bar.txt` is passed in, then folder `/foo` must already exist.
`data`	The new resource's contents. The data will be parsed to check if it contains a schema-based XML document, and the contents will be stored as schema-based in the schema's default table. Otherwise, it will be saved as binary data.
`datarow`	`REF` to an `XMLType` row to be used as the contents
`csid`	Character set id of the document. Must be a valid Oracle id; otherwise returns an error. If a zero CSID is specified then the data is defaulted to the database character set. Otherwise, the encoding of the data is determined as follows: ■ From the path extension, determine the resource's MIME type. ■ If the MIME type is */xml, then the encoding is detected based on Appendix F of the W3C XML 1.0 Reference at http://www.w3.org/TR/2000/REC-xml-20001006; otherwise, it is defaulted to the database character set.

Return Values

`TRUE` if operation successful; `FALSE`, otherwise.

DELETERESOURCE Procedure

This procedure deletes a resource from the hierarchy.

Syntax

```
DBMS_XDB.DELETERESOURCE(
   path          IN      VARCHAR2,
   delete_option IN      PLS_INTEGER);
```

Parameters

Table 120–11 DELETERESOURCE Procedure Parameters

Parameter	Description
`path`	Path name of the resource to delete
`delete_option`	The option that controls how a a resource is deleted; defined in Table 120–1 on page 120-4: ■ `DELETE_RESOURCE` ■ `DELETE_RECURSIVE` ■ `DELETE_FORCE` ■ `DELETE_RECURSIVE_FORCE`

DELETERESOURCEMETADATA Procedures

This procedure takes in a resource by absolute path and removes either the schema-based metadata identified by the REF, or the metadata identified by the namespace and name combination, which can be either schema-based or non-schema based. It will also take an additional (optional) parameter that specifies how to delete it. This parameter is only relevant for schema-based resource metadata that needs to be deleted. For non-schema based metadata, this parameter is ignored.

Syntax

Can be used only or schema-based metadata:

```
DBMS_XDB.DELETERESOURCEMETADATA (
   abspath        IN  VARCHAR2,
   metadata       IN  REF SYS.XMLTYPE,
   delete_option  IN  pls_integer := dbms_xdb.DELETE_RESOURCE_METADATA_CASCADE);
```

Can be used for schema-based or nonschema-based metadata:

```
DBMS_XDB.DELETERESOURCEMETADATA (
   abspath        IN VARCHAR2,
   metadatans     IN VARCHAR2,
   metadataname   IN VARCHAR2,
   delete_option  IN pls_integer := dbms_xdb.DELETE_RESOURCE_METADATA_CASCADE);
```

Parameters

Table 120–12 DELETERESOURCEMETADATA Procedure Parameters

Parameter	Description
`abspath`	Absolute path of the resource
`metadata`	`REF` to the piece of metadata (schema based) to be deleted
`mettadatans`	Namespace of the metadata fragment to be removed
`mettadataname`	Local name of the metadata fragment to be removed
`delete_option`	Only applicable for schema-based metadata, this can be one of the following: ■ `DELETE_RES_METADATA_CASCADE` - deletes the corresponding row in the metadata table ■ `DELETE_RES_METADATA_NOCASCADE` - does not delete the row in the metadata table

EXISTSRESOURCE Function

This function indicates if a resource is in the hierarchy. Matches resource by a string that represents its absolute path.

Syntax

```
DBMS_XDB.EXISTSRESOURCE(
   abspath    IN    VARCHAR2)
 RETURN BOOLEAN;
```

Parameters

Table 120–13 EXISTSRESOURCE Function Parameters

Parameter	Description
abspath	Path name of the resource whose ACL document is required

Return Values

TRUE if the resource is found.

GETACLDOCUMENT Function

This function retrieves ACL document that protects resource given its path name.

Syntax

```
DBMS_XDB.GETACLDOCUMENT(
   abspath  IN  VARCHAR2)
  RETURN sys.xmltype;
```

Parameters

Table 120–14 GETACLDOCUMENT Function Parameters

Parameter	Description
`abspath`	Path name of the resource whose ACL document is required

Return Values

The `XMLType` for ACL document.

GETFTPPORT Function

This procedure gets the value of the current FTP port.

Syntax

```
DBMS_XDB.GETFTPPORT
 RETURN NUMBER;
```

GETHTTPPORT Function

This procedure gets the value of the current HTTP port.

Syntax

```
DBMS_XDB.GETHTTPPORT
 RETURN NUMBER;
```

GETLOCKTOKEN Procedure

Given a path to a resource, this procedure returns that resource's lock token for the current user.

Syntax

```
DBMS_XDB.GETLOCKTOKEN(
   path          IN      VARCHAR2,
   locktoken     OUT     VARCHAR2);
```

Parameters

Table 120–15 GETLOCKTOKEN Procedure Parameters

Parameter	Description
`path`	Path name to the resource
`locktoken`	Logged-in user's lock token for the resource

Usage Notes

The user must have `READPROPERTIES` privilege on the resource.

GETPRIVILEGES Function

This function gets all privileges granted to the current user on the given resource.

Syntax

```
DBMS_XDB.GETPRIVILEGES(
   res_path    IN     VARCHAR2)
 RETURN sys.xmltype;
```

Parameters

Table 120–16 GETPRIVILEGES Function Parameters

Parameter	Description
res_path	Absolute path in the hierarchy of the resource

Return Values

An XMLType instance of <privilege> element, which contains the list of all leaf privileges granted on this resource to the current user.

GETRESOID Function

Returns the object ID of the resource from its absolute path.

Syntax

```
DBMS_XDB.GETRESOID(
   abspath IN VARCHAR2)
RETURN RAW;
```

Parameters

Table 120–17 GETRESOID Function Parameters

Parameter	Description
abspath_path	Absolute path of the resource

Return Values

NULL if the resource is not present.

GETXDB_TABLESPACE Function

This function returns the current tablespace of the XDB (user).

Syntax

```
DBMS_XDB.GETXDB_TABLESPACE
 RETURN VARCHAR2;
```

LINK Procedure

This procedure creates a link to an existing resource.

Syntax

```
DBMS_XDB.LINK(
   srcpath    IN  VARCHAR2,
   linkfolder IN  VARCHAR2,
   linkname   IN  VARCHAR2);
```

Parameters

Table 120–18 LINK Procedure Parameters

Parameter	Description
`srcpath`	Path name of the resource to which a link is made
`linkfolder`	Folder in which the new link is placed
`linkname`	Name of the new link

LOCKRESOURCE Function

Given a path to a resource, this function gets a WebDAV-style lock on that resource.

Syntax

```
DBMS_XDB.LOCKRESOURCE(
   path      IN  VARCHAR2,
   depthzero IN  BOOLEAN,
   shared    IN  boolean)
RETURN BOOLEAN;
```

Parameters

Table 120–19 LOCKRESOURCE Function Parameters

Parameter	Description
`path`	Path name of the resource to lock.
`depthzero`	Currently not supported
`shared`	Passing `TRUE` will obtain a shared write lock

Return Values

`TRUE` if successful.

Usage Notes

The user must have `UPDATE` privileges on the resource.

MOVEXDB_TABLESPACE Procedure

This procedure moves the XDB (user) to the specified tablespace.

Syntax

```
DBMS_XDB.MOVEXDB_TABLESPACE(
   new_tablespace   IN  VARCHAR2);
```

Parameters

Table 120–20 MOVEXDB_TABLESPACE Procedure Parameters

Parameter	Description
`new_tablespace`	Name of the tablespace where the XDB will be moved

Usage Notes

This operation waits for all concurrent XDB sessions to exit.

PURGERESOURCEMETADATA Procedure

This procedure deletes all user metadata from a resource. Schema-based metadata is removed in cascade mode, rows being deleted from the corresponding metadata tables.

Syntax

```
DBMS_XDB.PURGERESOURCEMETADATA(

 abspath  IN  VARCHAR2);
```

Parameters

Table 120–21 PURGERESOURCEMETADATA Procedure Parameters

Parameter	Description
abspath	Absolute path of the resource

REBUILDHIERARCHICALINDEX Procedure

This procedure rebuilds the hierarchical index after import or export operations. This is necessary because data cannot be exported from index tables.

Syntax

```
DBMS_XDB.REBUILDHIERARCHICALINDEX;
```

RENAMERESOURCE Procedure

This procedure renames the XDB resource.

Syntax

```
DBMS_XDB.RENAMERESOURCE(
   srcpath    IN  VARCHAR2,
   destfolder IN  CARCHAR2,
   newname    IN  VARCHAR2);
```

Parameters

Table 120–22 RENAMERESOURCE Procedure Parameters

Parameter	Description
srcpath	Absolute path in the Hierarchy for the source resource destination folder
destfolder	Absolute path in the Hierarchy for the destination folder
newname	Name of the child in the destination folder

SETACL Procedure

This procedure sets the ACL on the given resource to be the ACL specified by path.

Syntax

```
DBMS_XDB.SETACL(
   res_path   IN  VARCHAR2,
   acl_path   IN  VARCHAR2);
```

Parameters

Table 120–23 SETACL Procedure Parameters

Parameter	Description
`res_path`	Absolute path in the Hierarchy for resource
`acl_path`	Absolute path in the Hierarchy for ACL

Usage Notes

The user must have `<write-acl>` privileges on the resource.

SETFTPPORT Procedure

This procedure sets the FTP port to a new value.

Syntax

```
DBMS_XDB.SETFTPPORT(
   new_port  IN  NUMBER);
```

Parameters

Table 120–24 SETFTPPORT Procedure Parameters

Parameter	Description
new_port	Value to which the FTP port will be set

SETHTTPPORT Procedure

This procedure sets the HTTP port to a new value.

Syntax

```
DBMS_XDB.SETHTTPPORT(
   new_port  IN  NUMBER);
```

Parameters

Table 120–25 SETHTTPPORT Procedure Parameters

Parameter	Description
new_port	Value to which the HTTP port will be set

UPDATERESOURCEMETADATA Procedures

This procedure updates metadata for a resource. The procedure takes in a resource identified by absolute path and the metadata in it to replace identified by its REF. It replaces that piece of metadata with user-defined metadata which is either in the form of a REF to XMLTYPE or an XMLTYPE.

Syntax

Can be used to update schema-based metadata only. The new metadata must be schema-based:

```
DBMS_XDB.UPDATERESOURCEMETADATA(
   abspath  IN VARCHAR2,
   oldmetadata    IN   REF SYS.XMLTYPE,
   newmetadata    IN   REF SYS.XMLTYPE)
```

Can be used to update schema-based metadata only. The new metadata must be schema-based or nonschema-based:

```
DBMS_XDB.UPDATERESOURCEMETADATA(
   abspath        IN   VARCHAR2,
   oldmetadata    IN   REF SYS.XMLTYPE,
   newmetadata    IN   XMLTYPE);
```

Can be used for both schema-based and nonschema-based metadata:

```
DBMS_XDB.UPDATERESOURCEMETADATA(
   abspath        IN   VARCHAR2,
   oldns          IN   VARCHAR2,
   oldname IN VARCHAR,
   newmetadata IN XMLTYPE);
```

Can be used for both schema-based or nonschema-based metadata. New metadata must be schema-based:

```
DBMS_XDB.UPDATERESOURCEMETADATA(
   abspath       IN   VARCHAR2,
   oldns         IN   VARCHAR2,
   oldname       IN   VARCHAR,
   newmetadata   IN   REF SYS.XMLTYPE);
```

Parameters

Table 120–26 UPDATERESOURCEMETADATA Procedure Parameters

Parameter	Description
abspath	Absolute path of the resource
oldmetadata	REF to the old of metadata
newmetadata	REF to the new, replacement metadata (can be either schema-based or nonschema-based depending on the overload)
oldns	Namespace identifying old metadata
oldname	Local name identifying old metadata

Usage Notes

In the case of `REF`, it stores the `REF` in the resource and the metadata is stored in a separate table. Uniqueness of `REFs` is enforced. In the case where the `XMLTYPE` is passed in, data is parsed to determine if it is schema-based or not and is stored accordingly.

UNLOCKRESOURCE Function

This function unlocks the resource given a lock token and a path to the resource.

Syntax

```
DBMS_XDB.UNLOCKRESOURCE(
   path     IN  VARCHAR2,
   deltoken IN  VARCHAR2)
 RETURN BOOLEAN;
```

Parameters

Table 120–27 UNLOCKRESOURCE Function Parameters

Parameter	Description
`path`	Path name to the resource
`deltoken`	Lock token to be removed

Return Values

`TRUE` if operation successful.

Usage Notes

The user must have `UPDATE` privileges on the resource.

121

DBMS_XDB_VERSION

Oracle XML DB versioning APIs are found in the `DBMS_XBD_VERSION` package. Functions and procedures of `DBMS_XDB_VERSION` help to create a VCR and manage the versions in the version history.

This chapter contains the following topic:

- Summary of DBMS_XDB_VERSION Subprograms

See Also: *Oracle XML DB Developer's Guide*

Summary of DBMS_XDB_VERSION Subprograms

Table 121–1 DBMS_XDB_VERSION Package Subprograms

Method	Description
CHECKIN Function on page 121-3	Checks in a checked-out VCR and returns the resource id of the newly-created version
CHECKOUT Procedure on page 121-4	Checks out a VCR before updating or deleting it
GETCONTENTSBLOBBYRESID Function on page 121-5	Obtain contents as a `BLOB`
GETCONTENTSCLOBBYRESID Function on page 121-6	Obtain contents as a `CLOB`
GETCONTENTSXMLBYRESID Function on page 121-7	Obtain contents as an `XMLType`
GETPREDECESSORS Function on page 121-8	Retrieves the list of predecessors by path name
GETPREDSBYRESID Function on page 121-9	Retrieves the list of predecessors by resource id
GETRESOURCEBYRESID Function on page 121-10	Obtains the resource as an `XMLType`, given the resource object ID
GETSUCCESSORS Function on page 121-11	Retrieves the list of successors by path name
GETSUCCSBYRESID Function on page 121-12	Retrieves the list of successors by resource id
MAKEVERSIONED Function on page 121-13	Turns a regular resource whose path name is given into a version-controlled resource
UNCHECKOUT Function on page 121-14	Checks in a checked-out resource, returns the resource id of the version before the resource is checked out

CHECKIN Function

This function checks in a checked-out VCR and returns the resource id of the newly-created version.

Syntax

```
DBMS_XDB_VERSION.CHECKIN(
   pathname VARCHAR2)
 RETURN DBMS_XDB.resid_type;
```

Parameters

Table 121–2 CHECKIN Function Parameters

Parameter	Description
pathname	The path name of the checked-out resource.

Usage Notes

This is not an auto-commit SQL operation. CHECKIN Function doesn't have to take the same path name that was passed to CHECKOUT Procedure operation. However, the CHECKIN Function path name and the CHECKOUT Procedure path name must be of the same resource for the operations to function correctly. If the resource has been renamed, the new name must be used to CHECKIN Function because the old name is either invalid or is currently bound with a different resource. Exception is raised if the path name does not exist. If the path name has been changed, the new path name must be used to CHECKIN Function the resource.

CHECKOUT Procedure

This procedure checks out a VCR before updating or deleting it.

Syntax

```
DBMS_XDB_VERSION.Checkout(
   pathname    VARCHAR2);
```

Parameters

Table 121–3 CHECKOUT Procedure Parameters

Parameter	Description
pathname	The path name of the VCR to be checked out.

Usage Notes

This is not an auto-commit SQL operation. Two users of the same workspace cannot CHECKOUT Procedure the same VCR at the same time. If this happens, one user must rollback. As a result, it is good practice to commit the CHECKOUT Procedure operation before updating a resource and avoid loss of the update if the transaction is rolled back. An exception is raised if the given resource is not a VCR, if the VCR is already checked out, if the resource doesn't exist.

GETCONTENTSBLOBBYRESID Function

This function obtain contents as a BLOB.

Syntax

```
DBMS_XDB_VERSION.GETCONTENTSBLOBYRESID(
   resid      DBMS_XDB.resid_type)
 RETURN BLOB;
```

Parameters

Table 121–4 GETCONTENTSBLOBYRESID Function Parameters

Parameter	Description
resid	The resource id.

GETCONTENTSCLOBBYRESID Function

This function obtains contents as a CLOB.

Syntax

```
DBMS_XDB_VERSION.GETCONTENTSCLOBYRESID(
   resid     DBMS_XDB.resid_type)
 RETURN CLOB;
```

Parameters

Table 121–5 GETCONTENTSCLOBYRESID Function Parameters

Parameter	Description
resid	The resource id.

GETCONTENTSXMLBYRESID Function

This function obtains contents as an `XMLType`.

Syntax

```
DBMS_XDB_VERSION.GETCONTENTSXMLBYRESID(
   resid      DBMS_XDB.resid_type)
 RETURN XMLType;
```

Parameters

Table 121–6 GETCONTENTSXMLBYRESID Function Parameters

Parameter	Description
`resid`	The resource id.

Return Values

If the contents are not valid XML, returns `NULL`.

GETPREDECESSORS Function

This function retrieves the list of predecessors by the path name.

Syntax

```
DBMS_XDB_VERSION.GETPREDECESSORS(
   pathname       VARCHAR2)
 RETURN resid_list_type;
```

Parameters

Table 121–7 GETPREDECESSORS Function Parameters

Parameter	Description
pathname	The path name of the resource.

Return Values

An exception is raised if PATHNAME is illegal.

GETPREDSBYRESID Function

This function retrieves the list of predecessors by resource id.

Syntax

```
DBMS_XDB_VERSION.GETPREDSBYRESID(
   resid      resid_type)
 RETURN resid_list_type;
```

Parameters

Table 121–8 GETPREDSBYRESID Function Parameters

Parameter	Description
resid	The resource id.

Usage Notes

Getting predecessors by RESID is more efficient than by PATHNAME.

Exceptions

An exception is raised if the RESID is illegal.

GETRESOURCEBYRESID Function

This function obtains the resource as an `XMLType`, given the resource object ID. Because the system will not create a path name for versions, this function is useful for retrieving the resource using its resource id.

Syntax

```
DBMS_XDB_VERSION.GETRESOURCEBYRESID(
   resid      resid_type)
 RETURN XMLType;
```

Parameters

Table 121–9 GETRESOURCEBYRESID Function Parameters

Parameter	Description
`resid`	The resource id.

GETSUCCESSORS Function

Given a version resource or a VCR, this function retrieves the list of the successors of the resource by the path name.

Syntax

```
DBMS_XDB_VERSION.GETSUCCESSORS(
   pathname VARCHAR2)
 RETURN resid_list_type;
```

Parameters

Table 121–10 GETSUCCESSORS Function Parameters

Parameter	Description
pathname	The path name of the resource.

Usage Notes

Getting successors by RESID is more efficient than by PATHNAME.

Exceptions

An exception is raised if the PATHNAME is illegal.

GETSUCCSBYRESID Function

This function retrieves the list of the successors of the resource by resource id using version resource or VCR.

Syntax

```
DBMS_XDB_VERSION.GETSUCCSBYRESID(
   resid    resid_type)
  RETURN resid_list_type;
```

Parameters

Table 121–11 GETSUCCSBYRESID Function Parameters

Parameter	Description
resid	The resource id.

Usage Notes

Getting successors by RESID is more efficient than by PATHNAME.

Exceptions

An exception is raised if the PATHNAME is illegal.

MAKEVERSIONED Function

This function turns a regular resource whose path name is given into a version-controlled resource. This new resource is then put under version control. All other path names continue to refer to the original resource.

Syntax

```
DBMS_XDB_VERSION.MAKEVERSIONED(
   pathname   VARCHAR2)
 RETURN DBMS_XDB.resid_type;
```

Parameters

Table 121–12 MAKEVERSIONED Function Parameters

Parameter	Description
pathname	The path name of the resource to be put under version control.

Return Values

This function returns the resource ID of the first version, or root, of the VCR.

Usage Notes

If two or more path names are bound with the same resource, a copy of the resource will be created, and the given path name will be bound with the newly-created copy.

This is not an auto-commit SQL operation. An exception is raised if the resource doesn't exist.

- This call is legal for VCR, and neither exception nor warning is raised.
- This call is illegal for folder, version history, version resource, and ACL.
- No support for Schema-based resources is provided.

UNCHECKOUT Function

This function checks-in a checked-out resource and returns the resource id of the version before the resource is checked out.

Syntax

```
DBMS_XDB_VERSION.UNCHECKOUT(
   pathname     VARCHAR2)
 RETURN DBMS_XDB.resid_type;
```

Parameters

Table 121–13 UNCHECKOUT Function Parameters

Parameter	Description
pathname	The path name of the checked-out resource.

Usage Notes

This is not an auto-commit SQL operation. UNCHECKOUT FunctionCHECKOUT Procedure doesn't have to take the same path name that was passed to operation. However, the UNCHECKOUT Function path name and the CHECKOUT Procedure path name must be of the same resource for the operations to function correctly. If the resource has been renamed, the new name must be used to UNCHECKOUT Function, because the old name is either invalid or is currently bound with a different resource. If the path name has been changed, the new path name must be used to UNCHECKOUT Function the resource.

Exceptions

An exception is raised if the path name doesn't exist.

122

DBMS_XDBT

The DBMS_XDBT package provides a convenient mechanism for administrators to set up a CONTEXT index on the Oracle XML DB hierarchy. The package contains procedures to create default preferences, create the index and set up automatic synchronization of the CONTEXT index

The DBMS_XDBT package also contains a set of package variables that describe the configuration settings for the index. These are intended to cover the basic customizations that installations may require, but is by no means a complete set.

> **See Also:** *Oracle XML DB Developer's Guide*

This chapter contains the following topics:

Using DBMS_XDBT

Overview

The DBMS_XDBT package can be used in the following fashion:

- Customize the package to set up the appropriate configuration.
- DROPPREFERENCES ProcedureDrop any existing index preferences using the procedure.
- Create new index preferences using the CREATEPREFERENCES Procedure procedure.
- Create the CONTEXT index using the CREATEINDEX Procedure procedure.
- Set up automatic synchronization of the index using the CONFIGUREAUTOSYNC Procedureprocedure.

Operational Notes

The DBMS_XDBT package can be customized by using a PL/SQL procedure or an anonymous block to set the relevant package variables, configuration settings, and then execute the procedures. A more general approach would be to introduce the appropriate customizations by modifying this package in place, or as a copy. The system must be configured to use job queues, and the jobs can be viewed through the USER_JOBS catalog views. This section describes the configuration settings, or package variables, available to customize the DBMS_XDBT package.

Table 122–1 General Indexing Settings for Customizing DBMS_XDBT

Parameter	Default Value	Description
IndexName	XDB$CI	The name of the CONTEXT index.
IndexTablespace	XDB$RESINFO	Tablespace used by tables and indexes comprising the CONTEXT index.
IndexMemory	128M	Memory used by index creation and SYNC; less than or equal to the MAX_INDEX_MEMORY system parameter (see the CTX_ADMIN package).
LogFile	'XdbCtxLog'	The log file used for ROWID during indexing. The LOG_DIRECTORY system parameter must be set already. NULL turns off ROWID logging.

Table 122–2 Filtering Settings for Customizing DBMS_XDBT

Parameter	Default Value	Description
SkipFilter_Types	image/%, audio/%, video/%, model/%	List of mime types that should not be indexed.
NullFilter_Types	text/plain, text/html, text/xml	List of mime types that do not need to use the INSO filter. Use this for text-based documents.
FilterPref	XDB$CI_FILTER	Name of the filter preference.

Table 122–3 Stoplist Settings for Customizing DBMS_XDBT

Parameter	Default Value	Description
StoplistPref	XDB$CI_STOPLIST	Name of the stoplist.
StopWords	0..9; 'a'..'z'; 'A'..'Z'	List of stopwords, in excess of CTXSYS.DEFAULT_STOPLIST.

Table 122–4 Sectioning and Section Group Settings for Customizing DBMS_XDBT

Parameter	Default Value	Description
SectionGroup	HTML_SECTION_GROUP	Default sectioner. Use PATH_SECTION_GROUP or AUTO_SECTION_GROUP if repository contains mainly XML documents.
SectiongroupPref	XDB$CI_SECTIONGROUP	Name of the section group.

Table 122–5 Other Index Preference Settings for Customizing DBMS_XDBT

Parameter	Default Value	Description
DatastorePref	XDB$CI_DATASTORE	The name of the datastore preference.
StoragePref	XDB$CI_STORAGE	The name of the storage preference.
WordlistPref	XDB$CI_WORDLIST	The name of the wordlist preference.
DefaultLexerPref	XDB$CI_DEFAULT_LEXER	The name of the default lexer preference.

Table 122–6 SYNC (CONTEXT Synchronization) Settings for Customizing DBMS_XDBT

Parameter	Default Value	Description
AutoSyncPolicy	SYNC_BY_PENDING_COUNT	Indicates when the index should be SYNCed. One of SYNC_BY_PENDING_COUNT, SYNC_BY_TIME, or SYNC_BY_PENDING_COUNT_AND_TIME.
MaxPendingCount	2	Maximum number of documents in the CTX_USER_PENDING queue before an index SYNC is triggered. Only if the AutoSyncPolicy is SYNC_BY_PENDING_COUNT or SYNC_BY_PENDING_COUNT_AND_TIME.
CheckPendingCountInterval	10 minutes	How often, in minutes, the pending queue should be checked. Only if the AutoSyncPolicy is SYNC_BY_PENDING_COUNT or SYNC_BY_PENDING_COUNT_AND_TIME.
SyncInterval	60 minutes	Indicates how often, in minutes, the index should be SYNCed. Only if the AutoSyncPolicy is SYNC_BY_TIME or SYNC_BY_PENDING_COUNT_AND_TIME

Summary of DBMS_XDBT Subprograms

Table 122–7 DBMS_XDBT Package Subprograms

Subprogram	Description
CONFIGUREAUTOSYNC Procedure on page 122-7	Configures the CONTEXT index for automatic maintenance, SYNC
CREATEDATASTOREPREF Procedure on page 122-8	Creates a USER datastore preference for the CONTEXT index
CREATEFILTERPREF Procedure on page 122-9	Creates a filter preference for the CONTEXT index
CREATEINDEX Procedure on page 122-10	Creates the CONTEXT index on the XML DB hierarchy
CREATELEXERPREF Procedure on page 122-11	Creates a lexer preference for the CONTEXT index
CREATEPREFERENCES Procedure on page 122-12	Creates preferences required for the CONTEXT index on the XML DB hierarchy
CREATESECTIONGROUPPREF Procedure on page 122-13	Creates a storage preference for the CONTEXT index
CREATESTOPLISTPREF Procedure on page 122-14	Creates a section group for the CONTEXT index
CREATESTORAGEPREF Procedure on page 122-15	Creates a wordlist preference for the CONTEXT index
CREATEWORLDLISTPREF Procedure on page 122-16	Creates a stoplist for the CONTEXT index
DROPPREFERENCES Procedure on page 122-17	Drops any existing preferences

CONFIGUREAUTOSYNC Procedure

This procedure sets up jobs for automatic SYNCs of the CONTEXT index.

Syntax

```
DBMS_XDBT.CONFIGUREAUTOSYNC;
```

Usage Notes

- The system must be configured for job queues for automatic synchronization. The jobs can be viewed using the USER_JOBS catalog views
- The configuration parameter AutoSyncPolicy can be set to choose an appropriate synchronization policy.

The synchronization can be based on one of the following:

Sync Basis	Description
SYNC_BY_PENDING_COUNT	The SYNC is triggered when the number of documents in the pending queue is greater than a threshold (See the MaxPendingCount configuration setting on page 122-5). The pending queue is polled at regular intervals (See the CheckPendingCountInterval configuration parameter on page 122-5) to determine if the number of documents exceeds the threshold.
SYNC_BY_TIME	The SYNC is triggered at regular intervals. (See the SyncInterval configuration parameter on page 122-5).
SYNC_BY_PENDING_COUNT_ AND_TIME	A combination of both of the preceding options.

CREATEDATASTOREPREF Procedure

This procedure creates a user datastore preference for the CONTEXT index on the XML DB hierarchy.

Syntax

```
DBMS_XDBT.CREATEDATASTOREPREF;
```

Usage Notes

- The name of the datastore preference can be modified; see the DatastorePref configuration setting.
- The default USER datastore procedure also filters the incoming document. The DBMS_XDBT package provides a set of configuration settings that control the filtering process.
- The SkipFilter_Types array contains a list of regular expressions. Documents with a mime type that matches one of these expressions are not indexed. Some of the properties of the document metadata, such as author, remain unindexed.
  - The NullFilter_Types array contains a list of regular expressions. Documents with a mime type that matches one of these expressions are not filtered; however, they are still indexed. This is intended to be used for documents that are text-based, such as HTML, XML and plain-text.
  - All other documents use the INSO filter through the IFILTER API.

CREATEFILTERPREF Procedure

This procedure creates a `NULL` filter preference for the `CONTEXT` index on the XML DB hierarchy.

Syntax

```
DBMS_XDBT.CREATEFILTERPREF;
```

Usage Notes

- The name of the filter preference can be modified; see `FilterPref` configuration setting.
- The `USER` datastore procedure filters the incoming document; see CREATEDATASTOREPREF Procedurefor more details.

CREATEINDEX Procedure

This procedure creates the `CONTEXT` index on the XML DB hierarchy.

Syntax

```
DBMS_XDBT.CREATEINDEX;
```

Usage Notes

- The name of the index can be changed; see the `IndexName` configuration setting.
- Set the `LogFile` configuration parameter to enable `ROWID` logging during index creation.
- Set the `IndexMemory` configuration parameter to determine the amount of memory that index creation, and later SYNCs, will use.

CREATELEXERPREF Procedure

This procedure creates a BASIC `lexer` preference for the `CONTEXT` index on the XML DB hierarchy.

Syntax

```
DBMS_XDBT.CREATELEXERPREF;
```

Usage Notes

- The name of the `lexer` preference can be modified; see `LexerPref` configuration setting. No other configuration settings are provided.
- `MultiLexer` preferences are not supported.
- Base letter translation is turned on by default.

CREATEPREFERENCES Procedure

This procedure creates a set of default preferences based on the configuration settings.

Syntax

```
DBMS_XDBT.CREATEPREFERENCES;
```

CREATESECTIONGROUPPREF Procedure

This procedure creates a section group for the CONTEXT index on the XML DB hierarchy.

Syntax

```
DBMS_XDBT.CREATESECTIONGROUPPREF;
```

Usage Notes

- The name of the section group can be changed; see the SectiongroupPref configuration setting.
- The HTML sectioner is used by default. No zone sections are created by default. If the vast majority of documents are XML, consider using the AUTO_SECTION_GROUP or the PATH_SECTION_GROUP; see the SectionGroup configuration setting.

CREATESTOPLISTPREF Procedure

This procedure creates a stoplist for the `CONTEXT` index on the XML DB hierarchy.

Syntax

```
DBMS_XDBT.CREATESTOPLISTPREF;
```

Usage Notes

- The name of the stoplist can be modified; see the `StoplistPref` configuration setting.
- Numbers are not indexed.
- The `StopWords` array is a configurable list of stopwords. These are meant to be stopwords in addition to the set of stopwords in `CTXSYS.DEFAULT_STOPLIST`.

CREATESTORAGEPREF Procedure

This procedure creates a `BASIC_STORAGE` preference for the `CONTEXT` index on the XML DB hierarchy.

Syntax

```
DBMS_XDBT.CREATESTORAGEPREF;
```

Usage Notes

- The name of the storage preference can be modified; see the `StoragePref` configuration setting.
- A tablespace can be specified for the tables and indexes comprising the `CONTEXT` index; see the `IndexTablespace` configuration setting.
- Prefix and Substring indexing are not turned on by default.
- The `I_INDEX_CLAUSE` uses key compression.

CREATEWORLDLISTPREF Procedure

This procedure creates a wordlist preference for the `CONTEXT` index on the XML DB hierarchy.

Syntax

```
DBMS_XDBT.CREATEWORDLISTPREF;
```

Usage Notes

- The name of the wordlist preference can be modified; see the `WordlistPref` configuration setting. No other configuration settings are provided.
- `FUZZY_MATCH` and `STEMMER` attributes are set to `AUTO` (auto-language detection)

DROPPREFERENCES Procedure

This procedure drops any previously created preferences for the `CONTEXT` index on the XML DB hierarchy.

Syntax

```
DBMS_XDBT.DROPPREFERENCES;
```

123

DBMS_XDBZ

The DBMS_XDBZ package controls the Oracle XML DB repository security, which is based on Access Control Lists (ACLs).

This chapter contains the following topics:

See Also: *Oracle XML DB Developer's Guide*

Using DBMS_XDBZ

This section contains topics which relate to using the `DBMS_XDBZ` package.

Constants

The DBMS_XDBZ package uses the constants shown in following tables.

Table 123–1 DBMS_XDBZ Constants - Name Format

Constant	Type	Value	Description
NAME_FORMAT_SHORT	PLS_INTEGER	1	DB user name or LDAP nickname
NAME_FORMAT_DISTINGUISHED	PLS_INTEGER	2	LDAP distinguished name

Table 123–2 DBMS_XDBZ Constants - Enable Option

Constant	Type	Value	Description
ENABLE_CONTENTS	PLS_INTEGER	1	Enables hierarchy for contents and is used by users when calling enable_hierarchy
ENABLE_RESMETADATA	PLS_INTEGER	2	Enables hierarchy for resource metadata, that is, this table will store schema based custom metadata for resources

Table 123–3 DBMS_XDBZ Constants - Enable Option Exercised

Constant	Type	Value	Description
IS_ENABLED_CONTENTS	PLS_INTEGER	1	If hierarchy was enabled for contents, that is, the ENABLE_HIERARCHY Procedurewas called with hierarchy_type as ENABLE_CONTENTS
IS_ENABLED_RESMETADATA	PLS_INTEGER	2	If hierarchy was enabled for resource metadata, that is, the ENABLE_HIERARCHY Procedure was called with hierarchy_type as ENABLE_RESMETADATA

Summary of DBMS_XDBZ Subprograms

Table 123–4 DBMS_XDBZ Package Subprograms

Method	Description
DISABLE_HIERARCHY Procedure on page 123-5	Disables repository support for the specified `XMLTYPE` table or view
ENABLE_HIERARCHY Procedure on page 123-6	Enables repository support for the specified `XMLType` table or view
GET_ACLOID Function on page 123-7	Retrieves the ACL Object ID for the specified resource
GET_USERID Function on page 123-8	Retrieves the user ID for the specified user
IS_HIERARCHY_ENABLED Function on page 123-9	Determines if repository support for the specified `XMLType` table or view is enabled
PURGELDAPCACHE Function on page 123-10	Purges the LDAP nickname cache

DISABLE_HIERARCHY Procedure

This procedure disables repository support for a particular XMLType table or view.

Syntax

```
DBMS_XDBZ.DISABLE_HIERARCHY(
   object_schema IN VARCHAR2,
   object_name   IN VARCHAR2);
```

Parameters

Table 123–5 DISABLE_HIERARCHY Procedure Parameters

Parameter	Description
object_schema	The schema name of the XMLType table or view
object_name	The name of the XMLType table or view

ENABLE_HIERARCHY Procedure

This procedure enables repository support for a particular XMLType table or view. This allows the use of a uniform ACL-based security model across all documents in the repository.

Syntax

```
DBMS_XDBZ.ENABLE_HIERARCHY(
   object_schema   IN   VARCHAR2,
   object_name     IN   VARCHAR2,
   hierarchy_type  IN   PLS_INTEGER := DBMS_XDBZ.ENABLE_CONTENTS);
```

Parameters

Table 123–6 ENABLE_HIERARCHY Procedure Parameters

Parameter	Description
object_schema	The schema name of the XMLType table or view
object_name	The name of the XMLType table or view
hierarchy_type	How to enable the hierarchy. ■ ENABLE_CONTENTS : enable hierarchy for contents, that is, this table will store contents of resources in the repository ■ ENABLE_RESMETADATA : enable hierarchy for resource metadata, that is, this table will store schema based custom metadata for resources If this subprogram is called on a table, another call will have no effect. Note that you cannot enable hierarchy for both contents and resource metadata.

GET_ACLOID Function

This function retrieves the ACL Object ID for the specified resource, if the repository path is known.

Syntax

```
DBMS_XDBZ.GET_ACLOID(
   aclpath   IN   VARCHAR2,
   acloid    OUT  RAW)
 RETURN BOOLEAN;
```

Parameters

Table 123–7 GET_ACLOID Function Parameters

Parameter	Description
`aclpath`	ACL resource path for the repository
`acloid`	The returned Object ID

Return Values

Returns `TRUE` if successful.

GET_USERID Function

This function retrieves the user ID for the specified user name. The local database is searched first, and if found, the `USERID` is returned in 4-byte database format. Otherwise, the LDAP directory is searched, if available, and if found, the `USERID` is returned in 4-byte database format.

Syntax

```
DBMS_XDBZ.GET_USERID(
   username IN  VARCHAR2,
   userid   OUT RAW,
   format   IN  BINARY_INTEGER := NAME_FORMAT_SHORT)
 RETURN BOOLEAN;
```

Parameters

Table 123–8 GET_USERID Function Parameters

Parameter	Description
`username`	Name of the database or LDAP user.
`userid`	Return parameter for the matching user id.
`format`	Format of the specified user name; valid options are: ■ `DBMS_XDBZ.NAME_FORMAT_SHORT` (default) -- DB user name or LDAP nickname ■ `DBMS_XDBZ.NAME_FORMAT_DISTINGUISHIED` -- LDAP distinguished name.

Return Values

Returns `TRUE` if successful.

IS_HIERARCHY_ENABLED Function

This function determines if repository support for the specified XMLType table or view is enabled.

Syntax

```
DBMS_XDBZ.IS_HIERARCHY_ENABLED(
   object_schema    IN  VARCHAR2,
   object_name      IN  VARCHAR2,
   hierarchy_type   IN  PLS_INTEGER := IS_ENABLED_CONTENTS)
 RETURN BOOLEAN;
```

Parameters

Table 123–9 IS_HIERARCHY_ENABLED Function Parameters

Parameter	Description
object_schema	The schema name of the XMLType table or view
object_name	The name of the XMLType table or view
hierarchy_type	The type of hierarchy to check for. ■ IS_ENABLED_CONTENTS : if hierarchy was enabled for contents, that is, the ENABLE_HIERARCHY Procedurewas called with hierarchy_type as ENABLE_CONTENTS ■ IS_ENABLED_RESMETADATA : if hierarchy was enabled for resource metadata, that is, the ENABLE_HIERARCHY Procedure was called with hierarchy_type as ENABLE_RESMETADATA

Return Values

Returns TRUE if the given XMLTYPE table or view has the XDB Hierarchy enabled with the specified type.

PURGELDAPCACHE Function

This function purges the LDAP nickname cache. Returns `TRUE` if successful.

Syntax

```
DBMS_XDBZ.PURGELDAPCACHE
 RETURN BOOLEAN;
```

124

DBMS_XMLDOM

The DBMS_XMLDOM package is used to access XMLType objects, and implements the Document Object Model (DOM), an application programming interface for HTML and XML documents.

> **See Also:** *Oracle XML Developer's Kit Programmer's Guide*

This chapter contains the following topics:

Using DBMS_XMLDOM

Overview

The Document Object Model (DOM) is an application programming interface (API) for HTML and XML documents. It defines the logical structure of documents, and the manner in which they are accessed and manipulated. In the DOM specification, the term "document" is used in the broad sense. XML is being increasingly used to represent many different kinds of information that may be stored in diverse systems. This information has been traditionally be seen as "data"; nevertheless, XML presents this data as documents, and the `DBMS_XMLDOM` package allows you access to both schema-based and non schema-based documents.

Note:

- Before database startup, the read-from and write-to directories in the `initialization.ORA` file must be specified; for example: `UTL_FILE_DIR=/mypath/insidemypath`.
- Read-from and write-to files must be on the server file system.

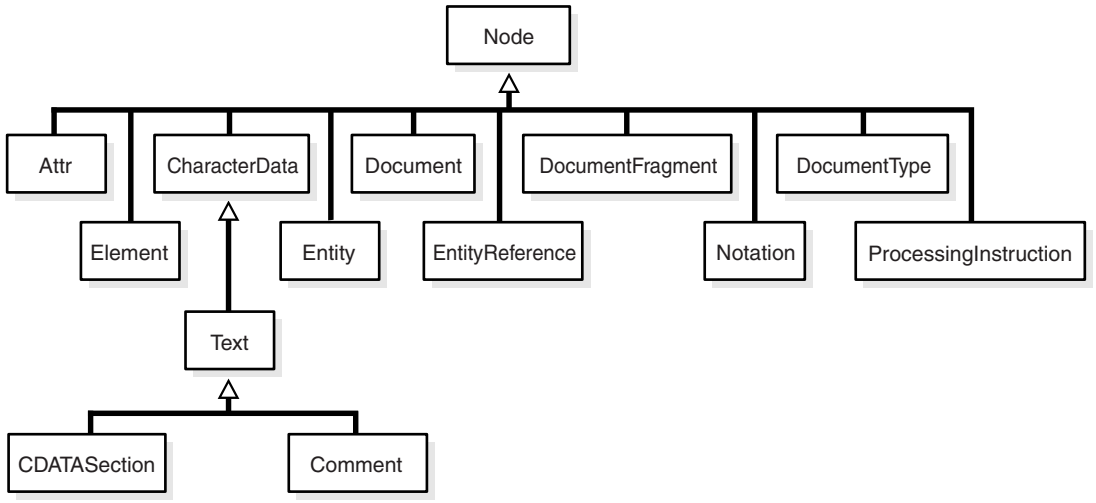
With DOM, anything found in an HTML or XML document can be accessed, changed, deleted, or added using the Document Object Model, with a few exceptions. In particular, the DOM interfaces for the XML internal and external subsets have not yet been specified.

One important objective of the W3C DOM specification is to provide a standard programming interface that can be used in a wide variety of environments, programming languages, and applications. Because the DOM standard is object-oriented while PL/SQL is essentially a procedural language, some changes had to be made:

- Various DOM interfaces such as `Node`, `Element`, and others have equivalent PL/SQL types `DOMNode`, `DOMElement`, respectively.
- Various `DOMException` codes such as `WRONG_DOCUMENT_ERR`, `HIERARCHY_REQUEST_ERR`, and others, have similarly named PL/SQL exceptions.
- Various DOM Node type codes such as `ELEMENT_NODE`, `ATTRIBUTE_NODE`, and others, have similarly named PL/SQL constants.
- Subprograms defined on a DOM type become functions or procedures that accept it as a parameter. For example, to perform APPENDCHILD Function on a `DOMNode n`, the APPENDCHILD FunctionPL/SQL function on page 124-40 is provided.
- To perform `setAttribute` on a `DOMElement elem`SETATTRIBUTE Procedures, use PL/SQL procedure on page 124-135.

DOM defines an inheritance hierarchy. For example, `Document`, `Element`, and `Attr` are defined to be subtypes of `Node` (see Figure 124–1). Thus, a method defined in the `Node` interface should be available in these as well. Since such inheritance is not supported in PL/SQL, it is implemented through direct invocation of the `MAKENODE` function. Calling `MAKENODE` on various DOM types converts these types into a `DOMNode`. The appropriate functions or procedures that accept `DOMNodes` can then be called to operate on these types. If, subsequently, type specific functionality is desired, the `DOMNode` can be converted back into the original type by the `make`*`XXX`* functions, where `DOM`*`XXX`* is the desired DOM type.

Figure 124–1 Inheritance Diagram for DOM Types



The implementation of this interface follows the REC-DOM-Level-1-19981001.

Constants

Defined constants of DBMS_XMLDOM are listed in Table 124–1.

Table 124–1 Defined Constants for DBMS_XMLDOM

Constant	Type	Value	Description
ELEMENT_NODE	PLS_INTEGER	1	The Node is an Element.
ATTRIBUTE_NODE	PLS_INTEGER	2	The Node is an Attribute.
TEXT_NODE	PLS_INTEGER	3	The Node is a Text node.
CDATA_SECTION_NODE	PLS_INTEGER	4	The Node is a CDataSection.
ENTITY_REFERENCE_NODE	PLS_INTEGER	5	The Node is an Entity Reference.
ENTITY_NODE	PLS_INTEGER	6	The Node is an Entity.
PROCESSING_INSTRUCTION_NODE	PLS_INTEGER	7	The Node is a Processing Instruction.
COMMENT_NODE	PLS_INTEGER	8	The Node is a Comment.
DOCUMENT_NODE	PLS_INTEGER	9	The Node is a Document.
DOCUMENT_TYPE_NODE	PLS_INTEGER	10	The Node is a Document Type Definition.
DOCUMENT_FRAGMENT_NODE	PLS_INTEGER	11	The Node is a Document fragment.
NOTATION_NODE	PLS_INTEGER	12	The Node is a Notation.

Types

The following types for DBMS_XMLDOM.DOMTYPE are defined in Table 124–2:

Table 124–2 XDB_XMLDOM Types

Type	Description
DOMATTR	Implements the DOM Attribute interface.
DOMCDATASECTION	Implements the DOM CDataSection interface.
DOMCHARACTERDATA	Implements the DOM Character Data interface.
DOMCOMMENT	Implements the DOM Comment interface.
DOMDOCUMENT	Implements the DOM Document interface.
DOMDOCUMENTFRAGMENT	Implements the DOM DocumentFragment interface.
DOMDOCUMENTTYPE	Implements the DOM Document Type interface.
DOMELEMENT	Implements the DOM Element interface.
DOMENTITY	Implements the DOM Entity interface.
DOMENTITYREFERENCE	Implements the DOM EntityReference interface.
DOMIMPLEMENTATION	Implements the DOM Implementation interface.
DOMNAMEDNODEMAP	Implements the DOM Named Node Map interface.
DOMNODE	Implements the DOM Node interface.
DOMNODELIST	Implements the DOM NodeList interface.
DOMNOTATION	Implements the DOM Notation interface.
DOMPROCESSINGINSTRUCTION	Implements the DOM Processing instruction interface.
DOMTEXT	Implements the DOM Text interface.

Exceptions

The exceptions listed in Table 124–3 are defined for DBMS_XMLDOM:

Table 124–3 Exceptions for DBMS_XMLDOM

Exception	Description
DOMSTRING_SIZE_ERR	If the specified range of text does not fit into a DOMString.
HIERARCHY_REQUEST_ERR	If any node is inserted somewhere it doesn't belong.
INDEX_SIZE_ERR	If index or size is negative, or greater than the allowed value.
INUSE_ATTRIBUTE_ERR	If an attempt is made to add an attribute that is already in use elsewhere.
INVALID_CHARACTER_ERR	If an invalid or illegal character is specified, such as in a name. See production 2 in the XML specification for the definition of a legal character, and production 5 for the definition of a legal name character.
NO_DATA_ALLOWED_ERROR	If data is specified for a node that does not support data.
NOT_FOUND_ERR	If an attempt is made to reference a node in a context where it does not exist.
NO_MODIFICATION_ALLOWED_ERR	If an attempt is made to modify an object where modifications are not allowed.
NOT_SUPPORTED_ERR	If the implementation does not support the requested type of object or operation.
WRONG_DOCUMENT_ERR	If a node is used in a different document than the one that created it (that doesn't support it).

Subprogram Groups

DBMS_XMLDOM subprograms are divided into groups according to W3C Interfaces.

DOMNode Subprograms

Table 124–4 Summary of DOMNode Subprograms; DBMS_XMLDOM

Subprogram	Description
ADOPTNODE Function on page 124-39	Adopts a node from another document.
APPENDCHILD Function on page 124-40	Appends a new child to the node.
CLONENODE Function on page 124-42	Clones the node.
FREENODE Procedure on page 124-57	Frees all resources associated with the node.
GETATTRIBUTES Function on page 124-60	Retrieves the attributes of the node.
GETCHILDNODES Function on page 124-61	Retrieves the children of the node.
GETEXPANDEDNAME Procedure and Functions on page 124-68	Retrieves the expanded name of the node.
GETFIRSTCHILD Function on page 124-69	Retrieves the first child of the node.
GETLASTCHILD Function on page 124-71	Retrieves the last child of the node.
GETLOCALNAME Procedure and Functions on page 124-73	Retrieves the local part of the qualified name.
GETNAMESPACE Procedure and Functions on page 124-76	Retrieves the node's namespace URI.
GETNEXTSIBLING Function on page 124-77	Retrieves the next sibling of the node.
GETNODENAME Function on page 124-78	Retrieves the Name of the Node.
GETNODETYPE Function on page 124-79	Retrieves the Type of the node.
GETNODEVALUE Function on page 124-80	Retrieves the Value of the Node.
GETOWNERDOCUMENT Function on page 124-84	Retrieves the owner document of the node.
GETPARENTNODE Function on page 124-86	Retrieves the parent of this node.
GETPREFIX Function on page 124-87	Retrieves the namespace prefix.
GETPREVIOUSSIBLING Function on page 124-88	Retrieves the previous sibling of the node.
GETSCHEMANODE Function on page 124-91	Retrieves the associated schema URI.
HASATTRIBUTES Function on page 124-100	Tests if the node has attributes.
HASCHILDNODES Function on page 124-101	Tests if the node has child nodes.
IMPORTNODE Function on page 124-103	Imports a node from another document.
INSERTBEFORE Function on page 124-104	Inserts a child before the reference child.
ISNULL Functions on page 124-106	Tests if the node is `NULL`
MAKEATTR Function on page 124-110	Casts the node to an Attribute.
MAKECDATASECTION Function on page 124-111	Casts the node to a CData Section.
MAKECHARACTERDATA Function on page 124-112	Casts the node to Character Data.
MAKECOMMENT Function on page 124-113	Casts the node to a Comment.
MAKEDOCUMENT Function on page 124-114	Casts the node to a DOM Document.

Table 124–4 (Cont.) Summary of DOMNode Subprograms; DBMS_XMLDOM

Subprogram	Description
MAKEDOCUMENTFRAGMENT Function on page 124-115	Casts the node to a DOM Document Fragment.
MAKEDOCUMENTTYPE Function on page 124-116	Casts the node to a DOM Document Type.
MAKEELEMENT Function on page 124-117	Casts the node to a DOM Element.
MAKEENTITY Function on page 124-118	Casts the node to a DOM Entity.
MAKEENTITYREFERENCE Function on page 124-119	Casts the node to a DOM Entity Reference.
MAKENOTATION Function on page 124-123	Casts the node to a DOM Notation.
MAKEPROCESSINGINSTRUCTION Function on page 124-124	Casts the node to a DOM Processing Instruction.
MAKETEXT Function on page 124-125	Casts the node to a DOM Text.
REMOVECHILD Function on page 124-130	Removes a specified child from a node.
REPLACECHILD Function on page 124-132	Replaces the old child with a new child.
SETNODEVALUE Procedure on page 124-140	Sets the Value of the node.
SETPREFIX Procedure on page 124-141	Sets the namespace prefix.
WRITETOBUFFER Procedures on page 124-147	Writes the contents of the node to a buffer.
WRITETOCLOB Procedures on page 124-148	Writes the contents of the node to a `CLOB`.
WRITETOFILE Procedures on page 124-149	Writes the contents of the node to a file.

DOMAttr Subprograms

Table 124–5 Summary of DOMAttr Subprograms; DBMS_XMLDOM

Method	Description
GETEXPANDEDNAME Procedure and Functions on page 124-68	Retrieves the expanded name of the attribute.
GETLOCALNAME Procedure and Functions on page 124-73	Retrieves the local name of the attribute.
GETNAME Functions on page 124-74	Retrieves the name of the attribute.
GETNAMESPACE Procedure and Functions on page 124-76	Retrieves the NS URI of the attribute.
GETOWNERELEMENT Function on page 124-85	Retrieves the Element node, parent of the attribute.
GETQUALIFIEDNAME Functions on page 124-90	Retrieves the Qualified Name of the attribute.
GETSPECIFIED Function on page 124-92	Tests if attribute was specified in the element.
GETVALUE Function on page 124-96	Retrieves the value of the attribute.
ISNULL Functions on page 124-106	Tests if the Attribute node is `NULL`.
MAKENODE Functions on page 124-120	Casts the Attribute to a node.
SETVALUE Procedure on page 124-143	Sets the value of the attribute.

DOMCDataSection Subprograms

Table 124–6 Summary of DOMCdata Subprograms; DBMS_XMLDOM

Method	Description
ISNULL Functions on page 124-106	Tests if the `CDataSection` is `NULL`.
MAKENODE Functions on page 124-120	Casts the `CDatasection` to a node.

DOMCharacterData Subprograms

Table 124–7 Summary of DOMCharacterData Subprograms; DBMS_XMLDOM

Method	Description
APPENDDATA Procedure on page 124-41	Appends the given data to the node data.
DELETEDATA Procedure on page 124-52	Deletes the data from the given offSets.
GETDATA Functions on page 124-63	Retrieves the data of the node.
GETLENGTH Functions on page 124-72	Retrieves the length of the data.
INSERTDATA Procedure on page 124-105	Inserts the data in the node at the given offSets.
ISNULL Functions on page 124-106	Tests if the CharacterData is `NULL`.
MAKENODE Functions on page 124-120	Casts the CharacterData to a node.
REPLACEDATA Procedure on page 124-133	Changes a range of characters in the node.
SETDATA Procedures on page 124-137	Sets the data to the node.
SUBSTRINGDATA Function on page 124-146	Retrieves the substring of the data.

DOMComment Subprograms

Table 124–8 Summary of DOMComment Subprograms; DBMS_XMLDOM

Method	Description
ISNULL Functions on page 124-106	Tests if the comment is `NULL`.
MAKENODE Functions on page 124-120	Casts the Comment to a node.

DOMDocument Subprograms

Table 124–9 Summary of DOMDocument Subprograms; DBMS_XMLDOM

Method	Description
CREATEATTRIBUTE Functions on page 124-43	Creates an Attribute.
CREATECDATASECTION Function on page 124-44	Creates a `CDataSection` node.
CREATECOMMENT Function on page 124-45	Creates a Comment node.
CREATEDOCUMENT Function on page 124-46	Creates a new Document.
CREATEDOCUMENTFRAGMENT Function on page 124-47	Creates a new Document Fragment.
CREATEELEMENT Functions on page 124-48	Creates a new Element.
CREATEENTITYREFERENCE Function on page 124-49	Creates an Entity reference.
CREATEPROCESSINGINSTRUCTION Function on page 124-50	Creates a Processing Instruction.
CREATETEXTNODE Function on page 124-51	Creates a Text node.
FREEDOCFRAG Procedure on page 124-55	Frees the document fragment.
FREEDOCUMENT Procedure on page 124-56	Frees the document.
GETDOCTYPE Function on page 124-64	Retrieves the DTD of the document.
GETDOCUMENTELEMENT Function on page 124-65	Retrieves the root element of the document.
GETELEMENTSBYTAGNAME Functions on page 124-66	Retrieves the elements in the by tag name.
GETIMPLEMENTATION Function on page 124-70	Retrieves the DOM implementation.
GETSTANDALONE Function on page 124-93	Retrieves the standalone property of the document.
GETVERSION Function on page 124-97	Retrieves the version of the document.
GETXMLTYPE Function on page 124-98	Retrieves the `XMLType` associated with the DOM Document.
ISNULL Functions on page 124-106	Tests if the document is `NULL`.
MAKENODE Functions on page 124-120	Casts the document to a node.
NEWDOMDOCUMENT Functions on page 124-126	Creates a new document.
SETDOCTYPE Procedure on page 124-138	Sets the DTD of the document.
SETSTANDALONE Procedure on page 124-142	Sets the standalone property of the document.
SETVERSION Procedure on page 124-144	Sets the version of the document.
WRITETOBUFFER Procedures on page 124-147	Writes the document to a buffer.
WRITETOCLOB Procedures on page 124-148	Writes the document to a `CLOB`.

Table 124–9 (Cont.) Summary of DOMDocument Subprograms; DBMS_XMLDOM

Method	Description
WRITETOFILE Procedures on page 124-149	Writes the document to a file.

DOMDocumentFragment Subprograms

Table 124–10 Summary of DOMDocumentFragment Subprograms; DBMS_XMLDOM

Method	Description
FREEDOCFRAG Procedure on page 124-55	Frees the specified document fragment.
ISNULL Functions on page 124-106	Tests if the DocumentFragment is `NULL`.
MAKENODE Functions on page 124-120	Casts the Document Fragment to a node.
WRITETOBUFFER Procedures on page 124-147	Writes the contents of a document fragment into a buffer.

DOMDocumentType Subprograms

Table 124–11 Summary of DOMDocumentType Subprograms; DBMS_XMLDOM

Method	Description
FINDENTITY Function on page 124-53	Finds the specified entity in the document type.
FINDNOTATION Function on page 124-54	Finds the specified notation in the document type.
GETENTITIES Function on page 124-67	Retrieves the nodemap of entities in the Document type.
GETNAME Functions on page 124-74	Retrieves the name of the Document type.
GETNOTATIONS Function on page 124-82	Retrieves the nodemap of the notations in the Document type.
GETPUBLICID Functions on page 124-89	Retrieves the public ID of the document type.
GETSYSTEMID Functions on page 124-94	Retrieves the system ID of the document type.
ISNULL Functions on page 124-106	Tests if the Document Type is `NULL`.
MAKENODE Functions on page 124-120	Casts the document type to a node.

DOMElement Subprograms

Table 124–12 Summary of DOMElement Subprograms; DBMS_XMLDOM

Method	Description
GETATTRIBUTE Functions on page 124-58	Retrieves the attribute node by name.
GETATTRIBUTENODE Functions on page 124-59	Retrieves the attribute node by name.
GETCHILDRENBYTAGNAME Functions on page 124-62	Retrieves children of the element by tag name.
GETELEMENTSBYTAGNAME Functions on page 124-66	Retrieves elements in the subtree by tagname.
GETEXPANDEDNAME Procedure and Functions on page 124-68	Retrieves the expanded name of the element.
GETLOCALNAME Procedure and Functions on page 124-73	Retrieves the local name of the element.
GETNAMESPACE Procedure and Functions on page 124-76	Retrieves the NS URI of the element.
GETQUALIFIEDNAME Functions on page 124-90	Retrieves the qualified name of the element.
GETTAGNAME Function on page 124-95	Retrieves the Tag name of the element.
HASATTRIBUTE Functions on page 124-99	Tests if an attribute exists .
ISNULL Functions on page 124-106	Tests if the Element is `NULL`.
MAKENODE Functions on page 124-120	Casts the Element to a node.
NORMALIZE Procedure on page 124-127	Normalizes the text children of the element.
REMOVEATTRIBUTE Procedures on page 124-128	Removes the attribute specified by the name.
REMOVEATTRIBUTENODE Function on page 124-129	Removes the attribute node in the element.
RESOLVENAMESPACEPREFIX Function on page 124-134	Resolve the prefix to a namespace URI.
SETATTRIBUTE Procedures on page 124-135	Sets the attribute specified by the name.
SETATTRIBUTENODE Functions on page 124-136	Sets the attribute node in the element.

DOMEntity Subprograms

Table 124–13 Summary of DOMEntity Subprograms; DBMS_XMLDOM

Method	Description
GETNOTATIONNAME Function on page 124-81	Retrieves the notation name of the entity.
GETPUBLICID Functions on page 124-89	Retrieves the public Id of the entity.
GETSYSTEMID Functions on page 124-94	Retrieves the system Id of the entity.
ISNULL Functions on page 124-106	Tests if the Entity is `NULL`.
MAKENODE Functions on page 124-120	Casts the Entity to a node.

DOMEntityReference Subprograms

Table 124–14 Summary of DOMEntityReference Subprograms; DBMS_XMLDOM

Method	Description
ISNULL Functions on page 124-106	Tests if the `DOMEntityReference` is `NULL`.
MAKENODE Functions on page 124-120	Casts the `DOMEntityReference` to `NULL`.

DOMImplementation Subprograms

Table 124–15 Summary of DOMImplementation Subprograms; DBMS_XMLDOM

Method	Description
ISNULL Functions on page 124-106	Tests if the `DOMImplementation` node is `NULL`.
HASFEATURE Function on page 124-102	Tests if the `DOMImplementation` implements a feature.

DOMNamedNodeMap Subprograms

Table 124–16 Summary of DOMNamedNodeMap Subprograms; DBMS_XMLDOM

Method	Description
GETLENGTH Functions on page 124-72	Retrieves the number of items in the map.
GETNAMEDITEM Function on page 124-75	Retrieves the item specified by the name.
ISNULL Functions on page 124-106	Tests if the `NamedNodeMap` is `NULL`.
ITEM Functions on page 124-109	Retrieves the item given the index in the map.
REMOVENAMEDITEM Function on page 124-131	Removes the item specified by name.
SETNAMEDITEM Function on page 124-139	Sets the item in the map specified by the name.

DOMNodeList Subprograms

Table 124–17 Summary of DOMNodeList Subprograms; DBMS_XMLDOM

Method	Description
GETLENGTH Functions on page 124-72	Retrieves the number of items in the list.
ISNULL Functions on page 124-106	Tests if the `NodeList` is `NULL`.
ITEM Functions on page 124-109	Retrieves the item given the index in the `NodeList`.

DOMNotation Subprograms

Table 124–18 Summary of DOMNotation Subprograms; DBMS_XMLDOM

Method	Description
GETPUBLICID Functions on page 124-89	Retrieves the public Id of the notation.
GETSYSTEMID Functions on page 124-94	Retrieves the system Id of the notation.
ISNULL Functions on page 124-106	Tests if the Notation is `NULL`.
MAKENODE Functions on page 124-120	Casts the notation to a node.

DOMProcessingInstruction Subprograms

Table 124–19 Summary of DOMProcessingInstruction Subprograms; DBMS_XMLDOM

Method	Description
GETDATA Functions on page 124-63	Retrieves the data of the processing instruction.
GETTARGET Function on page 124-83	Retrieves the target of the processing instruction.
ISNULL Functions on page 124-106	Tests if the Processing Instruction is `NULL`.
MAKENODE Functions on page 124-120	Casts the Processing Instruction to a node.
SETDATA Procedures on page 124-137	Sets the data of the processing instruction.

DOMText Subprograms

Table 124–20 Summary of DOMText Subprograms; DBMS_XMLDOM

Method	Description
ISNULL Functions on page 124-106	Tests if the text is `NULL`.
MAKENODE Functions on page 124-120	Casts the text to a node.
SPLITTEXT Function on page 124-145	Splits the contents of the text node into 2 text nodes.

Summary of DBMS_XMLDOM Subprograms

Table 124–21 Summary of DBMS_XMLDOM Package Subprogram

Subprogram	Description	Group
ADOPTNODE Function on page 124-39	Adopts a node from another document	DOMNode Subprograms on page 124-10
APPENDCHILD Function on page 124-40	Appends a new child to the node	DOMNode Subprograms on page 124-10
APPENDDATA Procedure on page 124-41	Appends the given data to the node data	DOMCharacterData Subprograms on page 124-14
CLONENODE Function on page 124-42	Clones the node	DOMNode Subprograms on page 124-10
CREATEATTRIBUTE Functions on page 124-43	Creates an Attribute	DOMDocument Subprograms on page 124-16
CREATECDATASECTION Function on page 124-44	Creates a `CDataSection` node	DOMDocument Subprograms on page 124-16
CREATECOMMENT Function on page 124-45	Creates a Comment node	DOMDocument Subprograms on page 124-16
CREATEDOCUMENT Function on page 124-46	Creates a new Document	DOMDocument Subprograms on page 124-16
CREATEDOCUMENTFRAGMENT Function on page 124-47	Creates a new Document Fragment	DOMDocument Subprograms on page 124-16
CREATEELEMENT Functions on page 124-48	Creates a new Element	DOMDocument Subprograms on page 124-16
CREATEENTITYREFERENCE Function on page 124-49	Creates an Entity reference	DOMDocument Subprograms on page 124-16
CREATEPROCESSINGINSTRUCTION Function on page 124-50	Creates a Processing Instruction	DOMDocument Subprograms on page 124-16
CREATETEXTNODE Function on page 124-51	Creates a Text node	DOMDocument Subprograms on page 124-16
DELETEDATA Procedure on page 124-52	Deletes the data from the given offSets	DOMCharacterData Subprograms on page 124-14
FINDENTITY Function on page 124-53	Finds the specified entity in the document type	DOMDocumentType Subprograms on page 124-19
FINDNOTATION Function on page 124-54	Finds the specified notation in the document type	DOMDocumentType Subprograms on page 124-19

Table 124–21 (Cont.) Summary of DBMS_XMLDOM Package Subprogram

Subprogram	Description	Group
FREEDOCFRAG Procedure on page 124-55	Frees the document fragment	DOMDocument Subprograms on page 124-16 and DOMDocumentFragment Subprograms on page 124-18
FREEDOCUMENT Procedure on page 124-56	Frees the document	DOMDocument Subprograms on page 124-16
FREENODE Procedure on page 124-57	Frees all resources associated with the node	DOMNode Subprograms on page 124-10
GETATTRIBUTE Functions on page 124-58	Retrieves the attribute node by name	DOMElement Subprograms on page 124-20
GETATTRIBUTENODE Functions on page 124-59	Retrieves the attribute node by name	DOMElement Subprograms on page 124-20
GETATTRIBUTES Function on page 124-60	Retrieves the attributes of the node	DOMNode Subprograms on page 124-10
GETCHILDNODES Function on page 124-61	Retrieves the children of the node	DOMNode Subprograms on page 124-10
GETCHILDRENBYTAGNAME Functions on page 124-62	Retrieves children of the element by tag name	DOMCharacterData Subprograms on page 124-14
GETDATA Functions on page 124-63	Retrieves ▪ the data of the node ▪ the data of the processing instruction	▪ DOMCharacterData Subprograms on page 124-14 ▪ DOMProcessingInstruction Subprograms on page 124-27
GETDOCTYPE Function on page 124-64	Retrieves the DTD of the document	DOMDocument Subprograms on page 124-16
GETDOCUMENTELEMENT Function on page 124-65	Retrieves the root element of the document	DOMDocument Subprograms on page 124-16
GETELEMENTSBYTAGNAME Functions on page 124-66	Retrieves ▪ the elements in the by tag name ▪ elements in the subtree by tagname	▪ DOMDocument Subprograms on page 124-16 ▪ DOMElement Subprograms on page 124-20
GETENTITIES Function on page 124-67	Retrieves the nodemap of entities in the Document type	DOMDocumentType Subprograms on page 124-19

Table 124–21 (Cont.) Summary of DBMS_XMLDOM Package Subprogram

Subprogram	Description	Group
GETEXPANDEDNAME Procedure and Functions on page 124-68	Retrieves ▪ the expanded name of the node ▪ the expanded name of the attribute ▪ the expanded name of the element	▪ DOMNode Subprograms on page 124-10 ▪ DOMAttr Subprograms on page 124-12 ▪ DOMElement Subprograms on page 124-20
GETFIRSTCHILD Function on page 124-69	Retrieves the first child of the node	DOMNode Subprograms on page 124-10
GETIMPLEMENTATION Function on page 124-70	Retrieves the DOM implementation	DOMDocument Subprograms on page 124-16
GETLASTCHILD Function on page 124-71	Retrieves the last child of the node	DOMNode Subprograms on page 124-10
GETLENGTH Functions on page 124-72	Retrieves ▪ the length of the data ▪ the number of items in the map ▪ the number of items in the list	▪ DOMCharacterData Subprograms on page 124-14 ▪ DOMNamedNodeMap Subprograms on page 124-24 ▪ DOMNodeList Subprograms on page 124-25
GETLOCALNAME Procedure and Functions on page 124-73	Retrieves ▪ the local part of the qualified name ▪ the local name of the attribute ▪ the local name of the element	▪ DOMNode Subprograms on page 124-10 ▪ DOMAttr Subprograms on page 124-12 ▪ DOMElement Subprograms on page 124-20
GETNAME Functions on page 124-74	Retrieves ▪ the name of the attribute ▪ the name of the Document type	▪ DOMAttr Subprograms on page 124-12 ▪ DOMDocumentType Subprograms on page 124-19
GETNAMEDITEM Function on page 124-75	Retrieves ▪ an item specified by name ▪ and namespace URI on page 124-24)	▪ DOMNamedNodeMap Subprograms on page 124-24 ▪ DOMNamedNodeMap Subprograms

Table 124–21 (Cont.) Summary of DBMS_XMLDOM Package Subprogram

Subprogram	Description	Group
GETNAMESPACE Procedure and Functions on page 124-76	Retrieves ■ the node's namespace URI ■ the NS URI of the attribute ■ the NS URI of the element	■ DOMNode Subprograms on page 124-10 ■ DOMAttr Subprograms on page 124-12 ■ DOMElement Subprograms on page 124-20
GETNEXTSIBLING Function on page 124-77	Retrieves the next sibling of the node	DOMNode Subprograms on page 124-10
GETNODENAME Function on page 124-78	Retrieves the Name of the Node	DOMNode Subprograms on page 124-10
GETNODETYPE Function on page 124-79	Retrieves the Type of the node	DOMNode Subprograms on page 124-10
GETNODEVALUE Function on page 124-80	Retrieves the Value of the Node	DOMNode Subprograms on page 124-10
GETNOTATIONNAME Function on page 124-81	Retrieves the notation name of the entity	DOMEntity Subprograms on page 124-21
GETNOTATIONS Function on page 124-82	Retrieves the nodemap of the notations in the Document type	DOMDocumentType Subprograms on page 124-19
GETTARGET Function on page 124-83	Retrieves the target of the processing instruction	DOMProcessingInstruction Subprograms on page 124-27
GETOWNERDOCUMENT Function on page 124-84	Retrieves the owner document of the node	DOMNode Subprograms on page 124-10
GETOWNERELEMENT Function on page 124-85	Retrieves the Element node, parent of the attribute	DOMAttr Subprograms on page 124-12
GETPARENTNODE Function on page 124-86	Retrieves the parent of this node	DOMNode Subprograms on page 124-10
GETPREFIX Function on page 124-87	Retrieves the namespace prefix)	DOMNode Subprograms on page 124-10
GETPREVIOUSSIBLING Function on page 124-88	Retrieves the previous sibling of the node	DOMNode Subprograms on page 124-10
GETPUBLICID Functions on page 124-89	Retrieves ■ the public ID of the document type ■ the public Id of the entity ■ the public Id of the notation	■ DOMDocumentType Subprograms on page 124-19 ■ DOMEntity Subprograms on page 124-21 ■ DOMNotation Subprograms on page 124-26
GETQUALIFIEDNAME Functions on page 124-90	Retrieves ■ the Qualified Name of the attribute ■ the qualified name of the element	■ DOMAttr Subprograms on page 124-12 ■ DOMElement Subprograms on page 124-20

Table 124–21 (Cont.) Summary of DBMS_XMLDOM Package Subprogram

Subprogram	Description	Group
GETSCHEMANODE Function on page 124-91	Retrieves the associated schema URI	DOMNode Subprograms on page 124-10
GETSPECIFIED Function on page 124-92	Tests if attribute was specified in the element.	DOMAttr Subprograms on page 124-12
GETSTANDALONE Function on page 124-93	Retrieves the standalone property of the document	DOMDocument Subprograms on page 124-16
GETSYSTEMID Functions on page 124-94	Retrieves ▪ the system ID of the document type ▪ the system Id of the entity ▪ the system Id of the notation	▪ DOMDocumentType Subprograms on page 124-19 ▪ DOMEntity Subprograms on page 124-21 ▪ DOMNotation Subprograms on page 124-26
GETTAGNAME Function on page 124-95	Retrieves the Tag name of the element	DOMElement Subprograms on page 124-20
GETVALUE Function on page 124-96	Retrieves the value of the attribute	DOMAttr Subprograms on page 124-12
GETVERSION Function on page 124-97	Retrieves the version of the document	DOMDocument Subprograms on page 124-16)
GETXMLTYPE Function on page 124-98	Retrieves the `XMLType` associated with the DOM Document	DOMDocument Subprograms on page 124-16
HASATTRIBUTES Function on page 124-100	Tests if the node has attributes	DOMNode Subprograms on page 124-10
HASATTRIBUTE Functions on page 124-99	Tests if an attribute exists	DOMElement Subprograms on page 124-20
HASCHILDNODES Function on page 124-101	Tests if the node has child nodes	DOMNode Subprograms on page 124-10
HASFEATURE Function on page 124-102	Tests if the `DOMImplementation` implements a feature	DOMImplementation Subprograms on page 124-23
IMPORTNODE Function on page 124-103	Imports a node from another document	DOMNode Subprograms on page 124-10
INSERTBEFORE Function on page 124-104	Inserts a child before the reference child	DOMNode Subprograms on page 124-10
INSERTDATA Procedure on page 124-105	Inserts the data in the node at the given offSets	DOMCharacterData Subprograms on page 124-14

Table 124–21 (Cont.) Summary of DBMS_XMLDOM Package Subprogram

Subprogram	Description	Group
ISNULL Functions on page 124-106	Tests ■ if the node is NULL ■ if the Attribute node is NULL ■ if the CDataSection is NULL ■ if the CharacterData is NULL ■ if the comment is NULL ■ if the document is NULL ■ if the DocumentFragment is NULL ■ if the Document Type is NULL ■ if the Element is NULL ■ if the Entity is NULL ■ if the DOMEntityReference is NULL ■ if the DOMImplementation node is NULL ■ if the NamedNodeMap is NULL ■ if the NodeList is NULL ■ if the Notation is NULL ■ if the Processing Instruction is NULL ■ if the text is NULL	■ DOMNode Subprograms on page 124-10 ■ DOMAttr Subprograms on page 124-12 ■ DOMCDataSection Subprograms on page 124-13 ■ DOMCharacterData Subprograms on page 124-14 ■ DOMComment Subprograms on page 124-15 ■ DOMDocument Subprograms on page 124-16 ■ DOMDocumentFragment Subprograms on page 124-18 ■ DOMDocumentType Subprograms on page 124-19 ■ DOMElement Subprograms on page 124-20 ■ DOMEntity Subprograms on page 124-21 ■ DOMEntityReference Subprograms on page 124-22 ■ DOMImplementation Subprograms on page 124-23 ■ DOMNamedNodeMap Subprograms on page 124-24 ■ DOMNodeList Subprograms on page 124-25 ■ DOMNotation Subprograms on page 124-26 ■ DOMProcessingInstruction Subprograms on page 124-27 ■ DOMText Subprograms on page 124-28

Table 124–21 (Cont.) Summary of DBMS_XMLDOM Package Subprogram

Subprogram	Description	Group
ITEM Functions on page 124-109	Retrieves ▪ the item given the index in the map ▪ the item given the index in the `NodeList`	▪ DOMNamedNodeMap Subprograms on page 124-24 ▪ DOMNodeList Subprograms on page 124-25
MAKEATTR Function on page 124-110	Casts the node to an Attribute	DOMNode Subprograms on page 124-10
MAKECDATASECTION Function on page 124-111	Casts the node to a CData Section	DOMNode Subprograms on page 124-10
MAKECHARACTERDATA Function on page 124-112	Casts the node to Character Data	DOMNode Subprograms on page 124-10
MAKECOMMENT Function on page 124-113	Casts the node to a Comment	DOMNode Subprograms on page 124-10
MAKEDOCUMENT Function on page 124-114	Casts the node to a DOM Document	DOMNode Subprograms on page 124-10
MAKEDOCUMENTFRAGMENT Function on page 124-115	Casts the node to a DOM Document Fragment	DOMNode Subprograms on page 124-10)
MAKEDOCUMENTTYPE Function on page 124-116	Casts the node to a DOM Document Type	DOMNode Subprograms on page 124-10
MAKEELEMENT Function on page 124-117	Casts the node to a DOM ElemenT	DOMNode Subprograms on page 124-10
MAKEENTITY Function on page 124-118	Casts the node to a DOM Entity	DOMNode Subprograms on page 124-10
MAKEENTITYREFERENCE Function on page 124-119	Casts the node to a DOM Entity Reference	DOMNode Subprograms on page 124-10

Table 124–21 (Cont.) Summary of DBMS_XMLDOM Package Subprogram

Subprogram	Description	Group
MAKENODE Functions on page 124-120	Casts ■ the Attribute to a node ■ the `CDatasection` to a node ■ the CharacterData to a node ■ the Comment to a node ■ the document to a node ■ the Document Fragment to a node ■ the document type to a node ■ the Element to a node ■ the Entity to a node ■ ■ the `DOMEntityReference` to `NULL` ■ the notation to a node ■ the Processing Instruction to a node ■ the text to a node	■ DOMAttr Subprograms on page 124-12 ■ DOMCDataSection Subprograms on page 124-13 ■ DOMCharacterData Subprograms on page 124-14 ■ DOMComment Subprograms on page 124-15 ■ DOMDocument Subprograms on page 16 ■ DOMDocumentFragment Subprograms on page 124-18 ■ DOMDocumentType Subprograms on page 124-19 ■ DOMElement Subprograms on page 124-20 ■ DOMEntity Subprograms on page 124-21 ■ DOMEntityReference Subprograms on page 124-22 ■ DOMNotation Subprograms on page 124-26 ■ DOMProcessingInstruction Subprograms on page 124-27 ■ DOMText Subprograms on page 124-28
MAKENOTATION Function on page 124-123	Casts the node to a DOM Notation	DOMNode Subprograms on page 124-10
MAKEPROCESSINGINSTRUCTION Function on page 124-124	Casts the node to a DOM Processing Instruction	DOMNode Subprograms on page 124-10
MAKETEXT Function on page 124-125	Casts the node to a DOM Text	DOMNode Subprograms on page 124-10
NEWDOMDOCUMENT Functions on page 124-126	Creates a new document	DOMDocument Subprograms on page 124-16
NORMALIZE Procedure on page 124-127	Normalizes the text children of the element	DOMElement Subprograms on page 124-20

Table 124–21 (Cont.) Summary of DBMS_XMLDOM Package Subprogram

Subprogram	Description	Group
REMOVEATTRIBUTE Procedures on page 124-128	Removes the attribute specified by the name	DOMElement Subprograms on page 124-20
REMOVEATTRIBUTENODE Function on page 124-129	Removes the attribute node in the element	DOMElement Subprograms on page 124-20
REMOVECHILD Function on page 124-130	Removes a specified child from a node	DOMNode Subprograms on page 124-10
REMOVENAMEDITEM Function on page 124-131	Removes the item specified by name	DOMNamedNodeMap Subprograms on page 124-24
REPLACECHILD Function on page 124-132	Replaces the old child with a new child	DOMNode Subprograms on page 124-10
REPLACEDATA Procedure on page 124-133	Changes a range of characters in the node	DOMCharacterData Subprograms on page 124-14
RESOLVENAMESPACEPREFIX Function on page 124-134	Resolve the prefix to a namespace URI	DOMElement Subprograms on page 124-20
SETATTRIBUTE Procedures on page 124-135	Sets the attribute specified by the name	DOMElement Subprograms on page 124-20
SETATTRIBUTENODE Functions on page 124-136	Sets the attribute node in the element	DOMElement Subprograms on page 124-20
SETDATA Procedures on page 124-137	Sets ■ the data to the node ■ the data of the processing instruction	■ DOMCharacterData Subprograms on page 124-14 ■ DOMProcessingInstruction Subprograms on page 124-27
SETDOCTYPE Procedure on page 124-138	Sets the DTD of the document.	DOMDocument Subprograms on page 124-16
SETNAMEDITEM Function on page 124-139	Sets the item in the map specified by the name	DOMNamedNodeMap Subprograms on page 124-24
SETNODEVALUE Procedure on page 124-140	Sets the Value of the node	DOMNode Subprograms on page 124-10
SETPREFIX Procedure on page 124-141	Sets the namespace prefix	DOMNode Subprograms on page 124-10
SETSTANDALONE Procedure on page 124-142	Sets the standalone property of the document	DOMDocument Subprograms on page 124-16
SETVALUE Procedure on page 124-143	Sets the value of the attribute	DOMAttr Subprograms on page 124-12
SETVERSION Procedure on page 124-144	Sets the version of the document	DOMDocument Subprograms on page 124-16

Table 124–21 (Cont.) Summary of DBMS_XMLDOM Package Subprogram

Subprogram	Description	Group
SPLITTEXT Function on page 124-145	Splits the contents of the text node into 2 text nodes	DOMText Subprograms on page 124-28
SUBSTRINGDATA Function on page 124-146	Retrieves the substring of the data	DOMCharacterData Subprograms on page 124-14
WRITETOBUFFER Procedures on page 124-147	Writes ■ the contents of the node to a buffer ■ the document to a buffer ■ the contents of a document fragment into a buffer	■ DOMNode Subprograms on page 124-10 ■ DOMDocument Subprograms on page 124-16 ■ DOMDocumentFragment Subprograms on page 124-18
WRITETOCLOB Procedures on page 124-148	Writes ■ the contents of the node to a `CLOB` ■ the document to a `CLOB`	■ DOMNode Subprograms on page 124-10 ■ DOMDocument Subprograms on page 124-16
WRITETOFILE Procedures on page 124-149	Writes ■ the contents of the node to a file ■ the document to a file	■ DOMNode Subprograms on page 124-10 ■ DOMDocument Subprograms on page 124-16

ADOPTNODE Function

This function adopts a node from another document, and returns this new node.

> **See Also:** DOMNode Subprograms on page 124-10 for other subprograms in this group

Syntax

```
DBMS_XMLDOM.ADOPTNODE(
   doc            IN   DOMDocument,
   importedNode   IN   DOMNode)
 RETURN DOMNODE;
```

Parameters

Table 124–22 ADOPTNODE Function Parameters

Parameter	Description
doc	Document that is adopting the node
importedNode	Node to adopt

APPENDCHILD Function

This function adds the node `newchild` to the end of the list of children of this node, and returns the newly added node. If the `newchild` is already in the tree, it is first removed.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.APPENDCHILD(
   n          IN    DOMNode,
   newchild   IN    DOMNode)
 RETURN DOMNODE;
```

Parameters

Table 124–23 APPENDCHILD Function Parameters

Parameter	Description
`n`	`DOMNode`
`newchild`	The child to be appended to the list of children of node `n`

APPENDDATA Procedure

This procedure appends the string to the end of the character data of the node. Upon success, data provides access to the concatenation of data and the specified string argument.

See Also: DOMCharacterData Subprograms on page 124-14

Syntax

```
DBMS_XMLDOM.APPENDDATA(
   cd      IN    DOMCHARACTERDATA,
   arg     IN    VARCHAR2);
```

Parameters

Table 124–24 APPENDDATA Procedure Parameters

Parameter	Description
cd	DOMCHARACTERDATA
arg	The data to append to the existing data

CLONENODE Function

This function returns a duplicate of this node, and serves as a generic copy constructor for nodes. The duplicate node has no parent, its parent node is NULL.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.CLONENODE(
   n       IN    DOMNODE,
   deep    IN    BOOLEAN)
 RETURN DOMNODE;
```

Parameters

Table 124–25 CLONENODE Function Parameters

Parameter	Description
n	DOMNODE
deep	Determines if children are to be cloned

Usage Notes

- Cloning an Element copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any text it contains unless it is a deep clone, since the text is contained in a child Text node.
- Cloning an Attribute directly, as opposed to be cloned as part of an Element cloning operation, returns a specified attribute (specified is TRUE).
- Cloning any other type of node simply returns a copy of this node.

CREATEATTRIBUTE Functions

This function creates a DOMATTR node.

See Also: DOMDocument Subprograms on page 124-16

Syntax

Creates a DOMATTR with the specified name:

```
DBMS_XMLDOM.CREATEATTRIBUTE(
   doc      IN    DOMDOCUMENT,
   name     IN    VARCHAR2)
 RETURN DOMATTR;
```

Creates a DOMATTR with the specified name and namespace URI:

```
DBMS_XMLDOM.CREATEATTRIBUTE(
   doc      IN    DOMDOCUMENT,
   qname     IN    VARCHAR2,
   ns       IN     VARCHAR2)
RETURN DOMATTR;
```

Parameters

Table 124–26 CREATEATTRIBUTE Function Parameters

Parameter	Description
doc	DOMDOCUMENT
qname	New attribute qualified name
ns	Namespace

CREATECDATASECTION Function

This function creates a DOMCDATASECTION node.

See Also: DOMDocument Subprograms on page 124-16

Syntax

```
DBMS_XMLDOM.CREATECDATASECTION(
   doc      IN      DOMDOCUMENT,
   data     IN      VARCHAR2)
 RETURN DOMCDATASECTION;
```

Parameters

Table 124–27 CREATECDATASECTION Function Parameters

Parameter	Description
doc	DOMDOCUMENT
data	Content of the DOMCDATASECTION node

CREATECOMMENT Function

This function creates a DOMCOMMENT node.

See Also: DOMDocument Subprograms on page 124-16

Syntax

```
DBMS_XMLDOM.CREATECOMMENT(
   doc      IN      DOMDOCUMENT,
   data     IN      VARCHAR2)
 RETURN DOMCOMMENT;
```

Parameters

Table 124–28 CREATECOMMENT Function Parameters

Parameter	Description
doc	DOMDOCUMENT
data	Content of the DOMComment node

CREATEDOCUMENT Function

This function creates a DOMDOCUMENT with specified namespace URI, root element name, DTD.

See Also: DOMDocument Subprograms on page 124-16

Syntax

```
DBMS_XMLDOM.CREATEDOCUMENT(
   namespaceURI      IN     VARCHAR2,
   qualifiedName     IN     VARCHAR2,
   doctype           IN     DOMTYPE := NULL)
 RETURN DOMDOCUMENT;
```

Parameters

Table 124–29 CREATEDOCUMENT Function Parameters

Parameter	Description
namespaceURI	Namespace URI
qualifiedName	Root element name
doctype	Document type

CREATEDOCUMENTFRAGMENT Function

This function creates a DOMDOCUMENTFRAGMENT.

See Also: DOMDocument Subprograms on page 124-16

Syntax

```
DBMS_XMLDOM.CREATEDOCUMENTFRAGMENT(
   doc       IN      DOMDOCUMENT)
 RETURN DOMDOCUMENTFRAGMENT;
```

Parameters

Table 124–30 CREATEDOCUMENTFRAGMENT Function Parameters

Parameter	Description
doc	DOMDocument

CREATEELEMENT Functions

This function creates a DOMELEMENT.

See Also: DOMDocument Subprograms on page 124-16

Syntax

Creates a DOMElement with specified name:

```
DBMS_XMLDOM.CREATEELEMENT(
   doc         IN      DOMDOCUMENT,
   tagName     IN      VARCHAR2)
 RETURN DOMELEMENT;
```

Creates a DOMElement with specified name and namespace URI:

```
DBMS_XMLDOM.CREATEELEMENT(
   doc         IN     DOMDOCUMENT,
   tagName     IN     VARCHAR2,
   ns          IN     VARCHAR2)
 RETURN DOMELEMENT;
```

Parameters

Table 124–31 CREATEELEMENT Function Parameters

Parameter	Description
doc	DOMDOCUMENT
tagName	Tagname for new DOMELEMENT
ns	Namespace

CREATEENTITYREFERENCE Function

This function creates a DOMENTITYREFERENCE node.

See Also: DOMDocument Subprograms on page 124-16

Syntax

```
DBMS_XMLDOM.CREATEENTITYREFERENCE(
   doc         IN     DOMDOCUMENT,
   name        IN     VARCHAR2)
 RETURN DOMENTITYREFERENCE;
```

Parameters

Table 124–32 CREATEENTITYREFERENCE Function Parameters

Parameter	Description
doc	DOMDOCUMENT
name	New entity reference name

CREATEPROCESSINGINSTRUCTION Function

This function creates a DOMPROCESSINGINSTRUCTION node.

See Also: DOMDocument Subprograms on page 124-16

Syntax

```
DBMS_XMLDOM.CREATEPROCESSINGINSTRUCTION(
   doc        IN      DOMDocument,
   target     IN      VARCHAR2,
   data       IN      VARCHAR2)
 RETURN DOMPROCESSINGINSTRUCTION;
```

Parameters

Table 124–33 CREATEPROCESSINGINSTRUCTION Function Parameters

Parameter	Description
doc	DOMDOCUMENT
target	Target of the new processing instruction
data	Content data of the new processing instruction

CREATETEXTNODE Function

This function creates a DOMTEXT node.

See Also: DOMDocument Subprograms on page 124-16

Syntax

```
DBMS_XMLDOM.CREATETEXTNODE(
   doc      IN     DOMDocument,
   data     IN     VARCHAR2)
 RETURN DOMTEXT;
```

Parameters

Table 124–34 CREATETEXTNODE Function Parameters

Parameter	Description
doc	DOMDOCUMENT
data	Content of the DOMText node

DELETEDATA Procedure

This procedure removes a range of characters from the node. Upon success, data and length reflect the change.

See Also: DOMCharacterData Subprograms on page 124-14

Syntax

```
DBMS_XMLDOM.DELETEDATA(
   cd        IN     DOMCHARACTERDATA,
   offset    IN     NUMBER,
   cnt       IN     NUMBER);
```

Parameters

Table 124–35 DELETEDATA PROCEDURE Parameters

Parameter	Description
cd	DOMCHARACTERDATA
offset	The offset from which to delete the data
cnt	The number of characters (starting from offset) to delete

FINDENTITY Function

This function finds an entity in the given DTD, and returns that entity if found.

See Also: DOMDocumentType Subprograms on page 124-19

Syntax

```
DBMS_XMLDOM.FINDENTITY(
   dt      IN     DOMDOCUMENTTYPE,
   name    IN     VARCHAR2,
   par     IN     BOOLEAN)
 RETURN  DOMENTITY;
```

Parameters

Table 124–36 FINDENTITY Function Parameters

Parameter	Description
dt	The DTD
name	Entity to find
par	Flag to indicate type of entity; TRUE for parameter entity and FALSE for normal entity

FINDNOTATION Function

This function finds the notation in the given DTD, and returns it, if found.

See Also: DOMDocumentType Subprograms on page 124-19

Syntax

```
DBMS_XMLDOM.FINDNOTATION(
   dt         IN     DOMDocumentType,
   name       IN     VARCHAR2)
 RETURN DOMNOTATION;
```

Parameters

Table 124–37 FINDNOTATION Function Parameters

Parameter	Description
dt	The DTD
name	The notation to find

FREEDOCFRAG Procedure

This procedure frees the specified document fragment.

> **See Also:** DOMDocument Subprograms on page 124-16 and DOMDocumentFragment Subprograms on page 124-18

Syntax

```
DBMS_XMLDOM.FREEDOCFRAG(
   df     IN     DOMDOCUMENTFRAGMENT);
```

Parameters

Table 124–38 FREEDOCFRAG Procedure Parameters

Parameter	Description
df	DOM document fragment

FREEDOCUMENT Procedure

This procedure frees DOMDOCUMENT object.

See Also: DOMDocument Subprograms on page 124-16

Syntax

```
DBMS_XMLDOM.FREEDOCUMENT(
   doc      IN      DOMDOCUMENT);
```

Parameters

Table 124–39 FREEDOCUMENT Procedure Parameters

Parameter	Description
doc	DOMDOCUMENT

FREENODE Procedure

This procedure frees all resources associated with a DOMNODE.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.FREENODE(
   n       IN      DOMNODE);
```

Parameters

Table 124–40 FREENODE Procedure Parameters

Parameter	Description
n	DOMNODE

GETATTRIBUTE Functions

This function returns the value of a DOMELEMENT's attribute by name.

See Also: DOMElement Subprograms on page 124-20

Syntax

Returns the value of a DOMELEMENT's attribute by name:

```
DBMS_XMLDOM.GETATTRIBUTE(
   elem       IN      DOMELEMENT,
   name       IN      VARCHAR2)
 RETURN VARCHAR2;
```

Returns the value of a DOMELEMENT's attribute by name and namespace URI:

```
DBMS_XMLDOM.GETATTRIBUTE(
   elem      IN     DOMELEMENT,
   name      IN     VARCHAR2,
   ns        IN     VARCHAR2)
 RETURN VARCHAR2;
```

Parameters

Table 124–41 GETATTRIBUTE Function Parameters

Parameter	Description
elem	The DOMELEMENT
name	Attribute name
ns	Namespace

GETATTRIBUTENODE Functions

This function returns an attribute node from the DOMELEMENT by name. The function is overloaded. The specific forms of functionality are described along with the syntax declarations.

See Also: DOMElement Subprograms on page 124-20

Syntax

Returns an attribute node from the DOMELEMENT by name:

```
DBMS_XMLDOM.GETATTRIBUTENODE(
   elem      IN     DOMElement,
   name      IN     VARCHAR2)
 RETURN DOMATTR;
```

Returns an attribute node from the DOMELEMENT by name and namespace URI:

```
DBMS_XMLDOM.GETATTRIBUTENODE(
   elem      IN     DOMElement,
   name      IN     VARCHAR2,
   ns        IN     VARCHAR2)
RETURN DOMATTR;
```

Parameters

Table 124–42 GETATTRIBUTENODE Function Parameters

Parameter	Description
elem	The DOMELEMENT
name	Attribute name; * matches any attribute
ns	Namespace

GETATTRIBUTES Function

This function retrieves a NAMEDNODEMAP containing the attributes of this node (if it is an Element) or NULL otherwise.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.GETATTRIBUTES(
   n       IN       DOMNode)
 RETURN DOMNAMEDNODEMAP;
```

Parameters

Table 124–43 GETATTRIBUTES Function Parameters

Parameter	Description
n	DOMNODE

GETCHILDNODES Function

This function retrieves a DOMNODELIST that contains all children of this node. If there are no children, this is a DOMNODELIST containing no nodes.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.GETCHILDNODES(
   n      IN    DOMNode)
 RETURN DOMNodeList;
```

Parameters

Table 124–44 GETCHILDNODES Function Parameters

Parameter	Description
n	DOMNODE

GETCHILDRENBYTAGNAME Functions

This function returns the children of the DOMELEMENT.

See Also: DOMElement Subprograms on page 124-20

Syntax

Returns children of the DOMELEMENT given the tag name:

```
DBMS_XMLDOM.GETCHILDRENBYTAGNAME(
   elem      IN      DOMElement,
   name      IN      VARCHAR2)
 RETURN DOMNODELIST;
```

Returns children of the DOMELEMENT given the tag name and namespace:

```
DBMS_XMLDOM.GETCHILDRENBYTAGNAME(
   elem      IN      DOMElement,
   name      IN      VARCHAR2,
   ns        IN      VARCHAR2)
RETURN DOMNODELIST;
```

Parameters

Table 124–45 GETCHILDRENBYTAGNAME Function Parameters

Parameter	Description
elem	The DOMELEMENT
name	Tag name
ns	Namespace

GETDATA Functions

This function is overloaded. The specific forms of functionality are described along with the syntax declarations.

Syntax

Gets the character data of the node that implements this interface (See Also: DOMCharacterData Subprograms on page 124-14):

```
DBMS_XMLDOM.GETDATA(
   cd      IN    DOMCHARACTERDATA)
 RETURN VARCHAR2;
```

Returns the content data of the `DOMProcessingInstruction` (See Also: DOMProcessingInstruction Subprograms on page 124-27):

```
DBMS_XMLDOM.GETDATA(
   pi      IN    DOMPROCESSINGINSTRUCTION)
 RETURN VARCHAR2;
```

Parameters

Table 124–46 GETDATA Function Parameters

Parameter	Description
`cd`	`DOMCHARACTERDATA`
`pi`	The `DOMPROCESSINGINSTRUCTION`

GETDOCTYPE Function

This function returns the DTD associated to the DOMDOCUMENT.

See Also: DOMDocument Subprograms on page 124-16

Syntax

```
DBMS_XMLDOM.GETDOCTYPE(
   doc       IN     DOMDOCUMENT)
RETURN DOMDOCUMENTTYPE;
```

Parameters

Table 124–47 GETDOCTYPE Function Parameters

Parameter	Description
doc	DOMDOCUMENT

GETDOCUMENTELEMENT Function

This function returns the root element of the DOMDOCUMENT.

See Also: DOMDocument Subprograms on page 124-16

Syntax

```
DBMS_XMLDOM.GETDOCUMENTELEMENT(
   doc       IN       DOMDOCUMENT)
 RETURN DOMELEMENT;
```

Parameters

Table 124–48 GETDOCUMENTELEMENT Function Parameters

Parameter	Description
doc	DOMDOCUMENT

GETELEMENTSBYTAGNAME Functions

This function is overloaded. The specific forms of functionality are described along with the syntax declarations.

Syntax

Returns a DOMNODELIST of all the elements with a given tagname (See Also: DOMDocument Subprograms on page 124-16):

```
DBMS_XMLDOM.GETELEMENTSBYTAGNAME(
   doc          IN      DOMDOCUMENT,
   tagname      IN      VARCHAR2)
 RETURN DOMNODELIST;
```

Returns the element children of the DOMELEMENT given the tag name (See Also: DOMElement Subprograms on page 124-20):

```
DBMS_XMLDOM.GETELEMENTSBYTAGNAME(
   elem       IN     DOMELEMENT,
   name       IN     VARCHAR2)
 RETURN DOMNODELIST;
```

Returns the element children of the DOMELEMENT given the tag name and namespace (See Also: DOMElement Subprograms on page 124-20):

```
DBMS_XMLDOM.GETELEMENTSBYTAGNAME(
   elem       IN     DOMELEMENT,
   name       IN     VARCHAR2,
   ns         IN     VARCHAR2)
 RETURN DOMNODELIST;
```

Parameters

Table 124–49 GETELEMENTSBYTAGNAME Function Parameters

Parameter	Description
doc	DOMDOCUMENT
tagname	Name of the tag to match on
elem	The DOMELEMENT
name	Tag name; using a wildcard(*) would match any tag
ns	Namespace

GETENTITIES Function

This function retrieves a DOMNAMEDNODEMAP containing the general entities, both external and internal, declared in the DTD.

See Also: DOMDocumentType Subprograms on page 124-19

Syntax

```
DBMS_XMLDOM.GETENTITIES(
   dt       IN     DOMDocumentType)
 RETURN DOMNAMEDNODEMAP;
```

Parameters

Table 124–50 GETENTITIES Function Parameters

Parameter	Description
dt	DOMDOCUMENTTYPE

GETEXPANDEDNAME Procedure and Functions

This subprogram is overloaded as a procedure and two functions. The specific forms of functionality are described along with the syntax declarations.

Syntax

Retrieves the expanded name of the `Node` if is in an `Element` or `Attribute` type; otherwise, returns `NULL` (See Also: DOMNode Subprograms on page 124-10)

```
DBMS_XMLDOM.GETEXPANDEDNAME(
   n       IN      DOMNODE
   data    OUT     VARCHAR);
```

Returns the expanded name of the `DOMAttr` (See Also: DOMAttr Subprograms on page 124-12):

```
DBMS_XMLDOM.GETEXPANDEDNAME(
   a       IN     DOMAttr)
 RETURN VARCHAR2;
```

Returns the expanded name of the `DOMElement` (See Also: DOMElement Subprograms on page 124-20):

```
DBMS_XMLDOM.GETEXPANDEDNAME(
   elem      IN    DOMELEMENT)
 RETURN VARCHAR2;
```

Parameters

Table 124–51 GETEXPANDEDNAME Procedure and Function Parameters

Parameter	Description
`n`	`DOMNODE`
`data`	Returned expanded name of the `Node`
`a`	`DOMATTR`
`elem`	`DOMELEMENT`

GETFIRSTCHILD Function

This function retrieves the first child of this node. If there is no such node, this returns NULL.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.GETFIRSTCHILD(
   n      IN      DOMNODE)
 RETURN DOMNODE;
```

Parameters

Table 124–52 GETFIRSTCHILD Function Parameters

Parameter	Description
n	DOMNODE

GETIMPLEMENTATION Function

This function returns the DOMIMPLEMENTATION object that handles this DOMDOCUMENT.

See Also: DOMDocument Subprograms on page 124-16

Syntax

```
DBMS_XMLDOM.GETIMPLEMENTATION(
   doc      IN     DOMDOCUMENT)
 RETURN DOMIMPLEMENTATION;
```

Parameters

Table 124–53 GETIMPLEMENTATION Function Parameters

Parameter	Description
doc	DOMDOCUMENT

GETLASTCHILD Function

This function retrieves the last child of this node. If there is no such node, this returns NULL.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.GETLASTCHILD(
   n      IN   DOMNODE)
 RETURN DOMNODE;
```

Parameters

Table 124–54 GETLASTCHILD Function Parameters

Parameter	Description
n	DOMNODE

GETLENGTH Functions

This function is overloaded. The specific forms of functionality are described along with the syntax declarations.

Syntax

Gets the number of characters in the data. This may have the value zero, because CharacterData nodes may be empty (See Also: DOMCharacterData Subprograms on page 124-14):

```
DBMS_XMLDOM.GETLENGTH(
   cd      IN      DOMCHARACTERDATA)
 RETURN NUMBER;
```

Gets the number of nodes in this map. The range of valid child node indexes is `0` to `length-1`, inclusive (See Also: DOMNamedNodeMap Subprograms on page 124-24):

```
DBMS_XMLDOM.GETLENGTH(
   nnm       IN      DOMNAMEDNODEMAP)
 RETURN NUMBER;
```

Gets the number of nodes in the list. The range of valid child node indexes is `0` to `length-1`, inclusive (See Also: DOMNodeList Subprograms on page 124-25):

```
DBMS_XMLDOM.GETLENGTH(
   nl      IN     DOMNODELIST)
 RETURN NUMBER;
```

Parameters

Table 124–55 GETLENGTH Function Parameters

Parameter	Description
cd	DOMCHARACTERDATA
nnm	DOMNAMEDNODEMAP
nl	DOMNODELIST

GETLOCALNAME Procedure and Functions

This function is overloaded as a procedure and two functions. The specific forms of functionality are described alongside the syntax declarations.

Syntax

Retrieves the local part of the node's qualified name (See Also: DOMNode Subprograms on page 124-10):

```
DBMS_XMLDOM.GETLOCALNAME(
   n       IN     DOMNODE,
   data    OUT    VARCHAR2);
```

Returns the local name of the `DOMAttr` (See Also: DOMAttr Subprograms on page 124-12):

```
DBMS_XMLDOM.GETLOCALNAME(
   a       IN     DOMATTR)
 RETURN VARCHAR2;
```

Returns the local name of the `DOMElement` (See Also: DOMElement Subprograms on page 124-20)

```
DBMS_XMLDOM.GETLOCALNAME(
   elem       IN     DOMELEMENT)
 RETURN VARCHAR2;
```

Parameters

Table 124–56 GETLOCALNAME Procedure and Function Parameters

Parameter	Description
`n`	`DOMNode`
`data`	Returned local name.
`a`	`DOMAttr`.
`elem`	`DOMElement`.

GETNAME Functions

This function is overloaded. The specific forms of functionality are described alongside the syntax declarations.

Syntax

Returns the name of this attribute (See Also: DOMAttr Subprograms on page 124-12):

```
DBMS_XMLDOM.GETNAME(
   a        IN     DOMATTR)
 RETURN VARCHAR2;
```

Retrieves the name of DTD, or the name immediately following the `DOCTYPE` keyword (See Also: DOMDocumentType Subprograms on page 124-19):

```
DBMS_XMLDOM.GETNAME(
   dt       IN     DOMDOCUMENTTYPE)
 RETURN VARCHAR2;
```

Parameters

Table 124–57 GETNAME Function Parameters

Parameter	Description
a	DOMATTR
dt	DOMDOCUMENTTYPE

GETNAMEDITEM Function

This function retrieves a node specified by name.

See Also: DOMNamedNodeMap Subprograms on page 124-24

Syntax

Retrieves a node specified by name:

```
DBMS_XMLDOM.GETNAMEDITEM(
   nnm    IN  DOMNAMEDNODEMAP,
   name   IN  VARCHAR2)
 RETURN DOMNODE;
```

Retrieves a node specified by name and namespace URI:

```
DBMS_XMLDOM.GETNAMEDITEM(
   nnm    IN  DOMNAMEDNODEMAP,
   name   IN  VARCHAR2,
   ns     IN  VARCHAR2)
 RETURN DOMNODE;
```

Parameters

Table 124–58 GETNAMEDITEM Function Parameters

Parameter	Description
nnm	DOMNAMEDNODEMAP
name	Name of the item to be retrieved
ns	Namespace

GETNAMESPACE Procedure and Functions

This subprogram is overloaded as a procedure and two functions. The specific forms of functionality are described alongside the syntax declarations.

Syntax

Retrieves the namespace URI associated with the node (See Also: DOMNode Subprograms on page 124-10):

```
DBMS_XMLDOM.GETNAMESPACE(
   n       IN     DOMNODE,
   data    OUT    VARCHAR2);
```

Retrieves the namespace of the DOMATTR (See Also: DOMAttr Subprograms on page 124-12):

```
DBMS_XMLDOM.GETNAMESPACE(
   a       IN     DOMATTR)
 RETURN VARCHAR2;
```

Retrieves the namespace of the DOMELEMENT (See Also: DOMElement Subprograms on page 124-20):

```
DBMS_XMLDOM.GETNAMESPACE(
   elem       IN     DOMELEMENT)
 RETURN VARCHAR2;
```

Parameters

Table 124–59 GETNAMESPACE Procedure and Function Parameters

Parameter	Description
n	DOMNODE
data	Returned namespace URI
a	DOMATTR
elem	DOMELEMENT

GETNEXTSIBLING Function

This function retrieves the node immediately following this node. If there is no such node, this returns `NULL`.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.GETNEXTSIBLING(
   n       IN     DOMNODE)
 RETURN DOMNode;
```

Parameters

Table 124–60 GETNEXTSIBLING Function Parameters

Parameter	Description
n	DOMNODE

GETNODENAME Function

This function gets the name of the node depending on its type.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.GETNODENAME(
   n        IN     DOMNODE)
 RETURN VARCHAR2;
```

Parameters

Table 124–61 GETNODENAME Function Parameters

Parameter	Description
n	DOMNODE

GETNODETYPE Function

This function retrieves a code representing the type of the underlying object.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.GETNODETYPE(
   n        IN     DOMNODE)
 RETURN NUMBER;
```

Parameters

Table 124–62 GETNODETYPE Function Parameters

Parameter	Description
n	DOMNODE

GETNODEVALUE Function

This function gets the value of this node, depending on its type.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.GETNODEVALUE(
   n        IN     DOMNODE)
 RETURN VARCHAR2;
```

Parameters

Table 124–63 GETNODEVALUE Function Parameters

Parameter	Description
n	DOMNODE

GETNOTATIONNAME Function

This function returns the notation name of the `DOMENTITY`.

See Also: DOMEntity Subprograms on page 124-21

Syntax

```
DBMS_XMLDOM.GETNOTATIONNAME(
   ent        IN     DOMENTITY)
 RETURN VARCHAR2;
```

Parameters

Table 124–64 GETNOTATIONNAME Function Parameters

Parameter	Description
ent	DOMENTITY

GETNOTATIONS Function

This function retrieves a DOMNAMEDNODEMAP containing the notations declared in the DTD.

See Also: DOMDocumentType Subprograms on page 124-19

Syntax

```
DBMS_XMLDOM.GETNOTATIONS(
   dt       IN     DOMDOCUMENTTYPE)
 RETURN DOMNAMEDNODEMAP;
```

Parameters

Table 124–65 GETNOTATIONS Function Parameters

Parameter	Description
dt	DOMDOCUMENTTYPE

GETTARGET Function

This function returns the target of the DOMPROCESSINGINSTRUCTION.

See Also: DOMProcessingInstruction Subprograms on page 124-27

Syntax

```
DBMS_XMLDOM.GETTARGET(
   pi       IN     DOMPROCESSINGINSTRUCTION)
 RETURN VARCHAR2;
```

Parameters

Table 124–66 GETTARGET Function Parameters

Parameter	Description
pi	DOMPROCESSINGINSTRUCTION

GETOWNERDOCUMENT Function

This function retrieves the Document object associated with this node. This is also the Document object used to create new nodes. When this node is a Document or a Document Type that is not used with any Document yet, this is NULL.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.GETOWNERDOCUMENT(
   n        IN     DOMNODE)
 RETURN DOMDOCUMENT;
```

Parameters

Table 124–67 GETOWNERDOCUMENT Function Parameters

Parameter	Description
n	DOMNODE

GETOWNERELEMENT Function

This function retrieves the Element node to which the specified Attribute is attached.

See Also: DOMAttr Subprograms on page 124-12

Syntax

```
DBMS_XMLDOM.GETOWNERELEMENT(
   a        IN     DOMATTR)
 RETURN DOMElement;
```

Parameters

Table 124–68 GETOWNERELEMENT Function Parameters

Parameter	Description
a	Attribute

GETPARENTNODE Function

This function retrieves the parent of this node. All nodes, except `Attr`, `Document`, `DocumentFragment`, `Entity`, and `Notation` may have a parent. However, if a node has just been created and not yet added to the tree, or if it has been removed from the tree, this is `NULL`.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.GETPARENTNODE(
   n        IN     DOMNODE)
 RETURN DOMNODE;
```

Parameters

Table 124–69 GETPARENTNODE Function Parameters

Parameter	Description
n	DOMNODE

GETPREFIX Function

This function retrieves the namespace prefix of the node.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.GETPREFIX(
   n        IN     DOMNODE)
 RETURN VARCHAR2;
```

Parameters

Table 124–70 GETPREFIX Function Parameters

Parameter	Description
n	DOMNODE

GETPREVIOUSSIBLING Function

This function retrieves the node immediately preceding this node. If there is no such node, this returns `NULL`.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.GETPREVIOUSSIBLING(
   n        IN     DOMNODE)
 RETURN DOMNODE;
```

Parameters

Table 124–71 GETPREVIOUSSIBLING Function Parameters

Parameter	Description
n	DOMNODE

GETPUBLICID Functions

This function is overloaded. The specific forms of functionality are described along with the syntax declarations.

Syntax

Returns the public identifier of the given DTD (See Also: DOMDocumentType Subprograms on page 124-19):

```
DBMS_XMLDOM.GETPUBLICID(
   dt        IN     DOMDOCUMENTTYPE)
 RETURN VARCHAR2;
```

Returns the public identifier of the DOMENTITY (See Also: DOMEntity Subprograms on page 124-21):

```
DBMS_XMLDOM.GETPUBLICID(
   ent       IN     DOMENTITY)
 RETURN VARCHAR2;
```

Returns the public identifier of the DOMNOTATION (See Also: DOMNotation Subprograms on page 124-26):

```
DBMS_XMLDOM.GETPUBLICID(
   n         IN     DOMNOTATION)
 RETURN VARCHAR2;
```

Parameters

Table 124–72 GETPUBLICID Function Parameters

Parameter	Description
dt	The DTD
ent	DOMENTITY
n	DOMNOTATION

GETQUALIFIEDNAME Functions

This function is overloaded. The specific forms of functionality are described along with the syntax declarations.

Syntax

Returns the qualified name of the `DOMATTR` (See Also: DOMAttr Subprograms on page 124-12):

```
DBMS_XMLDOM.GETQUALIFIEDNAME(
   a         IN     DOMATTR)
 RETURN VARCHAR2;
```

Returns the qualified name of the `DOMElement` (See Also: DOMElement Subprograms on page 124-20):

```
DBMS_XMLDOM.GETQUALIFIEDNAME(
   elem      IN     DOMELEMENT)
 RETURN VARCHAR2;
```

Parameters

Table 124–73 GETQUALIFIEDNAME Functions Parameters

Parameter	Description
a	DOMATTR
elem	DOMELEMENT

GETSCHEMANODE Function

This function retrieves the schema URI associated with the node.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.GETSCHEMANODE(
   n        IN     DOMNODE)
 RETURN DOMNODE;
```

Parameters

Table 124–74 GETSCHEMANODE Function Parameters

Parameter	Description
n	DOMNODE

GETSPECIFIED Function

If this attribute was explicitly given a value in the original document, this is true; otherwise, it is false.

See Also: DOMAttr Subprograms on page 124-12

Syntax

```
DBMS_XMLDOM.GETSPECIFIED(
   a       IN     DOMATTR)
 RETURN BOOLEAN;
```

Parameters

Table 124–75 GETSPECIFIED Function Parameters

Parameter	Description
a	DOMATTR

GETSTANDALONE Function

This function returns the standalone property associated with the DOMDOCUMENT.

See Also: DOMDocument Subprograms on page 124-16

Syntax

```
DBMS_XMLDOM.GETSTANDALONE(
   doc        IN      DOMDOCUMENT)
 RETURN VARCHAR2;
```

Parameters

Table 124–76 GETSTANDALONE Function Parameters

Parameter	Description
doc	DOMDOCUMENT.

GETSYSTEMID Functions

This function is overloaded. The specific forms of functionality are described along with the syntax declarations.

Syntax

Returns the system id of the given DTD (See Also: DOMDocumentType Subprograms on page 124-19):

```
DBMS_XMLDOM.GETSYSTEMID(
   dt       IN     DOMDOCUMENTTYPE)
 RETURN VARCHAR2;
```

Returns the system identifier of the DOMENTITY (See Also: DOMEntity Subprograms on page 124-21):

```
DBMS_XMLDOM.GETSYSTEMID(
   ent      IN     DOMENTITY)
 RETURN VARCHAR2;
```

Returns the system identifier of the DOMNOTATION (See Also: DOMNotation Subprograms on page 124-26):

```
DBMS_XMLDOM.GETSYSTEMID(
   n        IN     DOMNOTATION)
 RETURN VARCHAR2;
```

Parameters

Table 124–77 GETSYSTEMID Function Parameters

Parameter	Description
dt	The DTD.
ent	DOMEntity.
n	DOMNotation.

GETTAGNAME Function

This function returns the name of the DOMELEMENT.

See Also: DOMElement Subprograms on page 124-20

Syntax

```
DBMS_XMLDOM.GETTAGNAME(
   elem       IN     DOMELEMENT)
 RETURN VARCHAR2;
```

Parameters

Table 124–78 GETTAGNAME Function Parameters

Parameter	Description
elem	The DOMELEMENT

GETVALUE Function

This function retrieves the value of the attribute.

See Also: DOMAttr Subprograms on page 124-12

Syntax

```
DBMS_XMLDOM.GETVALUE(
   a        IN     DOMATTR)
 RETURN VARCHAR2;
```

Parameters

Table 124–79 GETVALUE Function Parameters

Parameter	Description
a	DOMATTR

GETVERSION Function

This function returns the version of the DOMDOCUMENT.

See Also: DOMDocument Subprograms on page 124-16

Syntax

```
DBMS_XMLDOM.GETVERSION(
   doc        IN     DOMDOCUMENT)
 RETURN VARCHAR2;
```

Parameters

Table 124–80 GETVERSION Function Parameters

Parameter	Description
doc	DOMDOCUMENT

GETXMLTYPE Function

This function returns the `XMLType` associated with the `DOMDOCUMENT`.

See Also: DOMDocument Subprograms on page 124-16

Syntax

```
DBMS_XMLDOM.GETXMLTYPE(
   doc        IN     DOMDOCUMENT)
 RETURN SYS.XMLTYPE;
```

Parameters

Table 124–81 GETXMLTYPE Function Parameters

Parameter	Description
`doc`	`DOMDOCUMENT`

HASATTRIBUTE Functions

Verifies whether an attribute has been defined for DOMELEMENT, or has a default value.

See Also: DOMElement Subprograms on page 124-20

Syntax

Verifies whether an attribute with the specified name has been defined for DOMElement:

```
DBMS_XMLDOM.HASATTRIBUTE(
   elem     IN  DOMELEMENT,
   name     IN  VARCHAR2)
 RETURN VARCHAR2;
```

Verifies whether an attribute with specified name and namespace URI has been defined for DOMELEMENT; namespace enabled:

```
DBMS_XMLDOM.HASATTRIBUTE(
   elem     IN  DOMELEMENT,
   name     IN  VARCHAR2,
   ns       IN  VARCHAR2)
 RETURN VARCHAR2;
```

Parameters

Table 124–82 HASATTRIBUTE Function Parameters

Parameter	Description
elem	The DOMELEMENT
name	Attribute name; * matches any attribute
ns	Namespace

HASATTRIBUTES Function

This function returns whether this node has any attributes.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.HASATTRIBUTES(
   n        IN     DOMNODE)
 RETURN BOOLEAN;
```

Parameters

Table 124–83 HASATTRIBUTES Function Parameters

Parameter	Description
n	DOMNODE

HASCHILDNODES Function

This function determines whether this node has any children.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.HASCHILDNODES(
   n        IN     DOMNODE)
 RETURN BOOLEAN;
```

Parameters

Table 124–84 HASCHILDNODES Function Parameters

Parameter	Description
n	DOMNODE

HASFEATURE Function

This function tests if the DOMIMPLEMENTATION implements a specific feature.

See Also: DOMImplementation Subprograms on page 124-23

Syntax

```
DBMS_XMLDOM.HASFEATURE(
   di       IN     DOMIMPLEMENTATION,
   feature  IN     VARCHAR2,
   version  IN     VARCHAR2)
 RETURN BOOLEAN;
```

Parameters

Table 124–85 HASFEATURE Function Parameters

Parameter	Description
di	DOMIMPLEMENTATION
feature	The feature to check for
version	The version of the DOM to check in

IMPORTNODE Function

This function imports a node from an external document and returns this new node.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.IMPORTNODE(
   doc             IN  DOMDOCUMENT,
   importedNode    IN  DOMNODE,
   deep            IN  BOOLEAN)
  RETURN DOMNODE;
```

Parameters

Table 124–86 IMPORTNODE Function Parameters

Parameter	Description
doc	Document from which the node is imported
importedNode	Node to import
deep	Setting for recursive import. ■ If this value is TRUE, the entire subtree of the node will be imported with the node. ■ If this value is FALSE, only the node itself will be imported.

INSERTBEFORE Function

This function inserts the node `newchild` before the existing child node `refchild`. If `refchild` is `NULL`, insert `newchild` at the end of the list of children.

If `newchild` is a `DOCUMENTFRAGMENT` object, all of its children are inserted, in the same order, before `refchild`. If the `newchild` is already in the tree, it is first removed.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.INSERTBEFORE(
   n          IN     DOMNODE,
   newchild   IN     DOMNODE,
   refchild   IN     DOMNODE)
  RETURN DOMNode;
```

Parameters

Table 124–87 INSERTBEFORE Function Parameters

Parameter	Description
`n`	`DOMNODE`
`newChild`	The child to be inserted in the `DOMNODE`
`refChild`	The reference node before which the `newchild` is to be inserted

INSERTDATA Procedure

This procedure inserts a string at the specified character offset.

See Also: DOMCharacterData Subprograms on page 124-14

Syntax

```
DBMS_XMLDOM.INSERTDATA(
   cd       IN     DOMCHARACTERDATA,
   offset   IN     NUMBER,
   arg      IN     VARCHAR2);
```

Parameters

Table 124–88 INSERTDATA Procedure Parameters

Parameter	Description
cd	DOMCHARACTERDATA
offset	The offset at which to insert the data
arg	The value to be inserted

ISNULL Functions

This function is overloaded. The specific forms of functionality are described along with the syntax declarations.

Syntax

Checks if the given DOMNODE is NULL. Returns TRUE if it is NULL, FALSE otherwise (See Also: DOMNode Subprograms on page 124-10):

```
DBMS_XMLDOM.ISNULL(
  n         IN     DOMNODE)
 RETURN BOOLEAN;
```

Checks that the given DOMATTR is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: DOMAttr Subprograms on page 124-12):

```
DBMS_XMLDOM.ISNULL(
   a        IN     DOMATTR)
 RETURN BOOLEAN;
```

Checks that the given DOMCDATASECTION is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: DOMCDataSection Subprograms on page 124-13):

```
DBMS_XMLDOM.ISNULL(
   cds      IN     DOMCDATASECTION)
 RETURN BOOLEAN;
```

Checks that the given DOMCHARACTERDATA is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: DOMCharacterData Subprograms on page 124-14):

```
DBMS_XMLDOM.ISNULL(
   cd       IN     DOMCHARACTERDATA)
 RETURN BOOLEAN;
```

Checks that the given DOMCOMMENT is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: DOMComment Subprograms on page 124-15):

```
DBMS_XMLDOM.ISNULL(
   com       IN     DOMCOMMENT)
 RETURN BOOLEAN;
```

Checks that the given DOMDOCUMENT is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: DOMDocument Subprograms on page 124-16):

```
DBMS_XMLDOM.ISNULL(
   doc       IN     DOMDOCUMENT)
 RETURN BOOLEAN;
```

Checks that the given DOMDOCUMENTFRAGMENT is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: DOMDocumentFragment Subprograms on page 124-18):

```
DBMS_XMLDOM.ISNULL(
   df       IN     DOMDOCUMENTFRAGMENT)
 RETURN BOOLEAN;
```

Checks that the given DOMDOCUMENTTYPE is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: DOMDocumentType Subprograms on page 124-19):

```
DBMS_XMLDOM.ISNULL(
   dt       IN     DOMDOCUMENTTYPE)
```

```
 RETURN BOOLEAN;
```

Checks that the given DOMELEMENT is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: DOMElement Subprograms on page 124-20):

```
DBMS_XMLDOM.ISNULL(
   elem     IN     DOMELEMENT)
 RETURN BOOLEAN;
```

Checks that the given DOMENTITY is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: DOMEntity Subprograms on page 124-21):

```
DBMS_XMLDOM.ISNULL(
   ent       IN     DOMENTITY)
 RETURN BOOLEAN;
```

Checks that the given DOMENTITYREFERENCE is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: DOMEntityReference Subprograms on page 124-22):

```
DBMS_XMLDOM.ISNULL(
   EREF       IN     DOMENTITYREFERENCE)
 RETURN BOOLEAN;
```

Checks that the DOMIMPLEMENTATION is NULL; returns TRUE if it is NULL (See Also: DOMImplementation Subprograms on page 124-23):

```
DBMS_XMLDOM.ISNULL(
   di       IN     DOMIMPLEMENTATION)
 RETURN BOOLEAN;
```

Checks that the given DOMNAMEDNODEMAP is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: DOMNamedNodeMap Subprograms on page 124-24):

```
DBMS_XMLDOM.ISNULL(
   nnm       IN     DOMNAMEDNODEMAP)
 RETURN BOOLEAN;
```

Checks that the given DOMNODELIST is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: DOMNodeList Subprograms on page 124-25):

```
DBMS_XMLDOM.ISNULL(
   nl       IN     DOMNODELIST)
 RETURN BOOLEAN;
```

Checks that the given DOMNOTATION is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: DOMNotation Subprograms on page 124-26):

```
DBMS_XMLDOM.ISNULL(
   n       IN     DOMNOTATION)
 RETURN BOOLEAN;
```

Checks that the given DOMPROCESSINGINSTRUCTION is NULL; returns TRUE if it is (See Also: DOMProcessingInstruction Subprograms on page 124-27):

```
DBMS_XMLDOM.ISNULL(
   pi       IN     DOMPROCESSINGINSTRUCTION)
 RETURN BOOLEAN;
```

Checks that the given DOMTEXT is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: DOMText Subprograms on page 124-28):

```
DBMS_XMLDOM.ISNULL(
   t       IN     DOMTEXT)
```

```
RETURN BOOLEAN;
```

Parameters

Table 124–89 ISNULL Function Parameters

Parameter	Description
n	DOMNODE to check
a	DOMATTR to check
cds	DOMCDATASECTION to check
cd	DOMCHARACTERDATA to check
com	DOMCOMMENT to check
doc	DOMDOCUMENT to check
dF	DOMDOCUMENTFRAGMENT to check
dt	DOMDOCUMENTTYPE to check
elem	DOMELEMENT to check
ent	DOMENTITY to check
eref	DOMENTITYREFERENCE to check
di	DOMIMPLEMENTATION to check
nnm	DOMNAMENODEMAP to check
nl	DOMNODELIST to check
n	DOMNOTATION to check
pi	DOMPROCESSINGINSTRUCTION to check
t	DOMTEXT to check

ITEM Functions

This function is overloaded. The specific forms of functionality are described along with the syntax declarations.

Syntax

Returns the item in the map which corresponds to the `INDEX` parameter. If `INDEX` is greater than or equal to the number of nodes in this map, this returns `NULL` (See Also: DOMNamedNodeMap Subprograms on page 124-24):

```
DBMS_XMLDOM.ITEM(
   nnm       IN     DOMNAMEDNODEMAP,
   index     IN     NUMBER)
 RETURN DOMNODE;
```

Returns the item in the collection which corresponds to the `INDEX` parameter. If index is greater than or equal to the number of nodes in the list, this returns `NULL` (See Also: DOMNodeList Subprograms on page 124-25):

```
DBMS_XMLDOM.ITEM(
   nl        IN     DOMNODELIST,
   index     IN     NUMBER)
 RETURN DOMNODE;
```

Parameters

Table 124–90 ITEM Function Parameters

Parameter	Description
`nnm`	`DOMNAMEDNODEMAP`
`index`	The index in the node map at which the item is to be retrieved
`nl`	`DOMNODELIST`
`index`	The index in the `NodeList` used to retrieve the item

MAKEATTR Function

This function casts a given DOMNODE to a DOMATTR, and returns the DOMATTR.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.MAKEATTR(
   n        IN     DOMNODE)
 RETURN DOMATTR;
```

Parameters

Table 124–91 MAKEATTR Function Parameters

Parameter	Description
n	DOMNODE to cast

MAKECDATASECTION Function

This function casts a given DOMNODE to a DOMCDATASECTION.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.MAKECDATASECTION(
   n        IN     DOMNODE)
 RETURN DOMCDATASECTION;
```

Parameters

Table 124–92 MAKECDATASECTION Function Parameters

Parameter	Description
n	DOMNODE to cast

MAKECHARACTERDATA Function

This function casts a given DOMNODE to a DOMCHARACTERDATA, and returns the DOMCHARACTERDATA.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.MAKECHARACTERDATA(
   n       IN     DOMNode)
 RETURN DOMCharacterData;
```

Parameters

Table 124–93 MAKECHARACTERDATA Function Parameters

Parameter	Description
n	DOMNODE to cast

MAKECOMMENT Function

This function casts a given DOMNODE to a DOMCOMMENT, and returns the DOMCOMMENT.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.MAKECOMMENT(
   n        IN      DOMNODE)
 RETURN DOMCOMMENT;
```

Parameters

Table 124–94 MAKECOMMENT Function Parameters

Parameter	Description
n	DOMNODE to cast

MAKEDOCUMENT Function

This function casts a given DOMNODE to a DOMDOCUMENT, and returns the DOMDOCUMENT.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.MAKEDOCUMENT(
   n        IN     DOMNODE)
 RETURN DOMDocument;
```

Parameters

Table 124–95 MAKEDOCUMENT Function Parameters

Parameter	Description
n	DOMNODE to cast

MAKEDOCUMENTFRAGMENT Function

This function casts a given DOMNODE to a DOMDOCUMENTFRAGMENT, and returns the DOMDOCUMENTFRAGMENT.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.MAKEDOCUMENTFRAGMENT(
   n        IN     DOMNODE)
 RETURN DOMDOCUMENTFRAGMENT;
```

Parameters

Table 124–96 MAKEDOCUMENTFRAGMENT Function Parameters

Parameter	Description
n	DOMNODE to cast

MAKEDOCUMENTTYPE Function

This function casts a given DOMNODE to a DOMDOCUMENTTYPE and returns the DOMDOCUMENTTYPE.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.MAKEDOCUMENTTYPE(
   n        IN     DOMNODE)
 RETURN DOMDOCUMENTTYPE;
```

Parameters

Table 124–97 MAKEDOCUMENTTYPE Function Parameters

Parameter	Description
n	DOMNODE to cast.

MAKEELEMENT Function

This function casts a given DOMNODE to a DOMELEMENT, and returns the DOMELEMENT.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.MAKEELEMENT(
   n        IN     DOMNODE)
 RETURN DOMELEMENT;
```

Parameters

Table 124–98 MAKEELEMENT Function Parameters

Parameter	Description
n	DOMNODE to cast

MAKEENTITY Function

This function casts a given DOMNODE to a DOMENTITY, and returns the DOMENTITY.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.MAKEENTITY(
   n        IN     DOMNODE)
 RETURN DOMENTITY;
```

Parameters

Table 124–99 MAKEENTITY Function Parameters

Parameter	Description
n	DOMNODE to cast

MAKEENTITYREFERENCE Function

This function casts a given DOMNODE to a DOMENTITYREFERENCE, and returns the DOMENTITYREFERENCE.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.MAKEENTITYREFERENCE(
   n       IN     DOMNODE)
 RETURN DOMENTITYREFERENCE;
```

Parameters

Table 124–100 MAKEENTITYREFERENCE Function Parameters

Parameter	Description
n	DOMNODE to cast

MAKENODE Functions

This function is overloaded. The specific forms of functionality are described along with the syntax declarations.

Syntax

Casts given DOMATTR to a DOMNODE, and returns the DOMNODE (See Also: DOMAttr Subprograms on page 124-12):

```
DBMS_XMLDOM.MAKENODE(
   a         IN     DOMATTR)
 RETURN DOMNODE;
```

Casts the DOMCDATASECTION to a DOMNODE, and returns that DOMNODE (See Also: DOMCDataSection Subprograms on page 124-13):

```
DBMS_XMLDOM.MAKENODE(
   cds       IN     DOMCDATASECTION)
 RETURN DOMNODE;
```

Casts the given DOMCHARACTERDATA as a DOMNODE, and returns that DOMNODE (See Also: DOMCharacterData Subprograms on page 124-14):

```
DBMS_XMLDOM.MAKENODE(
   cd        IN     DOMCHARACTERDATA)
 RETURN DOMNODE;
```

Casts the given DOMCOMMENT to a DOMNODE, and returns that DOMNODE (See Also: DOMComment Subprograms on page 124-15):

```
DBMS_XMLDOM.MAKENODE(
   com       IN     DOMCOMMENT)
 RETURN DOMNODE;
```

Casts the DOMDOCUMENT to a DOMNODE, and returns that DOMNODE (See Also: DOMDocument Subprograms on page 124-16):

```
DBMS_XMLDOM.MAKENODE(
   doc       IN     DOMDOCUMENT)
 RETURN DOMNODE;
```

Casts the given DOMDOCUMENTFRAGMENT to a DOMNODE, and returns that DOMNODE (See Also: DOMDocumentFragment Subprograms on page 124-18):

```
DBMS_XMLDOM.MAKENODE(
   df        IN     DOMDOCUMENTFRAGMENT)
 RETURN DOMNode;
```

Casts the given DOMDOCUMENTTYPE to a DOMNODE, and returns that DOMNODE (See Also: DOMDocumentType Subprograms on page 124-19):

```
DBMS_XMLDOM.MAKENODE(
   dt        IN     DOMDOCUMENTTYPE)
 RETURN DOMNODE;
```

Casts the given DOMELEMENT to a DOMNODE, and returns that DOMNODE (See Also: DOMElement Subprograms on page 124-20):

```
DBMS_XMLDOM.MAKENODE(
   elem        IN     DOMELEMENT)
```

```
 RETURN DOMNODE;
```

Casts given DOMENTITY to a DOMNODE, and returns that DOMNODE (See Also: DOMEntity Subprograms on page 124-21):

```
DBMS_XMLDOM.MAKENODE(
   ent       IN     DOMENTITY)
 RETURN DOMNODE;
```

Casts the DOMENTITYREFERENCE to a DOMNODE, and returns that DOMNODE (See Also: DOMEntityReference Subprograms on page 124-22):

```
DBMS_XMLDOM.MAKENODE(
   eref       IN     DOMENTITYREFERENCE)
 RETURN DOMNODE;
```

Casts the DOMNOTATION to a DOMNODE, and returns that DOMNODE (See Also: DOMNotation Subprograms on page 124-26):

```
DBMS_XMLDOM.MAKENODE(
   n       IN     DOMNOTATION)
 RETURN DOMNODE;
```

Casts the DOMPROCESSINGINSTRUCTION to a DOMNODE, and returns the DOMNODE (See Also: DOMProcessingInstruction Subprograms on page 124-27):

```
DBMS_XMLDOM.MAKENODE(
   pi       IN     DOMPROCESSINGINSTRUCTION)
 RETURN DOMNODE;
```

Casts the DOMTEXT to a DOMNODE, and returns that DOMNODE (See Also: DOMText Subprograms on page 124-28):

```
DBMS_XMLDOM.MAKENODE(
   t       IN     DOMTEXT)
 RETURN DOMNODE;
```

Parameters

Table 124–101 MAKENODE Function Parameters

Parameter	Description
a	DOMATTR to cast
cds	DOMCDATASECTION to cast
cd	DOMCHARACTERDATA to cast
com	DOMCOMMENT to cast
doc	DOMDOCUMENT to cast
df	DOMDOCUMENTFRAGMENT to cast
dt	DOMDOCUMENTTYPE to cast
elem	DOMELEMENT to cast
ent	DOMENTITY to cast
eref	DOMENTITYREFERENCE to cast
n	DOMNOTATION to cast
pi	DOMPROCESSINGINSTRUCTION to cast

Table 124–101 (Cont.) MAKENODE Function Parameters

Parameter	Description
t	DOMTEXT to cast

MAKENOTATION Function

This function casts a given DOMNODE to a DOMNOTATION, and returns the DOMNOTATION.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.MAKENOTATION(
   n       IN     DOMNODE)
 RETURN DOMNOTATION;
```

Parameters

Table 124–102 MAKENOTATION Function Parameters

Parameter	Description
n	DOMNODE to cast

MAKEPROCESSINGINSTRUCTION Function

This function casts a given DOMNODE to a DOMPROCESSINGINSTRUCTION, and returns the Domprocessinginstruction.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.MAKEPROCESSINGINSTRUCTION(
   n        IN     DOMNODE)
 RETURN DOMPROCESSINGINSTRUCTION;
```

Parameters

Table 124–103 MAKEPROCESSINGINSTRUCTION Function Parameters

Parameter	Description
n	DOMNODE to cast

MAKETEXT Function

This function casts a given DOMNODE to a DOMTEXT, and returns the DOMTEXT.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.MAKETEXT(
   n        IN     DOMNODE)
 RETURN DOMTEXT;
```

Parameters

Table 124–104 MAKETEXT Function Parameters

Parameter	Description
n	DOMNODE to cast

NEWDOMDOCUMENT Functions

This function returns a new DOMDOCUMENT instance.

See Also: DOMDocument Subprograms on page 124-16

Syntax

Returns a new DOMDOCUMENT instance:

```
DBMS_XMLDOM.NEWDOMDOCUMENT
 RETURN DOMDOCUMENT;
```

Returns a new DOMDOCUMENT instance created from the specified XMLType object:

```
DBMS_XMLDOM.NEWDOMDOCUMENT(
   xmldoc    IN SYS.XMLTYPE)
 RETURN DOMDOCUMENT;
```

Returns a new DOMDOCUMENT instance created from the specified CLOB:

```
DBMS_XMLDOM.NEWDOMDOCUMENT(
   cl       IN    CLOB)
 RETURN DOMDOCUMENT;
```

Parameters

Table 124–105 NEWDOMDOCUMENT Function Parameters

Parameter	Description
xmldoc	XMLType source for the DOMDOCUMENT
cl	CLOB source for the DOMDOCUMENT

NORMALIZE Procedure

This procedure normalizes the text children of the DOMELEMENT.

See Also: DOMElement Subprograms on page 124-20

Syntax

```
DBMS_XMLDOM.NORMALIZE(
   elem       IN     DOMELEMENT);
```

Parameters

Table 124–106 NORMALIZE Procedure Parameters

Parameter	Description
elem	The DOMELEMENT

REMOVEATTRIBUTE Procedures

This procedure removes an attribute from the DOMELEMENT by name.

See Also: DOMElement Subprograms on page 124-20

Syntax

Removes the value of a DOMELEMENT's attribute by name:

```
DBMS_XMLDOM.REMOVEATTRIBUTE(
   elem     IN    DOMELEMENT,
   name     IN    VARCHAR2);
```

Removes the value of a DOMELEMENT's attribute by name and namespace URI.

```
DBMS_XMLDOM.REMOVEATTRIBUTE(
   elem     IN    DOMELEMENT,
   name     IN    VARCHAR2,
   ns       IN    VARCHAR2);
```

Parameters

Table 124–107 REMOVEATTRIBUTE Procedure Parameters

Parameter	Description
elem	The DOMELEMENT
name	Attribute name
ns	Namespace

REMOVEATTRIBUTENODE Function

This function removes the specified attribute node from the DOMELEMENT. The method returns the removed node.

See Also: DOMElement Subprograms on page 124-20

Syntax

```
DBMS_XMLDOM.REMOVEATTRIBUTENODE(
   elem       IN     DOMELEMENT,
   oldAttr    IN     DOMATTR)
  RETURN DOMAttr;
```

Parameters

Table 124–108 REMOVEATTRIBUTENODE Function Parameters

Parameter	Description
elem	The DOMELEMENT.
oldAttr	The old DOMATTR.

REMOVECHILD Function

This function removes the child node indicated by `oldchild` from the list of children, and returns it.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.REMOVECHILD(
   n          IN     DOMNode,
   oldchild   IN     DOMNode)
 RETURN DOMNODE;
```

Parameters

Table 124–109 REMOVECHILD Function Parameters

Parameter	Description
`n`	`DOMNODE`
`oldCHild`	The child of the node `n` to be removed

REMOVENAMEDITEM Function

This function removes from the map a node specified by name, and returns this node. When this map contains the attributes attached to an element, if the removed attribute is known to have a default value, an attribute immediately appears containing the default value as well as the corresponding namespace URI, local name, and prefix when applicable.

See Also: DOMNamedNodeMap Subprograms on page 124-24

Syntax

Removes a node specified by name:

```
DBMS_XMLDOM.REMOVENAMEDITEM(
   nnm      IN     DOMNamedNodeMap,
   name     IN     VARCHAR2)
 RETURN DOMNode;
```

Removes a node specified by name and namespace URI:

```
DBMS_XMLDOM.REMOVENAMEDITEM(
   nnm      IN     DOMNamedNodeMap,
   name     IN     VARCHAR2,
   ns       IN     VARCHAR2)
 RETURN DOMNode;
```

Parameters

Table 124–110 REMOVENAMEDITEM Function Parameters

Parameter	Description
nnm	DOMNamedNodeMap
name	The name of the item to be removed from the map
ns	Namespace

REPLACECHILD Function

This function replaces the child node `oldchild` with `newchild` in the list of children, and returns the `oldchild` node. If `newchild` is a `DocumentFragment` object, `oldchild` is replaced by all of the `DocumentFragment` children, which are inserted in the same order. If the `newchild` is already in the tree, it is first removed.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.REPLACECHILD(
   n           IN     DOMNode,
   newchild    IN     DOMNode,
   oldchild    IN     DOMNode)
 RETURN DOMNode;
```

Parameters

Table 124–111 REPLACECHILD Function Parameters

Parameter	Description
`n`	`DOMNode`
`newchild`	The new child which is to replace the old child
`oldchild`	The child of the node n which is to be replaced

REPLACEDATA Procedure

This procedure changes a range of characters in the node. Upon success, data and length reflect the change.

See Also: DOMCharacterData Subprograms on page 124-14

Syntax

```
DBMS_XMLDOM.REPLACEDATA(
   cd        IN     DOMCHARACTERDATA,
   offset    IN     NUMBER,
   cnt       IN     NUMBER,
   arg       IN     VARCHAR2);
```

Parameters

Table 124–112 REPLACEDATA Procedure Parameters

Parameter	Description
cd	DOMCHARACTERDATA
offset	The offset at which to replace
cnt	The number of characters to replace
arg	The value to replace with

RESOLVENAMESPACEPREFIX Function

This function resolves the given namespace prefix, and returns the resolved namespace.

See Also: DOMElement Subprograms on page 124-20

Syntax

```
DBMS_XMLDOM.RESOLVENAMESPACEPREFIX(
   elem       IN     DOMELEMENT,
   prefix     IN     VARCHAR2)
 RETURN VARCHAR2;
```

Parameters

Table 124–113 RESOLVENAMESPACEPREFIX Function Parameters

Parameter	Description
elem	The DOMELEMENT
prefix	Namespace prefix

SETATTRIBUTE Procedures

Sets the value of a DOMELEMENT's attribute by name.

See Also: DOMElement Subprograms on page 124-20

Syntax

Sets the value of a DOMELEMENT's attribute by name:

```
DBMS_XMLDOM.SETATTRIBUTE(
   elem    IN  DOMELEMENT,
   name    IN  VARCHAR2,
   value   IN  VARCHAR2);
```

Sets the value of a DOMElement's attribute by name and namespace URI:

```
DBMS_XMLDOM.SETATTRIBUTE(
   elem    IN  DOMELEMENT,
   name    IN  VARCHAR2,
   value   IN  VARCHAR2,
   ns      IN  VARCHAR2);
```

Parameters

Table 124–114 SETATTRIBUTE Function Parameters

Parameter	Description
elem	The DOMELEMENT
name	Attribute name
value	Attribute value
ns	Namespace

SETATTRIBUTENODE Functions

Adds a new attribute node to the DOMELEMENT.

See Also: DOMElement Subprograms on page 124-20

Syntax

Adds a new attribute node to the DOMELEMENT:

```
DBMS_XMLDOM.SETATTRIBUTENODE(
   elem      IN  DOMELEMENT,
   newAttr   IN  DOMATTR)
 RETURN DOMATTR;
```

Adds a new attribute node to the DOMElement; namespace enabled:

```
DBMS_XMLDOM.SETATTRIBUTENODE(
   elem      IN  DOMELEMENT,
   newAttr   IN  DOMATTR,
   ns        IN  VARCHAR2)
 RETURN DOMATTR;
```

Parameters

Table 124–115 SETATTRIBUTENODE Function Parameters

Parameter	Description
elem	The DOMELEMENT
newAttr	The new DOMATTR
ns	The namespace

SETDATA Procedures

This procedure is overloaded. The specific forms of functionality are described along with the syntax declarations.

Syntax

Sets the character data of the node that implements this interface (See Also: DOMCharacterData Subprograms on page 124-14):

```
DBMS_XMLDOM.SETDATA(
   cd       IN     DOMCHARACTERDATA,
   data     IN     VARCHAR2);
```

Sets the content data of the DOMPROCESSINGINSTRUCTION (See Also: DOMProcessingInstruction Subprograms on page 124-14):

```
DBMS_XMLDOM.SETDATA(
   pi       IN     DOMPROCESSINGINSTRUCTION,
   data     IN     VARCHAR2);
```

Parameters

Table 124–116 SETDATA Procedure Parameters

Parameter	Description
cd	DOMCHARACTERDATA
data	The data to which the node is set
pi	DOMPROCESSINGINSTRUCTION
data	New processing instruction content data

SETDOCTYPE Procedure

Given a DOM document, this procedure creates a new DTD with the given name, system id and public id and sets it in the document. This DTD can later be retrieved using the GETDOCTYPE Function.

Syntax

```
DBMS_XMLDOM.SETDOCTYPE(
  doc     IN   DOMDocument,
  name    IN   VARCHAR2,
  sysid   IN   VARCHAR2,
  pubid   IN   VARCHAR2);
```

Parameters

Table 124–117 SETDOCTYPE Procedure Parameters

Parameter	Description
doc	The document whose DTD has to be set
name	The name that the doctype needs to be initialized with
sysid	The system ID that the doctype needs to be initialized with
pubid	The public ID that the doctype needs to be initialized with

SETNAMEDITEM Function

This function adds a node using its NodeName attribute. If a node with that name is already present in this map, it is replaced by the new one. The old node is returned on replacement; if no replacement is made, NULL is returned.

As the NodeName attribute is used to derive the name under which the node must be stored, multiple nodes of certain types, those that have a "special" string value, cannot be stored because the names would clash. This is seen as preferable to allowing nodes to be aliased.

See Also: DOMNamedNodeMap Subprograms on page 124-24

Syntax

Adds a node using its NodeName attribute:

```
DBMS_XMLDOM.SETNAMEDITEM(
   nnm      IN     DOMNAMEDNODEMAP,
   arg      IN     DOMNODE)
 RETURN DOMNode;
```

Adds a node using its NodeName attribute and namespace URI:

```
DBMS_XMLDOM.SETNAMEDITEM(
   nnm      IN     DOMNAMEDNODEMAP,
   arg       IN    DOMNODE,
   ns       IN     VARCHAR2)
 RETURN DOMNode;
```

Parameters

Table 124–118 SETNAMEDITEM Function Parameters

Parameter	Description
nnm	DOMNAMEDNODEMAP
arg	The Node to be added using its NodeName attribute
ns	Namespace

SETNODEVALUE Procedure

This procedure sets the value of this node, depending on its type. When it is defined to be NULL, setting it has no effect.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.SETNODEVALUE(
   n         IN     DOMNODE,
   nodeValue IN     VARCHAR2);
```

Parameters

Table 124–119 SETNODEVALUE Procedure Parameters

Parameter	Description
n	DOMNode
nodeValue	The value to which node is set

SETPREFIX Procedure

This procedure sets the namespace prefix for this node to the given value.

See Also: DOMNode Subprograms on page 124-10

Syntax

```
DBMS_XMLDOM.SETPREFIX(
   n       IN     DOMNODE,
   prefix  IN     VARCHAR2);
```

Parameters

Table 124–120 SETPREFIX Procedure Parameters

Parameter	Description
n	DOMNODE
prefix	The value for the namespace prefix of the node

SETSTANDALONE Procedure

This procedure sets the standalone property of the DOMDOCUMENT.

See Also: DOMDocument Subprograms on page 124-16

Syntax

```
DBMS_XMLDOM.SETSTANDALONE(
   doc          IN     DOMDOCUMENT,
   newvalue     IN     VARCHAR2);
```

Parameters

Table 124–121 SETSTANDALONE Procedure Parameters

Parameter	Description
doc	DOMDOCUMENT
newvalue	Value of the standalone property of the document

SETVALUE Procedure

This procedure sets the value of the attribute.

See Also: DOMAttr Subprograms on page 124-12

Syntax

```
DBMS_XMLDOM.SETVALUE(
   a       IN     DOMATTR,
   value   IN     VARCHAR2);
```

Parameters

Table 124–122 SETVALUE Procedure Parameters

Parameter	Description
a	DOMATTR
value	The value to which to set the attribute

SETVERSION Procedure

This procedure sets the version of the DOMDOCUMENT.

See Also: DOMDocument Subprograms on page 124-16

Syntax

```
DBMS_XMLDOM.SETVERSION(
   doc         IN     DOMDOCUMENT,
   version     IN     VARCHAR2);
```

Parameters

Table 124–123 SETVERSION Procedure Parameters

Parameter	Description
doc	DOMDOCUMENT
version	The version of the document

SPLITTEXT Function

This function breaks this DOMTEXT node into two DOMTEXT nodes at the specified offset.

See Also: DOMText Subprograms on page 124-28

Syntax

```
DBMS_XMLDOM.SPLITTEXT(
   t        IN     DOMTEXT,
   offset   IN     NUMBER)
 RETURN DOMText;
```

Parameters

Table 124–124 SPLITTEXT Function Parameters

Parameter	Description
t	DOMTEXT
offset	Offset at which to split

SUBSTRINGDATA Function

This function extracts a range of data from the node.

See Also: DOMCharacterData Subprograms on page 124-14

Syntax

```
DBMS_XMLDOM.SUBSTRINGDATA(
   cd         IN     DOMCHARACTERDATA,
   offset     IN     NUMBER,
   cnt        IN     NUMBER)
 RETURN VARCHAR2;
```

Parameters

Table 124–125 SUBSTRINGDATA Function Parameters

Parameter	Description
cd	DOMCHARACTERDATA
offset	The starting offset of the data from which to get the data
cnt	The number of characters (from the offset) of the data to get

WRITETOBUFFER Procedures

This procedure is overloaded. The specific forms of functionality are described along with the syntax declarations.

Syntax

Writes XML node to specified buffer using the database character set (See Also: DOMNode Subprograms on page 124-10):

```
DBMS_XMLDOM.WRITETOBUFFER(
   n        IN      DOMNODE,
   buffer   IN OUT  VARCHAR2);
```

Writes XML document to a specified buffer using database character set (See Also: DOMDocument Subprograms on page 124-16):

```
DBMS_XMLDOM.WRITETOBUFFER(
   doc       IN      DOMDOCUMENT,
   buffer    IN OUT  VARCHAR2);
```

Writes the contents of the specified document fragment into a buffer using the database character set (See Also: DOMDocumentFragment Subprograms on page 124-18):

```
DBMS_XMLDOM.WRITETOBUFFER(
   df        IN      DOMDOCUMENTFRAGMENT,
   buffer    IN OUT  VARCHAR2);
```

Parameters

Table 124–126 WRITETOBUFFER Procedure Parameters

Parameter	Description
n	DOMNODE
buffer	Buffer to which to write
doc	DOMDOCUMENT
df	DOM document fragment

WRITETOCLOB Procedures

This procedure is overloaded. The specific forms of functionality are described along with the syntax declarations.

Syntax

Writes XML node to specified CLOB using the database character set (See Also: DOMNode Subprograms on page 124-10):

```
DBMS_XMLDOM.WRITETOCLOB(
   n       IN      DOMNODE,
   cl      IN OUT  CLOB);
```

Writes XML document to a specified CLOB using database character set (See Also: DOMDocument Subprograms on page 124-16):

```
DBMS_XMLDOM.WRITETOCLOB(
   doc     IN      DOMDOCUMENT,
   cl      IN OUT  CLOB);
```

Parameters

Table 124–127 WRITETOCLOB Procedure Parameters

Parameter	Description
n	DOMNODE
cl	CLOB to which to write
doc	DOMDOCUMENT

WRITETOFILE Procedures

This procedure is overloaded. The specific forms of functionality are described along with the syntax declarations.

Syntax

Writes XML node to specified file using the database character set (See Also: DOMNode Subprograms):

```
DBMS_XMLDOM.WRITETOFILE(
   n          IN      DOMNODE,
   fileName   IN      VARCHAR2);
```

Writes XML node to specified file using the given character set, which is passed in as a separate parameter (See Also: DOMNode Subprograms):

```
DBMS_XMLDOM.WRITETOFILE(
   n          IN      DOMNODE,
   fileName   IN      VARCHAR2,
   charset    IN      VARCHAR2);
```

Writes an XML document to a specified file using database character set (See Also: DOMDocument Subprograms):

```
DBMS_XMLDOM.WRITETOFILE(
   doc        IN   DOMDOCUMENT,
   filename   IN   VARCHAR2);
```

Writes an XML document to a specified file using given character set (See Also: DOMDocument Subprograms):

```
DBMS_XMLDOM.WRITETOFILE(
   doc       IN   DOMDOCUMENT,
   fileName  IN   VARCHAR2,
   charset   IN   VARCHAR2);
```

Parameters

Table 124–128 WRITETOFILE Procedure Parameters

Parameter	Description
n	DOMNODE
fileName	File to which to write
charset	Given character set
doc	DOMDOCUMENT
charset	Character set

125

DBMS_XMLGEN

The DBMS_XMLGEN package converts the results of a SQL query to a canonical XML format. The package takes an arbitrary SQL query as input, converts it to XML format, and returns the result as a CLOB. This package is similar to the DBMS_XMLQUERY package, except that it is written in C and compiled into the kernel. This package can only be run on the database.

This chapter contains the following topic:

- Summary of DBMS_XMLGEN Subprograms

> **See Also:** *Oracle XML DB Developer's Guide*, for more information on XML support and on examples of using DBMS_XMLGEN

Summary of DBMS_XMLGEN Subprograms

Table 125–1 Summary of DBMS_XMLGEN Package Subprograms

Subprogram	Description
CLOSECONTEXT Procedure on page 125-3	Closes the context and releases all resources
CONVERT Functions on page 125-4	Converts the XML into the escaped or unescaped XML equivalent
GETNUMROWSPROCESSED Function on page 125-5	Gets the number of SQL rows that were processed in the last call to GETXML Functions
GETXML Functions on page 125-6	Gets the XML document
GETXMLTYPE Functions on page 125-7	Gets the XML document and returns it as `XMLType`
NEWCONTEXT Functions on page 125-8	Creates a new context handle
RESTARTQUERY Procedure on page 125-9	Restarts the query to start fetching from the beginning
SETCONVERTSPECIALCHARS Procedure on page 125-10	Sets whether special characters such as `$`, which are non-XML characters, should be converted or not to their escaped representation
SETMAXROWS Procedure on page 125-11	Sets the maximum number of rows to be fetched each time
SETNULLHANDLING Procedure on page 125-12	Sets `NULL` handling options
SETROWSETTAG Procedure on page 125-13	Sets the name of the element enclosing the entire result
SETROWTAG Procedure on page 125-14	Sets the name of the element enclosing each row of the result
SETSKIPROWS Procedure on page 125-15	Sets the number of rows to skip every time before generating the XML.
USEITEMTAGSFORCOLL Procedure on page 125-16	Forces the use of the collection column name appended with the tag `_ITEM` for collection elements
USENULLATTRIBUTEINDICATOR Procedure on page 125-17	Specified weather to use an XML attribute to indicate `NULL`ness, or to do it by omitting the inclusion of the particular entity in the XML document.

CLOSECONTEXT Procedure

This procedure closes a given context and releases all resources associated with it, including the SQL cursor and bind and define buffers. After this call, the handle cannot be used for a subsequent function call.

Syntax

```
DBMS_XMLGEN.CLOSECONTEXT (
   ctx  IN ctxHandle);
```

Parameters

Table 125–2 CLOSECONTEXT Procedure Parameters

Parameter	Description
ctx	The context handle to close.

CONVERT Functions

This function converts the XML data into the escaped or unescapes XML equivalent, and returns XML CLOB data in encoded or decoded format. There are several version of the function.

Syntax

Uses XMLDATA in string form (VARCHAR2):

```
DBMS_XMLGEN.CONVERT (
   xmlData IN VARCHAR2,
   flag    IN NUMBER := ENTITY_ENCODE)
RETURN VARCHAR2;
```

Uses XMLDATA in CLOB form:

```
DBMS_XMLGEN.CONVERT (
   xmlData IN CLOB,
   flag    IN NUMBER := ENTITY_ENCODE)
 RETURN CLOB;
```

Parameters

Table 125–3 CONVERT Function Parameters

Parameter	Description
xmlData	The XML CLOB data to be encoded or decoded.
flag	The flag setting; ENTITY_ENCODE (default) for encode, and ENTITY_DECODE for decode.

Usage Notes

This function escapes the XML data if the ENTITY_ENCODE is specified. For example, the escaped form of the character < is <. Unescaping is the reverse transformation.

GETNUMROWSPROCESSED Function

This function retrieves the number of SQL rows processed when generating the XML using the GETXML Functions call. This count does not include the number of rows skipped before generating the XML. Note that GETXML Functions always generates an XML document, even if there are no rows present.

Syntax

```
DBMS_XMLGEN.GETNUMROWSPROCESSED (
   ctx     IN    ctxHandle)
RETURN NUMBER;
```

Parameters

Table 125–4 GETNUMROWSPROCESSED Function Parameters

Parameter	Description
ctx	The context handle obtained from the NEWCONTEXT Functions call on page 125-8.

Usage Notes

This function is used to determine the terminating condition if calling GETXML Functions in a loop.

GETXML Functions

This function gets the XML document. The function is overloaded.

Syntax

Gets the XML document by fetching the maximum number of rows specified. It appends the XML document to the CLOB passed in. Use this version of GETXML Functions to avoid any extra CLOB copies and to reuse the same CLOB for subsequent calls. Because of the CLOB reuse, this GETXML Functionscall is potentially more efficient:

```
DBMS_XMLGEN.GETXML (
   ctx          IN ctxHandle,
   tmpclob      IN OUT NCOPY CLOB,
   dtdOrSchema  IN number := NONE)
 RETURN BOOLEAN;
```

Generates the XML document and returns it as a temporary CLOB. The temporary CLOB obtained from this function must be freed using the DBMS_ LOB.FREETEMPORARY call:

```
DBMS_XMLGEN.GETXML (
   ctx          IN ctxHandle,
   dtdOrSchema  IN number := NONE)
 RETURN CLOB;
```

Converts the results from the SQL query string to XML format, and returns the XML as a temporary CLOB, which must be subsequently freed using the DBMS_ LOB.FREETEMPORARY call:

```
DBMS_XMLGEN.GETXML (
   sqlQuery     IN VARCHAR2,
   dtdOrSchema  IN number := NONE)
 RETURN CLOB;
```

Parameters

Table 125–5 GETXML Function Parameters

Parameter	Description
ctx	The context handle obtained from the newContext call.
tmpclob	The CLOB to which the XML document is appended.
sqlQuery	The SQL query string.
dtdOrSchema	Generate a DTD or a schema? Only NONE is supported.

Usage Notes

When the rows indicated by the SETSKIPROWS Procedure call are skipped, the maximum number of rows as specified by the SETMAXROWS Procedure call (or the entire result if not specified) is fetched and converted to XML. Use the GETNUMROWSPROCESSED Function to check if any rows were retrieved.

GETXMLTYPE Functions

This function gets the XML document and returns it as an XMLTYPE. XMLTYPE operations can be performed on the results.

This function is overloaded.

Syntax

Generates the XML document and returns it as a sys.XMLType:

```
DBMS_XMLGEN.GETXMLTYPE (
   ctx           IN ctxhandle,
   dtdOrSchema   IN number := NONE)
 RETURN sys.XMLType;
```

Converts the results from the SQL query string to XML format, and returns the XML as a sys.XMLType:

```
DBMS_XMLGEN.GETXMLTYPE (
   sqlQuery     IN VARCHAR2,
   dtdOrSchema  IN number := NONE)
 RETURN sys.XMLType
```

Parameters

Table 125–6 GETXMLTYPE Function Parameters

Parameter	Description
ctx	The context handle obtained from the newContext call.
sqlQuery	The SQL query string.
dtdOrSchema	Generate a DTD or a schema? Only NONE is supported.

NEWCONTEXT Functions

This function generates and returns a new context handle. This context handle is used in GETXML Functions and other functions to get XML back from the result. There are several version of the function.

Syntax

Generates a new context handle from a query:

```
DBMS_XMLGEN.NEWCONTEXT (
      query     IN VARCHAR2)
 RETURN ctxHandle;
```

Generates a new context handle from a query string in the form of a PL/SQL ref cursor:

```
DBMS_XMLGEN.NEWCONTEXT (
   queryString  IN SYS_REFCURSOR)
 RETURN ctxHandle;
```

Parameters

Table 125–7 NEWCONTEXT Function Parameters

Parameter	Description
`query`	The query, in the form of a `VARCHAR`, the result of which must be converted to XML.
`queryString`	The query string in the form of a PL/SQL ref cursor, the result of which must be converted to XML.

RESTARTQUERY Procedure

This procedure restarts the query and generates the XML from the first row. It can be used to start executing the query again, without having to create a new context.

Syntax

```
DBMS_XMLGEN.RESTARTQUERY (
ctx  IN ctxHandle);
```

Parameters

Table 125–8 RESTARTQUERY Procedure Parameters

Parameter	Description
ctx	The context handle corresponding to the current query.

SETCONVERTSPECIALCHARS Procedure

This procedure sets whether or not special characters in the XML data must be converted into their escaped XML equivalent. For example, the < sign is converted to `<`. The default is to perform conversions. This function improves performance of XML processing when the input data cannot contain any special characters such as <, >, `"`, `'`, which must be escaped. It is expensive to scan the character data to replace the special characters, particularly if it involves a lot of data.

Syntax

```
DBMS_XMLGEN.SETCONVERTSPECIALCHARS (
ctx   IN ctxHandle,
conv  IN BOOLEAN);
```

Parameters

Table 125–9 SETCONVERTSPECIALCHARS Procedure Parameters

Parameter	Description
`ctx`	The context handle obtained from one of the NEWCONTEXT Functions call on page 125-8.
`conv`	`TRUE` indicates that conversion is needed.

SETMAXROWS Procedure

This procedure sets the maximum number of rows to fetch from the SQL query result for every invokation of the GETXML Functions call. It is used when generating paginated results. For example, when generating a page of XML or HTML data, restrict the number of rows converted to XML or HTML by setting the `maxrows` parameter.

Syntax

```
DBMS_XMLGEN.SETMAXROWS (
ctx      IN ctxHandle,
maxRows  IN NUMBER);
```

Parameters

Table 125–10 SETMAXROWS Procedure Parameters

Parameter	Description
`ctx`	The context handle corresponding to the query executed.
`maxRows`	The maximum number of rows to get for each call to GETXML Functions

SETNULLHANDLING Procedure

This procedure sets NULL handling options, handled through the flag parameter setting.

Syntax

```
DBMS_XMLGEN.SETNULLHANDLING(
ctx  IN ctx,
flag IN NUMBER);
```

Parameters

Table 125–11 SETNULLHANDLING Procedure Parameters

Parameter	Description
ctx	The context handle corresponding to the query executed.
flag	The NULL handling option set. ■ DROP_NULLS CONSTANT NUMBER:= 0; (Default) Leaves out the tag for NULL elements. ■ NULL_ATTR CONSTANT NUMBER:= 1; Sets xsi:nil="true". ■ EMPTY_TAG CONSTANT NUMBER:= 2; Sets, for example, <foo/>.

SETROWSETTAG Procedure

This procedure sets the name of the root element of the document. The default name is `ROWSET`.

Syntax

```
DBMS_XMLGEN.SETROWSETTAG (
ctx            IN ctxHandle,
rowSetTagName  IN VARCHAR2);
```

Parameters

Table 125–12 SETROWSETTAG Procedure Parameters

Parameter	Description
ctx	The context handle obtained from the NEWCONTEXT Functions call on page 125-8.
rowSetTagName	The name of the document element. Passing `NULL` indicates that you do not want the `ROWSET` element present.

Usage Notes

The user can set the `rowSetTag` to `NULL` to suppress the printing of this element. However, an error is produced if both the row and the rowset are `NULL` and there is more than one column or row in the output . This is because the generated XML would not have a top-level enclosing tag, and so would be invalid.

SETROWTAG Procedure

This procedure sets the name of the element separating all the rows. The default name is `ROW`.

Syntax

```
DBMS_XMLGEN.SETROWTAG (
ctx         IN ctxHandle,
rowTagName  IN VARCHAR2);
```

Parameters

Table 125–13 SETROWTAG Procedure Parameters

Parameter	Description
`ctx`	The context handle obtained from the NEWCONTEXT Functions call on page 125-8.
`rowTagName`	The name of the `ROW` element. Passing `NULL` indicates that you do not want the `ROW` element present.

Usage Notes

The user can set the name of the element to `NULL` to suppress the `ROW` element itself. However, an error is produced if both the row and the rowset are `NULL` and there is more than one column or row in the output. This is because the generated XML would not have a top-level enclosing tag, and so would be invalid.

SETSKIPROWS Procedure

This procedure skips a given number of rows before generating the XML output for every call to the GETXML Functions. It is used when generating paginated results for stateless Web pages using this utility. For example, when generating the first page of XML or HTML data, set `skiprows` to zero. For the next set, set the `skiprows` to the number of rows obtained in the first case. See GETNUMROWSPROCESSED Function on page 125-5.

Syntax

```
DBMS_XMLGEN.SETSKIPROWS (
ctx       IN ctxHandle,
skipRows  IN NUMBER);
```

Parameters

Table 125–14 SETSKIPROWS Procedure Parameters

Parameter	Description
`ctx`	The context handle corresponding to the query executed.
`skipRows`	The number of rows to skip for each call to `getXML`.

USEITEMTAGSFORCOLL Procedure

This procedure overrides the default name of the collection elements. The default name for collection elements is the type name itself.

Syntax

```
DBMS_XMLGEN.USEITEMTAGSFORCOLL (
   ctx  IN ctxHandle);
```

Parameters

Table 125–15 USEITEMTAGSFORCOLL Procedure Parameters

Parameter	Description
ctx	The context handle.

Usage Notes

Using this procedure, you can override the default to use the name of the column with the _ITEM tag appended to it. If there is a collection of NUMBER, the default tag name for the collection elements is NUMBER.

USENULLATTRIBUTEINDICATOR Procedure

This procedure specifies whether to use an XML attribute to indicate NULLness, or to do it by omitting the inclusion of the particular entity in the XML document. It is used as a shortcut for the SETNULLHANDLING Procedure .

Syntax

```
DBMS_XMLGEN.USENULLATTRIBUTEINDICATOR(
ctx       IN   ctxType,
attrind   IN   BOOLEAN := TRUE);
```

Parameters

Table 125–16 USENULLATTRIBUTEINDICATOR Procedure Parameters

Parameter	Description
ctx	Context handle.
attrind	Use attribute to indicate NULL?

126

DBMS_XMLPARSER

Using `DBMS_XMLPARSER`, you can access the contents and structure of XML documents. XML describes a class of data XML document objects. It partially describes the behavior of computer programs which process them. By construction, XML documents are conforming SGML documents.

XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form character data, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure. XML provides a mechanism to impose constraints on the storage layout and logical structure.

A software module called an XML processor is used to read XML documents and provide access to their content and structure. It is assumed that an XML processor is doing its work on behalf of another module, called the application. This PL/SQL implementation of the XML processor (or parser) follows the W3C XML specification REC-xml-19980210 and includes the required behavior of an XML processor in terms of how it must read XML data and the information it must provide to the application.

The default behavior for this PL/SQL XML parser is to build a parse tree that can be accessed by DOM APIs, validate it if a DTD is found (otherwise, it is non-validating), and record errors if an error log is specified. If parsing fails, an application error is raised.

This chapter contains the following topics:

See Also: *Oracle XML DB Developer's Guide*

Summary of DBMS_XMLPARSER Subprograms

Table 126–1 DBMS_XMLPARSER Package Subprograms

Method	Description
FREEPARSER on page 126-3	Frees a parser object.
GETDOCTYPE on page 126-4	Gets parsed DTD.
GETDOCUMENT on page 126-5	Gets DOM document.
GETRELEASEVERSION on page 126-5	Returns the release version of Oracle XML Parser for PL/SQL.
GETVALIDATIONMODE on page 126-7	Returns validation mode.
NEWPARSER on page 126-8	Returns a new parser instance
PARSE on page 126-9	Parses XML stored in the given url/file.
PARSEBUFFER on page 126-10	Parses XML stored in the given buffer
PARSECLOB on page 126-10	Parses XML stored in the given clob
PARSEDTD on page 126-12	Parses DTD stored in the given url/file
PARSEDTDBUFFER on page 126-13	Parses DTD stored in the given buffer
PARSEDTDCLOB on page 126-14	Parses DTD stored in the given clob
SETBASEDIR on page 126-15	Sets base directory used to resolve relative URLs.
SETDOCTYPE on page 126-16	Sets DTD.
SETERRORLOG on page 126-17	Sets errors to be sent to the specified file
SETPRESERVEWHITESPACE on page 126-18	Sets white space preserve mode
SETVALIDATIONMODE on page 126-19	Sets validation mode.
SHOWWARNINGS on page 126-20	Turns warnings on or off.

FREEPARSER

Frees a parser object.

Syntax

```
PROCEDURE freeParser(
    p Parser);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.

GETDOCTYPE

Returns the parsed DTD; this function must be called only after a DTD is parsed.

Syntax

```
FUNCTION getDoctype(
p Parser)
RETURN DOMDocumentType;
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.

GETDOCUMENT

Returns the document node of a DOM tree document built by the parser; this function must be called only after a document is parsed.

Syntax

```
FUNCTION GETDOCUMENT(
p Parser)
RETURN DOMDocument;
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.

GETRELEASEVERSION

Returns the release version of the Oracle XML parser for PL/SQL.

Syntax

```
FUNCTION getReleaseVersion
RETURN VARCHAR2;
```

GETVALIDATIONMODE

Retrieves validation mode; `TRUE` for validating, `FALSE` otherwise.

Syntax

```
FUNCTION GETVALIDATIONMODE(
p Parser)
RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.

NEWPARSER

Returns a new parser instance. This function must be called before the default behavior of Parser can be changed and if other parse methods need to be used.

Syntax

```
FUNCTION newParser
RETURN Parser;
```

PARSE

Parses XML stored in the given URL or file. An application error is raised if parsing fails. There are several versions of this method.

Syntax	Description

`FUNCTION parse(`
`  url VARCHAR2)`
`RETURN DOMDocument;`
	Returns the built DOM Document.
	This is meant to be used when the default parser behavior is acceptable and just a url/file needs to be parsed.

`PROCEDURE parse(`
`  p  Parser,`
`  url  VARCHAR2);`
	Any changes to the default parser behavior should be effected before calling this procedure.

Parameter	IN / OUT	Description
`url`	`(IN)`	Complete path of the url/file to be parsed.
`p`	`(IN)`	Parser instance.

PARSEBUFFER

Parses XML stored in the given buffer. Any changes to the default parser behavior should be effected before calling this procedure. An application error is raised if parsing fails.

Syntax

```
PROCEDURE PARSEBUFFER(
p   Parser,
doc VARCHAR2);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
doc	(IN)	XML document buffer to parse.

PARSECLOB

Parses XML stored in the given clob. Any changes to the default parser behavior should be effected before calling this procedure. An application error is raised if parsing fails.

Syntax

```
PROCEDURE PARSECLOB(
p   Parser,
doc CLOB);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
doc	(IN)	XML document buffer to parse.

PARSEDTD

Parses the DTD stored in the given URL or file. Any changes to the default parser behavior should be effected before calling this procedure. An application error is raised if parsing fails.

Syntax

```
PROCEDURE PARSEDTD(
p     Parser,
url   VARCHAR2,
root  VARCHAR2);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
url	(IN)	Complete path of the URL or file to be parsed.
root	(IN)	Name of the root element.

PARSEDTDBUFFER

Parses the DTD stored in the given buffer. Any changes to the default parser behavior should be effected before calling this procedure. An application error is raised if parsing fails.

Syntax

```
PROCEDURE PARSEDTDBUFFER(
p    Parser,
dtd  VARCHAR2,
root VARCHAR2);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
dtd	(IN)	DTD buffer to parse.
root	(IN)	Name of the root element.

PARSEDTDCLOB

Parses the DTD stored in the given clob. Any changes to the default parser behavior should be effected before calling this procedure. An application error is raised if parsing fails.

Syntax

```
PROCEDURE PARSEDTDCLOB(
p    Parser,
dtd  CLOB,
root VARCHAR2);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
dtd	(IN)	DTD Clob to parse.
root	(IN)	Name of the root element.

SETBASEDIR

Sets base directory used to resolve relative URLs. An application error is raised if parsing fails.

Syntax

```
PROCEDURE setBaseDir(
p   Parser,
dir VARCHAR2);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
dir	(IN)	Directory used as a base directory.

SETDOCTYPE

Sets a DTD to be used by the parser for validation. This call should be made before the document is parsed.

Syntax

```
PROCEDURE setDoctype(
p   Parser,
dtd DOMDocumentType);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
dtd	(IN)	DTD to set.

SETERRORLOG

Sets errors to be sent to the specified file.

Syntax

```
PROCEDURE setErrorLog(
p        Parser,
fileName VARCHAR2);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
fileName	(IN)	Complete path of the file to use as the error log.

SETPRESERVEWHITESPACE

Sets whitespace preserving mode.

Syntax

```
PROCEDURE setPreserveWhitespace(
p   Parser,
yes BOOLEAN);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
yes	(IN)	Mode to set: TRUE - preserve, FALSE - don't preserve.

SETVALIDATIONMODE

Sets validation mode.

Syntax

```
PROCEDURE setValidationMode(
p   Parser,
yes BOOLEAN);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
yes	(IN)	Mode to set: TRUE - validate, FALSE - don't validate.

SHOWWARNINGS

Turns warnings on or off.

Syntax

```
PROCEDURE showWarnings(
p   Parser,
yes BOOLEAN);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
yes	(IN)	Mode to set: TRUE - show warnings, FALSE - don't show warnings.

127

DBMS_XMLQUERY

DBMS_XMLQUERY provides database-to-XMLType functionality. Whenever possible, use DBMS_XMLGEN, a built-in package in C, instead of DBMS_XMLQUERY.

See Also: *Oracle XML DB Developer's Guide*

This chapter contains the following topics:

Using DBMS_XMLQUERY

Constants

Table 127–1 Constants of DBMS_XMLQUERY

Constant	Description
DB_ENCODING	Used to signal that the DB character encoding is to be used.
DEFAULT_ ROWSETTAG	The tag name for the element enclosing the XML generated from the result set (that is, for most cases the root node tag name) -- ROWSET.
DEFAULT_ERRORTAG	The default tag to enclose raised errors -- ERROR.
DEFAULT_ ROWIDATTR	The default name for the cardinality attribute of XML elements corresponding to db.records -- NUM
DEFAULT_ROWTAG	The default tag name for the element corresponding to db. records -- ROW
DEFAULT_DATE_ FORMAT	Default date mask --'MM/dd/yyyy HH:mm:ss'
ALL_ROWS	Indicates that all rows are needed in the output.
NONE	Used to specifies that the output should not contain any XML metadata (for example, no DTD).
DTD	Used to specify that the generation of the DTD is desired.
SCHEMA	Used to specify that the generation of the XML Schema is desired.
LOWER_CASE	Use lower case tag names.
UPPER_CASE	Use upper case tag names.

Types

Table 127–2 Types of DBMS_XMLQUERY

Type	Description
`ctxType`	The type of the query context handle. This is the return type of NEWCONTEXT

Summary of DBMS_XMLQUERY Subprograms

Table 127–3 DBMS_XMLQUERY Package Subprograms

Method	Description
CLOSECONTEXT on page 127-7	Closes or deallocates a particular query context.
GETDTD on page 127-8	Generates the DTD.
GETEXCEPTIONCONTENT on page 127-9	Returns the thrown exception's error code and error message.
GETNUMROWSPROCESSED on page 127-10	Returns the number of rows processed for the query.
GETVERSION on page 127-11	Prints the version of the XSU in use.
GETXML on page 127-12	Generates the XML document.
NEWCONTEXT on page 127-13	Creates a query context and it returns the context handle.
PROPAGATEORIGINALEXCEPTION on page 127-14	Tells the XSU that if an exception is raised, and is being thrown, the XSU should throw the very exception raised; rather then, wrapping it with an `OracleXMLSQLException`.
REMOVEXSLTPARAM on page 127-15	Removes a particular top-level stylesheet parameter.
SETBINDVALUE on page 127-16	Sets a value for a particular bind name.
SETCOLLIDATTRNAME on page 127-17	Sets the name of the id attribute of the collection element's separator tag.
SETDATAHEADER on page 127-17	Sets the XML data header.
SETDATEFORMAT on page 127-19	Sets the format of the generated dates in the XML document.
SETENCODINGTAG on page 127-20	Sets the encoding processing instruction in the XML document.
SETERRORTAG on page 127-21	Sets the tag to be used to enclose the XML error documents.
SETMAXROWS on page 127-22	Sets the maximum number of rows to be converted to XML.
SETMETAHEADER on page 127-23	Sets the XML meta header.
SETRAISEEXCEPTION on page 127-24	Tells the XSU to throw the raised exceptions.
SETRAISENOROWSEXCEPTION on page 127-25	Tells the XSU to throw or not to throw an `OracleXMLNoRowsException` in the case when for one reason or another, the XML document generated is empty.
SETROWIDATTRNAME on page 127-26	Sets the name of the id attribute of the row enclosing tag.
SETROWIDATTRVALUE on page 127-27	Specifies the scalar column whose value is to be assigned to the id attribute of the row enclosing tag.
SETROWSETTAG on page 127-28	Sets the tag to be used to enclose the XML dataset.
SETROWTAG on page 127-29	Sets the tag to be used to enclose the XML element.
SETSKIPROWS on page 127-30	Sets the number of rows to skip.

Table 127–3 (Cont.) DBMS_XMLQUERY Package Subprograms

Method	Description
SETSQLTOXMLNAMEESCAPING on page 127-31	This turns on or off escaping of XML tags in the case that the SQL object name, which is mapped to a XML identifier, is not a valid XML identifier.
SETSTYLESHEETHEADER on page 127-32	Sets the stylesheet header.
SETTAGCASE on page 127-33	Specified the case of the generated XML tags.
SETXSLT on page 127-34	Registers a stylesheet to be applied to generated XML.
SETXSLTPARAM on page 127-35	Sets the value of a top-level stylesheet parameter.
USENULLATTRIBUTEINDICATOR on page 127-36	Specifies weather to use an XML attribute to indicate `NULL`ness.
USETYPEFORCOLLELEMTAG on page 127-37	Tells the XSU to use the collection element's type name as the collection element tag name.

CLOSECONTEXT

Closes or deallocates a particular query context

Syntax

```
PROCEDURE CLOSECONTEXT(
ctxHdl IN ctxType);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.

GETDTD

Generates and returns the DTD based on the SQL query used to initialize the context. The options are described in the following table.

Syntax	Description

`FUNCTION GETDTD(`
`ctxHdl IN ctxType,`
`withVer IN BOOLEAN := false)`
`RETURN CLOB;`
	Function that generates the DTD based on the SQL query used to initialize the context.

`PROCEDURE GETDTD(`
`ctxHdl IN ctxType,`
`xDoc IN CLOB,`
`withVer IN BOOLEAN := false);`
	Procedure that generates the DTD based on the SQL query used to initialize the context; specifies the output `CLOB` for XML document result.

Parameter	IN / OUT	Description
`ctxHdl`	`(IN)`	Context handle.
`withVer`	`(IN)`	Generate the version information? `TRUE` for yes .
`xDoc`	`(IN)`	`CLOB` into which to write the generated XML document.

GETEXCEPTIONCONTENT

Returns the thrown exception's SQL error code and error message through the procedure's `OUT` parameters. This procedure is a work around the JVM functionality that obscures the original exception by its own exception, rendering PL/SQL unable to access the original exception content.

Syntax

```
PROCEDURE GETEXCEPTIONCONTENT(
ctxHdl IN ctxType,
errNo OUT NUMBER,
errMsg OUT VARCHAR2);
```

Parameter	IN / OUT	Description
`ctxHdl`	`(IN)`	Context handle.
`errNo`	`(OUT)`	Error number.
`errMsg`	`(OUT)`	Error message.

GETNUMROWSPROCESSED

Return the number of rows processed for the query.

Syntax

```
FUNCTION GETNUMROWSPROCESSED(
ctxHdl IN ctxType)
RETURN NUMBER;
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.

GETVERSION

Prints the version of the XSU in use.

Syntax

```
PROCEDURE GETVERSION();
```

GETXML

Creates the new context, executes the query, gets the XML back and closes the context. This is a convenience function. The context doesn't have to be explicitly opened or closed. The options are described in the following table.

Syntax	Description

`FUNCTION GETXML(`
`sqlQuery IN VARCHAR2,`
`metaType IN NUMBER := NONE)`
`RETURN CLOB;`
	This function uses a SQL query in string form.

`FUNCTION GETXML(`
`sqlQuery IN CLOB,`
`metaType IN NUMBER := NONE)`
`RETURN CLOB;`
	This function uses a SQL query in `CLOB` form.

`FUNCTION GETXML(`
`ctxHdl IN ctxType,`
`metaType IN NUMBER := NONE)`
`RETURN CLOB;`
	This function generates the XML document based on a SQL query used to initialize the context.

`PROCEDURE GETXML(`
`ctxHdl IN ctxType,`
`xDoc IN CLOB,`
`metaType IN NUMBER := NONE);`
	This procedure generates the XML document based on the SQL query used to initialize the context.

Parameter	IN / OUT	Description
`ctxHdl`	`(IN)`	Context handle.
`metaType`	`(IN)`	XML metadata type (`NONE`, `DTD`, or `SCHEMA`).
`sqlQuery`	`(IN)`	SQL query.
`xDoc`	`(IN)`	`CLOB` into which to write the generated XML document.

NEWCONTEXT

Creates a query context and it returns the context handle. The options are described in the following table.

Syntax	Description

`FUNCTION NEWCONTEXT(`
`  sqlQuery IN VARCHAR2)`
`RETURN ctxType;`
	Creates a query context from a string.

`FUNCTION NEWCONTEXT(`
`  sqlQuery IN CLOB)`
`RETURN ctxType;`
	Creates a query context from a `CLOB`.

Parameter	IN / OUT	Description
`sqlQuery`	`(IN)`	SQL query, the results of which to convert to XML.

PROPAGATEORIGINALEXCEPTION

Specifies whether to throw every original exception raised or to wrap it in an `OracleXMLSQLException`.

Syntax

```
PROCEDURE PROPAGATEORIGINALEXCEPTION(
ctxHdl IN ctxType,
flag IN BOOLEAN);
```

Parameter	IN / OUT	Description
`ctxHdl`	`(IN)`	Context handle.
`flag`	`(IN)`	`TRUE` if want to propagate original exception, `FALSE` to wrap in `OracleXMLException`.

REMOVEXSLTPARAM

Removes the value of a top-level stylesheet parameter. If no stylesheet is registered, this method is not operational.

Syntax

```
PROCEDURE REMOVEXSLTPARAM(
ctxHdl IN ctxType,
name IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
name	(IN)	Name of the top level stylesheet parameter.

SETBINDVALUE

Sets a value for a particular bind name.

Syntax

```
PROCEDURE SETBINDVALUE(
ctxHdl IN ctxType,
bindName IN VARCHAR2,
bindValue IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
bindName	(IN)	Bind name.
bindValue	(IN)	Bind value.

SETCOLLIDATTRNAME

Sets the name of the id attribute of the collection element's separator tag. Passing `NULL` or an empty string for the tag causes the row id attribute to be omitted.

Syntax

```
PROCEDURE SETCOLLIDATTRNAME(
ctxHdl IN ctxType,
attrName IN VARCHAR2);
```

Parameter	IN / OUT	Description
`ctxHdl`	`(IN)`	Context handle.
`attrName`	`(IN)`	Attribute name.

SETDATAHEADER

Sets the XML data header. The data header is an XML entity that is appended at the beginning of the query-generated XML entity, the `rowset`. The two entities are enclosed by the `docTag` argument. The last data header specified is used. Passing in `NULL` for the `header` parameter unsets the data header.

Syntax

```
PROCEDURE SETDATAHEADER(
ctxHdl IN ctxType,
header IN CLOB := null,
tag IN VARCHAR2 := null);
```

Parameter	IN / OUT	Description
`ctxHdl`	`(IN)`	Context handle.
`header`	`(IN)`	Header.
`tag`	`(IN)`	Tag used to enclose the data header and the rowset.

SETDATEFORMAT

Sets the format of the generated dates in the XML document. The syntax of the date format pattern, the date mask, should conform to the requirements of the `java.text.SimpleDateFormat` class. Setting the mask to `NULL` or an empty string sets the default mask -- `DEFAULT_DATE_FORMAT`.

Syntax

```
PROCEDURE SETDATEFORMAT(
ctxHdl IN ctxType,
mask IN VARCHAR2);
```

Parameter	IN / OUT	Description
`ctxHdl`	`(IN)`	Context handle.
`mask`	`(IN)`	The date mask.

SETENCODINGTAG

Sets the encoding processing instruction in the XML document.

Syntax

```
PROCEDURE SETENCODINGTAG(
ctxHdl IN ctxType,
enc IN VARCHAR2 := DB_ENCODING);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
enc	(IN)	The encoding to use.

SETERRORTAG

Sets the tag to be used to enclose the XML error documents.

Syntax

```
PROCEDURE SETERRORTAG(
ctxHdl IN ctxType,
tag IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
tag	(IN)	Tag name.

SETMAXROWS

Sets the maximum number of rows to be converted to XML. By default, there is no set maximum.

Syntax

```
PROCEDURE SETMAXROWS (
ctxHdl IN ctxType,
rows IN NUMBER);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
rows	(IN)	Maximum number of rows to generate.

SETMETAHEADER

Sets the XML meta header. When set, the header is inserted at the beginning of the metadata part (DTD or XMLSchema) of each XML document generated by this object. The last meta header specified is used. Passing in `NULL` for the `header` parameter unsets the meta header.

Syntax

```
PROCEDURE SETMETAHEADER(
ctxHdl IN ctxType,
header IN CLOB := null);
```

Parameter	IN / OUT	Description
`ctxHdl`	`(IN)`	Context handle.
`Header`	`(IN)`	Header.

SETRAISEEXCEPTION

Specifies whether to throw raised exceptions. If this call isn't made or if `FALSE` is passed to the `flag` argument, the XSU catches the SQL exceptions and generates an XML document from the exception message.

Syntax

```
PROCEDURE SETRAISEEXCEPTION(
ctxHdl IN ctxType,
flag IN BOOLEAN:=true);
```

Parameter	IN / OUT	Description
`ctxHdl`	`(IN)`	Context handle.
`flag`	`(IN)`	Throw raised exceptions? `TRUE` for yes, otherwise `FALSE`.

SETRAISENOROWSEXCEPTION

Specifies whether to throw an `OracleXMLNoRowsException` when the generated XML document is empty. By default, the exception is not thrown.

Syntax

```
PROCEDURE SETRAISENOROWSEXCEPTION(
ctxHdl IN ctxType,
flag IN BOOLEAN:=false);
```

Parameter	IN / OUT	Description
`ctxHdl`	`(IN)`	Context handle.
`flag`	`(IN)`	Throws an `OracleXMLNoRowsException` if set to `TRUE`.

SETROWIDATTRNAME

Sets the name of the id attribute of the row enclosing tag. Passing `NULL` or an empty string for the tag causes the row id attribute to be omitted.

Syntax

```
PROCEDURE SETROWIDATTRNAME(
ctxHdl IN ctxType,
attrName IN VARCHAR2);
```

Parameter	IN / OUT	Description
`ctxHdl`	`(IN)`	Context handle.
`attrName`	`(IN)`	Attribute name.

SETROWIDATTRVALUE

Specifies the scalar column whose value is to be assigned to the id attribute of the row enclosing tag. Passing `NULL` or an empty string for the `colName` assigns the row count value (0, 1, 2 and so on) to the row id attribute.

Syntax

```
PROCEDURE SETROWIDATTRVALUE(
ctxHdl IN ctxType,
colName IN VARCHAR2);
```

Parameter	IN / OUT	Description
`ctxHdl`	`(IN)`	Context handle.
`colName`	`(IN)`	Column whose value is to be assigned to the row id attribute.

SETROWSETTAG

Sets the tag to be used to enclose the XML dataset.

Syntax

```
PROCEDURE SETROWSETTAG(
ctxHdl IN ctxType,
tag IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
tag	(IN)	Tag name.

SETROWTAG

Sets the tag to be used to enclose the XML element corresponding to a `db.record`.

Syntax

```
PROCEDURE SETROWTAG(
ctxHdl IN ctxType,
tag IN VARCHAR2);
```

Parameter	IN / OUT	Description
`ctxHdl`	`(IN)`	Context handle.
`tag`	`(IN)`	Tag name.

SETSKIPROWS

Sets the number of rows to skip. By default, 0 rows are skipped.

Syntax

```
PROCEDURE SETSKIPROWS(
ctxHdl IN ctxType,
rows IN NUMBER);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
rows	(IN)	Maximum number of rows to skip.

SETSQLTOXMLNAMEESCAPING

This turns on or off escaping of XML tags in the case that the SQL object name, which is mapped to a XML identifier, is not a valid XML identifier.

Syntax

```
PROCEDURE SETSQLTOXMLNAMEESCAPING(
ctxHdl IN ctxType,
flag IN BOOLEAN := true);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
flag	(IN)	Turn on escaping? TRUE for yes, otherwise FALSE.

SETSTYLESHEETHEADER

Sets the stylesheet header (the stylesheet processing instructions) in the generated XML document. Passing `NULL` for the `uri` argument will unset the stylesheet header and the stylesheet type.

Syntax

```
PROCEDURE SETSTYLESHEETHEADER(
ctxHdl IN ctxType,
uri IN VARCHAR2,
type IN VARCHAR2 := 'text/xsl');
```

Parameter	IN / OUT	Description
`ctxHdl`	`(IN)`	Context handle.
`uri`	`(IN)`	Stylesheet URI.
`type`	`(IN)`	Stylesheet type; defaults to "`text/xsl`".

SETTAGCASE

Specifies the case of the generated XML tags.

Syntax

```
PROCEDURE SETTAGCASE(
ctxHdl IN ctxType,
tCase IN NUMBER);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
tCase	(IN)	The tag's case: ■ 0 for as are ■ 1 for lower case ■ 2 for upper case

SETXSLT

Registers a stylesheet to be applied to generated XML. If a stylesheet was already registered, it is replaced by the new one. The options are described in the following table. Passing `NULL` for the `uri` argument or an empty string for the `stylesheet` argument will unset the stylesheet header and type.

Syntax	Description
`PROCEDURE SETXSLT( ctxHdl IN ctxType, uri IN VARCHAR2, ref IN VARCHAR2 := null);`	To un-register the stylesheet pass in a null for the uri.
`PROCEDURE SETXSLT( ctxHdl IN ctxType, stylesheet CLOB, ref IN VARCHAR2 := null);`	To un-register the stylesheet pass in a null or an empty string for the stylesheet.

Parameter	IN / OUT	Description
`ctxHdl`	`(IN)`	Context handle.
`uri`	`(IN)`	Stylesheet URI.
`stylesheet`	`(IN)`	Stylesheet.
`ref`	`(IN)`	URL to include, imported and external entities.

SETXSLTPARAM

Sets the value of a top-level stylesheet parameter. The parameter value is expected to be a valid XPath expression; the string literal values would therefore have to be quoted explicitly. If no stylesheet is registered, this method is not operational.

Syntax

```
PROCEDURE SETXSLTPARAM(
ctxHdl IN ctxType,
name IN VARCHAR2,
value IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
name	(IN)	Name of the top level stylesheet parameter.
value	(IN)	Value to be assigned to the stylesheet parameter.

USENULLATTRIBUTEINDICATOR

Specifies whether to use an XML attribute to indicate NULLness, or to do this by omitting the particular entity in the XML document.

Syntax

```
PROCEDURE SETNULLATTRIBUTEINDICATOR(
ctxHdl IN ctxType,
flag IN BOOLEAN);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
flag	(IN)	Sets attribute to NULL if TRUE, omits from XML document if FALSE.

USETYPEFORCOLLELEMTAG

Specifies whether to use the collection element's type name as its element tag name. By default, the tag name for elements of a collection is the collection's tag name followed by `_item`.

Syntax

```
PROCEDURE USETYPEFORCOLLELEMTAG(
ctxHdl IN ctxType,
flag IN BOOLEAN := true);
```

Parameter	IN / OUT	Description
`ctxHdl`	`(IN)`	Context handle.
`flag`	`(IN)`	Turn on use of the type name?

128

DBMS_XMLSAVE

DBMS_XMLSAVE provides XML to database-type functionality.

See Also: *Oracle XML DB Developer's Guide*

This chapter contains the following topics:

Using DBMS_XMLSAVE

Constants

Table 128–1 Constants of DBMS_XMLSAVE

Constant	Description
DEFAULT_ROWTAG	The default tag name for the element corresponding to database records -- ROW
DEFAULT_DATE_ FORMAT	Default date mask:'MM/dd/yyyy HH:mm:ss'
MATCH_CASE	Used to specify that when mapping XML elements to database entities; the XSU should be case sensitive.
IGNORE_CASE	Used to specify that when mapping XML elements to database. entities the XSU should be case insensitive.

Types

Table 128–2 Types of DBMS_XMLSAVE

Type	Description
`ctxType`	The type of the query context handle. The type of the query context handle. This the return type of NEWCONTEXT.

Summary of DBMS_XMLSAVE Subprograms

Table 128–3 DBMS_XMLSAVE Package Subprograms

Method	Description
CLEARKEYCOLUMNLIST on page 128-6	Clears the key column list.
CLEARUPDATECOLUMNLIST on page 128-7	Clears the update column list.
CLOSECONTEXT on page 128-8	It closes/deallocates a particular save context.
DELETEXML on page 128-9	Deletes records specified by data from the XML document, from the table specified at the context creation time.
GETEXCEPTIONCONTENT on page 128-10	Via its arguments, this method returns the thrown exception's error code and error message.
INSERTXML on page 128-11	Inserts the XML document into the table specified at the context creation time.
NEWCONTEXT on page 128-12	Creates a save context, and returns the context handle.
PROPAGATEORIGINALEXCEPTION on page 128-13	Tells the XSU that if an exception is raised, and is being thrown, the XSU should throw the very exception raised; rather then, wrapping it with an `OracleXMLSQLException`.
REMOVEXSLTPARAM on page 128-14	Removes the value of a top-level stylesheet parameter
SETBATCHSIZE on page 128-15	Changes the batch size used during DML operations.
SETCOMMITBATCH on page 128-17	Sets the commit batch size.
SETDATEFORMAT on page 128-17	Sets the format of the generated dates in the XML document.
SETIGNORECASE on page 128-18	The XSU does mapping of XML elements to database.
SETKEYCOLUMN on page 128-19	This methods adds a column to the key column list.
SETPRESERVEWHITESPACE on page 128-20	Tells the XSU whether to preserve whitespace or not.
SETROWTAG on page 128-21	Names the tag used in the XML document to enclose the XML elements corresponding to database.
SETSQLTOXMLNAMEESCAPING on page 128-22	This turns on or off escaping of XML tags in the case that the SQL object name, which is mapped to a XML identifier, is not a valid XML identifier.
SETUPDATECOLUMN on page 128-23	Adds a column to the update column list.
SETXSLT on page 128-24	Registers a XSL transform to be applied to the XML to be saved.
SETXSLTPARAM on page 128-25	Sets the value of a top-level stylesheet parameter.
UPDATEXML on page 128-26	Updates the table given the XML document.

CLEARKEYCOLUMNLIST

Clears the key column list.

Syntax

```
PROCEDURE clearKeyColumnList(
   ctxHdl IN ctxType);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.

CLEARUPDATECOLUMNLIST

Clears the update column list.

Syntax

```
PROCEDURE clearUpdateColumnList(
   ctxHdl IN ctxType);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.

CLOSECONTEXT

Closes/deallocates a particular save context.

Syntax

```
PROCEDURE closeContext(
   ctxHdl IN ctxType);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.

DELETEXML

Deletes records specified by data from the XML document from the table specified at the context creation time, and returns the number of rows deleted. The options are described in the following table.

Syntax	Description

`FUNCTION deleteXML(`
`ctxHdl IN ctxPType,`
`xDoc IN VARCHAR2)`
`RETURN NUMBER;`
	Uses a `VARCHAR2` type for the `xDoc` parameter.

`FUNCTION deleteXML(`
`ctxHdl IN ctxType,`
`xDoc IN CLOB)`
`RETURN NUMBER;`
	Uses a `CLOB` type for the `xDoc` parameter.

Parameter	IN / OUT	Description
`ctxHdl`	`(IN)`	Context handle.
`xDoc`	`(IN)`	String containing the XML document.

GETEXCEPTIONCONTENT

Through its arguments, this method returns the thrown exception's error code and error message, SQL error code. This is to get around the fact that the JVM throws an exception on top of whatever exception was raised; thus, rendering PL/SQL unable to access the original exception.

Syntax

```
PROCEDURE getExceptionContent(
   ctxHdl IN ctxType,
   errNo OUT NUMBER,
   errMsg OUT VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
errNo	(IN)	Error number.
errMsg	(IN)	Error message.

INSERTXML

Inserts the XML document into the table specified at the context creation time, and returns the number of rows inserted. The options are described in the following table.

Syntax	Description
`FUNCTION insertXML( ctxHdl IN ctxType, xDoc IN VARCHAR2) RETURN NUMBER;`	Passes in the `xDoc` parameter as a `VARCHAR2`.
`FUNCTION insertXML( ctxHdl IN ctxType, xDoc IN CLOB) RETURN NUMBER;`	Passes in the `xDoc` parameter as a `CLOB`.

Parameter	IN / OUT	Description
`ctxHdl`	`(IN)`	Context handle.
`xDoc`	`(IN)`	String containing the XML document.

NEWCONTEXT

Creates a save context, and returns the context handle.

Syntax

```
FUNCTION newContext(
   targetTable IN VARCHAR2)
RETURN ctxType;
```

Parameter	IN / OUT	Description
targetTable	(IN)	The target table into which to load the XML document.

PROPAGATEORIGINALEXCEPTION

Tells the XSU that if an exception is raised, and is being thrown, the XSU should throw the very exception raised; rather then, wrapping it with an `OracleXMLSQLException`.

Syntax

```
PROCEDURE propagateOriginalException(
   ctxHdl IN ctxType,
   flag IN BOOLEAN);
```

Parameter	IN / OUT	Description
`ctxHdl`	`(IN)`	Context handle.
`flag`	`(IN)`	Propagate the original exception? 0=`FALSE`, 1=`TRUE`.

REMOVEXSLTPARAM

Removes the value of a top-level stylesheet parameter.

Syntax

```
PROCEDURE removeXSLTParam(
   ctxHdl IN ctxType,
   name IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
name	(IN)	Parameter name.

SETBATCHSIZE

Changes the batch size used during DML operations. When performing inserts, updates or deletes, it is better to batch the operations so that they get executed in one shot rather than as separate statements. The flip side is that more memory is needed to buffer all the bind values. Note that when batching is used, a commit occurs only after a batch is executed. So if one of the statement inside a batch fails, the whole batch is rolled back. This is a small price to pay considering the performance gain; nevertheless, if this behavior is unacceptable, then set the batch size to 1.

Syntax

```
PROCEDURE setBatchSize(
   ctxHdl IN ctxType,
   batchSize IN NUMBER);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
batchSize	(IN)	Batch size.

SETCOMMITBATCH

Sets the commit batch size. The commit batch size refers to the number or records inserted after which a commit should follow. If `batchSize` is less than `1` or the session is in "auto-commit" mode, using the XSU does not make any explicit commits. By default, `commitBatch` is `0`.

Syntax

```
PROCEDURE setCommitBatch(
   ctxHdl IN ctxType,
   batchSize IN NUMBER);
```

Parameter	IN / OUT	Description
`ctxHdl`	`(IN)`	Context handle.
`batchSize`	`(IN)`	Commit batch size.

SETDATEFORMAT

Sets the format of the generated dates in the XML document. The syntax of the date format patern, the date mask, should conform to the requirements of the class `java.text.SimpleDateFormat`. Setting the mask to `<code>null</code>` or an empty string unsets the date mask.

Syntax

```
PROCEDURE setDateFormat(
   ctxHdl IN ctxType,
   mask IN VARCHAR2);
```

Parameter	IN / OUT	Description
`ctxHdl`	`(IN)`	Context handle.
`mask`	`(IN)`	Syntax of the date format pattern..

SETIGNORECASE

The XSU does mapping of XML elements to db columns/attributes based on the element names (XML tags). This function tells the XSU to do this match case insensitive.

Syntax

```
PROCEDURE setIgnoreCase(
   ctxHdl IN ctxType,
   flag IN NUMBER);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
flag	(IN)	Ignore tag case in the XML doc? 0=FALSE, 1=TRUE.

SETKEYCOLUMN

This method adds a column to the "key column list". The value for the column cannot be NULL. In case of update or delete, the columns in the key column list make up the WHERE clause of the statement. The key columns list must be specified before updates can complete; this is optional for delete operations.

Syntax

```
PROCEDURE setKeyColumn(
   ctxHdl IN ctxType,
   colName IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
colName	(IN)	Column to be added to the key column list; cannot be NULL.

SETPRESERVEWHITESPACE

Tells the XSU whether or not to preserve whitespace.

Syntax

```
PROCEDURE setPreserveWhitespace(
   ctxHdl IN ctxType,
   flag IN BOOLEAN := true);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
flag	(IN)	Should XSU preserve whitespace?

SETROWTAG

Names the tag used in the XML document to enclose the XML elements corresponding to db. records.

Syntax

```
PROCEDURE setRowTag(
   ctxHdl IN ctxType,
   tag IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
tag	(IN)	Tag name.

SETSQLTOXMLNAMEESCAPING

Turns on or off escaping of XML tags in the case that the SQL object name, which is mapped to a XML identifier, is not a valid XML identifier.

Syntax

```
PROCEDURE setSQLToXMLNameEscaping(
   ctxHdl IN ctxType,
   flag IN BOOLEAN := true);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
flag	(IN)	Turn on escaping?

SETUPDATECOLUMN

Adds a column to the update column list. In case of insert, the default is to insert values to all the columns in the table; on the other hand, in case of updates, the default is to only update the columns corresponding to the tags present in the ROW element of the XML document. When the update column list is specified, the columns making up this list alone will get updated or inserted into.

Syntax

```
PROCEDURE setUpdateColumn(
   ctxHdl IN ctxType,
   colName IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
colName	(IN)	Column to be added to the update column list.

SETXSLT

Registers an XSL transform to be applied to the XML to be saved. If a stylesheet was already registered, it gets replaced by the new one. To un-register the stylesheet, pass in null for the URI. The options are described in the following table.

Syntax	Description
`PROCEDURE setXSLT( ctxHdl IN ctxType, uri IN VARCHAR2, ref IN VARCHAR2 := null);`	Passes in the stylesheet through a URI.
`PROCEDURE setXSLT( ctxHdl IN ctxType, stylesheet IN CLOB, ref IN VARCHAR2 := null);`	Passes in the stylesheet through a `CLOB`.

Parameter	IN / OUT	Description
`ctxHdl`	`(IN)`	Context handle.
`uri`	`(IN)`	URI to the stylesheet to register.
`ref`	`(IN)`	URL for include, import, and external entities.
`stylesheet`	`(IN)`	`CLOB` containing the stylesheet to register .

SETXSLTPARAM

Sets the value of a top-level stylesheet parameter. The parameter is expected to be a valid XPath expression; literal values would therefore have to be explicitly quoted.

Syntax

```
PROCEDURE setXSLTParam(
   ctxHdl IN ctxType,
   name IN VARCHAR2,
   value IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
name	(IN)	Parameter name.
value	(IN)	Parameter value as an XPath expression

UPDATEXML

Updates the table specified at the context creation time with data from the XML document, and returns the number of rows updated. The options are described in the following table.

Syntax	Description
`FUNCTION updateXML( ctxHdl IN ctxType, xDoc IN VARCHAR2) RETURN NUMBER;`	Passes in the `xDoc` parameter as a `VARCHAR2`.
`FUNCTION updateXML( ctxHdl IN ctxType, xDoc IN CLOB) RETURN NUMBER;`	Passes in the `xDoc` parameter as a `CLOB`.

Parameter	IN / OUT	Description
`ctxHdl`	`(IN)`	Context handle.
`xDoc`	`(IN)`	String containing the XML document.

129

DBMS_XMLSCHEMA

DBMS_XMLSCHEMA package provides procedures to manage XML schemas. It is created by script dbmsxsch.sql during Oracle database installation.

See Also: *Oracle XML DB Developer's Guide*

This chapter contains the following topics:

Using DBMS_XMLSCHEMA

This section contains topics which relate to using the `DBMS_XMLSCHEMA` package.

Overview

This package provides subprograms to

- Register an XML schema
- Delete a previously registered XML schema
- Re-compile a previously registered XML schema
- Generate an XML schema
- Evolves an XML schema

Constants

The DBMS_XMLSCHEMA package uses the constants shown in following tables.

- DBMS_XMLSCHEMA Constants - Delete Option
- DBMS_XMLSCHEMA Constants - Enable Hierarchy
- DBMS_XMLSCHEMA Constants - Register CSID

Table 129–1 DBMS_XMLSCHEMA Constants - Delete Option

Constant	Type	Value	Description
DELETE_RESTRICT	NUMBER	1	Deletion of an XML schema fails if there are any tables or XML schemas that depend on it
DELETE_INVALIDATE	NUMBER	2	Deletion of an XML schema does not fail if there are tables or XML schemas that depend on it. All dependent tables and schemas are invalidated.
DELETE_CASCADE	NUMBER	3	Deletion of an XML schema also drops all SQL types and default tables associated with it. SQL types are dropped only if gentypes argument was set to TRUE during registration of the XML schema. However, deletion of the XML schema fails if there are any instance documents conforming to the schema or any dependent XML schemas.
DELETE_CASCADE_FORCE	NUMBER	4	This option is similar to DELETE_CASCADE except that it does not check for any stored instance documents conforming to the schema or any dependent XML schemas. Also, it ignores any errors.

Table 129–2 DBMS_XMLSCHEMA Constants - Enable Hierarchy

Constant	Type	Value	Description
ENABLE_HIERARCHY_NONE	PLS_INTEGER	1	The ENABLE_HIERARCHY procedure of the DBMS_XDBZ package will not be called on any tables created while registering that schema
ENABLE_HIERARCHY_CONTENTS	PLS_INTEGER	2	The ENABLE_HIERARCHY procedure of the DBMS_XDBZ package will be called for all tables created during schema registration with hierarchy_type as DBMS_XDBZ.ENABLE_CONTENTS

Table 129–2 (Cont.) DBMS_XMLSCHEMA Constants - Enable Hierarchy

Constant	Type	Value	Description
ENABLE_HIERARCHY_ RESMETADATA	PLS_INTEGER	3	The ENABLE_HIERARCHY procedure of the DBMS_XDBZ package will be called on all tables created during schema registration with hierarchy_ type as DBMS_XDBZ.ENABLE_ RESMETADATA. Users should pass in DBMS_ XMLSCHEMA.ENABLE_ RESMETADATA for schemas they intend to use as resource metadata tables.

Table 129–3 DBMS_XMLSCHEMA Constants - Register CSID

Constant	Type	Value	Description
REGISTER_NODOCID	NUMBER	1	If a schema is registered for metadata use (using the value ENABLE_HIER_ RESMETADATA for parameter enablehierarchy during registration), a column named DOCID is added to all tables created during schema registration. This constant can be used in the options argument of REGISTERSCHEMA to prevent the creation of this column if the user wishes to optimize on storage
REGISTER_CSID_NULL	NUMBER	-1	If user wishes to not specify the character set of the input schema document when invoking REGISTERSCHEMA, this value can be used for the csid parameter

Views

The DBMS_XMLSCHEMA package uses the views shown in Table 129–4. The columns of these views are described in detail in the *Oracle Database Reference.*

Table 129–4 Summary of Views used by DBMS_XMLSCHEMA

Schema	Description
USER_XML_SCHEMAS	All registered XML Schemas owned by the user
ALL_XML_SCHEMAS	All registered XML Schemas usable by the current user
DBA_XML_SCHEMAS	All registered XML Schemas in the database
DBA_XML_TABLES	All XMLType tables in the system
USER_XML_TABLES	All XMLType tables owned by the current user
ALL_XML_TABLES	All XMLType tables usable by the current user
DBA_XML_TAB_COLS	All XMLType table columns in the system
USER_XML_TAB_COLS	All XMLType table columns in tables owned by the current user
ALL_XML_TAB_COLS	All XMLType table columns in tables usable by the current user
DBA_XML_VIEWS	All XMLType views in the system
USER_XML_VIEWS	All XMlType views owned by the current user
ALL_XML_VIEWS	All XMLType views usable by the current user
DBA_XML_VIEW_COLS	All XMLType view columns in the system
USER_XML_VIEW_COLS	All XMLType view columns in views owned by the current user
ALL_XML_VIEW_COLS	All XMLType view columns in views usable by the current user

Summary of DBMS_XMLSCHEMA Subprograms

Table 129–5 DBMS_XMLSCHEMA Package Subprograms

Method	Description
COMPILESCHEMA Procedure on page 129-8	Used to re-compile an already registered XML schema. This is useful for bringing a schema in an invalid state to a valid state.
COPYEVOLVE Procedure on page 129-9	Evolves registered schemas so that existing XML instances remain valid
DELETESCHEMA Procedure on page 129-11	Removes the schema from the database
GENERATEBEAN Procedure on page 129-12	Generates the Java bean code corresponding to a registered XML schema
GENERATESCHEMA Function on page 129-13	Generates an XML schema from an oracle type name
GENERATESCHEMAS Function on page 129-14	Generates several XML schemas from an oracle type name
REGISTERSCHEMA Procedures on page 129-15	Registers the specified schema for use by Oracle. This schema can then be used to store documents conforming to this.
REGISTERURI Procedure on page 129-19	Registers an XML schema specified by a URI name

COMPILESCHEMA Procedure

This procedure can be used to re-compile an already registered XML schema. This is useful for bringing a schema in an invalid state to a valid state. Can result in a `ORA-31001` exception: invalid resource handle or path name.

Syntax

```
DBMS_XMLSCHEMA.COMPILESCHEMA(
   schemaurl IN VARCHAR2);
```

Parameters

Table 129–6 COMPILESCHEMA Procedure Parameters

Parameter	Description
`schemaurl`	URL identifying the schema

COPYEVOLVE Procedure

This procedure evolves registered schemas so that existing XML instances remain valid.

This procedure is accomplished in according to the following basic scenario (alternative actions are controlled by the procedure's parameters):

- copies data in schema based `XMLType` tables to temporary table storage
- drops old tables
- deletes old schemas
- registers new schemas
- creates new `XMLType` tables
- Populates new tables with data in temporary storage; auxiliary structures (constraints, triggers, indexes, and others) are not preserved
- drops temporary tables

See Also:

- "Schema Evolution" chapter of the *Oracle XML DB Developer's Guide* for examples on how to evolve existing schemas
- *Oracle Database Error Messages* for information on exceptions specific to schema evolution, ORA-30142 through ORA-30946.

Syntax

```
DBMS_XMLSCHEMA.COPYEVOLVE(
   schemaurls       IN  XDB$STRUBG_LIST_T,
   newschemas       IN  XMLSequenceType,
   transforms       IN  XMLSequenceType :=NULL,
   preserveolddocs  IN  BOOLEAN :=FALSE,
   maptablename     IN  VARCHAR2 :=NULL,
   generatetables   IN  BOOLEAN :=TRUE,
   force            IN  BOOLEAN :=FALSE,
   schemaowners     IN  XDB$STRING_LIST_T :=NULL);
```

Parameters

Table 129–7 COPYEVOLVE Procedure Parameters

Parameter	Description
`schemaurls`	`VARRAY` of URLs of all schemas to be evolved. Should include the dependent schemas. Unless the `FORCE` parameter is `TRUE`, URLs should be in the order of dependency.
`newschemas`	`VARRAY` of new schema documents. Should be specified in same order as the corresponding URLs.
`transforms`	`VARRAY` of transforming XSL documents to be applied to schema-based documents. Should be specified in same order as the corresponding URLs. Optional if no transformations are required.
`preserveolddocs`	Default is `FALSE`, and temporary tables with old data are dropped. If `TRUE`, these table are still available after schema evolution is complete.

Table 129–7 (Cont.) COPYEVOLVE Procedure Parameters

Parameter	Description
maptabname	Specifies the name of the table mapping permanent to temporary tables during the evolution process. Valid columns are: ■ SCHEMA_URL - VARCHAR2(700) - URL of schema to which this table conforms ■ SCHEMA_OWNER -VARCHAR2(30) - Owner of the schema ■ ELEMENT_NAME - VARCHAR2(256)- Element to which this table conforms ■ TAB_NAME - VARCHAR2(65) - Qualified table name: *<owner_name>.<table_name>* ■ COL_NAME - VARCHAR2(4000) - Name of the column (NULL for XMLType tables) ■ TEMP_TABNAME - VARCHAR2(30) - Name of temporary tables which holds data for this table.
generatetables	Default is TRUE, and new tables will be generated. If FALSE: ■ new tables will not be generated after registration of new schemas ■ preserveolddocs must be TRUE ■ maptablename must be non-NULL
force	Default is FALSE. If TRUE, ignores errors generated during schema evolution. Used when there are circular dependencies among schemas to ensure that all schemas are stored despite possible errors in registration.
schemaowners	VARRAY of names of schema owners. Should be specified in same order as the corresponding URLs. Default is NULL, assuming that all schemas are owned by the current user.

Usage Notes

You should back up all schemas and documents prior to invocation because COPYEVOLVE Procedure deletes all conforming documents prior to implementing the schema evolution.

DELETESCHEMA Procedure

This procedure deletes the XML Schema specified by the URL.

Syntax

```
DBMS_XMLSCHEMA.DELETESCHEMA(
   schemaurl      IN  VARCHAR2,
   delete_option  IN  PLS_INTEGER := DELETE_RESTRICT);
```

> **See Also:** "XMLSCHEMA Storage and Query: Basic" chapter of the *Oracle XML DB Developer's Guide*

Parameters

Table 129–8 DELETESCHEMA Procedure Parameters

Parameter	Description
`schemaurl`	URL identifying the schema to be deleted

Exceptions

Table 129–9 DELETESCHEMA Procedure Exceptions

Exception	Description
`ORA-31001`	Invalid resource handle or path name

GENERATEBEAN Procedure

This procedure can be used to generate the Java bean code corresponding to a registered XML schema.

Syntax

```
DBMS_XMLSCHEMA.GENERATEBEAN(
   schemaurl  IN  VARCHAR2);
```

Parameters

Table 129–10 GENERATEBEAN Procedure Parameters

Parameter	Description
schemaurl	Name identifying a registered XML schema

Exceptions

Table 129–11 GENERATEBEAN Procedure Exceptions

Exception	Description
ORA-31001	Invalid resource handle or path name

Usage Notes

Note that there is also an option to generate the beans as part of the registration procedure itself (see the genbean parameter of the REGISTERSCHEMA Procedures on page 129-15).

GENERATESCHEMA Function

This function generates XML schema(s) from an Oracle type name. It inlines all in one schema (`XMLType`).

> **See Also:** "XMLSCHEMA Storage and Query: Advanced" chapter of the *Oracle XML DB Developer's Guide*

Syntax

```
DBMS_XMLSCHEMA.GENERATESCHEMA(
   schemaname    IN  VARCHAR2,
   typename      IN  VARCHAR2,
   elementname   IN  VARCHAR2 := NULL,
   recurse       IN  BOOLEAN  := TRUE,
   annotate      IN  BOOLEAN  := TRUE,
   embedcoll     IN  BOOLEAN  := TRUE)
RETURN SYS.XMLTYPE;
```

Parameters

Table 129–12 GENERATESCHEMA Function Parameters

Parameter	Description
`schemaname`	Name of the database schema containing the type
`typename`	Name of the Oracle type
`elementname`	The name of the top level element in the XML Schema. Defaults to `typename`.
`recurse`	Whether or not to also generate schema for all types referred to by the type specified
`annotate`	Whether or not to put the SQL annotations in the XML Schema
`embedcoll`	Determines whether the collections should be embedded in the type which refers to them, or create a `complextype`. Cannot be `FALSE` if annotations are turned on

Exceptions

Table 129–13 GENERATESCHEMA Procedure Exceptions

Exception	Description
`ORA-31001`	Invalid resource handle or path name

GENERATESCHEMAS Function

This function generates XML schema(s) from an Oracle type name. It returns a collection of XMLTypes, one XML Schema document for each database schema.

> **See Also:** "XMLSCHEMA Storage and Query: Advanced" chapter of the *Oracle XML DB Developer's Guide*

Syntax

```
DBMS_XMLSCHEMA.GENERATESCHEMAS(
   schemaname   IN  VARCHAR2,
   typename     IN  VARCHAR2,
   elementname  IN  VARCHAR2 := NULL,
   schemaurl    IN  VARCHAR2 := NULL,
   annotate     IN  BOOLEAN := TRUE,
   embedcoll    IN  BOOLEAN := TRUE )
 RETURN SYS.XMLTYPE;
```

Parameters

Table 129–14 GENERATESCHEMAS Procedure Parameters

Parameter	Description
schemaname	Name of the database schema containing the type
typename	Name of the Oracle type
elementname	The name of the top level element in the XML Schema defaults to typeName
schemaurl	Specifies base URL where schemas will be stored, needed by top level schema for import statement
annotate	Whether or not to put the SQL annotations in the XML Schema
embedcoll	Determines whether the collections be embedded in the type which refers to them, or create a complextype. Cannot be FALSE if annotations are turned on

Exceptions

Table 129–15 GENERATESCHEMAS Procedure Exceptions

Exception	Description
ORA-31001	Invalid resource handle or path name

REGISTERSCHEMA Procedures

This procedure registers the specified schema for use by the database. The procedure is overloaded. The different functionality of each form of syntax is presented along with the definition.

> **See Also:** "XMLSCHEMA Storage and Query: Basic" chapter of the *Oracle XML DB Developer's Guide*

Syntax

Registers a schema specified as a VARCHAR2:

```
DBMS_XMLSCHEMA.REGISTERSCHEMA(
    schemaurl        IN  VARCHAR2,
    schemadoc        IN  VARCHAR2,
    local            IN  BOOLEAN := TRUE,
    gentypes         IN  BOOLEAN := TRUE,
    genbean          IN  BOOLEAN := FALSE,
    gentables        IN  BOOLEAN := TRUE,
    force            IN  BOOLEAN := FALSE,
    owner            IN  VARCHAR2 := NULL,
    enablehierarchy  IN  PLS_INTEGER := DBMS_XMLSCHEMA.ENABLE_CONTENTS,
    options          IN  PLS_INTEGER := 0);
```

Registers the schema specified as a BFILE. The contents of the schema document must be in the database character set:

```
DBMS_XMLSCHEMA.REGISTERSCHEMA(
   schemaurl        IN  VARCHAR2,
   schemadoc        IN  BFILE,
   local            IN  BOOLEAN := TRUE,
   gentypes         IN  BOOLEAN := TRUE,
   genbean          IN  BOOLEAN := FALSE,
   force            IN  BOOLEAN := FALSE,
   owner            IN  VARCHAR2 := NULL,
   enablehierarchy  IN  PLS_INTEGER := DBMS_XMLSCHEMA.ENABLE_CONTENTS,
   options          IN  PLS_INTEGER := 0);
```

Registers the schema specified as a BFILE and identifies the character set id of the schema document:

```
DBMS_XMLSCHEMA.REGISTERSCHEMA(
   schemaurl        IN  VARCHAR2,
   schemadoc        IN  BFILE,
   local            IN  BOOLEAN := TRUE,
   gentypes         IN  BOOLEAN := TRUE,
   genbean          IN  BOOLEAN := TRUE,
   gentables        IN  BOOLEAN := TRUE,
   force            IN  BOOLEAN := TRUE,
   owner            IN  VARCHAR2 := '',
   csid             IN  NUMBER,
   enablehierarchy  IN  PLS_INTEGER := DBMS_XMLSCHEMA.ENABLE_CONTENTS,
   options          IN  PLS_INTEGER := 0);
```

Registers the schema specified as a BLOB. The contents of the schema document must be in the database character set:

```
DBMS_XMLSCHEMA.REGISTERSCHEMA(
   schemaurl        IN  VARCHAR2,
```

```
   schemadoc        IN  BLOB,
   local            IN  BOOLEAN := TRUE,
   genTypes         IN  BOOLEAN := TRUE,
   genBean          IN  BOOLEAN := FASLE,
   force            IN  BOOLEAN := FALSE,
   owner            IN  VARCHAR2 := NULL,
   enablehierarchy  IN  PLS_INTEGER := DBMS_XMLSCHEMA.ENABLE_CONTENTS,
   options          IN  PLS_INTEGER := 0);
```

Registers the schema specified as a `BLOB` and identifies the character set id of the schema document:

```
DBMS_XMLSCHEMA.REGISTERSCHEMA(
   schemaurl        IN  VARCHAR2,
   schemadoc        IN  BLOB,
   local            IN  BOOLEAN := TRUE,
   gentypes         IN  BOOLEAN := TRUE,
   genbean          IN  BOOLEAN := TRUE,
   gentables        IN  BOOLEAN := TRUE,
   force            IN  BOOLEAN := TRUE,
   owner            IN  VARCHAR2 := '',
   csid             IN  NUMBER,
   enablehierarchy  IN  PLS_INTEGER := DBMS_XMLSCHEMA.ENABLE_CONTENTS,
   options          IN  PLS_INTEGER := 0);
```

Registers the schema specified as a `CLOB`

```
DBMS_XMLSCHEMA.REGISTERSCHEMA(
   schemaurl        IN  VARCHAR2,
   schemadoc        IN  CLOB,
   local            IN  BOOLEAN := TRUE,
   gentypes         IN  BOOLEAN := TRUE,
   genbean          IN  BOOLEAN := FALSE,
   force            IN  BOOLEAN := FALSE,
   owner            IN  VARCHAR2 := NULL,
   options          IN  PLS_INTEGER := 0);
```

Registers the schema specified as an `XMLTYPE`.

```
DBMS_XMLSCHEMA.REGISTERSCHEMA(
   schemaurl        IN  VARCHAR2,
   schemadoc        IN  SYS.XMLTYPE,
   local            IN  BOOLEAN := TRUE,
   gentypes         IN  BOOLEAN := TRUE,
   genbean          IN  BOOLEAN := FALSE,
   force            IN  BOOLEAN := FALSE,
   owner            IN  VARCHAR2 := NULL,
   enablehierarchy  IN  PLS_INTEGER := DBMS_XMLSCHEMA.ENABLE_CONTENTS,
   options          IN  PLS_INTEGER := 0);
```

Registers the schema specified as a `BLOB`. The contents of the schema document must be in the database character set:

```
DBMS_XMLSCHEMA.REGISTERSCHEMA(
   schemaurl        IN  VARCHAR2,
   schemadoc        IN  SYS.URIType,
   local            IN  BOOLEAN := TRUE,
   gentypes         IN  BOOLEAN := TRUE,
   genbean          IN  BOOLEAN := FALSE,
   force            IN  BOOLEAN := FALSE,
   owner            IN  VARCHAR2 := NULL,
   enablehierarchy  IN  PLS_INTEGER := DBMS_XMLSCHEMA.ENABLE_CONTENTS,
```

```
   options          IN  PLS_INTEGER := 0);
```

Parameters

Table 129–16 REGSITERSCHEMA Procedure Parameters

Parameter	Description
schemaurl	URL that uniquely identifies the schema document. This value is used to derive the path name of the schema document within the database hierarchy. Can be used inside `schemalocation` attribute of XML Schema import element.
schemadoc	A valid XML schema document
local	Is this a local or global schema? ■ By default, all schemas are registered as local schemas, under `/sys/schemas/<username>/...` ■ If a schema is registered as global, it is added under `/sys/schemas/PUBLIC/...` You need write privileges on the directory to be able to register a schema as global.
gentypes	Determines whether the schema compiler generates object types. By default, `TRUE`.
genbean	Determines whether the schema compiler generates Java beans. By default, `FALSE`.
gentables	Determines whether the schema compiler generates default tables. By default, `TRUE`
force	If this parameter is set to `TRUE`, the schema registration will not raise errors. Instead, it creates an invalid XML schema object in case of any errors. By default, the value of this parameter is `FALSE`.
owner	This parameter specifies the name of the database user owning the XML schema object. By default, the user registering the schema owns the XML schema object. This parameter can be used to register a XML schema to be owned by a different database user.
csid	Identifies the character set of the input schema document. If this value is `0`, the schema document's encoding is determined by the current rule for "text/xml" MIME type.
enablehierarchy	■ `ENABLE_HIERARCHY_NONE` - enable hierarchy will not be called on any tables created while registering that schema ■ `ENABLE_HIERARCHY_CONTENTS` - enable hierarchy will be called for all tables created during schema registration with `hierarchy_type` as `DBMS_XDBZ.ENABLE_CONTENTS`. This is the default. ■ `ENABLE_HIERARCHY_RESMETADATA` - enable hierarchy will be called on all tables created during schema registration with `hierarchy_type` as `DBMS_XDBZ.ENABLE_RESMETADATA`. Users should pass in `DBMS_XMLSCHEMA.ENABLE_RESMETADATA` for schemas they intend to use as resource metadata tables.

Table 129–16 (Cont.) REGSITERSCHEMA Procedure Parameters

Parameter	Description
`options`	Additional options to specify how the schema should be registered. The various options are represented as bits of an integer and the options parameter should be constructed by doing a `BITOR` of the desired bits. Possible bits: ▪ `REGISTER_NODOCID` - this will suppress the creation of the `DOCID` column for out of line tables. This is a storage optimization which might be desirable when we do not need to join back to the document table (for example if we do not care about rewriting certain queries that could be rewritten by making use of the `DOCID` column)

REGISTERURI Procedure

This procedure registers an XML Schema specified by a URI name.

Syntax

```
DBMS_XMLSCHEMA.REGISTERURI(
   schemaurl      IN  VARCHAR2,
   schemadocuri   IN  VARCHAR2,
   local          IN  BOOLEAN := TRUE,
   gentypes       IN  BOOLEAN := TRUE,
   genbean        IN  BOOLEAN := FALSE,
   gentables      IN  BOOLEAN := TRUE,
   force          IN  BOOLEAN := FALSE,
   owner          IN  VARCHAR2 := NULL,
   options          IN  PLS_INTEGER := 0);
```

Parameters

Table 129–17 REGISTERURI Procedure Parameters

Parameter	Description
`schemaurl`	Uniquely identifies the schema document. Can be used inside `schemaLocation` attribute of XML Schema import element.
`schemadocuri`	Pathname (URI) corresponding to the physical location of the schema document. The URI path could be based on HTTP, FTP, DB or Oracle XML DB protocols. This function constructs a `URIType` instance using the `urifactory` - and invokes the REGISTERSCHEMA Procedures function.
`local`	Determines whether this is a local or global schema. By default, all schemas are registered as local schemas, under `/sys/schemas/ <username>/...` If a schema is registered as global, it is added under `/sys/schemas/PUBLIC/...` The user needs write privileges on the directory to register a global schema.
`gentypes`	Determines whether the compiler generate object types. By default, `TRUE`.
`genbean`	Determines whether the compiler generate Java beans. By default, `FALSE`.
`gentables`	Determines whether the compiler generate default tables. `TRUE` by default.
`force`	`TRUE`: schema registration will not raise errors. Instead, it creates an invalid XML schema object in case of any errors. By default, the value of this parameter is `FALSE`.
`owner`	This parameter specifies the name of the database user owning the XML schema object. By default, the user registering the schema owns the XML schema object. This parameter can be used to register a XML schema to be owned by a different database user.

Table 129–17 (Cont.) REGISTERURI Procedure Parameters

Parameter	Description
`options`	Additional options to specify how the schema should be registered. The various options are represented as bits of an integer and the options parameter should be constructed by doing a `BITOR` of the desired bits. Possible bits: ▪ `REGISTER_NODOCID` - this will suppress the creation of the `DOCID` column for out of line tables. This is a storage optimization which might be desirable when we do not need to join back to the document table (for example if we do not care about rewriting certain queries that could be rewritten by making use of the `DOCID` column)

130

DBMS_XMLSTORE

DBMS_XMLSTORE provides the ability to store XML data in relational tables.

See Also:

- *Oracle XML DB Developer's Guide*

This chapter contains the following sections:

Using DBMS_XMLSTORE

Types

Table 130–1 Types of DBMS_XMLSTORE

Type	Description
`ctxType`	The type of the query context handle. This is the return type of NEWCONTEXT.

Summary of DBMS_XMLSTORE Subprograms

Table 130–2 DBMS_XMLSTORE Package Subprograms

Method	Description
CLEARKEYCOLUMNLIST on page 130-5	Clears the key column list.
CLEARUPDATECOLUMNLIST on page 130-6	Clears the update column list.
CLOSECONTEXT on page 130-7	It closes/deallocates a particular save context.
DELETEXML on page 130-8	Deletes records specified by data from the XML document, from the table specified at the context creation time.
INSERTXML on page 130-9	Inserts the XML document into the table specified at the context creation time.
NEWCONTEXT on page 130-10	Creates a save context, and returns the context handle.
SETKEYCOLUMN on page 130-11	This method adds a column to the key column list.
SETROWTAG on page 130-12	Names the tag used in the XML document., to enclose the XML elements corresponding to the database.
SETUPDATECOLUMN on page 130-13	Adds a column to the "update column list".
UPDATEXML on page 130-14	Updates the table given the XML document.

CLEARKEYCOLUMNLIST

Clears the key column list.

Syntax

```
PROCEDURE clearKeyColumnList(
   ctxHdl IN ctxType);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.

CLEARUPDATECOLUMNLIST

Clears the update column list.

Syntax

```
PROCEDURE clearUpdateColumnList(
   ctxHdl IN ctxType);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.

CLOSECONTEXT

Closes/deallocates a particular save context.

Syntax

```
PROCEDURE closeContext(
   ctxHdl IN ctxType);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.

DELETEXML

Deletes records specified by data from the XML document from the table specified at the context creation time, and returns the number of rows deleted.

Syntax	Description

`FUNCTION deleteXML(`
`ctxHdl IN ctxPType,`
`xDoc IN VARCHAR2)`
`RETURN NUMBER;`
	Uses a `VARCHAR2` type for the `xDoc` parameter.

`FUNCTION deleteXML(`
`ctxHdl IN ctxType,`
`xDoc IN CLOB)`
`RETURN NUMBER;`
	Uses a `CLOB` type for the `xDoc` parameter.

`FUNCTION deleteXML(`
`ctxHdl IN ctxType,`
`xDoc IN XMLType)`
`RETURN NUMBER;`
	Uses an `XMLType` type for the `xDoc` parameter.

Parameter	IN / OUT	Description
`ctxHdl`	`(IN)`	Context handle.
`xDoc`	`(IN)`	String containing the XML document.

INSERTXML

Inserts the XML document into the table specified at the context creation time, and returns the number of rows inserted.

Syntax	Description
`FUNCTION insertXML( ctxHdl IN ctxType, xDoc IN VARCHAR2) RETURN NUMBER;`	Passes in the `xDoc` parameter as a `VARCHAR2`.
`FUNCTION insertXML( ctxHdl IN ctxType, xDoc IN CLOB) RETURN NUMBER;`	Passes in the `xDoc` parameter as a `CLOB`.
`FUNCTION insertXML( ctxHdl IN ctxType, xDoc IN XMLType) RETURN NUMBER;`	Passes in the `xDoc` parameter as an `XMLType`.

Parameter	IN / OUT	Description
`ctxHdl`	`(IN)`	Context handle.
`xDoc`	`(IN)`	String containing the XML document.

NEWCONTEXT

Creates a save context, and returns the context handle.

Syntax

```
FUNCTION newContext(
   targetTable IN VARCHAR2)
RETURN ctxType;
```

Parameter	IN / OUT	Description
targetTable	(IN)	The target table into which to load the XML document.

SETKEYCOLUMN

This method adds a column to the "key column list". The value for the column cannot be NULL. In case of update or delete, the columns in the key column list make up the WHERE clause of the statement. The key columns list must be specified before updates can complete; this is optional for delete operations

Syntax

```
PROCEDURE setKeyColumn(
   ctxHdl IN ctxType,
   colName IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
colName	(IN)	Column to be added to the key column list; cannot be NULL.

SETROWTAG

Names the tag used in the XML document, to enclose the XML elements corresponding to databse records.

Syntax

```
PROCEDURE setRowTag(
   ctxHdl IN ctxType,
   tag IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
tag	(IN)	Tag name.

SETUPDATECOLUMN

Adds a column to the update column list. In case of insert, the default is to insert values to all the columns in the table; on the other hand, in case of updates, the default is to only update the columns corresponding to the tags present in the `ROW` element of the XML document. When the update column list is specified, the columns making up this list alone will get updated or inserted into.

Syntax

```
PROCEDURE setUpdateColumn(
   ctxHdl IN ctxType,
   colName IN VARCHAR2);
```

Parameter	IN / OUT	Description
`ctxHdl`	`(IN)`	Context handle.
`colName`	`(IN)`	Column to be added to the update column list.

UPDATEXML

Updates the table specified at the context creation time with data from the XML document, and returns the number of rows updated. The options are described in the following table.

Syntax	Description
`FUNCTION updateXML( ctxHdl IN ctxType, xDoc IN VARCHAR2) RETURN NUMBER;`	Passes in the `xDoc` parameter as a `VARCHAR2`.
`FUNCTION updateXML( ctxHdl IN ctxType, xDoc IN CLOB) RETURN NUMBER;`	Passes in the `xDoc` parameter as a `CLOB`.
`FUNCTION updateXML( ctxHdl IN ctxType, xDoc IN XMLType) RETURN NUMBER;`	Passes in the `xDoc` parameter as a `XMLType`.

Parameter	IN / OUT	Description
`ctxHdl`	`(IN)`	Context handle.
`xDoc`	`(IN)`	String containing the XML document.

131

DBMS_XPLAN

The DBMS_XPLAN package provides an easy way to display the output of the EXPLAIN PLAN command in several, predefined formats. You can also use the DBMS_XPLAN package to display the plan of a statement stored in the Automatic Workload Repository (AWR) or stored in a SQL tuning set. It further provides a way to display the SQL execution plan and SQL execution runtime statistics for cached SQL cursors based on the information stored in the V$SQL_PLAN and V$SQL_PLAN_STATISTICS_ALL fixed views.

See Also:

- For more information on the EXPLAIN PLAN command, the AWR, and SQL tuning set, see *Oracle Database Performance Tuning Guide*.
- For more information on the V$SQL_PLAN and V$SQL_PLAN_STATISTICS fixed views, see *Oracle Database Reference.*

This chapter contains the following topics:

Using DBMS_XPLAN

Overview

The DBMS_XPLAN package supplies four table functions:

- DISPLAY - to format and display the contents of a plan table.
- DISPLAY_CURSOR - to format and display the contents of the execution plan of any loaded cursor.
- DISPLAY_AWR - to format and display the contents of the execution plan of a stored SQL statement in the AWR.
- DISPLAY_SQLSET - to format and display the contents of the execution plan of statements stored in a SQL tuning set.

Security Model

This package runs with the privileges of the calling user, not the package owner (SYS). The table function DISPLAY_CURSOR requires to have select privileges on the following fixed views: VSQL_PLAN, VSESSION and V$SQL_PLAN_STATISTICS_ALL.

Using the DISPLAY_AWR function requires the user to have SELECT privileges on DBA_HIST_SQL_PLAN, DBA_HIST_SQLTEXT, and V$DATABASE.

To use the DISPLAY_SQLSET functionality, the calling user must have SELECT privilege on ALL_SQLSET_STATEMENTS and ALL_SQLSET_PLANS.

All these privileges are automatically granted as part of the SELECT_CATALOG role.

Examples

Displaying a Plan Table Using DBMS_XPLAN.DISPLAY

Execute an explain plan command on a SELECT statement:

```
EXPLAIN PLAN FOR
SELECT * FROM emp e, dept d
   WHERE e.deptno = d.deptno
   AND e.ename='benoit';
```

Display the plan using the DBMS_XPLAN.DISPLAY table function

```
SET LINESIZE 130
SET PAGESIZE 0
SELECT * FROM table(DBMS_XPLAN.DISPLAY);
```

This query produces the following output:

```
Plan hash value: 3693697075
--------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |    57 |     6  (34)| 00:00:01 |
|*  1 |  HASH JOIN         |      |     1 |    57 |     6  (34)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL| EMP  |     1 |    37 |     3  (34)| 00:00:01 |
|   3 |   TABLE ACCESS FULL| DEPT |     4 |    80 |     3  (34)| 00:00:01 |
--------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("E"."DEPTNO"="D"."DEPTNO")
   2 - filter("E"."ENAME"='benoit')

15 rows selected.
```

Displaying a Cursor Execution Plan Using DBMS_XPLAN.DISPLAY_CURSOR

By default, the table function DISPLAY_CURSOR formats the execution plan for the last SQL statement executed by the session. For example:

```
SELECT ename  FROM  emp e, dept d
   WHERE   e.deptno = d.deptno
   AND   e.empno=7369;

ENAME
----------
SMITH
```

To display the execution plan of the last executed statement for that session:

```
SET PAGESIZE 0
SELECT * FROM table(DBMS_XPLAN.DISPLAY_CURSOR);
```

This query produces the following output:

```
Plan hash value: 3693697075, SQL hash value: 2096952573, child number: 0
-------------------------------------------------------------------
SELECT ename FROM emp e, dept d WHERE e.deptno = d.deptno
AND e.empno=7369

--------------------------------------------------------------------------------
```

```
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |       |       |            |          |
|*  1 |  HASH JOIN         |      |     1 |    16 |     6  (34)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL| EMP  |     1 |    13 |     3  (34)| 00:00:01 |
|   3 |   TABLE ACCESS FULL| DEPT |     4 |    12 |     3  (34)| 00:00:01 |
---------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("E"."DEPTNO"="D"."DEPTNO")
   2 - filter("E"."EMPNO"=7369)

21 rows selected.
```

You can also use the table function `DISPLAY_CURSOR` to display the execution plan for any loaded cursor stored in the cursor cache. In that case, you must supply a reference to the child cursor to the table function. This includes the SQL ID of the statement and optionally the child number.

Run a query with a distinctive comment:

```
SELECT /* TOTO */ ename, dname
FROM dept d join emp e USING (deptno);
```

Get `sql_id` and `child_number` for the preceding statement:

```
SELECT sql_id, child_number
FROM v$sql
WHERE sql_text LIKE '%TOTO%';

SQL_ID         CHILD_NUMBER
----------     ------------------------------
gwp663cqh5qbf   0
```

Display the execution plan for the cursor:

```
SELECT * FROM table(DBMS_XPLAN.DISPLAY_CURSOR(('gwp663cqh5qbf',0));

Plan hash value: 3693697075, SQL ID: gwp663cqh5qbf, child number: 0
-------------------------------------------------------
SELECT /* TOTO */ ename, dname
FROM dept d JOIN emp e USING (deptno);

-----------------------------------------------------------------------------
| Id  | Operation           | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |      |       |       |     7 (100)|          |
|   1 |  SORT GROUP BY      |      |     4 |    64 |     7  (43)| 00:00:01 |
|*  2 |   HASH JOIN         |      |    14 |   224 |     6  (34)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| DEPT |     4 |    44 |     3  (34)| 00:00:01 |
|   4 |    TABLE ACCESS FULL| EMP  |    14 |    70 |     3  (34)| 00:00:01 |
-----------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("E"."DEPTNO"="D"."DEPTNO")
```

Instead of issuing two queries, one to the get the sql_id and child_number pair and one to display the plan, you can combine these in a single query:

Display the execution plan of all cursors matching the string 'TOTO':

```
SELECT t.*
FROM v$sql s, table(DBMS_XPLAN.DISPLAY_CURSOR(s.sql_id, s.child_number)) t WHERE
sql_text LIKE '%TOTO%';
```

Displaying a Plan Table with Parallel Information

By default, only relevant information is reported by the display and `display_cursor` table functions. In Displaying a Plan Table Using DBMS_XPLAN.DISPLAY on page 131-5, the query does not execute in parallel. Hence, information related to the parallelization of the plan is not reported. As shown in the following example, parallel information is reported only if the query executes in parallel.

```
ALTER TABLE emp PARALLEL;
EXPLAIN PLAN for
SELECT * FROM emp e, dept d
   WHERE e.deptno = d.deptno
   AND e.ename    ='hermann'
   ORDER BY e.empno;
```

Display the plan using the `DBMS_XPLAN.DISPLAY` table function

```
SET LINESIZE 130
SET PAGESIZE 0
SELECT * FROM table(DBMS_XPLAN.DISPLAY);
Plan hash value: 3693697345
```

```
-------------------------------------------------------------------------------------------------------------
| Id | Operation          | Name    | Rows | Bytes | Cost (%CPU)| Time     | TQ    |INOUT |PQ Distrib   |
-------------------------------------------------------------------------------------------------------------
| 0  | SELECT STATEMENT   |         | 1    | 117   | 6 (50)     | 00:00:01 |       |      |             |
| 1  | PX COORDINATOR     |         |      |       |            |          |       |      |             |
| 2  | PX SEND QC (ORDER) |:TQ10003 | 1    | 117   | 6 (50)     | 00:00:01 | Q1,03 | P->S | QC (ORDER)  |
| 3  | SORT ORDER BY      |         | 1    | 117   | 6 (50)     | 00:00:01 | Q1,03 | PCWP |             |
| 4  | PX RECEIVE         |         | 1    | 117   | 5 (40)     | 00:00:01 | Q1,03 | PCWP |             |
| 5  | PX SEND RANGE      |:TQ10002 | 1    | 117   | 5 (40)     | 00:00:01 | Q1,02 | P->P | RANGE       |
|* 6 | HASH JOIN          |         | 1    | 117   | 5 (40)     | 00:00:01 | Q1,02 | PCWP |             |
| 7  | PX RECEIVE         |         | 1    | 87    | 2 (50)     | 00:00:01 | Q1,02 | PCWP |             |
| 8  | PX SEND HASH       |:TQ10001 | 1    | 87    | 2 (50)     | 00:00:01 | Q1,01 | P->P | HASH        |
| 9  | PX BLOCK ITERATOR  |         | 1    | 87    | 2 (50)     | 00:00:01 | Q1,01 | PCWC |             |
|* 10| TABLE ACCESS FULL  | EMP     | 1    | 87    | 2 (50)     | 00:00:01 | Q1,01 | PCWP |             |
| 11 | BUFFER SORT        |         |      |       |            |          | Q1,02 | PCWC |             |
| 12 | PX RECEIVE         |         | 4    | 120   | 3 (34)     | 00:00:01 | Q1,02 | PCWP |             |
| 13 | PX SEND HASH       |:TQ10000 | 4    | 120   | 3 (34)     | 00:00:01 |       | S->P | HASH        |
| 14 | TABLE ACCESS FULL  | DEPT    | 4    | 120   | 3 (34)     | 00:00:01 |       |      |             |
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
6 - access("E"."DEPTNO"="D"."DEPTNO")
10 - filter("E"."ENAME"='hermann')
---------------------------------------------------
```

When the query is parallel, information related to parallelism is reported: table queue number (`TQ` column), table queue type (`INOUT`) and table queue distribution method (`PQ Distrib`).

By default, if several plans in the plan table match the `statement_id` parameter passed to the display table function (default value is `NULL`), only the plan corresponding to the last `EXPLAIN PLAN` command is displayed. Hence, there is no need to purge the plan table after each `EXPLAIN PLAN`. However, you should purge the plan table regularly to ensure good performance in the execution of the `DISPLAY` table function. If no plan table is created, Oracle will use a global temporary table to store any plan information for individual users and will preserve its content

throughout the lifespan of a session. Note that you cannot truncate the content of a global temporary table.

For ease of use, you can define a view on top of the display table function and then use that view to display the output of the `EXPLAIN PLAN` command:

Using a View to Display Last Explain Plan

```
# define plan view
CREATE VIEW PLAN AS SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

# display the output of the last explain plan command
SELECT * FROM PLAN;
```

Summary of DBMS_XPLAN Subprograms

Table 131–1 DBMS_XPLAN Package Subprograms

Subprogram	Description
DISPLAY Function on page 131-10	Displays the contents of the plan table
DISPLAY_AWR Function on page 131-13	Displays the contents of an execution plan stored in the AWR
DISPLAY_CURSOR Function on page 131-16	Displays the execution plan of any cursor in the cursor cache
DISPLAY_SQLSET Function on page 131-19	Displays the execution plan of a given statement stored in a SQL tuning set

DISPLAY Function

This table function displays the contents of the plan table.

In addition, you can use this table function to display any plan (with or without statistics) stored in a table as long as the columns of this table are named the same as columns of the plan table (or V$SQL_PLAN_STATISTICS_ALL if statistics are included). You can apply a predicate on the specified table to select rows of the plan to display.

Syntax

```
DBMS_XPLAN.DISPLAY(
   table_name    IN  VARCHAR2  DEFAULT 'PLAN_TABLE',
   statement_id  IN  VARCHAR2  DEFAULT  NULL,
   format        IN  VARCHAR2  DEFAULT  'TYPICAL',
   filter_preds  IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 131–2 DISPLAY Function Parameters

Parameter	Description
table_name	Specifies the table name where the plan is stored. This parameter defaults to PLAN_TABLE, which is the default plan table for the EXPLAIN PLAN command. If NULL is specified it also defaults to PLAN_TABLE.
statement_id	Specifies the statement_id of the plan to be displayed. This parameter defaults to NULL, which is the default when the EXPLAIN PLAN command is executed without a set statement_id clause.If no statement_id is specified, the function will show you the plan of the most recent explained statement.

Table 131–2 (Cont.) DISPLAY Function Parameters

Parameter	Description
format	Controls the level of details for the plan. It accepts four values: ■ BASIC: Displays the minimum information in the plan—the operation ID, the operation name and its option. ■ TYPICAL: This is the default. Displays the most relevant information in the plan (operation id, name and option, #rows, #bytes and optimizer cost). Pruning, parallel and predicate information are only displayed when applicable. Excludes only PROJECTION, ALIAS and REMOTE SQL information (see below). ■ SERIAL: Like TYPICAL except that the parallel information is not displayed, even if the plan executes in parallel. ■ ALL: Maximum user level. Includes information displayed with the TYPICAL level with additional information (PROJECTION, ALIAS and information about REMOTE SQL if the operation is distributed). For finer control on the display output, the following keywords can be added to the above three standard format options to customize their default behavior. Each keyword either represents a logical group of plan table columns (such as PARTITION) or logical additions to the base plan table output (such as PREDICATE). Format keywords must be separated by either a comma or a space: ■ ROWS - if relevant, shows the number of rows estimated by the optimizer ■ BYTES - if relevant, shows the number of bytes estimated by the optimizer ■ COST - if relevant, shows optimizer cost information ■ PARTITION - if relevant, shows partition pruning information ■ PARALLEL - if relevant, shows PX information (distribution method and table queue information) ■ PREDICATE - if relevant, shows the predicate section ■ PROJECTION -if relevant, shows the projection section ■ ALIAS - if relevant, shows the "Query Block Name / Object Alias" section ■ REMOTE - if relevant, shows the information for distributed query (for example, remote from serial distribution and remote SQL) ■ NOTE - if relevant, shows the note section of the explain plan Format keywords can be prefixed by the sign '-' to exclude the specified information. For example, '-PROJECTION' excludes projection information. If the target plan table (see table_name parameter) also stores plan statistics columns (for example, it is a table used to capture the content of the fixed view V$SQL_PLAN_STATISTICS_ALL), additional format keywords can be used to specify which class of statistics to display when using the DISPLAY Function. These additional format keywords are IOSTATS, MEMSTATS, ALLSTATS and LAST (see the DISPLAY_CURSOR Function or the DISPLAY_SQLSET Function for a full description of these four keywords).

Table 131–2 (Cont.) DISPLAY Function Parameters

Parameter	Description
filter_preds	SQL filter predicate(s) to restrict the set of rows selected from the table where the plan is stored. When value is NULL (the default), the plan displayed corresponds to the last executed explain plan. For example: filter_preds=>'plan_id = 10' Can reference any column of the table where the plan is stored and can contain any SQL construct (for example, sub-query, function calls (see **WARNING** under Usage Notes)

Usage Notes

Here are some ways you might use variations on the format parameter:

- Use 'ALL -PROJECTION -NOTE' to display everything except the projection and note sections.
- Use 'TYPICAL PROJECTION' to display using the typical format with the additional projection section (which is normally excluded under the typical format). Since typical is default, using simply 'PROJECTION' is equivalent.
- Use '-BYTES -COST -PREDICATE' to display using the typical format but excluding optimizer cost and byte estimates as well as the predicate section.
- Use 'BASIC ROWS' to display basic information with the additional number of rows estimated by the optimizer.

WARNING: Application developers should expose the filter_preds **parameter to end-users only after careful consideration because this could expose the application to SQL injection. Indeed,** filter_preds **can potentially reference any table or execute any server function for which the database user invoking the table function has privileges.**

Examples

To display the result of the last EXPLAIN PLAN command stored in the plan table:

```
SELECT * FROM table (DBMS_XPLAN.DISPLAY);
```

To display from other than the default plan table, "my_plan_table":

```
SELECT * FROM table (DBMS_XPLAN.DISPLAY('my_plan_table'));
```

To display the minimum plan information:

```
SELECT * FROM table (DBMS_XPLAN.DISPLAY('plan_table', null, 'basic'));
```

To display the plan for a statement identified by 'foo', such as statement_id='foo':

```
SELECT * FROM table (DBMS_XPLAN.DISPLAY('plan_table', 'foo'));
```

DISPLAY_AWR Function

This table function displays the contents of an execution plan stored in the AWR.

Syntax

```
DBMS_XPLAN.DISPLAY_AWR(
   sql_id            IN      VARCHAR2,
   plan_hash_value   IN      NUMBER DEFAULT NULL,
   db_id             IN      NUMBER DEFAULT NULL,
   format            IN      VARCHAR2 DEFAULT TYPICAL);
```

Parameters

Table 131–3 DISPLAY_AWR Table Function Parameters

Parameter	Description
sql_id	Specifies the SQL_ID of the SQL statement. You can retrieve the appropriate value for the SQL statement of interest by querying the column SQL_ID in DBA_HIST_SQLTEXT.
plan_hash_value	Specifies the PLAN_HASH_VALUE of a SQL statement. This parameter is optional. If omitted, the table function will return all stored execution plans for a given SQL_ID.
db_id	Specifies the database_id for which the plan of the SQL statement, identified by SQL_ID should be displayed. If not supplied, the database_id of the local database will be used, as shown in V$DATABASE.
format	Controls the level of details for the plan. It accepts four values: ■ BASIC: Displays the minimum information in the plan—the operation ID, the operation name and its option. ■ TYPICAL: This is the default. Displays the most relevant information in the plan (operation id, name and option, #rows, #bytes and optimizer cost). Pruning, parallel and predicate information are only displayed when applicable. Excludes only PROJECTION, ALIAS and REMOTE SQL information (see below). ■ SERIAL: Like TYPICAL except that the parallel information is not displayed, even if the plan executes in parallel. ■ ALL: Maximum user level. Includes information displayed with the TYPICAL level with additional information (PROJECTION, ALIAS and information about REMOTE SQL if the operation is distributed).

Table 131–3 (Cont.) DISPLAY_AWR Table Function Parameters

Parameter	Description
	For finer control on the display output, the following keywords can be added to the above three standard format options to customize their default behavior. Each keyword either represents a logical group of plan table columns (such as PARTITION) or logical additions to the base plan table output (such as PREDICATE). Format keywords must be separated by either a comma or a space: ■ ROWS - if relevant, shows the number of rows estimated by the optimizer ■ BYTES - if relevant, shows the number of bytes estimated by the optimizer ■ COST - if relevant, shows optimizer cost information ■ PARTITION - if relevant, shows partition pruning information ■ PARALLEL - if relevant, shows PX information (distribution method and table queue information) ■ PREDICATE - if relevant, shows the predicate section ■ PROJECTION -if relevant, shows the projection section ■ ALIAS - if relevant, shows the "Query Block Name / Object Alias" section ■ REMOTE - if relevant, shows the information for distributed query (for example, remote from serial distribution and remote SQL) ■ NOTE - if relevant, shows the note section of the explain plan Format keywords can be prefixed by the sign '-' to exclude the specified information. For example, '-PROJECTION' excludes projection information.

Usage Notes

- To use the DISPLAY_AWR functionality, the calling user must have SELECT privilege on DBA_HIST_SQL_PLAN. DBA_HIST_SQLTEXT, and V$DATABASE, otherwise it will show an appropriate error message.
- Here are some ways you might use variations on the format parameter:
  - Use 'ALL -PROJECTION -NOTE' to display everything except the projection and note sections.
  - Use 'TYPICAL PROJECTION' to display using the typical format with the additional projection section (which is normally excluded under the typical format). Since typical is default, using simply 'PROJECTION' is equivalent.
  - Use '-BYTES -COST -PREDICATE' to display using the typical format but excluding optimizer cost and byte estimates as well as the predicate section.
  - Use 'BASIC ROWS' to display basic information with the additional number of rows estimated by the optimizer.

Examples

To display the different execution plans associated with the SQL ID 'atfwcg8anrykp':

```
SELECT * FROM table(DBMS_XPLAN.DISPLAY_AWR('atfwcg8anrykp'));
```

To display all execution plans of all stored SQL statements containing the string 'TOTO':

```
SELECT tf.* FROM DBA_HIST_SQLTEXT ht, table
    (DBMS_XPLAN.DISPLAY_AWR(ht.sql_id,null, null,  'ALL' )) tf
 WHERE ht.sql_text like '%TOTO%';
```

DISPLAY_CURSOR Function

This table function displays the explain plan of any cursor loaded in the cursor cache. In addition to the explain plan, various plan statistics (such as. I/O, memory and timing) can be reported (based on the V$SQL_PLAN_STATISTICS_ALL VIEWS).

Syntax

```
DBMS_XPLAN.DISPLAY_CURSOR(
   sql_id        IN  VARCHAR2  DEFAULT  NULL,
   child_number  IN  NUMBER    DEFAULT  NULL,
   format        IN  VARCHAR2  DEFAULT  'TYPICAL');
```

Parameters

Table 131–4 DISPLAY_CURSOR Function Parameters

Parameter	Description
sql_id	Specifies the SQL_ID of the SQL statement in the cursor cache. You can retrieve the appropriate value by querying the column SQL_ID in V$SQL or V$SQLAREA. Alternatively, you could choose the column PREV_SQL_ID for a specific session out of V$SESSION. This parameter defaults to NULL in which case the plan of the last cursor executed by the session will be displayed.
child_number	Child number of the cursor to display. If not supplied, the execution plan of all cursors matching the supplied sql_id parameter are displayed. The child_number can be specified only if sql_id is specified.
format	Controls the level of details for the plan. It accepts four values: ■ BASIC: Displays the minimum information in the plan—the operation ID, the operation name and its option. ■ TYPICAL: This is the default. Displays the most relevant information in the plan (operation id, name and option, #rows, #bytes and optimizer cost). Pruning, parallel and predicate information are only displayed when applicable. Excludes only PROJECTION, ALIAS and REMOTE SQL information (see below). ■ SERIAL: Like TYPICAL except that the parallel information is not displayed, even if the plan executes in parallel. ■ ALL: Maximum user level. Includes information displayed with the TYPICAL level with additional information (PROJECTION, ALIAS and information about REMOTE SQL if the operation is distributed). For finer control on the display output, the following keywords can be added to the above three standard format options to customize their default behavior. Each keyword either represents a logical group of plan table columns (such as PARTITION) or logical additions to the base plan table output (such as PREDICATE).

Table 131–4 (Cont.) DISPLAY_CURSOR Function Parameters

Parameter	Description
	Format keywords must be separated by either a comma or a space: ■ ROWS - if relevant, shows the number of rows estimated by the optimizer ■ BYTES - if relevant, shows the number of bytes estimated by the optimizer ■ COST - if relevant, shows optimizer cost information ■ PARTITION - if relevant, shows partition pruning information ■ PARALLEL - if relevant, shows PX information (distribution method and table queue information) ■ PREDICATE - if relevant, shows the predicate section ■ PROJECTION -if relevant, shows the projection section ■ ALIAS - if relevant, shows the "Query Block Name / Object Alias" section ■ REMOTE - if relevant, shows the information for distributed query (for example, remote from serial distribution and remote SQL) ■ NOTE - if relevant, shows the note section of the explain plan ■ IOSTATS - assuming that basic plan statistics are collected when SQL statements are executed (either by using the gather_plan_statistics hint or by setting the parameter statistics_level to ALL), this format will show IO statistics for ALL (or only for the LAST as shown below) executions of the cursor. ■ MEMSTATS - Assuming that PGA memory management is enabled (that is, pga_aggregate_target parameter is set to a non 0 value), this format allows to display memory management statistics (for example, execution mode of the operator, how much memory was used, number of bytes spilled to disk, and so on). These statistics only apply to memory intensive operations like hash-joins, sort or some bitmap operators. ■ ALLSTATS - A shortcut for 'IOSTATS MEMSTATS' ■ LAST - By default, plan statistics are shown for all executions of the cursor. The keyword LAST can be specified to see only the statistics for the last execution. The following two formats are deprecated but supported for backward compatibility: ■ RUNSTATS_TOT - Same as IOSTATS, that is, displays IO statistics for all executions of the specified cursor. ■ RUNSTATS_LAST - Same as IOSTATS LAST, that is, displays the runtime statistics for the last execution of the cursor Format keywords can be prefixed by the sign '-' to exclude the specified information. For example, '-PROJECTION' excludes projection information.

Usage Notes

- To use the DISPLAY_CURSOR functionality, the calling user must have SELECT privilege on the fixed views V$SQL_PLAN_STATISTICS_ALL, V$SQL and V$SQL_PLAN, otherwise it will show an appropriate error message.
- Here are some ways you might use variations on the format parameter:

- Use 'ALL -PROJECTION -NOTE' to display everything except the projection and note sections.
- Use 'TYPICAL PROJECTION' to display using the typical format with the additional projection section (which is normally excluded under the typical format). Since typical is default, using simply 'PROJECTION' is equivalent.
- Use '-BYTES -COST -PREDICATE' to display using the typical format but excluding optimizer cost and byte estimates as well as the predicate section.
- Use 'BASIC ROWS' to display basic information with the additional number of rows estimated by the optimizer.

Examples

To display the execution plan of the last SQL statement executed by the current session:

```
SELECT * FROM table (
   DBMS_XPLAN.DISPLAY_CURSOR);
```

To display the execution plan of all children associated with the SQL ID 'atfwcg8anrykp':

```
SELECT * FROM table (
   DBMS_XPLAN.DISPLAY_CURSOR('atfwcg8anrykp'));
```

To display runtime statistics for the cursor included in the preceding statement:

```
SELECT * FROM table (
   DBMS_XPLAN.DISPLAY_CURSOR('atfwcg8anrykp', NULL, 'ALLSTATS LAST');
```

DISPLAY_SQLSET Function

This table function displays the execution plan of a given statement stored in a SQL tuning set.

Syntax

```
DBMS_XPLAN.DISPLAY_SQLSET(
   sqlset_name      IN  VARCHAR2,
   sql_id           IN  VARCHAR2,
   plan_hash_value  IN NUMBER := NULL,
   format           IN  VARCHAR2  := 'TYPICAL',
   sqlset_owner     IN  VARCHAR2  := NULL)
  RETURN DBMS_XPLAN_TYPE_TABLE PIPELINED;
```

Parameters

Table 131–5 DISPLAY_SQLSET Function Parameters

Parameter	Description
sqlset_name	Name of the SQL Tuning Set
sql_id	Specifies the sql_id value for a SQL statement having its plan stored in the SQL tuning set. You can find all stored SQL statements by querying table function DBMS_SQLTUNE.SELECT_SQLSET
plan_hash_value	Optional parameter. Identifies a specific stored execution plan for a SQL statement. If suppressed, all stored execution plans are shown.
format	Controls the level of details for the plan. It accepts four values: ■ BASIC: Displays the minimum information in the plan—the operation ID, the operation name and its option. ■ TYPICAL: This is the default. Displays the most relevant information in the plan (operation id, name and option, #rows, #bytes and optimizer cost). Pruning, parallel and predicate information are only displayed when applicable. Excludes only PROJECTION, ALIAS and REMOTE SQL information (see below). ■ SERIAL: Like TYPICAL except that the parallel information is not displayed, even if the plan executes in parallel. ■ ALL: Maximum user level. Includes information displayed with the TYPICAL level with additional information (PROJECTION, ALIAS and information about REMOTE SQL if the operation is distributed).

Table 131–5 (Cont.) DISPLAY_SQLSET Function Parameters

Parameter	Description
	For finer control on the display output, the following keywords can be added to the above three standard format options to customize their default behavior. Each keyword either represents a logical group of plan table columns (such as `PARTITION`) or logical additions to the base plan table output (such as `PREDICATE`). Format keywords must be separated by either a comma or a space: ■ `ROWS` - if relevant, shows the number of rows estimated by the optimizer ■ `BYTES` - if relevant, shows the number of bytes estimated by the optimizer ■ `COST` - if relevant, shows optimizer cost information ■ `PARTITION` - if relevant, shows partition pruning information ■ `PARALLEL` - if relevant, shows PX information (distribution method and table queue information) ■ `PREDICATE` - if relevant, shows the predicate section ■ `PROJECTION` -if relevant, shows the projection section ■ `ALIAS` - if relevant, shows the "Query Block Name / Object Alias" section ■ `REMOTE` - if relevant, shows the information for distributed query (for example, remote from serial distribution and remote SQL) ■ `NOTE` - if relevant, shows the note section of the explain plan ■ `IOSTATS` - assuming that basic plan statistics are collected when SQL statements are executed (either by using the `gather_plan_statistics` hint or by setting the parameter statistics_level to ALL), this format will show IO statistics for `ALL` (or only for the `LAST` as shown below) executions of the cursor. ■ `MEMSTATS` - Assuming that PGA memory management is enabled (that is, `pga_aggregate_target` parameter is set to a non 0 value), this format allows to display memory management statistics (for example, execution mode of the operator, how much memory was used, number of bytes spilled to disk, and so on). These statistics only apply to memory intensive operations like hash-joins, sort or some bitmap operators. ■ `ALLSTATS` - A shortcut for `'IOSTATS MEMSTATS'` ■ `LAST` - By default, plan statistics are shown for all executions of the cursor. The keyword `LAST` can be specified to see only the statistics for the last execution. The following two formats are deprecated but supported for backward compatibility: ■ `RUNSTATS_TOT` - Same as `IOSTATS`, that is, displays IO statistics for all executions of the specified cursor. ■ `RUNSTATS_LAST` - Same as `IOSTATS LAST`, that is, displays the runtime statistics for the last execution of the cursor Format keywords can be prefixed by the sign '`-`' to exclude the specified information. For example, '`-PROJECTION`' excludes projection information.
`sqlset_owner`	The owner of the SQL tuning set. The default is the current user.

Usage Notes

Here are some ways you might use variations on the `format` parameter:

- Use '`ALL -PROJECTION -NOTE`' to display everything except the projection and note sections.
- Use '`TYPICAL PROJECTION`' to display using the typical format with the additional projection section (which is normally excluded under the typical format). Since typical is default, using simply '`PROJECTION`' is equivalent.
- Use '`-BYTES -COST -PREDICATE`' to display using the typical format but excluding optimizer cost and byte estimates as well as the predicate section.
- Use '`BASIC ROWS`' to display basic information with the additional number of rows estimated by the optimizer.

Examples

To display the execution plan for the SQL statement associated with SQL ID '`gwp663cqh5qbf`' and `PLAN HASH 3693697075` in the SQL Tuning Set called '`OLTP_optimization_0405`":

```
SELECT * FROM table (
   DBMS_XPLAN.DISPLAY_SQLSET(
       'OLTP_optimization_0405','gwp663cqh5qbf', 3693697075));
```

To display all execution plans of the SQL ID '`atfwcg8anrykp`' stored in the SQL tuning set:

```
SELECT * FROM table (
   DBMS_XPLAN.DISPLAY_SQLSET(
      'OLTP_optimization_0405','gwp663cqh5qbf'));
```

To display runtime statistics for the SQL statement included in the preceding statement:

```
SELECT * FROM table (
   DBMS_XPLAN.DISPLAY_SQLSET(
      'OLTP_optimization_0405', 'gwp663cqh5qbf', NULL, 'ALLSTATS LAST');
```

132

DBMS_XSLPROCESSOR

The DBMS_XSLPROCESSOR package provides an interface to manage the contents and structure of XML documents.

This chapter contains the following topics:

See Also:

- *Oracle XML DB Developer's Guide*

Using DBMS_XSLPROCESSOR

This section contains topics which relate to using the `DBMS_XSLPROCESSOR` package.

Overview

The DBMS_XSLPROCESSOR package provides an interface to manage the contents and structure of XML documents.

Standards

This PL/SQL implementation of the XSL processor follows the W3C XSLT working draft rev WD-xslt-19990813 and includes the required behavior of an XSL processor in terms of how it must read XSLT stylesheets and the transformation it must effect.

Concepts

The Extensible Stylesheet Language Transformation (XSLT) describes rules for transforming a source tree into a result tree. A transformation expressed in XSLT is called a stylesheet. The transformation specified is achieved by associating patterns with templates defined in the stylesheet. A template is instantiated to create part of the result tree.

Implementation

The following is the default behavior for this PL/SQL XSL Processor:

- A result tree which can be accessed by DOM programmatic interface
- Errors are not recorded unless an error log is specified; however, an application error will be raised if parsing fails

Summary of DBMS_XSLPROCESSOR Subprograms

Table 132–1 DBMS_XSLPROCESSOR Package Subprograms

Method	Description
CLOB2FILE Procedure on page 132-5	Writes content of a `CLOB` into a file
FREEPROCESSOR Procedure on page 132-6	Frees a processor object
FREESTYLESHEET Procedure on page 132-7	Frees a stylesheet object
NEWPROCESSOR Function on page 132-7	Returns a new processor instance
NEWSTYLESHEET Functions on page 132-9	Creates a new stylesheet from input and reference URLs
PROCESSXSL Functions and Procedures on page 132-10	Transforms an input XML document
READ2CLOB Function on page 132-12	Reads content of the file into a `CLOB`
REMOVEPARAM Procedure on page 132-13	Removes a top-level stylesheet parameter
RESETPARAMS Procedure on page 132-14	Resets the top-level stylesheet parameters
SELECTNODES Function on page 132-15	Selects nodes from a DOM tree that match a pattern
SELECTSINGLENODE Function on page 132-16	Selects the first node from the tree that matches a pattern
SETERRORLOG Procedure on page 132-17	Sets errors to be sent to the specified file
SETPARAM Procedure on page 132-18	Sets a top-level parameter in the stylesheet
SHOWWARNINGS Procedure on page 132-19	Turns warnings on or off
TRANSFORMNODE Function on page 132-20	Transforms a node in a DOM tree using a stylesheet
VALUEOF Function and Procedure on page 132-21	Gets the value of the first node that matches a pattern

CLOB2FILE Procedure

This procedure writes content of a CLOB into a file.

Syntax

```
DBMS_XSLPROCESSOR.CLOB2FILE(
  cl          IN  CLOB;
  flocation   IN  VARCHAR2,
  fname       IN  VARCHAR2,
  csid        IN  NUMBER:=0);
```

Parameters

Table 132–2 CLOB2FILE Procedure Parameters

Parameter	Description
CLOB	File directory
flocation	File directory
fname	File name
csid	Character set id of the file ■ Must be a valid Oracle id; otherwise returns an error ■ If 0, content of the output file will be in the database character set

FREEPROCESSOR Procedure

This procedure Frees a `Processor` object.

Syntax

```
DBMS_XSLPROCESSOR.FREEPROCESSOR(
  p  IN  Processor);
```

Parameters

Table 132–3 FREEPROCESSOR Procedure Parameters

Parameter	Description
p	Processor

FREESTYLESHEET Procedure

This procedure frees a `Stylesheet` object.

Syntax

```
DBMS_XSLPROCESSOR.FREESTYLESHEET(
  ss  IN  Stylesheet);
```

Parameters

Table 132–4 FREESTYLESHEET Procedure Parameters

Parameter	Description
`ss`	Stylesheet

NEWPROCESSOR Function

This function returns a new `Processor` instance. The function must be called before the default behavior of `Processor` can be changed and if other processor methods need to be used.

Syntax

```
DBMS_XSLPROCESSOR.NEWPROCESSOR
 RETURN Processor;
```

NEWSTYLESHEET Functions

This function creates and returns a new `Stylesheet` instance. The options are described in the following table.

Syntax

Creates and returns a new stylesheet instance using the given `DOMDOCUMENT` and reference URLs:

```
DBMS_XSLPROCESSOR.NEWSTYLESHEET(
   xmldoc  IN   DOMDOCUMENT,
   ref     IN   VARCHAR2)
 RETURN Stylesheet;
```

Creates and returns a new `Stylesheet` instance using the given input and reference URLs:

```
DBMS_XSLPROCESSOR.NEWSTYLESHEET(
   inp    IN   VARCHAR2,
   ref    IN   VARCHAR2)
 RETURN Stylesheet;
```

Parameters

Table 132–5 NEWSTYLESHEET Function Parameters

Parameter	Description
`xmldoc`	`DOMDocument` to use for construction
`inp`	Input URL to use for construction
`ref`	Reference URL

PROCESSXSL Functions and Procedures

This function transforms input XMLDocument. Any changes to the default processor behavior should be effected before calling this procedure. An application error is raised if processing fails.

Syntax

Transforms input XMLDocument using given DOMDocument and stylesheet, and returns the resultant document fragment:

```
DBMS_XSLPROCESSOR.PROCESSXSL(
   p      IN   Processor,
   ss     IN   Stylesheet,
   xmldoc IN   DOMDOCUMENT),
 RETURN DOMDOCUMENTFRAGMENT;
```

Transforms input XMLDocument using given document as URL and the Stylesheet, and returns the resultant document fragment:

```
DBMS_XSLPROCESSOR.PROCESSXSL(
   p     IN   Processor,
   ss    IN   Stylesheet,
   url   IN   VARCHAR2,
 RETURN DOMDOCUMENTFRAGMENT;
```

Transforms input XMLDocument using given document as CLOB and the Stylesheet, and returns the resultant document fragment:

```
DBMS_XSLPROCESSOR.PROCESSXSL(
   p     IN   Processor,
   ss    IN  Stylesheet,
   clb   IN  CLOB)
 RETURN DOMDOCUMENTFRAGMENT;
```

Transforms input XMLDocument using given DOMDOCUMENT and the stylesheet, and writes the output to the specified file:

```
DBMS_XSLPROCESSOR.DBMS_XSLPROCESSOR.(
   p           IN    Processor,
   ss          IN    Stylesheet,
   xmldoc      IN    DOMDOCUMENT,
   dir         IN    VARCHAR2,
   fileName    IN    VARCHAR2);
```

Transforms input XMLDocument using given URL and the stylesheet, and writes the output to the specified file in a specified directory:

```
DBMS_XSLPROCESSOR.PROCESSXSL(
   p          IN    Processor,
   ss         IN    Stylesheet,
   url        IN    VARCHAR2,
   dir        IN    VARCHAR2,
   fileName   IN    VARCHAR2);
```

Transforms input XMLDocument using given DOMDOCUMENT and the stylesheet, and writes the output to a CLOB:

```
DBMS_XSLPROCESSOR.PROCESSXSL(
   p          IN      Processor,
```

```
   ss         IN      Stylesheet,
   xmldoc     IN      DOMDOCUMENT,
   cl         IN OUT  CLOB);
```

Transforms input XMLDocument using given DOMDOCUMENTFRAGMENT and the stylesheet, and returns the resultant document fragment:

```
DBMS_XSLPROCESSOR.PROCESSXSL(
   p         IN    Processor,
   ss        IN    Stylesheet,
   xmldf     IN    DOMDOCUMENTFRAGMENT)
 RETURN DOMDOCUMENTFRAGMENT;
```

Transforms input XMLDocumentFragment using given DOMDocumentFragment and the stylesheet, and writes the output to the specified file in a specified directory:

```
DBMS_XSLPROCESSOR.PROCESSXSL(
   p         IN    Processor,
   ss        IN    Stylesheet,
   xmldf     IN    DOMDOCUMENTFRAGMENT,
   dir       IN    VARCHAR2,
   filename  IN    VARCHAR2);
```

Transforms input XMLDocumentFragment using given DOMDOCUMENTFRAGMENT and the stylesheet, and writes the output to a buffer:

```
DBMS_XSLPROCESSOR.PROCESSXSL(
   p          IN       Processor,
   ss         IN       Stylesheet,
   xmldf      IN       DOMDOCUMENTFRAGMENT,
   buf        IN OUT   VARCHAR2);
```

Transforms input XMLDocumentFragment using given DOMDOCUMENTFRAGMENT and the stylesheet, and writes the output to a CLOB:

```
DBMS_XSLPROCESSOR.PROCESSXSL(
   p          IN      Processor,
   ss         IN      Stylesheet,
   xmldf      IN      DOMDOCUMENTFRAGMENT,
   cl         IN OUT  CLOB);
```

Parameters

Table 132–6 PROCESSXSL Function and Procedure Parameters

Parameter	Description
p	Processor instance
ss	Stylesheet instance
xmldoc	XML document being transformed
url	URL for the information being transformed
clb	CLOB containing information to be transformed
dir	Directory where processing output file is saved
filename	Processing output file
cl	CLOB to which the processing output is saved
xmldf	XMLDocumentFragment being transformed

READ2CLOB Function

This function reads content of a file into a CLOB.

Syntax

```
DBMS_XSLPROCESSOR.READ2CLOB(
   flocation     IN   VARCHAR2,
   fname         IN   VARCHAR2,
   csid          IN   NUMBER:=0)
 RETURN CLOB;
```

Parameters

Table 132–7 READ2CLOB Function Parameters

Parameter	Description
flocation	File directory
fname	File name
csid	Character set id of the file ■ Must be a valid Oracle id; otherwise returns an error ■ If 0, input file is assumed to be in the database character set

REMOVEPARAM Procedure

This procedure removes a top level stylesheet parameter.

Syntax

```
DBMS_XSLPROCESSOR.REMOVEPARAM(
   ss     IN  Stylesheet,
   name   IN  VARCHAR2);
```

Parameters

Table 132–8 REMOVEPARAM Procedure Parameters

Parameter	Description
ss	`Stylesheet` instance
name	Name of the parameter

RESETPARAMS Procedure

This procedure resets the top-level stylesheet parameters.

Syntax

```
DBMS_XSLPROCESSOR.RESETPARAMS(
   ss  IN   Stylesheet);
```

Parameters

Table 132–9 RESETPARAMS Procedure Parameters

Parameter	Description
ss	Stylesheet instance

SELECTNODES Function

This function selects nodes which match the given pattern from a DOM tree, and returns the result of the selection.

Syntax

```
DBMS_XSLPROCESSOR.SELECTNODES(
   n        IN   DOMNODE,
   pattern  IN   VARCHAR2)
 RETURN DOMNODELIST;
```

Parameters

Table 132–10 SELECTNODES Function Parameters

Parameter	Description
`n`	Root `DOMNode` of the tree
`pattern`	Pattern to use

SELECTSINGLENODE Function

This function selects the first node from the tree that matches the given pattern, and returns that node.

Syntax

```
DBMS_XSLPROCESSOR.SELECTSINGLENODE(
   n         IN   DOMNODE,
   pattern   IN   VARCHAR2)
 RETURN DOMNode;
```

Parameters

Table 132–11

Parameter	Description
`n`	Root `DOMNode` of the tree
`pattern`	Pattern to use

SETERRORLOG Procedure

This procedure sets errors to be sent to the specified file.

> **Note:** This subprogram has been deprecated, and is included only for reasons of backward compatibility.

Syntax

```
DBMS_XSLPROCESSOR.SETERRORLOG(
   p         IN   Processor,
   fileName  IN   VARCHAR2);
```

Parameters

Table 132–12 SETERRORLOG Procedure Parameters

Parameter	Description
`p`	`Processor` instance
`fileName`	Complete path of the file to use as the error log

SETPARAM Procedure

This procedure sets a top level parameter in the stylesheet. The parameter value must be a valid `XPath` expression. Literal string values must be quoted.

Syntax

```
DBMS_XSLPROCESSOR.SETPARAM(
   ss      IN   Stylesheet,
   name    IN   VARCHAR2,
   value   IN   VARCHAR2);
```

Parameters

Table 132–13 SETPARAM Procedure Parameters

Parameter	Description
`ss`	`Stylesheet` instance
`name`	Name of the parameter
`value`	Value of the parameter

SHOWWARNINGS Procedure

This procedure turns warnings on (TRUE) or off (FALSE).

Syntax

```
DBMS_XSLPROCESSOR.SHOWWARNINGS(
   p     IN   Processor,
   yes   IN   BOOLEAN);
```

Parameters

Table 132–14 SHOWWARNINGS Procedure Parameters

Parameter	Description
p	Processor instance
yes	Mode to set: TRUE to show warnings, FALSE otherwise

TRANSFORMNODE Function

This function transforms a node in a DOM tree using the given stylesheet, and returns the result of the transformation as a `DOMDocumentFragment`.

Syntax

```
DBMS_XSLPROCESSOR.TRANSFORMNODE(
   n    IN  DOMNODE,
   ss   IN  Stylesheet)
 RETURN DOMDocumentFragment;
```

Parameters

Table 132–15 TRANSFORMNODE Function Parameters

Parameter	Description
`n`	`DOMNode` to transform
`ss`	Stylesheet to use

VALUEOF Function and Procedure

This subprogram retrieves the value of the first node from the tree that matches the given pattern. You can use either a function or a procedure.

Syntax

```
DBMS_XSLPROCESSOR.VALUEOF(
  n           IN    DBMS_XMLDOM.DOMNODE,
  pattern     IN    VARCHAR2,
  namespace   IN    VARCHAR2 := NULL)
 RETURN VARCHAR2;

DBMS_XSLPROCESSOR.VALUEOF(
  n           IN    DBMS_XMLDOM.DOMNODE,
  pattern     IN    VARCHAR2,
  val         OUT   VARCHAR2,
  namespace   IN    VARCHAR2 := NULL);
```

Parameters

Table 132–16 VALUEOF Function and Procedure Parameters

Parameter	Description
n	Node whose value is being retrieved
pattern	Pattern to use
val	Retrieved value
namespace	Namespace to use

133

DEBUG_EXTPROC

The DEBUG_EXTPROC package enables you to start up the extproc agent within a session. This utility package can help you debug external procedures.

This chapter contains the following topics:

Using DEBUG_EXTPROC

Security Model

Your Oracle account must have `EXECUTE` privileges on the package and `CREATE LIBRARY` privileges.

Operational Notes

To install the package, run the script DBGEXTP.SQL.

- Install/load this package in the Oracle USER where you want to debug the 'extproc' process.
- Ensure that you have execute privileges on package DEBUG_EXTPROC

```
SELECT SUBSTR(OBJECT_NAME, 1, 20)
FROM USER_OBJECTS
WHERE OBJECT_NAME = 'DEBUG_EXTPROC';
```

- You can install this package as any other user, as long as you have EXECUTE privileges on the package.

> **Note:** These notes assumes that you built your shared library with debug symbols to aid in the debugging process. Please check the C compiler manual pages for the appropriate C compiler switches to build the shared library with debug symbols.

Having installed the package, proceed accordingly:

- Start a new Oracle session through SQL*Plus or OCI program by connecting to ORACLE.
- Execute procedure DEBUG_EXTPROC.STARTUP_EXTPROC_AGENT to startup the extproc agent in this session; for example, execute DEBUG_EXTPROC.STARTUP_EXTPROC_AGENT; Do not exit this session, because that terminates the extproc agent.
- Determine the PID of the extproc agent that was started up for this session.
- Using a debugger (for example, gdb, dbx, or the native system debugger), load the extproc executable and attach to the running process.
- Set a breakpoint on function 'pextproc' and let the debugger continue with its execution.
- Now execute your external procedure in the same session where you first executed DEBUG_EXTPROC.STARTUP_EXTPROC_AGENT
- Your debugger should now break in function 'pextproc'. At this point in time, the shared library referenced by your PL/SQL external function would have been loaded and the function resolved. Now set a breakpoint in your C function and let the debugger continue its execution.

Because PL/SQL loads the shared library at runtime, the debugger you use may or may not automatically be able to track the new symbols from the shared library. You may have to issue some debugger command to load the symbols (for example, 'share' in gdb)

- The debugger should now break in your C function. Its assumed that you had built the shared library with debugging symbols.
- Now proceed with your debugging.

Rules and Limits

Note: DEBUG_EXTPROC works only on platforms with debuggers that can attach to a running process.

Summary of DEBUG_EXTPROC Subprograms

Table 133–1 DEBUG_EXTPROC Package Subprograms

Subprogram	Description
STARTUP_EXTPROC_AGENT Procedure on page 133-7	Starts up the extproc agent process in the session

STARTUP_EXTPROC_AGENT Procedure

This procedure starts up the extproc agent process in the session.This enables you to get the PID of the executing process. This PID is needed to be able to attach to the running process using a debugger.

Syntax

```
DEBUG_EXTPROC.STARTUP_EXTPROC_AGENT;
```

134

HTF

The HTF (hypertext functions) and HTP (hypertext procedures) packages generate HTML tags. For example, the HTF.ANCHOR function generates the HTML anchor tag, <A>.

> **See Also:** For more information about implementation of this package:
>
> - *Oracle HTTP Server Administrator's Guide*
> - *Oracle Application Server mod_plsql User's Guide*

This chapter contains the following topics:

Using HTF

Operational Notes

For every HTF function that generates one or more HTML tags, there is a corresponding HTP procedure with identical parameters with the following exception:

- The PRINTS Procedure and the PS Procedure do not have HTF function equivalents. Use the ESCAPE_SC Function or the ESCAPE_URL Function if you need a string conversion function. Note that while there is a ESCAPE_SC Procedure that performs the same operation as the PRINTS Procedure and the PS Procedure, there is no procedural equivalent for the ESCAPE_URL Function.
- The FORMAT_CELL Function does not have an HTP equivalent. The function formats column values inside an HTML table using TABLEDATA Function which does have an HTP equivalent in the TABLEDATA Procedure. The advantage of this using the FORMAT_CELL Function is that it allows for better control over the HTML tables.

The function versions do not directly generate output in your web page. Instead, they pass their output as return values to the statements that invoked them. Use these functions when you need to nest calls. To print the output of HTF functions, call the functions from within the HTF.`PRINT` function. It then prints its parameters to the generated web page.

Rules and Limits

If you use values of the LONG data type in functions such as `HTF.PRINT`, `HTF.PRN`, `HTF.PA` or `OWA_UTIL.CELLSPRINT`, only the first 32 K of the `LONG` data is used. The `LONG` data is bound to a `VARCHAR2` data type in the function.

Examples

The following commands generate a simple HTML document:

```
CREATE OR REPLACE PROCEDURE hello AS
BEGIN
   HTP.P (HTF.HTMLOPEN); -- generates <HTML>
   HTP.P (HTF.HEADOPEN); -- generates <HEAD>
   HTP.P (HTF.TITLE('Hello')); -- generates <TITLE>Hello</TITLE>
   HTP.P (HTF.HEADCLOSE); -- generates </HEAD>
   HTP.P (HTF.BODYOPEN); -- generates <BODY>
   HTP.P (HTF.HEADER(1, 'Hello')); -- generates <H1>Hello</H1>
   HTP.P (HTF.BODYCLOSE); -- generates </BODY>
   HTP.P (HTF.HTMLCLOSE); -- generates </HTML>
END;
```

Summary of Tags

HTML, HEAD, and BODY Tags

HTMLOPEN Function, HTMLCLOSE Function - generate <HTML> and </HTML>

HEADOPEN Function, HEADCLOSE Function - generate <HEAD> and </HEAD>

BODYOPEN Function, BODYCLOSE Function - generate <BODY> and </BODY>

Comment Tag

COMMENT Function - generates <!-- and -->

http://www.w3.org.BASE Function - generates <BASE>

LINKREL Function - generates <LINK> with the REL attribute

LINKREV Function - generates <LINK> with the REV attribute

TITLE Function - generates <TITLE>

META Function - generates <META>

SCRIPT Function - generates <SCRIPT>

STYLE Function - generates <STYLE>

ISINDEX Function - generates <ISINDEX>

Applet Tags

APPLETOPEN Function, APPLETCLOSE Function - generate <APPLET> and </APPLET>

PARAM Function - generates <PARAM>

List Tags

OLISTOPEN Function, OLISTCLOSE Function - generate <OL> and </OL>

ULISTOPEN Function, ULISTCLOSE Function - generate <UL> and </UL>

DLISTOPEN Function, DLISTCLOSE Function- generate <DL> and </DL>

DLISTTERM Function - generates <DT>

DLISTDEF Function - generates <DD>

DIRLISTOPEN Function, DIRLISTCLOSE Function - generate <DIR> and </DIR>

LISTHEADER Function - generates <LH>

LISTINGOPEN Function, LISTINGCLOSE Function - generate <LISTING> and </LISTING>

MENULISTOPEN Function - generate <MENU> and </MENU>

LISTITEM Function - generates <LI>

Form Tags

FORMOPEN Function, FORMCLOSE Function - generate <FORM> and </FORM>

FORMCHECKBOX Function - generates <INPUT TYPE="CHECKBOX">

FORMHIDDEN Function - generates <INPUT TYPE="HIDDEN">

FORMIMAGE Function - generates <INPUT TYPE="IMAGE">

FORMPASSWORD Function - generates <INPUT TYPE="PASSWORD">

FORMRADIO Function - generates <INPUT TYPE="RADIO">

FORMSELECTOPEN Function, FORMSELECTCLOSE Function - generate <SELECT> and </SELECT>

FORMSELECTOPTION Function - generates <OPTION>

FORMTEXT Function - generates <INPUT TYPE="TEXT">

FORMTEXTAREA Function - generate <TEXTAREA>

FORMTEXTAREAOPEN Function, FORMTEXTAREACLOSE Function - generate <TEXTAREA> and </TEXTAREA>

FORMRESET Function - generates <INPUT TYPE="RESET">

FORMSUBMIT Function - generates <INPUT TYPE="SUBMIT">

Table Tags

TABLEOPEN Function, TABLECLOSE Function - generate <TABLE> and </TABLE>

TABLECAPTION Function - generates <CAPTION>

TABLEROWOPEN Function, TABLEROWCLOSE Function - generate <TR> and </TR>

TABLEHEADER Function - generates <TH>

TABLEDATA Function - generates <TD>

IMG, HR, and A Tags

HR Function, LINE Function - generate <HR>

IMG Function, IMG2 Function - generate <IMG>

ANCHOR Function, ANCHOR2 Function - generate <A>

MAPOPEN Function, MAPCLOSE Function - generate <MAP> and </MAP>

Paragraph Formatting Tags

HEADER Function - generates heading tags (<H1> to <H6>)

PARA Function, PARAGRAPH Function - generate <P>

PRN Functions, PRINT Functions - generate any text that is passed in

PRN Functions, S Function - generate any text that is passed in; special characters in HTML are escaped

PREOPEN Function, PRECLOSE Function - generate <PRE> and </PRE>

BLOCKQUOTEOPEN Function, BLOCKQUOTECLOSE Function - generate <BLOCKQUOTE> and </BLOCKQUOTE>

DIV Function - generates <DIV>

NL Function, BR Function - generate

NOBR Function - generates <NOBR>

WBR Function - generates <WBR>

PLAINTEXT Function - generates <PLAINTEXT>

ADDRESS Function - generates <ADDRESS>

MAILTO Function - generates <A> with the MAILTO attribute

AREA Function - generates <AREA>

BGSOUND Function - generates <BGSOUND>

Character Formatting Tags

BASEFONT Function - generates <BASEFONT>

BIG Function - generates <BIG>

BOLD Function - generates <B>

CENTER Function - generates <CENTER> and </CENTER>

CENTEROPEN Function, CENTERCLOSE Function - generate <CENTER> and </CENTER>

CITE Function - generates <CITE>

CODE Function - generates <CODE>

DFN Function - generates <DFN>

EM Function, EMPHASIS Function - generate <EM>

FONTOPEN Function, FONTCLOSE Function - generate <FONT> and </FONT>

ITALIC Function - generates <I>

KBD Function, KEYBOARD Function - generate <KBD> and </KBD>

S Function - generates <S>

SAMPLE Function - generates <SAMP>

SMALL Function - generates <SMALL>

STRIKE Function - generates <STRIKE>

STRONG Function - generates <STRONG>

SUB Function - generates <SUB>

SUP Function - generates <SUP>

TELETYPE Function - generates <TT>

UNDERLINE Function - generates <U>

VARIABLE Function - generates <VAR>

Frame Tags

FRAME Function - generates <FRAME>

FRAMESETOPEN Function, FRAMESETCLOSE Function - generate <FRAMESET> and </FRAMESET>

NOFRAMESOPEN Function, NOFRAMESCLOSE Function - generate <NOFRAMES> and </NOFRAMES>

Summary of HTF Subprograms

Table 134–1 HTF Package Subprograms

Subprogram	Description
ADDRESS Function on page 134-15	Generates the <ADDRESS> and </ADDRESS> tags which specify the address, author and signature of a document
ANCHOR Function on page 134-16	Generates the <A> and </A> tags which specify the source or destination of a hypertext link
ANCHOR2 Function on page 134-17	Generates the <A> and </A> tags which specify the source or destination of a hypertext link
APPLETCLOSE Function on page 134-18	Closes the applet invocation with the </APPLET> tag
APPLETOPEN Function on page 134-19	Generates the <APPLET> tag which begins the invocation of a Java applet
AREA Function on page 134-20	Generates the <AREA> tag, which defines a client-side image map
BASE Function on page 134-21	Generates the <BASE> tag which records the URL of the document
BASEFONT Function on page 134-22	Generates the <BASEFONT> tag which specifies the base font size for a Web page
BGSOUND Function on page 134-23	Generates the <BGSOUND> tag which includes audio for a Web page
BIG Function on page 134-24	Generates the <BIG> and </BIG> tags which direct the browser to render the text in a bigger font
BLOCKQUOTECLOSE Function on page 134-25	Generates the </BLOCKQUOTE> tag which mark the end of a section of quoted text
BLOCKQUOTEOPEN Function on page 134-26	Generates the <BLOCKQUOTE> tag, which marks the beginning of a section of quoted text
BODYCLOSE Function on page 134-27	Generates the </BODY> tag which marks the end of a body section of an HTML document
BODYOPEN Function on page 134-28	Generates the <BODY> tag which marks the beginning of the body section of an HTML document
BOLD Function on page 134-29	Generates the <B> and </B> tags which direct the browser to display the text in boldface
BR Function on page 134-30	Generates the tag which begins a new line of text
CENTER Function on page 134-31	Generates the <CENTER> and </CENTER> tags which center a section of text within a Web page
CENTERCLOSE Function on page 134-32	Generates the </CENTER> tag which marks the end of a section of text to center
CENTEROPEN Function on page 134-33	Generates the <CENTER> tag which mark the beginning of a section of text to center
CITE Function on page 134-34	Generates the <CITE> and </CITE> tags which direct the browser to render the text as a citation
CODE Function on page 134-35	Generates the <CODE> and </CODE> tags which direct the browser to render the text in monospace font or however "code" is defined stylistically

Table 134–1 (Cont.) HTF Package Subprograms

Subprogram	Description
COMMENT Function on page 134-36	Generates This function generates the comment tags <!-- ctext -->
DFN Function on page 134-37	Generates the <DFN> and </DFN> tags which direct the browser to mark the text as italics or however "definition" is defined stylistically
DIRLISTCLOSE Function on page 134-38	Generates the </DIR> tag which ends a directory list section
DIRLISTOPEN Function on page 134-39	Generates the <DIR> which starts a directory list section
DIV Function on page 134-40	Generates the <DIV> tag which creates document divisions
DLISTCLOSE Function on page 134-41	Generates the </DL> tag which ends a definition list
DLISTDEF Function on page 134-42	Generates the <DD> tag, which inserts definitions of terms
DLISTOPEN Function on page 134-43	Generates the <DL> tag which starts a definition list
DLISTTERM Function on page 134-44	Generates the <DT> tag which defines a term in a definition list <DL>
EM Function on page 134-45	Generates the <EM> and </EM> tags, which define text to be emphasized
EMPHASIS Function on page 134-46	Generates the <EM> and </EM> tags, which define text to be emphasized
ESCAPE_SC Function on page 134-47	Replaces characters that have special meaning in HTML with their escape sequences
ESCAPE_URL Function on page 134-48	Replaces characters that have special meaning in HTML and HTTP with their escape sequences
FONTCLOSE Function on page 134-50	Generates the </FONT> tag which marks the end of a section of text with the specified font characteristics
FONTOPEN Function on page 134-50	Generates the <FONT> which marks the beginning of section of text with the specified font characteristics
FORMAT_CELL Function on page 134-51	formats column values inside an HTML table using the TABLEDATA Function
FORMCHECKBOX Function on page 134-52	Generates the <INPUT> tag with TYPE="checkbox" which inserts a checkbox element in a form
FORMCLOSE Function on page 134-53	Generates the </FORM> tag which marks the end of a form section in an HTML document
FORMFILE Function on page 134-54	Generates the <INPUT> tag with TYPE="file" which inserts a file form element, and is used for file uploading for a given page
FORMHIDDEN Function on page 134-55	Generates the <INPUT> tag with TYPE="hidden"which inserts a hidden form element
FORMIMAGE Function on page 134-56	Generates the <INPUT> tag with TYPE="image" which creates an image field that the user clicks to submit the form immediately
FORMOPEN Function on page 134-57	Generates the <FORM> tag which marks the beginning of a form section in an HTML document

Table 134–1 (Cont.) HTF Package Subprograms

Subprogram	Description
FORMPASSWORD Function on page 134-58	Generates the <INPUT> tag with TYPE="password" which creates a single-line text entry field
FORMRADIO Function on page 134-59	Generates the <INPUT> tag with TYPE="radio", which creates a radio button on the HTML form
FORMRESET Function on page 134-60	Generates the <INPUT> tag with TYPE="reset" which creates a button that, when selected, resets the form fields to their initial values
FORMSELECTCLOSE Function on page 134-61	Generates the </SELECT> tag which marks the end of a Select form element
FORMSELECTOPEN Function on page 134-62	Generates the </SELECT> tag which marks the beginning of a Select form element
FORMSELECTOPTION Function on page 134-63	Generates the <OPTION> tag which represents one choice in a Select element
FORMSUBMIT Function on page 134-64	Generates the <INPUT> tag with TYPE="submit" which creates a button that, when clicked, submits the form
FORMTEXT Function on page 134-65	Generates the <INPUT> tag with TYPE="text", which creates a field for a single line of text
FORMTEXTAREA Function on page 134-66	Generates the <TEXTAREA> tag, which creates a text field that has no predefined text in the text area
FORMTEXTAREA2 Function on page 134-67	Generates the <TEXTAREA> tag, which creates a text field that has no predefined text in the text area with the ability to specify a wrap style
FORMTEXTAREACLOSE Function on page 134-68	Generates the </TEXTAREA> tag which ends a text area form element
FORMTEXTAREAOPEN Function on page 134-69	Generates the <TEXTAREA> which marks the beginning of a text area form element
FORMTEXTAREAOPEN2 Function on page 134-70	Generates the <TEXTAREA> which marks the beginning of a text area form element with the ability to specify a wrap style
FRAME Function on page 134-71	Generates the <FRAME> tag which defines the characteristics of a frame created by a <FRAMESET> tag
FRAMESETCLOSE Function on page 134-72	Generates the </FRAMESET> tag which ends a frameset section
FRAMESETOPEN Function on page 134-73	Generates the </FRAMESET> tag which begins a frameset section
HEADCLOSE Function on page 134-74	Generates the </HEAD> tag which marks the end of an HTML document head section
HEADER Function on page 134-75	Generates opening heading tags (<H1> to <H6>) and their corresponding closing tags (</H1> to </H6>)
HEADOPEN Function on page 134-76	Generates the <HEAD> tag which marks the beginning of the HTML document head section
HR Function on page 134-77	Generates the <HR> tag, which generates a line in the HTML document
HTMLCLOSE Function on page 134-79	Generates the </HTML> tag which marks the end of an HTML document

Table 134–1 (Cont.) HTF Package Subprograms

Subprogram	Description
HTMLOPEN Function on page 134-79	Generates the <HTML> tag which marks the beginning of an HTML document
IMG Function on page 134-80	Generates the <IMG> tag which directs the browser to load an image onto the HTML page
IMG2 Function on page 134-81	Generates the <IMG> tag which directs the browser to load an image onto the HTML page with the option of specifying values for the USEMAP attribute
ISINDEX Function on page 134-82	Creates a single entry field with a prompting text, such as "*enter value*," then sends that value to the URL of the page or program
ITALIC Function on page 134-83	Generates the <I> and </I> tags which direct the browser to render the text in italics
KBD Function on page 134-84	Generates the <KBD> and </KBD> tags which direct the browser to render the text in monospace font
KEYBOARD Function on page 134-85	Generates the <KBD> and </KBD> tags, which direct the browser to render the text in monospace font
LINE Function on page 134-86	Generates the <HR> tag, which generates a line in the HTML document
LINKREL Function on page 134-87	Generates the <LINK> tag with the REL attribute which delineates the relationship described by the hypertext link from the anchor to the target
LINKREV Function on page 134-88	Generates the <LINK> tag with the REV attribute which delineates the relationship described by the hypertext link from the target to the anchor
LISTHEADER Function on page 134-89	Generates the <LH> and </LH> tags which print an HTML tag at the beginning of the list
LISTINGCLOSE Function on page 134-90	Generates the </LISTING> tags which marks the end of a section of fixed-width text in the body of an HTML page
LISTINGOPEN Function on page 134-91	Generates the <LISTING> tag which marks the beginning of a section of fixed-width text in the body of an HTML page
LISTITEM Function on page 134-92	Generates the <LI> tag, which indicates a list item
MAILTO Function on page 134-93	Generates the <A> tag with the HREF set to 'mailto' prepended to the mail address argument
MAPCLOSE Function on page 134-94	Generates the </MAP> tag which marks the end of a set of regions in a client-side image map
MAPOPEN Function on page 134-95	Generates the <MAP> tag which mark the beginning of a set of regions in a client-side image map
MENULISTCLOSE Function on page 134-96	Generates the </MENU> tag which ends a list that presents one line for each item
MENULISTOPEN Function on page 134-97	Generates the <MENU> tag which create a list that presents one line for each item
META Function on page 134-98	Generates the <META> tag, which embeds meta-information about the document and also specifies values for HTTP headers

Table 134–1 (Cont.) HTF Package Subprograms

Subprogram	Description
NL Function on page 134-86	Generates the tag which begins a new line of text
NOBR Function on page 134-100	Generates the <NOBR> and </NOBR> tags which turn off line-breaking in a section of text
NOFRAMESCLOSE Function on page 134-101	Generates the </NOFRAMES> tag which marks the end of a no-frames section
NOFRAMESOPEN Function on page 134-102	Generates the <NOFRAMES> tag which mark the beginning of a no-frames section
OLISTCLOSE Function on page 134-103	Generates the </OL> tag which defines the end of an ordered list
OLISTOPEN Function on page 134-104	Generates the <OL> tag which marks the beginning of an ordered list
PARA Function on page 134-105	Generates the <P> tag which indicates that the text that comes after the tag is to be formatted as a paragraph
PARAGRAPH Function on page 134-106	Adds attributes to the <P> tag
PARAM Function on page 134-107	Generates the <PARAM> tag which specifies parameter values for Java applets
PLAINTEXT Function on page 134-108	Generates the <PLAINTEXT> and </PLAINTEXT> tags which direct the browser to render the text they surround in fixed-width type
PRECLOSE Function on page 134-109	Generates the </PRE> tag which marks the end of a section of preformatted text in the body of the HTML page
PREOPEN Function on page 134-110	Generates the <PRE> tag which marks the beginning of a section of preformatted text in the body of the HTML page
PRINT Functions on page 134-111	Generates the specified parameter as a string terminated with the \n newline character
PRN Functions on page 134-112	Generates the specified parameter as a string
S Function on page 134-113	Generates the <S> and </S> tags which direct the browser to render the text they surround in strikethrough type
SAMPLE Function on page 134-114	Generates the <SAMP> and </SAMP> tags which direct the browser to render the text they surround in monospace font or however "sample" is defined stylistically
SCRIPT Function on page 134-115	Generates the <SCRIPT> and </SCRIPT> tags which contain a script written in languages such as JavaScript and VBscript
SMALL Function on page 134-116	Generates the <SMALL> and </SMALL> tags, which direct the browser to render the text they surround using a small font
STRIKE Function on page 134-117	Generates the <STRIKE> and </STRIKE> tags which direct the browser to render the text they surround in strikethrough type
STRONG Function on page 134-118	Generates the <STRONG> and </STRONG> tags which direct the browser to render the text they surround in bold or however "strong" is defined stylistically
STYLE Function on page 134-119	Generates the <STYLE> and </STYLE> tags which include a style sheet in a Web page

Table 134–1 (Cont.) HTF Package Subprograms

Subprogram	Description
SUB Function on page 134-120	Generates the _{and} tags which direct the browser to render the text they surround as subscript
SUP Function on page 134-121	Generates the ^{and} tags which direct the browser to render the text they surround as superscript
TABLECAPTION Function on page 134-122	Generates the <CAPTION> and </CAPTION> tags which place a caption in an HTML table
TABLECLOSE Function on page 134-123	Generates the </TABLE> tag which marks the end of an HTML table
TABLEDATA Function on page 134-124	Generates the <TD> and </TD> tags which insert data into a cell of an HTML table
TABLEHEADER Function on page 134-125	Generates the <TH> and </TH> tags which insert a header cell in an HTML table.
TABLEOPEN Function on page 134-126	Generates the <TABLE> tag which marks the beginning of an HTML table
TABLEROWCLOSE Function on page 134-127	Generates the </TR> tag which marks the end of a new row in an HTML table
TABLEROWOPEN Function on page 134-128	Generates the <TR> tag which marks the beginning of a new row in an HTML table
TELETYPE Function on page 134-129	Generates the <TT> and </TT> tags which direct the browser to render the text they surround in a fixed width typewriter font, for example, the courier font
TITLE Function on page 134-130	Generates the <TITLE> and </TITLE> tags which specify the text to display in the titlebar of the browser window
ULISTCLOSE Function on page 134-131	Generates the </UL> tag which marks the end of an unordered list
ULISTOPEN Function on page 134-132	Generates the <UL> tag which marks the beginning of an unordered list
UNDERLINE Function on page 134-133	Generates the <U> and </U> tags, which direct the browser to render the text they surround with an underline
VARIABLE Function on page 134-134	Generates the <VAR> and </VAR> tags which direct the browser to render the text they surround in italics or however "variable" is defined stylistically.
WBR Function on page 134-135	Generates the <WBR> tag, which inserts a soft line break within a section of NOBR text

ADDRESS Function

This function generates the <ADDRESS> and </ADDRESS> tags which specify the address, author and signature of a document.

Syntax

```
HTF.ADDRESS (
   cvalue          IN       VARCHAR2
   cnowrap         IN       VARCHAR2   DEFAULT NULL
   cclear          IN       VARCHAR2   DEFAULT NULL
   cattributes     IN       VARCHAR2   DEFAULT NULL)
 RETURN VARCHAR2;
```

Parameters

Table 134–2 ADDRESS Function Parameters

Parameter	Description
cvalue	The string that goes between the <ADDRESS> and </ADDRESS> tags.
cnowrap	If the value for this parameter is not NULL, the NOWRAP attribute is included in the tag
cclear	The value for the CLEAR attribute.
cattributes	The other attributes to be included as-is in the tag

Examples

This function generates

```
<ADDRESS CLEAR="cclear" NOWRAP cattributes>cvalue</ADDRESS>
```

ANCHOR Function

This function and the ANCHOR2 Function functions generate the <A> and </A> HTML tags which specify the source or destination of a hypertext link. The difference between these subprograms is that the ANCHOR2 Function provides a target and therefore can be used for a frame.

Syntax

```
HTF.ANCHOR (
   curl           IN       VARCHAR2,
   ctext          IN       VARCHAR2,
   cname          IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
 RETURN VARCHAR2;
```

Parameters

Table 134–3 ANCHOR Function Parameters

Parameter	Description
curl	The value for the HREF attribute.
ctext	The string that goes between the <A> and </A> tags.
cname	The value for the NAME attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<A HREF="curl" NAME="cname" cattributes>ctext</A>
```

Usage Notes

This tag accepts several attributes, but either HREF or NAME is required. HREF specifies to where to link. NAME allows this tag to be a target of a hypertext link.

ANCHOR2 Function

This function and the ANCHOR Function generate the <A> and </A> HTML tags which specify the source or destination of a hypertext link. The difference between these subprograms is that this functions provides a target and therefore can be used for a frame.

Syntax

```
HTF.ANCHOR2 (
   curl           IN       VARCHAR2,
   ctext          IN       VARCHAR2,
   cname          IN       VARCHAR2   DEFAULT NULL,
   ctarget        in       varchar2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
 RETURN VARCHAR2;
```

Parameters

Table 134–4 ANCHOR2 Function Parameters

Parameter	Description
curl	The value for the HREF attribute.
ctext	The string that goes between the <A> and </A> tags.
cname	The value for the NAME attribute
ctarget	The value for the TARGET attribute.
cattributes	The other attributes to be included as-is in the tag

Examples

This function generates

```
<A HREF="curl" NAME="cname" TARGET = "ctarget" cattributes>ctext</A>
```

APPLETCLOSE Function

This function closes the applet invocation with the `</APPLET>` tag. You must first invoke the a Java applet using APPLETOPEN Function on page 134-19

Syntax

```
HTF.APPLETCLOSE
 RETURN VARCHAR2;
```

APPLETOPEN Function

This function generates the <APPLET> tag which begins the invocation of a Java applet. You close the applet invocation with APPLETCLOSE Function on page 134-18 which generates the </APPLET> tag.

Syntax

```
HTF.APPLETOPEN (
   ccode          IN       VARCHAR2,
   cheight        IN       NUMBER,
   cwidth         IN       NUMBER,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–5 APPLETOPEN Function Parameters

Parameter	Description
ccode	The the value for the CODE attribute which specifies the name of the applet class.
cheight	The value for the HEIGHT attribute.
cwidth	The value for the WIDTH attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<APPLET CODE=ccode HEIGHT=cheight WIDTH=cwidth cattributes>
```

so that, for example,

```
HTF.appletopen('testclass.class', 100, 200, 'CODEBASE="/ows-applets"')
```

generates

```
<APPLET CODE="testclass.class" height=100 width=200 CODEBASE="/ows-applets">
```

Usage Notes

- Specify parameters to the Java applet using the PARAM Function function on page 134-107.
- Use the cattributes parameter to specify the CODEBASE attribute since the PL/SQL cartridge does not know where to find the class files. The CODEBASE attribute specifies the virtual path containing the class files.

AREA Function

This function generates the <AREA> tag, which defines a client-side image map. The <AREA> tag defines areas within the image and destinations for the areas.

Syntax

```
HTF.AREA (
   ccoords        IN       VARCHAR2
   cshape         IN       VARCHAR2   DEFAULT NULL,
   chref          IN       VARCHAR2   DEFAULT NULL,
   cnohref        IN       VARCHAR2   DEFAULT NULL,
   ctarget        IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–6 AREA Function Parameters

Parameter	Description
ccords	The the value for the COORDS attribute.
cshape	The value for the SHAPE attribute.
chref	The value for the HREF attribute.
cnohref	If the value for this parameter is not NULL, the NOHREF attribute is added to the tag.
ctarget	The value for the TARGET attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<AREA COORDS="ccoords" SHAPE="cshape" HREF="chref" NOHREF TARGET="ctarget"
cattributes>
```

BASE Function

This function generates the <BASE> tag which records the URL of the document.

Syntax

```
HTF.BASE (
   ctarget        IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–7 BASE Function Parameters

Parameter	Description
ctarget	The value for the TARGET attribute which establishes a window name to which all links in this document are targeted.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<BASE HREF="<current URL>" TARGET="ctarget" cattributes>
```

BASEFONT Function

This function generates the <BASEFONT> tag which specifies the base font size for a Web page.

Syntax

```
HTF.BASEFONT (
   nsize    IN    INTEGER)
  RETURN VARCHAR2;
```

Parameters

Table 134–8 BASEFONT Function Parameters

Parameter	Description
nsize	The value for the SIZE attribute.

Examples

This function generates

```
<BASEFONT SIZE="nsize">
```

BGSOUND Function

This function generates the <BGSOUND> tag which includes audio for a Web page.

Syntax

```
HTF.BGSOUND (
   csrc           IN       VARCHAR2,
   cloop          IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–9 BGSOUND Function Parameters

Parameter	Description
csrc	The value for the SRC attribute.
cloop	The value for the LOOP attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<BGSOUND SRC="csrc" LOOP="cloop" cattributes>
```

BIG Function

This function generates the <BIG> and </BIG> tags which direct the browser to render the text in a bigger font.

Syntax

```
HTF.BIG (
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–10 BIG Function Parameters

Parameter	Description
ctext	The the text that goes between the tags.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<BIG cattributes>ctext</BIG>
```

BLOCKQUOTECLOSE Function

This function generates the </BLOCKQUOTE> tag which mark the end of a section of quoted text. You mark the beginning of a section of text by means of the BLOCKQUOTEOPEN Function.

Syntax

```
HTF.BLOCKQUOTECLOSE
  RETURN VARCHAR2;
```

Examples

This function generates

```
</BLOCKQUOTE>
```

BLOCKQUOTEOPEN Function

This function generates the <BLOCKQUOTE> tag, which marks the beginning of a section of quoted text. You mark the end of a section of text by means of the BLOCKQUOTECLOSE Function.

Syntax

```
HTF.BLOCKQUOTEOPEN (
   cnowrap        IN       VARCHAR2   DEFAULT NULL,
   cclear         IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–11 BLOCKQUOTEOPEN Function Parameters

Parameter	Description
cnowrap	If the value for this parameter is not NULL, the NOWRAP attribute is added to the tag.
cclear	The value for the CLEAR attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<BLOCKQUOTE CLEAR="cclear" NOWRAP cattributes>
```

BODYCLOSE Function

This function generates the </BODY> tag which marks the end of a body section of an HTML document.You mark the beginning of a body section by means of the BODYOPEN Function.

Syntax

```
HTF.BODYCLOSE
  RETURN VARCHAR2;
```

Examples

This function generates

```
</BODY>
```

BODYOPEN Function

This function generates the <BODY> tag which marks the beginning of the body section of an HTML document. You mark the end of a body section by means of the BODYCLOSE Function.

Syntax

```
HTF.BODYOPEN (
   cbackground    IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–12 BODYOPEN Function Parameters

Parameter	Description
cbackground	The value for the BACKGROUND attribute which specifies a graphic file to use for the background of the document.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<BODY background="cbackground" cattributes>
```

so that

```
HTF.BODYOPEN('/img/background.gif')
  RETURN VARCHAR2;
```

generates:

```
<BODY background="/img/background.gif">
```

BOLD Function

This function generates the <B> and </B> tags which direct the browser to display the text in boldface.

Syntax

```
HTF.BOLD (
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–13 BOLD Function Parameters

Parameter	Description
ctext	The text that goes between the tags.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<B cattributes>ctext</B>
```

BR Function

This function generates the <BR> tag which begins a new line of text. It performs the same operation as the NL Function.

Syntax

```
HTF.BR(
   cclear         IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–14 BR Function Parameters

Parameter	Description
cclear	The value for the CLEAR attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<BR CLEAR="cclear" cattributes>
```

CENTER Function

This function generates the <CENTER> and </CENTER> tags which center a section of text within a Web page.

Syntax

```
HTF.CENTER (
   ctext          IN       VARCHAR2)
  RETURN VARCHAR2;
```

Parameters

Table 134–15 CENTER Parameters

Parameter	Description
ctext	The text that goes between the tags.

Examples

This function generates

```
<CENTER>ctext</CENTER>
```

CENTERCLOSE Function

This function generates the </CENTER> tag which marks the end of a section of text to center. You mark the beginning of a of a section of text to center by means of the CENTEROPEN Function.

Syntax

```
HTF.CENTERCLOSE
  RETURN VARCHAR2;
```

Examples

This function generates

```
</CENTER>
```

CENTEROPEN Function

This function generates the <CENTER> tag which mark the beginning of a section of text to center.You mark the beginning of a of a section of text to center by means of the CENTERCLOSE Function.

Syntax

```
HTF.CENTEROPEN
  RETURN VARCHAR2;
```

Examples

This function generates

```
<CENTER>
```

CITE Function

This function generates the <CITE> and </CITE> tags which direct the browser to render the text as a citation.

Syntax

```
HTF.CITE (
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–16 CITE Function Parameters

Parameter	Description
ctext	The text to render as citation.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<CITE cattributes>ctext</CITE>
```

CODE Function

This function generates the <CODE> and </CODE> tags which direct the browser to render the text in monospace font or however "code" is defined stylistically.

Syntax

```
HTF.CODE (
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–17 CODE Function Parameters

Parameter	Description
ctext	The text to render as code.
cattributes	The other attributes to be included as-is in the tag

Examples

This function generates

```
<CODE cattributes>ctext</CODE>
```

COMMENT Function

This function generates the comment tags.

Syntax

```
HTF.COMMENT (
   ctext           IN       VARCHAR2)
  RETURN VARCHAR2;
```

Parameters

Table 134–18 COMMENT Function Parameters

Parameter	Description
ctext	The comment.

Examples

This function generates

```
<!-- ctext -->
```

DFN Function

This function generates the <DFN> and </DFN> tags which direct the browser to mark the text in italics or however "definition" is described stylistically.

Syntax

```
HTF.DFN (
   ctext          IN       VARCHAR2)
  RETURN VARCHAR2;
```

Parameters

Table 134–19 DFN Function Parameters

Parameter	Description
ctext	The text to render in italics.

Examples

This function generates

```
<DFN>ctext</DFN>
```

DIRLISTCLOSE Function

This function generates the </DIR> tag which ends a directory list section. You start a directory list section with the DIRLISTOPEN Function.

Syntax

```
HTF.DIRLISTCLOSE
  RETURN VARCHAR2;
```

Usage Notes

A directory list presents a list of items that contains up to 20 characters. Items in this list are typically arranged in columns, 24 characters wide. Insert the <LI> tag directly or invoke the LISTITEM Function so that the <LI> tag appears directly after the </DIR> tag to define the items as a list.

Examples

This function generates

```
</DIR>
```

DIRLISTOPEN Function

This function generates the <DIR> which starts a directory list section. You end a directory list section with the DIRLISTCLOSE Function.

Syntax

```
HTF.DIRLISTOPEN
  RETURN VARCHAR2;
```

Usage Notes

A directory list presents a list of items that contains up to 20 characters. Items in this list are typically arranged in columns, 24 characters wide. Insert the <LI> tag directly or invoke the LISTITEM Function so that the <LI> tag appears directly after the </DIR> tag to define the items as a list.

Examples

This function generates

```
<DIR>
```

DIV Function

This function generates the <DIV> tag which creates document divisions.

Syntax

```
HTF.DIV (
   calign         IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–20 DIV Function Parameters

Parameter	Description
calign	The value for the ALIGN attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<DIV ALIGN="calign" cattributes>
```

DLISTCLOSE Function

This function generates the </DL> tag which ends a definition list. You start a definition list by means of the DLISTOPEN Function.

Syntax

```
HTF.DLISTCLOSE
  RETURN VARCHAR2;
```

Usage Notes

A definition list looks like a glossary: it contains terms and definitions. Terms are inserted using the DLISTTERM Function and definitions are inserted using the DLISTDEF Function.

Examples

This function generates

```
</DL>
```

DLISTDEF Function

This function generates the <DD> tag, which inserts definitions of terms. Use this tag for a definition list <DL>. Terms are tagged <DT> and definitions are tagged <DD>.

Syntax

```
HTF.DLISTDEF (
   ctext          IN       VARCHAR2   DEFAULT NULL,
   cclear         IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–21 DLISTDEF Function Parameters

Parameter	Description
ctext	The definition of the term.
cclear	The value for the CLEAR attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<DD CLEAR="cclear" cattributes>ctext
```

DLISTOPEN Function

This function generates the <DL> tag which starts a definition list. You end a definition list by means of the DLISTCLOSE Function.

Syntax

```
HTF.DLISTOPEN (
   cclear         IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–22 DLISTOPEN Function Parameters

Parameter	Description
cclear	The value for the CLEAR attribute.
cattributes	The other attributes to be included as-is in the tag.

Usage Notes

A definition list looks like a glossary: it contains terms and definitions. Terms are inserted using the DLISTTERM Function and definitions are inserted using the DLISTDEF Function.

Examples

This function generates

```
<DL CLEAR="cclear" cattributes>
```

DLISTTERM Function

This function generates the <DT> tag which defines a term in a definition list <DL>.

Syntax

```
HTF.DLISTTERM (
   ctext          IN       VARCHAR2   DEFAULT NULL,
   cclear         IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–23 DLISTTERM Function Parameters

Parameter	Description
ctext	The term.
cclear	The value for the CLEAR attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<DT CLEAR="cclear" cattributes>ctext
```

EM Function

This function generates the <EM> and </EM> tags, which define text to be emphasized. It performs the same task as the EMPHASIS Function.

Syntax

```
HTF.EM(
     ctext          IN       VARCHAR2,
     cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–24 EM Function Parameters

Parameter	Description
ctext	The text to emphasize.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<EM cattributes>ctext</EM>
```

EMPHASIS Function

This function generates the <EM> and </EM> tags, which define text to be emphasized. It performs the same task as the EM Function.

Syntax

```
HTF.EMPHASIS(
     ctext          IN       VARCHAR2,
     cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–25 EMPHASIS Function Parameters

Parameter	Description
ctext	The text to emphasize.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<EM cattributes>ctext</EM>
```

ESCAPE_SC Function

This function replaces characters that have special meaning in HTML with their escape sequences. The following characters are converted:

- `&` to `&`
- `"` to `&quot:`
- `<` to `<`
- `>` to `>`

This function performs the same operation as HTP. PRINTS Procedure and HTP. PS Procedure.

Syntax

```
HTF.ESCAPE_SC(
     ctext          IN       VARCHAR2);
```

Parameters

Table 134–26 ESCAPE_SC Procedure Parameters

Parameter	Description
`ctext`	The text string to convert.

ESCAPE_URL Function

This function replaces characters that have special meaning in HTML and HTTP with their escape sequences. The following characters are converted:

- `&` to `&`
- `"` to `&quot:`
- `<` to `<`
- `>` to `>`
- `%` to `&25`

Syntax

```
HTF.ESCAPE_URL(
     p_url          IN       VARCHAR2);
```

Parameters

Table 134–27 ESCAPE_URL Procedure Parameters

Parameter	Description
`p_url`	The string to convert.

FONTCLOSE Function

This function generates the</FONT> tag which marks the end of a section of text with the specified font characteristics. You mark the beginning of the section text by means of the FONTOPEN Function.

Syntax

```
HTF.FONTCLOSE
  RETURN VARCHAR2;
```

Examples

This function generates

</FONT>

FONTOPEN Function

This function generates the <FONT> which marks the beginning of section of text with the specified font characteristics. You mark the end of the section text by means of the FONTCLOSE Function.

Syntax

```
HTF.FONTOPEN(
   ccolor          IN       VARCHAR2   DEFAULT NULL,
   cface           IN       VARCHAR2   DEFAULT NULL,
   csize           IN       VARCHAR2   DEFAULT NULL,
   cattributes     IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–28 FONTOPEN Function Parameters

Parameter	Description
ccolor	The value for the COLOR attribute.
cface	The value for the FACE attribute
csize	The value for the SIZE attribute
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<FONT COLOR="ccolor" FACE="cface" SIZE="csize" cattributes>
```

FORMAT_CELL Function

This function formats column values inside an HTML table using the TABLEDATA Function. It allows for better control over the HTML tables.

Syntax

```
HTF.FORMAT_CELL(
   columnValue          IN       VARCHAR2
   format_numbers       IN       VARCHAR2   DEFAULT NULL
 RETURN VARCHAR2;
```

Parameters

Table 134–29 FORMAT_CELL Function Parameters

Parameter	Description
`columnValue`	The value that needs to be formatted in an HTML table.
`format_numbers`	The format that numeric data is displayed in. If the value of this parameter is not `NULL`, the number fields are right-justified and rounded to two decimal places.

Examples

This function generates

```
<TD >columnValue</TD>
```

FORMCHECKBOX Function

This function generates the <INPUT> tag with TYPE="checkbox" which inserts a checkbox element in a form. A checkbox element is a button that the user toggles on or off.

Syntax

```
HTF.FORMCHECKBOX(
   cname          IN       VARCHAR2,
   cvalue         IN       VARCHAR2   DEFAULT 'ON',
   cchecked       IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–30 FORMCHECKBOX Function Parameters

Parameter	Description
cname	The value for the NAME attribute.
cvalue	The value for the VALUE attribute.
cchecked	If the value for this parameter is not NULL, the CHECKED attribute is added to the tag.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<INPUT TYPE="checkbox" NAME="cname" VALUE="cvalue" CHECKED cattributes>
```

FORMCLOSE Function

This function generates the </FORM> tag which marks the end of a form section in an HTML document.You mark the beginning of the form section by means of the FORMOPEN Function.

Syntax

```
HTF.FORMCLOSE
  RETURN VARCHAR2;
```

Examples

This function generates

```
</FORM>
```

FORMFILE Function

This function generates the <INPUT> tag with TYPE="file" which inserts a file form element. This is used for file uploading for a given page.

Syntax

```
HTF.FORMFILE(
   cname          IN       VARCHAR2,
   caccept        IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–31 FORMFILE Function Parameters

Parameter	Description
cname	The value for the NAME attribute.
caccept	A comma-delimited list of MIME types for upload.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<INPUT TYPE="file" NAME="cname" ACCEPT="caccept" cattributes>
```

FORMHIDDEN Function

This function generates the <INPUT> tag with TYPE="hidden", which inserts a hidden form element. This element is not seen by the user. It submits additional values to the script.

Syntax

```
HTF.FORMHIDDEN(
   cname          IN       VARCHAR2,
   cvalue         IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–32 FORMHIDDEN Function Parameters

Parameter	Description
cname	The value for the NAME attribute.
cvalue	The value for the VALUE attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<INPUT TYPE="hidden" NAME="cname" VALUE="cvalue" cattributes>
```

FORMIMAGE Function

This function generates the <INPUT> tag with TYPE="image" which creates an image field that the user clicks to submit the form immediately. The coordinates of the selected point are measured in pixels, and returned (along with other contents of the form) in two name/value pairs. The x coordinate is submitted under the name of the field with .x appended, and the y coordinate with .y appended. Any VALUE attribute is ignored.

Syntax

```
HTF.FORMIMAGE(
   cname          IN       VARCHAR2,
   csrc           IN       VARCHAR2,
   calign         IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–33 FORMIMAGE Function Parameters

Parameter	Description
cname	The value for the NAME attribute.
csrc	The value for the SRC attribute that specifies the image file.
calign	The value for the ALIGN attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<INPUT TYPE="image" NAME="cname" SRC="csrc" ALIGN="calign" cattributes>
```

FORMOPEN Function

This function generates the <FORM> tag which marks the beginning of a form section in an HTML document. You mark the end of the form section by means of the FORMCLOSE Function.

Syntax

```
HTF.FORMOPEN(
   curl           IN       VARCHAR2,
   cmethod        IN       VARCHAR2   DEFAULT 'POST',
   ctarget        IN       VARCHAR2   DEFAULT NULL,
   cenctype       IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–34 FORMOPEN Function Parameters

Parameter	Description
curl	The URL of the Web Request Broker or CGI script where the contents of the form is sent. This parameter is required.
cmethod	The value for the METHOD attribute. The value can be "GET" or "POST".
ctarget	The value for the TARGET attribute.
cenctype	The value for the ENCTYPE attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<FORM ACTION="curl" METHOD="cmethod" TARGET="ctarget" ENCTYPE="cenctype"
cattributes>
```

FORMPASSWORD Function

This function generates the <INPUT> tag with TYPE="password" which creates a single-line text entry field. When the user enters text in the field, each character is represented by one asterisk. This is used for entering passwords.

Syntax

```
HTF.FORMPASSWORD(
   cname          IN       VARCHAR2,
   csize          IN       VARCHAR2,
   cmaxlength     IN       VARCHAR2   DEFAULT NULL,
   cvalue         IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–35 FORMPASSWORD Function Parameters

Parameter	Description
cname	The value for the NAME attribute.
csize	The value for the SIZE attribute.
cmaxlength	The value for the MAXLENGTH attribute.
cvalue	The value for the VALUE attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<INPUT TYPE="password" NAME="cname" SIZE="csize" MAXLENGTH="cmaxlength"
VALUE="cvalue" cattributes>
```

FORMRADIO Function

This function generates the <INPUT> tag with TYPE="radio", which creates a radio button on the HTML form. Within a set of radio buttons, the user selects only one. Each radio button in the same set has the same name, but different values. The selected radio button generates a name/value pair.

Syntax

```
HTF.FORMRADIO(
   cname          IN       VARCHAR2,
   cvalue         IN       VARCHAR2,
   cchecked       IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–36 FORMRADIO Function Parameters

Parameter	Description
cname	The value for the NAME attribute.
cvalue	The value for the VALUE attribute.
cchecked	If the value for this parameter is not NULL, the CHECKED attribute is added to the tag.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<INPUT TYPE="radio" NAME="cname" VALUE="cvalue" CHECKED cattributes>
```

FORMRESET Function

This function generates the <INPUT> tag with TYPE="reset" which creates a button that, when selected, resets the form fields to their initial values.

Syntax

```
HTF.FORMRESET(
   cvalue         IN       VARCHAR2   DEFAULT 'Reset',
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–37 FORMRESET Function Parameters

Parameter	Description
cvalue	The value for the VALUE attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<INPUT TYPE="reset" VALUE="cvalue" cattributes>
```

FORMSELECTCLOSE Function

This function generates the </SELECT> tag which marks the end of a Select form element. A Select form element is a listbox where the user selects one or more values. You mark the beginning of Select form element by means of the FORMSELECTOPEN Function.The values are inserted using FORMSELECTOPTION Function.

Syntax

```
HTF.FORMSELECTCLOSE
  RETURN VARCHAR2;
```

Examples

This function generates

```
</SELECT>
```

as shown under Examples of the FORMSELECTOPEN Function.

FORMSELECTOPEN Function

This function generates the <SELECT> tags which creates a Select form element. A Select form element is a listbox where the user selects one or more values. You mark the end of Select form element by means of the FORMSELECTCLOSE Function.The values are inserted using FORMSELECTOPTION Function.

Syntax

```
HTF.FORMSELECTOPEN(
   cname          IN       VARCHAR2,
   cprompt        IN       VARCHAR2   DEFAULT NULL,
   nsize          IN       INTEGER    DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–38 FORMSELECTOPEN Function Parameters

Parameter	Description
cname	The value for the NAME attribute.
cprompt	The string preceding the list box.
nsize	The value for the SIZE attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
cprompt <SELECT NAME="cname" SIZE="nsize" cattributes>
</SELECT>
```

so that

```
HTF.FORMSELECTOPEN('greatest_player';
    'Pick the greatest player:');
HTF.FORMSELECTOPTION('Messier');
HTF.FORMSELECTOPTION('Howe');
HTF.FORMSELECTOPTION('Gretzky');.
HTF.FORMSELECTCLOSE;
```

generates

```
Pick the greatest player:
<SELECT NAME="greatest_player">
<OPTION>Messier
<OPTION>Howe
<OPTION>Gretzky
</SELECT>
```

FORMSELECTOPTION Function

This function generates the <OPTION> tag which represents one choice in a Select element.

Syntax

```
HTF.FORMSELECTOPTION(
   cvalue         IN       VARCHAR2,
   cselected      IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–39 FORMSELECTOPTION Function Parameters

Parameter	Description
cvalue	The text for the option.
cvalue	If the value for this parameter is not NULL, the SELECTED attribute is added to the tag.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<OPTION SELECTED cattributes>cvalue
```

as shown under Examples of the FORMSELECTOPEN Function.

FORMSUBMIT Function

This function generates the <INPUT> tag with TYPE="submit" which creates a button that, when clicked, submits the form. If the button has a NAME attribute, the button contributes a name/value pair to the submitted data.

Syntax

```
HTF.FORMSUBMIT(
   cname          IN       VARCHAR2   DEFAULT NULL,
   cvalue         IN       VARCHAR2   DEFAULT 'Submit',
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–40 FORMSUBMIT Function Parameters

Parameter	Description
cname	The value for the NAME attribute.
cvalue	The value for the VALUE attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<INPUT TYPE="submit" NAME="cname" VALUE="cvalue" cattributes>
```

FORMTEXT Function

This function generates the <INPUT> tag with TYPE="text", which creates a field for a single line of text.

Syntax

```
HTF.FORMTEXT(
   cname          IN       VARCHAR2,
   csize          IN       VARCHAR2   DEFAULT NULL,
   cmaxlength     IN       VARCHAR2   DEFAULT NULL,
   cvalue         IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–41 FORMTEXT Function Parameters

Parameter	Description
cname	The value for the NAME attribute.
csize	The value for the SIZE attribute.
cmaxlength	The value for the MAXLENGTH attribute.
cvalue	The value for the VALUE attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<INPUT TYPE="text" NAME="cname" SIZE="csize" MAXLENGTH="cmaxlength" VALUE="cvalue"
cattributes>
```

FORMTEXTAREA Function

This function generates the <TEXTAREA> tag, which creates a text field that has no predefined text in the text area. This field enables entering several lines of text. The same operation is performed by the FORMTEXTAREA2 Function which in addition has the cwrap parameter that lets you specify a wrap style.

Syntax

```
HTF.FORMTEXTAREA(
   cname          IN       VARCHAR2,
   nrows          IN       INTEGER,
   ncolumns       IN       INTEGER,
   calign         IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–42 FORMTEXTAREA Function Parameters

Parameter	Description
cname	The value for the NAME attribute.
nrows	The value for the ROWS attribute.This is an integer.
ncolumns	The value for the COLS attribute.This is an integer.
calign	The value for the ALIGN attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolumns" ALIGN="calign"
cattributes></TEXTAREA>
```

FORMTEXTAREA2 Function

This function generates the <TEXTAREA> tag, which creates a text field that has no predefined text in the text area. This field enables entering several lines of text.The same operation is performed by the FORMTEXTAREA Function except that in that case you cannot specify a wrap style.

Syntax

```
HTF.FORMTEXTAREA2(
   cname          IN       VARCHAR2,
   nrows          IN       INTEGER,
   ncolumns       IN       INTEGER,
   calign         IN       VARCHAR2   DEFAULT NULL,
   cwrap          IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–43 FORMTEXTAREA2 Function Parameters

Parameter	Description
cname	The value for the NAME attribute.
nrows	The value for the ROWS attribute.This is an integer.
ncolumns	The value for the COLS attribute.This is an integer.
calign	The value for the ALIGN attribute.
cwrap	The value for the WRAP attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolumns" ALIGN="calign" WRAP="cwrap"
cattributes></TEXTAREA>
```

FORMTEXTAREACLOSE Function

This function generates the </TEXTAREA> tag which ends a text area form element. You open a text area element by means of either FORMTEXTAREAOPEN Function or FORMTEXTAREAOPEN2 Function.

Syntax

```
HTF.FORMTEXTAREACLOSE
  RETURN VARCHAR2;
```

Examples

This function generates

```
</TEXTAREA>
```

FORMTEXTAREAOPEN Function

This function generates the <TEXTAREA> which marks the beginning of a text area form element. The same operation is performed by the FORMTEXTAREAOPEN2 Function which in addition has the cwrap parameter that lets you specify a wrap style. You mark the end of a text area form element by means of the FORMTEXTAREACLOSE Function.

Syntax

```
HTF.FORMTEXTAREAOPEN(
   cname          IN       VARCHAR2,
   nrows          IN       INTEGER,
   ncolumns       IN       INTEGER,
   calign         IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–44 FORMTEXTAREAOPEN Function Parameters

Parameter	Description
cname	The value for the NAME attribute.
nrows	The value for the ROWS attribute.This is an integer.
ncolumns	The value for the COLS attribute.This is an integer.
calign	The value for the ALIGN attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolumns" ALIGN="calign" cattributes>
```

FORMTEXTAREAOPEN2 Function

This function generates the <TEXTAREA> which marks the beginning of a text area form element. The same operation is performed by the FORMTEXTAREAOPEN Function except that in that case you cannot specify a wrap style. You mark the end of a text area form element by means of the FORMTEXTAREACLOSE Function.

Syntax

```
HTF.FORMTEXTAREAOPEN2(
   cname          IN       VARCHAR2,
   nrows          IN       INTEGER,
   ncolumns       IN       INTEGER,
   calign         IN       VARCHAR2   DEFAULT NULL,
   cwrap          IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–45 FORMTEXTAREAOPEN2 Function Parameters

Parameter	Description
cname	The value for the NAME attribute.
nrows	The value for the ROWS attribute.This is an integer.
ncolumns	The value for the COLS attribute.This is an integer.
calign	The value for the ALIGN attribute.
cwrap	The value for the WRAP attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolumns" ALIGN="calign" WRAP = "cwrap"
cattributes>
```

FRAME Function

This function generates the <FRAME> tag which defines the characteristics of a frame created by a <FRAMESET> tag.

Syntax

```
HTF.FRAME(
   csrc           IN       VARCHAR2,
   cname          IN       VARCHAR2   DEFAULT NULL,
   cmarginwidth   IN       VARCHAR2   DEFAULT NULL,
   cmarginheight  IN       VARCHAR2   DEFAULT NULL,
   cscrolling     IN       VARCHAR2   DEFAULT NULL,
   cnoresize      IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–46 FRAME Function Parameters

Parameter	Description
csrc	The URL to display in the frame.
cname	The value for the NAME attribute.
cmarginwidth	The value for the MARGINWIDTH attribute.
cscrolling	The value for the SCROLLING attribute.
cnoresize	If the value for this parameter is not NULL, the NORESIZE attribute is added to the tag.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<FRAME SRC="csrc" NAME="cname" MARGINWIDTH="cmarginwidth"
MARGINHEIGHT="cmarginheight" SCROLLING="cscrolling" NORESIZE cattributes>
```

FRAMESETCLOSE Function

This function generates the </FRAMESET> tag which ends a frameset section. You mark the beginning of a frameset section by means of the FRAMESETOPEN Function.

Syntax

```
HTF.FRAMESETCLOSE
  RETURN VARCHAR2;
```

Examples

This function generates

```
</FRAMESET>
```

FRAMESETOPEN Function

This function generates the <FRAMESET> tag which define a frameset section. You mark the end of a frameset section by means of the FRAMESETCLOSE Function.

Syntax

```
HTF.FRAMESETOPEN(
   crows          IN       VARCHAR2   DEFAULT NULL,
   ccols          IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–47 FRAMESETOPEN Function Parameters

Parameter	Description
crows	The value for the ROWS attribute.
ccols	The value for the COLS attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<FRAMESET ROWS="crows" COLS="ccols" cattributes>
```

HEADCLOSE Function

This function generates the </HEAD> tag which marks the end of an HTML document head section. You mark the beginning of an HTML document head section by means of the HEADOPEN Function.

Syntax

```
HTF.HEADCLOSE
  RETURN VARCHAR2;
```

Examples

This function generates

```
</HEAD>
```

HEADER Function

This function generates opening heading tags (<H1> to <H6>) and their corresponding closing tags (</H1> to </H6>).

Syntax

```
HTF.HEADER(
   nsize          IN       INTEGER,
   cheader        IN       VARCHAR2,
   calign         IN       VARCHAR2   DEFAULT NULL,
   cnowrap        IN       VARCHAR2   DEFAULT NULL,
   cclear         IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–48 HEADER Function Parameters

Parameter	Description
nsize	The the heading level. This is an integer between 1 and 6.
cheader	The text to display in the heading.
calign	The value for the ALIGN attribute.
cnowrap	The value for the NOWRAP attribute.
cclear	The value for the CLEAR attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

```
HTF.header (1,'Overview')
  RETURN VARCHAR2;
```

produces:

```
<H1>Overview</H1>
```

HEADOPEN Function

This function generates the <HEAD> tag which marks the beginning of the HTML document head section. You mark the end of an HTML document head section by means of the HEADCLOSE Function.

Syntax

```
HTF.HEADOPEN
  RETURN VARCHAR2;
```

Examples

This function generates

```
<HEAD>
```

HR Function

This function generates the <HR> tag, which generates a line in the HTML document.This subprogram performs the same operation as the LINE Function.

Syntax

```
HTF.HR(
   cclear          IN       VARCHAR2   DEFAULT NULL,
   csrc            IN       VARCHAR2   DEFAULT NULL,
   cattributes     IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–49 HR Function Parameters

Parameter	Description
cclear	The value for the CLEAR attribute.
csrc	The value for the SRC attribute which specifies a custom image as the source of the line.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<HR CLEAR="cclear" SRC="csrc" cattributes>
```

HTMLCLOSE Function

This function generates the </HTML> tag which marks the end of an HTML document. You use the HTMLOPEN Function to mark the beginning of an HTML document.

Syntax

```
HTF.HTMLCLOSE
  RETURN VARCHAR2;
```

Examples

This function generates

```
</HTML>
```

HTMLOPEN Function

This function generates the <HTML> tag which marks the beginning of an HTML document. You use the HTMLCLOSE Function to mark the end of the an HTML document.

Syntax

```
HTF.HTMLOPEN
  RETURN VARCHAR2;
```

Examples

This function generates

```
<HTML>
```

IMG Function

This function generates the <IMG> tag which directs the browser to load an image onto the HTML page. The IMG2 Function performs the same operation but additionally uses the cusemap parameter.

Syntax

```
HTF.IMG(
   curl           IN       VARCHAR2   DEFAULT NULL,
   calign         IN       VARCHAR2   DEFAULT NULL,
   calt           IN       VARCHAR2   DEFAULT NULL,
   cismap         IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–50 IMG Function Parameters

Parameter	Description
curl	The value for the SRC attribute.
calign	The value for the ALIGN attribute.
calt	The value for the ALT attribute which specifies alternative text to display if the browser does not support images.
cismap	If the value for this parameter is not NULL, the ISMAP attribute is added to the tag. The attribute indicates that the image is an imagemap.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<IMG SRC="curl" ALIGN="calign" ALT="calt" ISMAP cattributes>
```

IMG2 Function

This function generates the <IMG> tag, which directs the browser to load an image onto the HTML page. The IMG Function performs the same operation but does not use the `cusemap` parameter.

Syntax

```
HTF.IMG2(
   curl           IN       VARCHAR2   DEFAULT NULL,
   calign         IN       VARCHAR2   DEFAULT NULL,
   calt           IN       VARCHAR2   DEFAULT NULL,
   cismap         IN       VARCHAR2   DEFAULT NULL,
   cusemap        IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–51 IMG2 Function Parameters

Parameter	Description
`curl`	The value for the SRC attribute.
`calign`	The value for the `ALIGN` attribute.
`calt`	The value for the `ALT` attribute which specifies alternative text to display if the browser does not support images.
`cismap`	If the value for this parameter is not `NULL`, the `ISMAP` attribute is added to the tag. The attribute indicates that the image is an imagemap.
`cusemap`	The value for the `USEMAP` attribute which specifies a client-side image map.
`cattributes`	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<IMG SRC="curl" ALIGN="calign" ALT="calt" ISMAP USEMAP="cusemap" cattributes>
```

ISINDEX Function

This function creates a single entry field with a prompting text, such as "*enter value,*" then sends that value to the URL of the page or program.

Syntax

```
HTF.ISINDEX(
   cprompt        IN       VARCHAR2    DEFAULT NULL,
   curl           IN       VARCHAR2    DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–52 ISINDEX Function Parameters

Parameter	Description
cprompt	The value for the PROMPT attribute.
curl	The value for the HREF attribute.

Examples

This function generates

```
<ISINDEX PROMPT="cprompt" HREF="curl">
```

ITALIC Function

This function generates

the <I> and

</I> tags which direct the browser to render the text in italics.

Syntax

```
HTF.ITALIC(
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–53 ITALIC Function Parameters

Parameter	Description
ctext	The text to be rendered in italics.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<I cattributes>ctext</I>
```

KBD Function

This function generates the <KBD> and </KBD> tags which direct the browser to render the text in monospace font. This subprogram performs the same operation as the KEYBOARD Function.

Syntax

```
HTF.KBD(
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–54 KBD Function Parameters

Parameter	Description
ctext	The text to be rendered in monospace.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<KBD cattributes>ctext</KBD>
```

KEYBOARD Function

This function generates the <KBD> and </KBD> tags, which direct the browser to render the text in monospace font. This subprogram performs the same operation as the KBD Function.

Syntax

```
HTF.KEYBOARD(
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–55 KEYBOARD Function Parameters

Parameter	Description
ctext	The text to be rendered in monospace.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<KBD cattributes>ctext</KBD>
```

LINE Function

This function generates the <HR> tag, which generates a line in the HTML document. This subprogram performs the same operation as the HR Function.

Syntax

```
HTF.LINE(
   cclear          IN       VARCHAR2   DEFAULT NULL,
   csrc            IN       VARCHAR2   DEFAULT NULL,
   cattributes     IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–56 LINE Function Parameters

Parameter	Description
cclear	The value for the CLEAR attribute.
csrc	The value for the SRC attribute which specifies a custom image as the source of the line.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<HR CLEAR="cclear" SRC="csrc" cattributes>
```

LINKREL Function

This function generates the <LINK> tag with the REL attribute which delineates the relationship described by the hypertext link from the anchor to the target. This is only used when the HREF attribute is present. This is the opposite of LINKREV Function. This tag indicates a relationship between documents but does not create a link. To create a link, use the ANCHOR Function.

Syntax

```
HTF.LINKREL(
   crel           IN       VARCHAR2,
   curl           IN       VARCHAR2,
   ctitle         IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–57 LINKREL Function Parameters

Parameter	Description
crel	The value for the REL attribute.
curl	The value for the URL attribute.
ctitle	The value for the TITLE attribute.

Examples

This function generates

```
<LINK REL="crel" HREF="curl" TITLE="ctitle">
```

LINKREV Function

This function generates the <LINK> tag with the REV attribute which delineates the relationship described by the hypertext link from the target to the anchor. This is the opposite of the LINKREL Function. This tag indicates a relationship between documents, but does not create a link. To create a link, use the ANCHOR Function.

Syntax

```
HTF.LINKREV(
   crev           IN       VARCHAR2,
   curl           IN       VARCHAR2,
   ctitle         IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–58 LINKREV Function Parameters

Parameter	Description
crev	The value for the REV attribute.
curl	The value for the URL attribute.
ctitle	The value for the TITLE attribute.

Examples

This function generates

```
<LINK REV="crev" HREF="curl" TITLE="ctitle">
```

LISTHEADER Function

This function generates the <LH> and </LH> tags which print an HTML tag at the beginning of the list.

Syntax

```
HTF.LISTHEADER(
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–59 LISTHEADER Function Parameters

Parameter	Description
ctext	The text to place between <LH> and </LH>.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<LH cattributes>ctext</LH>
```

LISTINGCLOSE Function

This function generates the </LISTING> tags which marks the end of a section of fixed-width text in the body of an HTML page. To mark the beginning of a section of fixed-width text in the body of an HTML page, use the LISTINGOPEN Function.

Syntax

```
HTF.LISTINGCLOSE
  RETURN VARCHAR2;
```

Examples

This function generates

```
</LISTING>
```

LISTINGOPEN Function

This function generates the <LISTING> tag which marks the beginning of a section of fixed-width text in the body of an HTML page. To mark the end of a section of fixed-width text in the body of an HTML page, use the LISTINGCLOSE Function.

Syntax

```
HTF.LISTINGOPEN
  RETURN VARCHAR2;
```

Examples

This function generates

```
<LISTING>
```

LISTITEM Function

This function generates the <LI> tag, which indicates a list item.

Syntax

```
HTF.LISTITEM(
   ctext          IN       VARCHAR2   DEFAULT NULL,
   cclear         IN       VARCHAR2   DEFAULT NULL,
   cdingbat       IN       VARCHAR2   DEFAULT NULL,
   csrc           IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–60 LISTITEM Function Parameters

Parameter	Description
ctext	The text for the list item.
cclear	The value for the CLEAR attribute.
cdingbat	The value for the DINGBAT attribute.
csrc	The value for the SRC attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<LI CLEAR="cclear" DINGBAT="cdingbat" SRC="csrc" cattributes>ctext
```

MAILTO Function

This function generates the <A> tag with the HREF set to 'mailto' prepended to the mail address argument.

Syntax

```
HTF.MAILTO(
   caddress       IN       VARCHAR2,
   ctext          IN       VARCHAR2,
   cname          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–61 MAILTO Function Parameters

Parameter	Description
caddress	The email address of the recipient.
ctext	The clickable portion of the link.
cname	The value for the NAME attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<A HREF="mailto:caddress" NAME="cname" cattributes>ctext</A>
```

so that

```
HTF.mailto('pres@white_house.gov','Send Email to the President');
```

generates:

```
<A HREF="mailto:pres@white_house.gov">Send Email to the President</A>
```

MAPCLOSE Function

This function generates the </MAP> tag which marks the end of a set of regions in a client-side image map. To mark the beginning of a set of regions in a client-side image map, use the MAPOPEN Function.

Syntax

```
HTF.MAPCLOSE
  RETURN VARCHAR2;
```

Examples

This function generates

```
</MAP>
```

MAPOPEN Function

This function generates the <MAP> tag which mark the beginning of a set of regions in a client-side image map. To mark the end of a set of regions in a client-side image map, use the MAPCLOSE Function.

Syntax

```
HTF.MAPOPEN(
   cname          IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–62 MAPOPEN Function Parameters

Parameter	Description
cname	The value for the NAME attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<MAP NAME="cname" cattributes>
```

MENULISTCLOSE Function

This function generates the </MENU> tag which ends a list that presents one line for each item. To begin a list of this kind, use the MENULISTOPEN Function. The items in the list appear more compact than an unordered list. The LISTITEM Function defines the list items in a menu list.

Syntax

```
HTF.MENULISTCLOSE
  RETURN VARCHAR2;
```

Examples

This function generates

```
</MENU>
```

MENULISTOPEN Function

This function generates the <MENU> tag which create a list that presents one line for each item. To end a list of this kind, use the MENULISTCLOSE Function.The items in the list appear more compact than an unordered list. The LISTITEM Function defines the list items in a menu list.

Syntax

```
HTF.MENULISTOPEN
  RETURN VARCHAR2;
```

Examples

This function generates

```
<MENU>
```

META Function

This function generates the <META> tag, which embeds meta-information about the document and also specifies values for HTTP headers. For example, you can specify the expiration date, keywords, and author name.

Syntax

```
HTF.META(
   chttp_equiv    IN       VARCHAR2,
   cname          IN       VARCHAR2,
   ccontent       IN       VARCHAR2)
  RETURN VARCHAR2;
```

Parameters

Table 134–63 META Function Parameters

Parameter	Description
chttp_equiv	The value for the CHTTP_EQUIV attribute.
cname	The value for the NAME attribute.
ccontent	The value for the CONTENT attribute.

Examples

This function generates

```
<META HTTP-EQUIV="chttp_equiv" NAME ="cname" CONTENT="ccontent">
```

so that

```
HTF.meta ('Refresh', NULL, 120);
```

generates

```
<META HTTP-EQUIV="Refresh" CONTENT=120>
```

On some Web browsers, this causes the current URL to be reloaded automatically every 120 seconds.

NL Function

This function generates the <BR> tag which begins a new line of text. It performs the same operation as the BR Function.

Syntax

```
HTF.NL(
   cclear         IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–64 NL Function Parameters

Parameter	Description
cclear	The value for the CLEAR attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<BR CLEAR="cclear" cattributes>
```

NOBR Function

This function generates the <NOBR> and </NOBR> tags which turn off line-breaking in a section of text.

Syntax

```
HTF.NOBR(
ctext        IN        VARCHAR2)
  RETURN VARCHAR2;
```

Parameters

Table 134–65 NOBR Function Parameters

Parameter	Description
ctext	The text that is to be rendered on one line.

Examples

This function generates

```
<NOBR>ctext</NOBR>
```

NOFRAMESCLOSE Function

This function generates the </NOFRAMES> tag which marks the end of a no-frames section. To mark the beginning of a no-frames section, use the FRAMESETOPEN Function. See also FRAME Function, FRAMESETOPEN Function and FRAMESETCLOSE Function.

Syntax

```
HTF.NOFRAMESCLOSE
  RETURN VARCHAR2;
```

Examples

This function generates

```
</NOFRAMES>
```

NOFRAMESOPEN Function

This function generates the <NOFRAMES> tag which mark the beginning of a no-frames section. To mark the end of a no-frames section, use the FRAMESETCLOSE Function. See also FRAME Function, FRAMESETOPEN Function and FRAMESETCLOSE Function.

Syntax

```
HTF.NOFRAMESOPEN
  RETURN VARCHAR2;
```

Examples

This function generates

```
<NOFRAMES>
```

OLISTCLOSE Function

This function generates the </OL> tag which defines the end of an ordered list. An ordered list presents a list of numbered items. To mark the beginning of a list of this kind, use the OLISTOPEN Function. Numbered items are added using LISTITEM Function.

Syntax

```
HTF.OLISTCLOSE
  RETURN VARCHAR2;
```

Examples

This function generates

```
</OL>
```

OLISTOPEN Function

This function generates the <OL> tag which marks the beginning of an ordered list. An ordered list presents a list of numbered items. To mark the end of a list of this kind, use the OLISTCLOSE Function. Numbered items are added using LISTITEM Function.

Syntax

```
HTF.OLISTOPEN(
   cclear         IN       VARCHAR2   DEFAULT NULL,
   cwrap          IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–66 OLISTOPEN Function Parameters

Parameter	Description
cclear	The value for the CLEAR attribute.
cwrap	The value for the WRAP attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<OL CLEAR="cclear" WRAP="cwrap" cattributes>
```

PARA Function

This function generates the <P> tag which indicates that the text that comes after the tag is to be formatted as a paragraph. You can add attributes to the tag by means of the PARAGRAPH Function.

Syntax

```
HTF.PARA
  RETURN VARCHAR2;
```

Examples

This function generates

```
<P>
```

PARAGRAPH Function

You can use this function to add attributes to the <P> tag created by the PARA Function.

Syntax

```
HTF.PARAGRAPH(
   calign         IN       VARCHAR2   DEFAULT NULL,
   cnowrap        IN       VARCHAR2   DEFAULT NULL,
   cclear         IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–67 PARAGRAPH Function Parameters

Parameter	Description
calign	The value for the ALIGN attribute.
cnowrap	If the value for this parameter is not NULL, the NOWRAP attribute is added to the tag.
cclear	The value for the CLEAR attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<P ALIGN="calign" NOWRAP CLEAR="cclear" cattributes>
```

PARAM Function

This function generates the <PARAM> tag which specifies parameter values for Java applets. The values can reference HTML variables. To invoke a Java applet from a Web page, use APPLETOPEN Function to begin the invocation. Use one PARAM Function for each desired name-value pair, and use APPLETCLOSE Function to end the applet invocation.

Syntax

```
HTF.PARAM(
   cname           IN       VARCHAR2
   cvalue          IN       VARCHAR2)
  RETURN VARCHAR2;
```

Parameters

Table 134–68 PARAM Function Parameters

Parameter	Description
cname	The value for the NAME attribute.
cvalue	The value for the VALUE attribute.

Examples

This function generates

```
<PARAM NAME=cname VALUE="cvalue">
```

PLAINTEXT Function

This function generates the <PLAINTEXT> and </PLAINTEXT> tags which direct the browser to render the text they surround in fixed-width type.

Syntax

```
HTF.PLAINTEXT(
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–69 PLAINTEXT Function Parameters

Parameter	Description
ctext	The text to be rendered in fixed-width font.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<PLAINTEXT cattributes>ctext</PLAINTEXT>
```

PRECLOSE Function

This function generates the </PRE> tag which marks the end of a section of preformatted text in the body of the HTML page. To mark the beginning of a section of preformatted text in the body of the HTML page, use the PREOPEN Function.

Syntax

```
HTF.PRECLOSE
  RETURN VARCHAR2;
```

Examples

This function generates

```
</PRE>
```

PREOPEN Function

This function generates the <PRE> tag which marks the beginning of a section of preformatted text in the body of the HTML page. To mark the end of a section of preformatted text in the body of the HTML page, use the PRECLOSE Function.

Syntax

```
HTF.PREOPEN(
   cclear         IN       VARCHAR2   DEFAULT NULL,
   cwidth         IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–70 PREOPEN Function Parameters

Parameter	Description
cclear	The value for the CLEAR attribute.
cwidth	The value for the WIDTH attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<PRE CLEAR="cclear" WIDTH="cwidth" cattributes>
```

PRINT Functions

These functions generate the specified parameter as a string terminated with the \n newline character. The PRN Functions performs the same operation but does not terminate with a newline character.

Syntax

```
HTF.PRINT (
   cbuf      IN       VARCHAR2)
  RETURN VARCHAR2;

HTF.PRINT (
   dbuf      IN       DATE)
  RETURN VARCHAR2;

HTF.PRINT (
   nbuf      IN       NUMBER)
  RETURN VARCHAR2;
```

Parameters

Table 134–71 PRINT Function Parameters

Parameter	Description
cbuf	The string to generate terminated by a newline.
dbuf	The string to generate terminated by a newline.
nbuf	The string to generate terminated by a newline.

Usage Notes

- The \n character is not the same as <BR>. The \n character formats the HTML source but it does not affect how the browser renders the HTML source. Use <BR> to control how the browser renders the HTML source.
- These functions do not have function equivalents.

PRN Functions

These functions generate the specified parameter as a string. Unlike the PRINT Functions the string is not terminated with the \n newline character.

Syntax

```
HTF.PRN (
   cbuf      IN       VARCHAR2)
  RETURN VARCHAR2;

HTF.PRN (
   dbuf      IN       DATE)
  RETURN VARCHAR2;

HTF.PRN (
   nbuf      IN       NUMBER)
  RETURN VARCHAR2;
```

Parameters

Table 134–72 PRN Function Parameters

Parameter	Description
cbuf	The string to generate (not terminated by a newline).
dbuf	The string to generate (not terminated by a newline).
nbuf	The string to generate (not terminated by a newline).

Usage Notes

These functions do not have function equivalents.

S Function

This function generates the <S> and </S> tags which direct the browser to render the text they surround in strikethrough type. This performs the same operation as STRIKE Function.

Syntax

```
HTF.S (
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–73 S Function Parameters

Parameter	Description
ctext	The text to be rendered in strikethrough type.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<S cattributes>ctext</S>
```

SAMPLE Function

This function generates the <SAMP> and </SAMP> tags which direct the browser to render the text they surround in monospace font or however "sample" is defined stylistically.

Syntax

```
HTF.SAMPLE (
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–74 SAMPLE Function Parameters

Parameter	Description
ctext	The text to be rendered in monospace font.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<SAMP cattributes>ctext</SAMP>
```

SCRIPT Function

This function generates the <SCRIPT> and </SCRIPT> tags which contain a script written in languages such as JavaScript and VBscript.

Syntax

```
HTF.SCRIPT (
   cscript        IN       VARCHAR2,
   clanguage      IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–75 SCRIPT Function Parameters

Parameter	Description
cscript	The text of the script. This is the text that makes up the script itself, not the name of a file containing the script.
clanguage	The language in which the script is written. If this parameter is omitted, the user's browser determines the scripting language.

Examples

This function generates

```
<SCRIPT LANGUAGE=clanguage>cscript</SCRIPT>
```

so that

```
HTF.script ('Erupting_Volcano', 'Javascript');
```

generates

```
<SCRIPT LANGUAGE=Javascript>"script text here"</SCRIPT>
```

This causes the browser to run the script enclosed in the tags.

SMALL Function

This function generates the <SMALL> and </SMALL> tags, which direct the browser to render the text they surround using a small font.

Syntax

```
HTF.SMALL (
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–76 SMALL Function Parameters

Parameter	Description
ctext	The text to be rendered in small font.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<SMALL cattributes>ctext</SMALL>
```

STRIKE Function

This function generates the <STRIKE> and </STRIKE> tags which direct the browser to render the text they surround in strikethrough type. This performs the same operation as S Function.

Syntax

```
STRIKE (
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–77 STRIKE Function Parameters

Parameter	Description
ctext	The text to be rendered in strikethrough type.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<STRIKE cattributes>ctext</STRIKE>
```

STRONG Function

This function generates the <STRONG> and </STRONG> tags which direct the browser to render the text they surround in bold or however "strong" is defined.

Syntax

```
HTF.STRONG(
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–78 STRONG Function Parameters

Parameter	Description
ctext	The text to be emphasized.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<STRONG cattributes>ctext</STRONG>
```

STYLE Function

This function generates the <STYLE> and </STYLE> tags which include a style sheet in a Web page. You can get more information about style sheets at http://www.w3.org. This feature is not compatible with browsers that support only HTML versions 2.0 or earlier. Such browsers will ignore this tag.

Syntax

```
HTF.STYLE(
   cstyle          IN       VARCHAR2)
  RETURN VARCHAR2;
```

Parameters

Table 134–79 STYLE Function Parameters

Parameter	Description
cstyle	The the style information to include.

Examples

This function generates

```
<STYLE>cstyle</STYLE>
```

SUB Function

This function generates the _{and} tags which direct the browser to render the text they surround as subscript.

Syntax

```
HTF.SUB(
   ctext          IN       VARCHAR2,
   calign         in       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–80 SUB Function Parameters

Parameter	Description
ctext	The text to render in subscript.
calign	The value for the ALIGN attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<SUB ALIGN="calign" cattributes>ctext</SUB>
```

SUP Function

This function generates the ^{and} tags which direct the browser to render the text they surround as superscript.

Syntax

```
HTF.SUP(
   ctext          IN       VARCHAR2,
   calign         in       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–81 SUP Function Parameters

Parameter	Description
ctext	The text to render in superscript.
calign	The value for the ALIGN attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<SUP ALIGN="calign" cattributes>ctext</SUP>
```

TABLECAPTION Function

This function generates the <CAPTION> and </CAPTION> tags which place a caption in an HTML table.

Syntax

```
HTF.TABLECAPTION(
   ccaption       IN       VARCHAR2,
   calign         in       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–82 TABLECAPTION Function Parameters

Parameter	Description
ctext	The text for the caption.
calign	The value for the ALIGN attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<CAPTION ALIGN="calign" cattributes>ccaption</CAPTION>
```

TABLECLOSE Function

This function generates the </TABLE> tag which marks the end of an HTML table. To define the beginning of an HTML table, use the TABLEOPEN Function on page 134-126.

Syntax

```
HTF.TABLECLOSE
  RETURN VARCHAR2;
```

Examples

This function generates

```
</TABLE>
```

TABLEDATA Function

This function generates the <TD> and </TD> tags which insert data into a cell of an HTML table.

Syntax

```
HTF.TABLEDATA(
   cvalue         IN       VARCHAR2   DEFAULT NULL,
   calign         IN       VARCHAR2   DEFAULT NULL,
   cdp            IN       VARCHAR2   DEFAULT NULL,
   cnowrap        IN       VARCHAR2   DEFAULT NULL,
   crowspan       IN       VARCHAR2   DEFAULT NULL,
   ccolspan       IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–83 TABLEDATA Function Parameters

Parameter	Description
cvalue	The data for the cell in the table.
calign	The value for the ALIGN attribute.
cdp	The value for the DP attribute.
cnowrap	If the value of this parameter is not NULL, the NOWRAP attribute is added to the tag.
ccolspan	The value for the COLSPAN attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<TD ALIGN="calign" DP="cdp" ROWSPAN="crowspan" COLSPAN="ccolspan" NOWRAP
cattributes>cvalue</TD>
```

TABLEHEADER Function

This function generates the <TH> and </TH> tags which insert a header cell in an HTML table. The <TH> tag is similar to the <TD> tag except that the text in this case the rows are usually rendered in bold type.

Syntax

```
HTF.TABLEHEADER(
   cvalue          IN       VARCHAR2   DEFAULT NULL,
   calign          IN       VARCHAR2   DEFAULT NULL,
   cdp             IN       VARCHAR2   DEFAULT NULL,
   cnowrap         IN       VARCHAR2   DEFAULT NULL,
   crowspan        IN       VARCHAR2   DEFAULT NULL,
   ccolspan        IN       VARCHAR2   DEFAULT NULL,
   cattributes     IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–84 TABLEHEADER Function Parameters

Parameter	Description
cvalue	The data for the cell in the table.
calign	The value for the ALIGN attribute.
cdp	The value for the DP attribute.
cnowrap	If the value of this parameter is not NULL, the NOWRAP attribute is added to the tag.
crispen	The value for the ROWSPAN attribute.
ccolspan	The value for the COLSPAN attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<TH ALIGN="calign" DP="cdp" ROWSPAN="crowspan" COLSPAN="ccolspan" NOWRAP
cattributes>cvalue</TH>
```

TABLEOPEN Function

This function generates the <TABLE> tag which marks the beginning of an HTML table. To define the end of an HTML table, use the TABLECLOSE Function.

Syntax

```
HTF.TABLEOPEN(
   cborder        IN       VARCHAR2   DEFAULT NULL
   calign         IN       VARCHAR2   DEFAULT NULL,
   cnowrap        IN       VARCHAR2   DEFAULT NULL,
   cclear         IN       VARCHAR2   DEFAULT NULL
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–85 TABLEOPEN Function Parameters

Parameter	Description
`border`	The value for the `BORDER` attribute.
`calign`	The value for the `ALIGN` attribute.
`cnowrap`	If the value of this parameter is not `NULL`, the `NOWRAP` attribute is added to the tag.
`cclear`	The value for the `CLEAR` attribute.
`cattributes`	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<TABLE "cborder" NOWRAP ALIGN="calign" CLEAR="cclear" cattributes>
```

TABLEROWCLOSE Function

This function generates the </TR> tag which marks the end of a new row in an HTML table. To mark the beginning of a new row, use the TABLEROWOPEN Function.

Syntax

```
HTF.TABLEROWCLOSE
  RETURN VARCHAR2;
```

Examples

This function generates

```
</TABLE>
```

TABLEROWOPEN Function

This function generates the <TR> tag which marks the beginning of a new row in an HTML table. To mark the end of a new row, use the TABLEROWCLOSE Function.

Syntax

```
HTF.TABLEROWOPEN(
   calign         IN       VARCHAR2   DEFAULT NULL,
   cvalign        IN       VARCHAR2   DEFAULT NULL,
   cdp            IN       VARCHAR2   DEFAULT NULL,
   cnowrap        IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–86 TABLEROWOPEN Function Parameters

Parameter	Description
calign	The value for the ALIGN attribute.
cvalign	The value for the VALIGN attribute.
cdp	The value for the DP attribute.
cnowrap	If the value of this parameter is not NULL, the NOWRAP attribute is added to the tag.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<<TR ALIGN="calign" VALIGN="cvalign" DP="cdp" NOWRAP cattributes>
```

TELETYPE Function

This function generates the <TT> and </TT> tags which direct the browser to render the text they surround in a fixed width typewriter font, for example, the courier font.

Syntax

```
HTF.TELETYPE(
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–87 TELETYPE Function Parameters

Parameter	Description
ctext	The text to render in a fixed width typewriter font.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<TT cattributes>ctext</TT>
```

TITLE Function

This function generates the <TITLE> and </TITLE> tags which specify the text to display in the titlebar of the browser window.

Syntax

```
HTF.TITLE(
   ctitle          IN       VARCHAR2)
  RETURN VARCHAR2;
```

Parameters

Table 134–88 TITLE Function Parameters

Parameter	Description
ctitle	The text to display in the titlebar of the browser window.

Examples

This function generates

```
<TITLE>ctitle</TITLE>
```

ULISTCLOSE Function

This function generates the </UL> tag which marks the end of an unordered list. An unordered list presents items with bullets. To mark the beginning of an unordered list, use the ULISTOPEN Function. Add list items with LISTITEM Function.

Syntax

```
HTF.ULISTCLOSE
  RETURN VARCHAR2;
```

Examples

This function generates

```
</UL>
```

ULISTOPEN Function

This function generates the <UL> tag which marks the beginning of an unordered list. An unordered list presents items with bullets. To mark the end of an unordered list, use the ULISTCLOSE Function. Add list items with LISTITEM Function.

Syntax

```
HTF.ULISTOPEN(
   cclear          IN       VARCHAR2   DEFAULT NULL,
   cwrap           IN       VARCHAR2   DEFAULT NULL,
   cdingbat        IN       VARCHAR2   DEFAULT NULL,
   csrc            IN       VARCHAR2   DEFAULT NULL,
   cattributes     IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–89 ULISTOPEN Function Parameters

Parameter	Description
cclear	The value for the CLEAR attribute.
cwrap	The value for the WRAP attribute.
cdingbat	The value for the DINGBAT attribute.
csrc	The value for the SRC attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<UL CLEAR="cclear" WRAP="cwrap" DINGBAT="cdingbat" SRC="csrc" cattributes>
```

UNDERLINE Function

This function generates the <U> and </U> tags, which direct the browser to render the text they surround with an underline.

Syntax

```
HTF.UNDERLINE(
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–90 UNDERLINE Function Parameters

Parameter	Description
ctext	The text to render with an underline.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<U cattributes>ctext</U>
```

VARIABLE Function

This function generates the <VAR> and </VAR> tags which direct the browser to render the text they surround in italics or however "variable" is defined stylistically.

Syntax

```
HTF.VARIABLE(
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 134–91 VARIABLE Function Parameters

Parameter	Description
ctext	The text to render in italics.
cattributes	The other attributes to be included as-is in the tag.

Examples

This function generates

```
<VAR cattributes>ctext</VAR>
```

WBR Function

This function generates the <WBR> tag, which inserts a soft line break within a section of NOBR text.

Syntax

```
HTF.WBR
  RETURN VARCHAR2;
```

Examples

This function generates

```
<WBR>
```

135

HTMLDB_CUSTOM_AUTH

The HTMLDB_CUSTOM_AUTH package provides an interface for authentication and session management.

Documentation of HTMLDB_CUSTOM_AUTH

For a complete description of this package within the context of HTMLDB, see `HTMLDB_CUSTOM_AUTH` in the *Oracle HTML DB User's Guide*.

136

HTMLDB_APPLICATION

The `HTMLDB_APPLICATION` package enables users to take advantage of global variables.

- Documentation of HTMLDB_APPLICATION

Documentation of HTMLDB_APPLICATION

For a complete description of this package within the context of HTMLDB, see `HTMLDB_APPLICATION` in the *Oracle HTML DB User's Guide*.

137

HTMLDB_ITEM

The HTMLDB_ITEM package enables users to create form elements dynamically based on a SQL query instead of creating individual items page by page.

Documentation of HTMLDB_ITEM

For a complete description of this package within the context of HTMLDB, see `HTMLDB_ITEM` in the *Oracle HTML DB User's Guide.*

138

HTMLDB_UTIL

The HTMLDB_UTIL package provides utilities for getting and setting session state, getting files, checking authorizations for users, resetting different states for users, and also getting and setting preferences for users.

Documentation of HTMLDB_UTIL

For a complete description of this package within the context of HTMLDB, see `HTMLDB_UTIL` in the *Oracle HTML DB User's Guide.*

139

HTP

The HTP (hypertext procedures) and HTF (hypertext functions) packages generate HTML tags. For example, the HTP.ANCHOR procedure generates the HTML anchor tag, <A>.

> **See Also:** For more information about implementation of this package:
>
> - *Oracle HTTP Server Administrator's Guide*
> - *Oracle Application Server mod_plsql User's Guide*

This chapter contains the following topics:

Using HTP

Operational Notes

For every HTP procedure that generates one or more HTML tags, there is a corresponding HTF function with identical parameters with the following exception:

- The PRINTS Procedure and the PS Procedure do not have HTF function equivalents. Use the ESCAPE_SC Function or the ESCAPE_URL Function if you need a string conversion function. Note that while there is a ESCAPE_SC Procedure that performs the same operation as the PRINTS Procedure and the PS Procedure, there is no procedural equivalent for the ESCAPE_URL Function.
- The FORMAT_CELL Function does not have an HTP equivalent. The function formats column values inside an HTML table using TABLEDATA Function which does have an HTP equivalent in the TABLEDATA Procedure. The advantage of this using the FORMAT_CELL Function is that it allows for better control over the HTML tables.

The function versions do not directly generate output in your web page. Instead, they pass their output as return values to the statements that invoked them. Use these functions when you need to nest calls. To print the output of HTF functions, call the functions from within the HTP.`PRINT` procedure. It then prints its parameters to the generated web page.

Rules and Limits

If you use values of the LONG data type in procedures such as HTP.PRINT, HTP.PRN, HTP.PRINTS, HTP.PA or OWA_UTIL.CELLSPRINT, only the first 32 K of the LONG data is used. The LONG data is bound to a VARCHAR2 data type in the procedure.

Examples

The following commands generate a simple HTML document:

```
CREATE OR REPLACE PROCEDURE hello AS
BEGIN
    HTP.HTMLOPEN;           -- generates <HTML>
    HTP.HEADOPEN;           -- generates <HEAD>
    HTP.TITLE('Hello');     -- generates <TITLE>Hello</TITLE>
    HTP.HEADCLOSE;          -- generates </HEAD>
    HTP.BODYOPEN;           -- generates <BODY>
    HTP.HEADER(1, 'Hello'); -- generates <H1>Hello</H1>
    HTP.BODYCLOSE;          -- generates </BODY>
    HTP.HTMLCLOSE;          -- generates </HTML>
END;
```

Summary of Tags

HTML, HEAD, and BODY Tags

HTMLOPEN Procedure, HTMLCLOSE Procedure - generate <HTML> and </HTML>

HEADOPEN Procedure, HEADCLOSE Procedure - generate <HEAD> and </HEAD>

BODYOPEN Procedure, BODYCLOSE Procedure - generate <BODY> and </BODY>

Comment Tag

COMMENT Procedure - generates <!-- and -->

Tags in the <HEAD> Area

BASE Procedure - generates <BASE>

LINKREL Procedure - generates <LINK> with the REL attribute

LINKREV Procedure - generates <LINK> with the REV attribute

TITLE Procedure - generates <TITLE>

META Procedure - generates <META>

SCRIPT Procedure - generates <SCRIPT>

STYLE Procedure - generates <STYLE>

ISINDEX Procedure - generates <ISINDEX>

Applet Tags

APPLETOPEN Procedure, APPLETCLOSE Procedure - generate <APPLET> and </APPLET>

PARAM Procedure - generates <PARAM>

List Tags

OLISTOPEN Procedure, OLISTCLOSE Procedure - generate <OL> and </OL>

ULISTOPEN Procedure, ULISTCLOSE Procedure - generate <UL> and </UL>

DLISTOPEN Procedure, DLISTCLOSE Procedure- generate <DL> and </DL>

DLISTTERM Procedure - generates <DT>

DLISTDEF Procedure - generates <DD>

DIRLISTOPEN Procedure, DIRLISTCLOSE Procedure - generate <DIR> and </DIR>

LISTHEADER Procedure - generates <LH>

LISTINGOPEN Procedure, LISTINGCLOSE Procedure - generate <LISTING> and </LISTING>

MENULISTOPEN Procedure - generate <MENU> and </MENU>

LISTITEM Procedure - generates <LI>

Form Tags

FORMOPEN Procedure, FORMCLOSE Procedure - generate <FORM> and </FORM>

FORMCHECKBOX Procedure - generates <INPUT TYPE="CHECKBOX">

FORMHIDDEN Procedure - generates <INPUT TYPE="HIDDEN">

FORMIMAGE Procedure - generates <INPUT TYPE="IMAGE">

FORMPASSWORD Procedure - generates <INPUT TYPE="PASSWORD">

FORMRADIO Procedure - generates <INPUT TYPE="RADIO">

FORMSELECTOPEN Procedure, FORMSELECTCLOSE Procedure - generate <SELECT> and </SELECT>

FORMSELECTOPTION Procedure - generates <OPTION>

FORMTEXT Procedure - generates <INPUT TYPE="TEXT">

FORMTEXTAREA Procedure - generate <TEXTAREA>

FORMTEXTAREAOPEN Procedure, FORMTEXTAREACLOSE Procedure - generate <TEXTAREA> and </TEXTAREA>

FORMRESET Procedure - generates <INPUT TYPE="RESET">

FORMSUBMIT Procedure - generates <INPUT TYPE="SUBMIT">

Table Tags

TABLEOPEN Procedure, TABLECLOSE Procedure - generate <TABLE> and </TABLE>

TABLECAPTION Procedure - generates <CAPTION>

TABLEROWOPEN Procedure, TABLEROWCLOSE Procedure - generate <TR> and </TR>

TABLEHEADER Procedure - generates <TH>

TABLEDATA Procedure - generates <TD>

IMG, HR, and A Tags

HR Procedure, LINE Procedure - generate <HR>

IMG Procedure, IMG2 Procedure - generate <IMG>

ANCHOR Procedure, ANCHOR2 Procedure - generate <A>

MAPOPEN Procedure, MAPCLOSE Procedure - generate <MAP> and </MAP>

Paragraph Formatting Tags

HEADER Procedure - generates heading tags (<H1> to <H6>)

PARA Procedure, PARAGRAPH Procedure - generate <P>

PRN Procedures, PRINT Procedures - generate any text that is passed in

PRINTS Procedure, PS Procedure - generate any text that is passed in; special characters in HTML are escaped

PREOPEN Procedure, PRECLOSE Procedure - generate <PRE> and </PRE>

BLOCKQUOTEOPEN Procedure, BLOCKQUOTECLOSE Procedure - generate <BLOCKQUOTE> and </BLOCKQUOTE>

DIV Procedure - generates <DIV>

NL Procedure, BR Procedure - generate

NOBR Procedure - generates <NOBR>

WBR Procedure - generates <WBR>

PLAINTEXT Procedure - generates <PLAINTEXT>

ADDRESS Procedure - generates <ADDRESS>

MAILTO Procedure - generates <A> with the MAILTO attribute

AREA Procedure - generates <AREA>

BGSOUND Procedure - generates <BGSOUND>

Character Formatting Tags

BASEFONT Procedure - generates <BASEFONT>

BIG Procedure - generates <BIG>

BOLD Procedure - generates <B>

CENTER Procedure - generates <CENTER> and </CENTER>

CENTEROPEN Procedure, CENTERCLOSE Procedure - generate <CENTER> and </CENTER>

CITE Procedure - generates <CITE>

CODE Procedure - generates <CODE>

DFN Procedure - generates <DFN>

EM Procedure, EMPHASIS Procedure - generate <EM>

FONTOPEN Procedure, FONTCLOSE Procedure - generate <FONT> and </FONT>

ITALIC Procedure - generates <I>

KBD Procedure, KEYBOARD Procedure - generate <KBD> and </KBD>

S Procedure - generates <S>

SAMPLE Procedure - generates <SAMP>

SMALL Procedure - generates <SMALL>

STRIKE Procedure - generates <STRIKE>

STRONG Procedure - generates <STRONG>

SUB Procedure - generates <SUB>

SUP Procedure - generates <SUP>

TELETYPE Procedure - generates <TT>

UNDERLINE Procedure - generates <U>

VARIABLE Procedure - generates <VAR>

Frame Tags

FRAME Procedure - generates <FRAME>

FRAMESETOPEN Procedure, FRAMESETCLOSE Procedure - generate <FRAMESET> and </FRAMESET>

NOFRAMESOPEN Procedure, NOFRAMESCLOSE Procedure - generate <NOFRAMES> and </NOFRAMES>

Summary of HTP Subprograms

Table 139–1 HTP Package Subprograms

Subprogram	Description
ADDRESS Procedure on page 139-16	Generate s the <ADDRESS> and </ADDRESS> tags which specify the address, author and signature of a document
ANCHOR Procedure on page 139-17	Generate s the <A> and </A> tags which specify the source or destination of a hypertext link
ANCHOR2 Procedure on page 139-18	Generates the <A> and </A> tags which specify the source or destination of a hypertext link
APPLETCLOSE Procedure on page 139-19	Closes the applet invocation with the </APPLET> tag
APPLETOPEN Procedure on page 139-20	Generates the <APPLET> tag which begins the invocation of a Java applet
AREA Procedure on page 139-21	Generates the <AREA> tag, which defines a client-side image map
BASE Procedure on page 139-22	Generates the <BASE> tag which records the URL of the document
BASEFONT Procedure on page 139-23	Generates the <BASEFONT> tag which specifies the base font size for a Web page
BGSOUND Procedure on page 139-24	Generates the <BGSOUND> tag which includes audio for a Web page
BIG Procedure on page 139-25	Generates the <BIG> and </BIG> tags which direct the browser to render the text in a bigger font
BLOCKQUOTECLOSE Procedure on page 139-26	Generates the </BLOCKQUOTE> tag which mark the end of a section of quoted text
BLOCKQUOTEOPEN Procedure on page 139-27	Generates the <BLOCKQUOTE> tag, which marks the beginning of a section of quoted text
BODYCLOSE Procedure on page 139-28	Generates the </BODY> tag which marks the end of a body section of an HTML document
BODYOPEN Procedure on page 139-29	Generates the <BODY> tag which marks the beginning of the body section of an HTML document
BOLD Procedure on page 139-30	Generates the <B> and </B> tags which direct the browser to display the text in boldface
BR Procedure on page 139-31	Generates the tag which begins a new line of text
CENTER Procedure on page 139-32	Generates the <CENTER> and </CENTER> tags which center a section of text within a Web page

Table 139–1 (Cont.) HTP Package Subprograms

Subprogram	Description
CENTERCLOSE Procedure on page 139-33	Generates the `</CENTER>` tag which marks the end of a section of text to center
CENTEROPEN Procedure on page 139-34	Generates the `<CENTER>` tag which mark the beginning of a section of text to center
CITE Procedure on page 139-35	Generates the `<CITE>` and `</CITE>` tags which direct the browser to render the text as a citation
CODE Procedure on page 139-36	Generates the `<CODE>` and `</CODE>` tags which direct the browser to render the text in monospace font or however "code" is defined stylistically
COMMENT Procedure on page 139-37	Generates This procedure generates the comment tags `<!-- ctext -->`
DFN Procedure on page 139-38	Generates the `<DFN>` and `</DFN>` tags which direct the browser to mark the text as italics or however "definition" is defined stylistically
DIRLISTCLOSE Procedure on page 139-39	Generates the `</DIR>` tag which ends a directory list section
DIRLISTOPEN Procedure on page 139-40	Generates the `<DIR>` which starts a directory list section
DIV Procedure on page 139-41	Generates the `<DIV>` tag which creates document divisions
DLISTCLOSE Procedure on page 139-42	Generates the `</DL>` tag which ends a definition list
DLISTDEF Procedure on page 139-43	Generates the `<DD>` tag, which inserts definitions of terms
DLISTOPEN Procedure on page 139-44	Generates the `<DL>` tag which starts a definition list
DLISTTERM Procedure on page 139-45	Generates the `<DT>` tag which defines a term in a definition list `<DL>`
EM Procedure on page 139-46	Generates the `<EM>` and `</EM>` tags, which define text to be emphasized
EMPHASIS Procedure on page 139-47	Generates the `<EM>` and `</EM>` tags, which define text to be emphasized
ESCAPE_SC Procedure on page 139-48	Replaces characters that have special meaning in HTML with their escape sequences
FONTCLOSE Procedure on page 139-49	Generates the `</FONT>` tag which marks the end of a section of text with the specified font characteristics
FONTOPEN Procedure on page 139-50	Generates the `<FONT>` which marks the beginning of section of text with the specified font characteristics
FORMCHECKBOX Procedure on page 139-51	Generates the `<INPUT>` tag with `TYPE="checkbox"` which inserts a checkbox element in a form
FORMCLOSE Procedure on page 139-52	Generates the `</FORM>` tag which marks the end of a form section in an HTML document
FORMOPEN Procedure on page 139-53	Generates the `<FORM>` tag which marks the beginning of a form section in an HTML document

Table 139–1 (Cont.) HTP Package Subprograms

Subprogram	Description
FORMFILE Procedure on page 139-54	Generates the <INPUT> tag with TYPE="file" which inserts a file form element, and is used for file uploading for a given page
FORMHIDDEN Procedure on page 139-55	Generates the <INPUT> tag with TYPE="hidden"which inserts a hidden form element
FORMIMAGE Procedure on page 139-56	Generates the <INPUT> tag with TYPE="image" which creates an image field that the user clicks to submit the form immediately
FORMPASSWORD Procedure on page 139-57	Generates the <INPUT> tag with TYPE="password" which creates a single-line text entry field
FORMRADIO Procedure on page 139-58	Generates the <INPUT> tag with TYPE="radio", which creates a radio button on the HTML form
FORMRESET Procedure on page 139-59	Generates the <INPUT> tag with TYPE="reset" which creates a button that, when selected, resets the form fields to their initial values
FORMSELECTCLOSE Procedure on page 139-60	Generates the </SELECT> tag which marks the end of a Select form element
FORMSELECTOPEN Procedure on page 139-61	Generates the </SELECT> tag which marks the beginning of a Select form element
FORMSELECTOPTION Procedure on page 139-62	Generates the <OPTION> tag which represents one choice in a Select element
FORMSUBMIT Procedure on page 139-63	Generates the <INPUT> tag with TYPE="submit" which creates a button that, when clicked, submits the form
FORMTEXT Procedure on page 139-64	Generates the <INPUT> tag with TYPE="text", which creates a field for a single line of text
FORMTEXTAREA Procedure on page 139-65	Generates the <TEXTAREA> tag, which creates a text field that has no predefined text in the text area
FORMTEXTAREA2 Procedure on page 139-66	Generates the <TEXTAREA> tag, which creates a text field that has no predefined text in the text area with the ability to specify a wrap style
FORMTEXTAREACLOSE Procedure on page 139-67	Generates the </TEXTAREA> tag which ends a text area form element
FORMTEXTAREAOPEN Procedure on page 139-68	Generates the <TEXTAREA> which marks the beginning of a text area form element
FORMTEXTAREAOPEN2 Procedure on page 139-69	Generates the <TEXTAREA> which marks the beginning of a text area form element with the ability to specify a wrap style
FRAME Procedure on page 139-70	Generates the <FRAME> tag which defines the characteristics of a frame created by a <FRAMESET> tag
FRAMESETCLOSE Procedure on page 139-71	Generates the </FRAMESET> tag which ends a frameset section
FRAMESETOPEN Procedure on page 139-72	Generates the </FRAMESET> tag which begins a frameset section
HEADCLOSE Procedure on page 139-73	Generates the </HEAD> tag which marks the end of an HTML document head section

Table 139–1 (Cont.) HTP Package Subprograms

Subprogram	Description
HEADER Procedure on page 139-74	Generates opening heading tags (<H1> to <H6>) and their corresponding closing tags (</H1> to </H6>)
HEADOPEN Procedure on page 139-75	Generates the <HEAD> tag which marks the beginning of the HTML document head section
HR Procedure on page 139-76	Generates the <HR> tag, which generates a line in the HTML document
HTMLCLOSE Procedure on page 139-77	Generates the </HTML> tag which marks the end of an HTML document
HTMLOPEN Procedure on page 139-78	Generates the <HTML> tag which marks the beginning of an HTML document
IMG Procedure on page 139-79	Generates the <IMG> tag which directs the browser to load an image onto the HTML page
IMG2 Procedure on page 139-80	Generates the <IMG> tag which directs the browser to load an image onto the HTML page with the option of specifying values for the USEMAP attribute
ISINDEX Procedure on page 139-81	Creates a single entry field with a prompting text, such as "*enter value*," then sends that value to the URL of the page or program
ITALIC Procedure on page 139-82	Generates the <I> and </I> tags which direct the browser to render the text in italics
KBD Procedure on page 139-83	Generates the <KBD> and </KBD> tags which direct the browser to render the text in monospace font
KEYBOARD Procedure on page 139-84	Generates the <KBD> and </KBD> tags, which direct the browser to render the text in monospace font
LINE Procedure on page 139-85	Generates the <HR> tag, which generates a line in the HTML document
LINKREL Procedure on page 139-86	Generates the <LINK> tag with the REL attribute which delineates the relationship described by the hypertext link from the anchor to the target
LINKREV Procedure on page 139-87	Generates the <LINK> tag with the REV attribute which delineates the relationship described by the hypertext link from the target to the anchor
LISTHEADER Procedure on page 139-88	Generates the <LH> and </LH> tags which print an HTML tag at the beginning of the list
LISTINGCLOSE Procedure on page 139-89	Generates the </LISTING> tags which marks the end of a section of fixed-width text in the body of an HTML page
LISTINGOPEN Procedure on page 139-90	Generates the <LISTING> tag which marks the beginning of a section of fixed-width text in the body of an HTML page
LISTITEM Procedure on page 139-91	Generates the <LI> tag, which indicates a list item
MAILTO Procedure on page 139-92	Generates the <A> tag with the HREF set to 'mailto' prepended to the mail address argument

Table 139–1 (Cont.) HTP Package Subprograms

Subprogram	Description
MAPCLOSE Procedure on page 139-93	Generates the </MAP> tag which marks the end of a set of regions in a client-side image map
MAPOPEN Procedure on page 139-94	Generates the <MAP> tag which mark the beginning of a set of regions in a client-side image map
MENULISTCLOSE Procedure on page 139-95	Generates the </MENU> tag which ends a list that presents one line for each item
MENULISTOPEN Procedure on page 139-96	Generates the <MENU> tag which create a list that presents one line for each item
META Procedure on page 139-97	Generates the <META> tag, which embeds meta-information about the document and also specifies values for HTTP headers
NL Procedure on page 139-98	Generates the tag which begins a new line of text
NOBR Procedure on page 139-99	Generates the <NOBR> and </NOBR> tags which turn off line-breaking in a section of text
NOFRAMESCLOSE Procedure on page 139-100	Generates the </NOFRAMES> tag which marks the end of a no-frames section
NOFRAMESOPEN Procedure on page 139-101	Generates the <NOFRAMES> tag which mark the beginning of a no-frames section
OLISTCLOSE Procedure on page 139-102	Generates the </OL> tag which defines the end of an ordered list
OLISTOPEN Procedure on page 139-103	Generates the <OL> tag which marks the beginning of an ordered list
PARA Procedure on page 139-104	Generates the <P> tag which indicates that the text that comes after the tag is to be formatted as a paragraph
PARAGRAPH Procedure on page 139-105	Adds attributes to the <P> tag
PARAM Procedure on page 139-106	Generates the <PARAM> tag which specifies parameter values for Java applets
PLAINTEXT Procedure on page 139-107	Generates the <PLAINTEXT> and </PLAINTEXT> tags which direct the browser to render the text they surround in fixed-width type
PRECLOSE Procedure on page 139-108	Generates the </PRE> tag which marks the end of a section of preformatted text in the body of the HTML page
PREOPEN Procedure on page 139-109	Generates the <PRE> tag which marks the beginning of a section of preformatted text in the body of the HTML page
PRINT Procedures on page 139-110	Generates the specified parameter as a string terminated with the \n newline character
PRINTS Procedure on page 139-111	Generates a string and replaces the following characters with the corresponding escape sequence
PRN Procedures on page 139-112	Generates the specified parameter as a string
PS Procedure on page 139-113	Generates a string and replaces the following characters with the corresponding escape sequence.
S Procedure on page 139-114	Generates the <S> and </S> tags which direct the browser to render the text they surround in strikethrough type

Table 139–1 (Cont.) HTP Package Subprograms

Subprogram	Description
SAMPLE Procedure on page 139-115	Generates the <SAMP> and </SAMP> tags which direct the browser to render the text they surround in monospace font or however "sample" is defined stylistically
SCRIPT Procedure on page 139-116	Generates the <SCRIPT> and </SCRIPT> tags which contain a script written in languages such as JavaScript and VBscript
SMALL Procedure on page 139-117	Generates the <SMALL> and </SMALL> tags, which direct the browser to render the text they surround using a small font
STRIKE Procedure on page 139-118	Generates the <STRIKE> and </STRIKE> tags which direct the browser to render the text they surround in strikethrough type
STRONG Procedure on page 139-119	Generates the <STRONG> and </STRONG> tags which direct the browser to render the text they surround in bold or however "strong" is defined stylistically
STYLE Procedure on page 139-120	Generates the <STYLE> and </STYLE> tags which include a style sheet in a Web page
SUB Procedure on page 139-121	Generates the _{and} tags which direct the browser to render the text they surround as subscript
SUP Procedure on page 139-122	Generates the ^{and} tags which direct the browser to render the text they surround as superscript
TABLECAPTION Procedure on page 139-123	Generates the <CAPTION> and </CAPTION> tags which place a caption in an HTML table
TABLECLOSE Procedure on page 139-124	Generates the </TABLE> tag which marks the end of an HTML table
TABLEDATA Procedure on page 139-125	Generates the <TD> and </TD> tags which insert data into a cell of an HTML table
TABLEHEADER Procedure on page 139-126	Generates the <TH> and </TH> tags which insert a header cell in an HTML table.
TABLEOPEN Procedure on page 139-127	Generates the <TABLE> tag which marks the beginning of an HTML table
TABLEROWCLOSE Procedure on page 139-128	Generates the </TR> tag which marks the end of a new row in an HTML table
TABLEROWOPEN Procedure on page 139-129	Generates the <TR> tag which marks the beginning of a new row in an HTML table
TELETYPE Procedure on page 139-130	Generates the <TT> and </TT> tags which direct the browser to render the text they surround in a fixed width typewriter font, for example, the courier font
TITLE Procedure on page 139-131	Generates the <TITLE> and </TITLE> tags which specify the text to display in the titlebar of the browser window
ULISTCLOSE Procedure on page 139-132	Generates the </UL> tag which marks the end of an unordered list
ULISTOPEN Procedure on page 139-133	Generates the <UL> tag which marks the beginning of an unordered list
UNDERLINE Procedure on page 139-134	Generates the <U> and </U> tags, which direct the browser to render the text they surround with an underline

Table 139–1 (Cont.) HTP Package Subprograms

Subprogram	Description
VARIABLE Procedure on page 139-135	Generates the <VAR> and </VAR> tags which direct the browser to render the text they surround in italics or however "variable" is defined stylistically.
WBR Procedure on page 139-136	Generates the <WBR> tag, which inserts a soft line break within a section of NOBR text

ADDRESS Procedure

This procedure generates the <ADDRESS> and </ADDRESS> tags which specify the address, author and signature of a document.

Syntax

```
HTP.ADDRESS (
   cvalue         IN       VARCHAR2
   cnowrap        IN       VARCHAR2   DEFAULT NULL
   cclear         IN       VARCHAR2   DEFAULT NULL
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–2 ADDRESS Procedure Parameters

Parameter	Description
cvalue	The string that goes between the <ADDRESS> and </ADDRESS> tags.
cnowrap	If the value for this parameter is not NULL, the NOWRAP attribute is included in the tag
cclear	The value for the CLEAR attribute.
cattributes	The other attributes to be included as-is in the tag

Examples

This procedure generates

```
<ADDRESS CLEAR="cclear" NOWRAP cattributes>cvalue</ADDRESS>
```

ANCHOR Procedure

This procedure and the ANCHOR2 Procedure procedures generate the <A> and </A> HTML tags which specify the source or destination of a hypertext link. The difference between these subprograms is that the ANCHOR2 Procedure provides a target and therefore can be used for a frame.

Syntax

```
HTP.ANCHOR (
   curl           IN       VARCHAR2,
   ctext          IN       VARCHAR2,
   cname          IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–3 ANCHOR Procedure Parameters

Parameter	Description
curl	The value for the HREF attribute.
ctext	The string that goes between the <A> and </A> tags.
cname	The value for the NAME attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<A HREF="curl" NAME="cname" cattributes>ctext</A>
```

Usage Notes

This tag accepts several attributes, but either HREF or NAME is required. HREF specifies to where to link. NAME allows this tag to be a target of a hypertext link.

ANCHOR2 Procedure

This procedure and the ANCHOR Procedure generate the <A> and </A> HTML tags which specify the source or destination of a hypertext link. The difference between these subprograms is that this procedures provides a target and therefore can be used for a frame.

Syntax

```
HTP.ANCHOR2 (
   curl           IN       VARCHAR2,
   ctext          IN       VARCHAR2,
   cname          IN       VARCHAR2   DEFAULT NULL,
   ctarget        in       varchar2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–4 ANCHOR2 Procedure Parameters

Parameter	Description
curl	The value for the HREF attribute.
ctext	The string that goes between the <A> and </A> tags.
cname	The value for the NAME attribute
ctarget	The value for the TARGET attribute.
cattributes	The other attributes to be included as-is in the tag

Examples

This procedure generates

```
<A HREF="curl" NAME="cname" TARGET = "ctarget" cattributes>ctext</A>
```

APPLETCLOSE Procedure

This procedure closes the applet invocation with the `</APPLET>` tag. You must first invoke the a Java applet using APPLETOPEN Procedure.

Syntax

```
HTP.APPLETCLOSE;
```

APPLETOPEN Procedure

This procedure generates the <APPLET> tag which begins the invocation of a Java applet. You close the applet invocation with APPLETCLOSE Procedure which generates the </APPLET> tag.

Syntax

```
HTP.APPLETOPEN (
   ccode          IN       VARCHAR2,
   cheight        IN       NUMBER,
   cwidth         IN       NUMBER,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–5 APPLETOPEN Procedure Parameters

Parameter	Description
ccode	The the value for the CODE attribute which specifies the name of the applet class.
cheight	The value for the HEIGHT attribute.
cwidth	The value for the WIDTH attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<APPLET CODE=ccode HEIGHT=cheight WIDTH=cwidth cattributes>
```

so that, for example,

```
HTP.appletopen('testclass.class', 100, 200, 'CODEBASE="/ows-applets"')
```

generates

```
<APPLET CODE="testclass.class" height=100 width=200 CODEBASE="/ows-applets">
```

Usage Notes

- Specify parameters to the Java applet using the PARAM Procedure procedure.
- Use the cattributes parameter to specify the CODEBASE attribute since the PL/SQL cartridge does not know where to find the class files. The CODEBASE attribute specifies the virtual path containing the class files.

AREA Procedure

This procedure generates the <AREA> tag, which defines a client-side image map. The <AREA> tag defines areas within the image and destinations for the areas.

Syntax

```
HTP.AREA (
   ccoords        IN       VARCHAR2
   cshape         IN       VARCHAR2   DEFAULT NULL,
   chref          IN       VARCHAR2   DEFAULT NULL,
   cnohref        IN       VARCHAR2   DEFAULT NULL,
   ctarget        IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–6 AREA Procedure Parameters

Parameter	Description
ccords	The the value for the COORDS attribute.
cshape	The value for the SHAPE attribute.
chref	The value for the HREF attribute.
cnohref	If the value for this parameter is not NULL, the NOHREF attribute is added to the tag.
ctarget	The value for the TARGET attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<AREA COORDS="ccoords" SHAPE="cshape" HREF="chref" NOHREF TARGET="ctarget"
cattributes>
```

BASE Procedure

This procedure generates the <BASE> tag which records the URL of the document.

Syntax

```
HTP.BASE (
   ctarget        IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–7 BASE Procedure Parameters

Parameter	Description
ctarget	The value for the TARGET attribute which establishes a window name to which all links in this document are targeted.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<BASE HREF="<current URL>" TARGET="ctarget" cattributes>
```

BASEFONT Procedure

This procedure generates the <BASEFONT> tag which specifies the base font size for a Web page.

Syntax

```
HTP.BASEFONT (
   nsize    IN    INTEGER);
```

Parameters

Table 139–8 BASEFONT Procedure Parameters

Parameter	Description
nsize	The value for the SIZE attribute.

Examples

This procedure generates

```
<BASEFONT SIZE="nsize">
```

BGSOUND Procedure

This procedure generates the <BGSOUND> tag which includes audio for a Web page.

Syntax

```
HTP.BGSOUND (
   csrc           IN       VARCHAR2,
   cloop          IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–9 BGSOUND Procedure Parameters

Parameter	Description
csrc	The value for the SRC attribute.
cloop	The value for the LOOP attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<BGSOUND SRC="csrc" LOOP="cloop" cattributes>
```

BIG Procedure

This procedure generates the <BIG> and </BIG> tags which direct the browser to render the text in a bigger font.

Syntax

```
HTP.BIG (
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–10 BIG Procedure Parameters

Parameter	Description
ctext	The the text that goes between the tags.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

<BIG *cattributes*>*ctext*</BIG>

BLOCKQUOTECLOSE Procedure

This procedure generates the </BLOCKQUOTE> tag which mark the end of a section of quoted text. You mark the beginning of a section of text by means of the BLOCKQUOTEOPEN Procedure.

Syntax

```
HTP.BLOCKQUOTECLOSE;
```

Examples

This procedure generates

```
</BLOCKQUOTE>
```

BLOCKQUOTEOPEN Procedure

This procedure generates the <BLOCKQUOTE> tag, which marks the beginning of a section of quoted text. You mark the end of a section of text by means of the BLOCKQUOTECLOSE Procedure.

Syntax

```
HTP.BLOCKQUOTEOPEN (
   cnowrap         IN       VARCHAR2   DEFAULT NULL,
   cclear          IN       VARCHAR2   DEFAULT NULL,
   cattributes     IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–11 BLOCKQUOTEOPEN Procedure Parameters

Parameter	Description
cnowrap	If the value for this parameter is not NULL, the NOWRAP attribute is added to the tag.
cclear	The value for the CLEAR attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<BLOCKQUOTE CLEAR="cclear" NOWRAP cattributes>
```

BODYCLOSE Procedure

This procedure generates the </BODY> tag which marks the end of a body section of an HTML document.You mark the beginning of a body section by means of the BODYOPEN Procedure.

Syntax

```
HTP.BODYCLOSE;
```

Examples

This procedure generates

```
</BODY>
```

BODYOPEN Procedure

This procedure generates the <BODY> tag which marks the beginning of the body section of an HTML document. You mark the end of a body section by means of the BODYCLOSE Procedure.

Syntax

```
HTP.BODYOPEN (
   cbackground    IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–12 BODYOPEN Procedure Parameters

Parameter	Description
cbackground	The value for the BACKGROUND attribute which specifies a graphic file to use for the background of the document.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<BODY background="cbackground" cattributes>
```

so that

```
HTP.BODYOPEN('/img/background.gif');
```

generates:

```
<BODY background="/img/background.gif">
```

BOLD Procedure

This procedure generates the <B> and </B> tags which direct the browser to display the text in boldface.

Syntax

```
HTP.BOLD (
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–13 BOLD Procedure Parameters

Parameter	Description
ctext	The text that goes between the tags.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<B cattributes>ctext</B>
```

BR Procedure

This procedure generates the <BR> tag which begins a new line of text. It performs the same operation as the NL Procedure.

Syntax

```
HTP.BR(
   cclear         IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–14 BR Procedure Parameters

Parameter	Description
cclear	The value for the CLEAR attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<BR CLEAR="cclear" cattributes>
```

CENTER Procedure

This procedure generates the <CENTER> and </CENTER> tags which center a section of text within a Web page.

Syntax

```
HTP.CENTER (
   ctext           IN       VARCHAR2);
```

Parameters

Table 139–15 CENTER Parameters

Parameter	Description
ctext	The text that goes between the tags.

Examples

This procedure generates

```
<CENTER>ctext</CENTER>
```

CENTERCLOSE Procedure

This procedure generates the </CENTER> tag which marks the end of a section of text to center. You mark the beginning of a of a section of text to center by means of the CENTEROPEN Procedure.

Syntax

```
HTP.CENTERCLOSE;
```

Examples

This procedure generates

```
</CENTER>
```

CENTEROPEN Procedure

This procedure generates the <CENTER> tag which mark the beginning of a section of text to center.You mark the beginning of a of a section of text to center by means of the CENTERCLOSE Procedure.

Syntax

```
HTP.CENTEROPEN;
```

Examples

This procedure generates

```
<CENTER>
```

CITE Procedure

This procedure generates the <CITE> and </CITE> tags which direct the browser to render the text as a citation.

Syntax

```
HTP.CITE (
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–16 CITE Procedure Parameters

Parameter	Description
ctext	The text to render as citation.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

<CITE *cattributes*>*ctext*</CITE>

CODE Procedure

This procedure generates the <CODE> and </CODE> tags which direct the browser to render the text in monospace font or however "code" is defined stylistically.

Syntax

```
HTP.CODE (
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–17 CODE Procedure Parameters

Parameter	Description
ctext	The text to render as code.
cattributes	The other attributes to be included as-is in the tag

Examples

This procedure generates

```
<CODE cattributes>ctext</CODE>
```

COMMENT Procedure

This procedure generates the comment tags.

Syntax

```
HTP.COMMENT (
   ctext           IN       VARCHAR2);
```

Parameters

Table 139–18 COMMENT Procedure Parameters

Parameter	Description
ctext	The comment.

Examples

This procedure generates

```
<!-- ctext -->
```

DFN Procedure

This procedure generates the <DFN> and </DFN> tags which direct the browser to mark the text in italics or however "definition" is described stylistically.

Syntax

```
HTP.DFN (
   ctext           IN       VARCHAR2);
```

Parameters

Table 139–19 DFN Procedure Parameters

Parameter	Description
ctext	The text to render in italics.

Examples

This procedure generates

```
<DFN>ctext</DFN>
```

DIRLISTCLOSE Procedure

This procedure generates the </DIR> tag which ends a directory list section. You start a directory list section with the DIRLISTOPEN Procedure.

Syntax

```
HTP.DIRLISTCLOSE;
```

Usage Notes

A directory list presents a list of items that contains up to 20 characters. Items in this list are typically arranged in columns, 24 characters wide. Insert the <LI> tag directly or invoke the LISTITEM Procedure so that the <LI> tag appears directly after the </DIR> tag to define the items as a list.

Examples

This procedure generates

```
</DIR>
```

DIRLISTOPEN Procedure

This procedure generates the <DIR> which starts a directory list section. You end a directory list section with the DIRLISTCLOSE Procedure.

Syntax

```
HTP.DIRLISTOPEN;
```

Usage Notes

A directory list presents a list of items that contains up to 20 characters. Items in this list are typically arranged in columns, 24 characters wide. Insert the <LI> tag directly or invoke the LISTITEM Procedure so that the <LI> tag appears directly after the </DIR> tag to define the items as a list.

Examples

This procedure generates

```
<DIR>
```

DIV Procedure

This procedure generates the <DIV> tag which creates document divisions.

Syntax

```
HTP.DIV (
   calign         IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–20 DIV Procedure Parameters

Parameter	Description
calign	The value for the ALIGN attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<DIV ALIGN="calign" cattributes>
```

DLISTCLOSE Procedure

This procedure generates the </DL> tag which ends a definition list. You start a definition list by means of the DLISTOPEN Procedure.

Syntax

```
HTP.DLISTCLOSE;
```

Usage Notes

A definition list looks like a glossary: it contains terms and definitions. Terms are inserted using the DLISTTERM Procedure and definitions are inserted using the DLISTDEF Procedure.

Examples

This procedure generates

```
</DL>
```

DLISTDEF Procedure

This procedure generates the <DD> tag, which inserts definitions of terms. Use this tag for a definition list <DL>. Terms are tagged <DT> and definitions are tagged <DD>.

Syntax

```
HTP.DLISTDEF (
   ctext          IN       VARCHAR2   DEFAULT NULL,
   cclear         IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–21 DLISTDEF Procedure Parameters

Parameter	Description
ctext	The definition of the term.
cclear	The value for the CLEAR attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<DD CLEAR="cclear" cattributes>ctext
```

DLISTOPEN Procedure

This procedure generates the <DL> tag which starts a definition list. You end a definition list by means of the DLISTCLOSE Procedure.

Syntax

```
HTP.DLISTOPEN (
   cclear         IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–22 DLISTOPEN Procedure Parameters

Parameter	Description
cclear	The value for the CLEAR attribute.
cattributes	The other attributes to be included as-is in the tag.

Usage Notes

A definition list looks like a glossary: it contains terms and definitions. Terms are inserted using the DLISTTERM Procedure and definitions are inserted using the DLISTDEF Procedure.

Examples

This procedure generates

```
<DL CLEAR="cclear" cattributes>
```

DLISTTERM Procedure

This procedure generates the <DT> tag which defines a term in a definition list <DL>.

Syntax

```
HTP.DLISTTERM (
   ctext          IN       VARCHAR2   DEFAULT NULL,
   cclear         IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–23 DLISTERM Procedure Parameters

Parameter	Description
ctext	The term.
cclear	The value for the CLEAR attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<DT CLEAR="cclear" cattributes>ctext
```

EM Procedure

This procedure generates the <EM> and </EM> tags, which define text to be emphasized. It performs the same task as the EMPHASIS Procedure.

Syntax

```
HTP.EM(
     ctext          IN       VARCHAR2,
     cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–24 EM Procedure Parameters

Parameter	Description
ctext	The text to emphasize.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<EM cattributes>ctext</EM>
```

EMPHASIS Procedure

This procedure generates the <EM> and </EM> tags, which define text to be emphasized. It performs the same task as the EM Procedure.

Syntax

```
HTP.EMPHASIS(
     ctext          IN       VARCHAR2,
     cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–25 EMPHASIS Procedure Parameters

Parameter	Description
ctext	The text to emphasize.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

<EM *cattributes*>*ctext*</EM>

ESCAPE_SC Procedure

This procedure replaces characters that have special meaning in HTML with their escape sequences. The following characters are converted:

- `&` to `&`
- `"` to `&quot:`
- `<` to `<`
- `>` to `>`

This procedure performs the same operation as PRINTS Procedures and PS Procedure.

Syntax

```
HTP.ESCAPE_SC(
     ctext          IN       VARCHAR2);
```

Parameters

Table 139–26 ESCAPE_SC Procedure Parameters

Parameter	Description
`ctext`	The text string to convert.

FONTCLOSE Procedure

This procedure generates the</FONT> tag which marks the end of a section of text with the specified font characteristics. You mark the beginning of the section text by means of the FONTOPEN Procedure.

Syntax

```
HTP.FONTCLOSE;
```

Examples

This procedure generates

```
</FONT>
```

FONTOPEN Procedure

This procedure generates the <FONT> which marks the beginning of section of text with the specified font characteristics. You mark the end of the section text by means of the FONTCLOSE Procedure.

Syntax

```
HTP.FONTOPEN(
   ccolor         IN       VARCHAR2   DEFAULT NULL,
   cface          IN       VARCHAR2   DEFAULT NULL,
   csize          IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–27 FONTOPEN Procedure Parameters

Parameter	Description
ccolor	The value for the COLOR attribute.
cface	The value for the FACE attribute
csize	The value for the SIZE attribute
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<FONT COLOR="ccolor" FACE="cface" SIZE="csize" cattributes>
```

FORMCHECKBOX Procedure

This procedure generates the <INPUT> tag with TYPE="checkbox" which inserts a checkbox element in a form. A checkbox element is a button that the user toggles on or off.

Syntax

```
HTP.FORMCHECKBOX(
   cname          IN       VARCHAR2,
   cvalue         IN       VARCHAR2   DEFAULT 'ON',
   cchecked       IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–28 FORMCHECKBOX Procedure Parameters

Parameter	Description
cname	The value for the NAME attribute.
cvalue	The value for the VALUE attribute.
cchecked	If the value for this parameter is not NULL, the CHECKED attribute is added to the tag.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<INPUT TYPE="checkbox" NAME="cname" VALUE="cvalue" CHECKED cattributes>
```

FORMCLOSE Procedure

This procedure generates the `</FORM>` tag which marks the end of a form section in an HTML document.You mark the beginning of the form section by means of the FORMOPEN Procedure.

Syntax

```
HTP.FORMCLOSE;
```

Examples

This procedure generates

```
</FORM>
```

FORMOPEN Procedure

This procedure generates the <FORM> tag which marks the beginning of a form section in an HTML document. You mark the end of the form section by means of the FORMCLOSE Procedure.

Syntax

```
HTP.FORMOPEN(
   curl           IN       VARCHAR2,
   cmethod        IN       VARCHAR2   DEFAULT 'POST',
   ctarget        IN       VARCHAR2   DEFAULT NULL,
   cenctype       IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–29 FORMOPEN Procedure Parameters

Parameter	Description
curl	The URL of the WRB or CGI script where the contents of the form is sent. This parameter is required.
cmethod	The value for the METHOD attribute. The value can be "GET" or "POST".
ctarget	The value for the TARGET attribute.
cenctype	The value for the ENCTYPE attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<FORM ACTION="curl" METHOD="cmethod" TARGET="ctarget" ENCTYPE="cenctype"
cattributes>
```

FORMFILE Procedure

This procedure generates the <INPUT> tag with TYPE="file" which inserts a file form element. This is used for file uploading for a given page.

Syntax

```
HTP.FORMFILE(
   cname          IN       VARCHAR2,
   caccept        IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–30 FORMFILE Procedure Parameters

Parameter	Description
cname	The value for the NAME attribute.
caccept	A comma-delimited list of MIME types for upload.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<INPUT TYPE="file" NAME="cname" ACCEPT="caccept" cattributes>
```

FORMHIDDEN Procedure

This procedure generates the <INPUT> tag with TYPE="hidden", which inserts a hidden form element. This element is not seen by the user. It submits additional values to the script.

Syntax

```
HTP.FORMHIDDEN(
   cname          IN       VARCHAR2,
   cvalue         IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–31 FORMHIDDEN Procedure Parameters

Parameter	Description
cname	The value for the NAME attribute.
cvalue	The value for the VALUE attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<INPUT TYPE="hidden" NAME="cname" VALUE="cvalue" cattributes>
```

FORMIMAGE Procedure

This procedure generates the <INPUT> tag with TYPE="image" which creates an image field that the user clicks to submit the form immediately. The coordinates of the selected point are measured in pixels, and returned (along with other contents of the form) in two name/value pairs. The x coordinate is submitted under the name of the field with .x appended, and the y coordinate with .y appended. Any VALUE attribute is ignored.

Syntax

```
HTP.FORMIMAGE(
   cname          IN       VARCHAR2,
   csrc           IN       VARCHAR2,
   calign         IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–32 FORMIMAGE Procedure Parameters

Parameter	Description
cname	The value for the NAME attribute.
csrc	The value for the SRC attribute that specifies the image file.
calign	The value for the ALIGN attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<INPUT TYPE="image" NAME="cname" SRC="csrc" ALIGN="calign" cattributes>
```

FORMPASSWORD Procedure

This procedure generates the <INPUT> tag with TYPE="password" which creates a single-line text entry field. When the user enters text in the field, each character is represented by one asterisk. This is used for entering passwords.

Syntax

```
HTP.FORMPASSWORD(
   cname          IN       VARCHAR2,
   csize          IN       VARCHAR2,
   cmaxlength     IN       VARCHAR2   DEFAULT NULL,
   cvalue         IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–33 FORMPASSWORD Procedure Parameters

Parameter	Description
cname	The value for the NAME attribute.
csize	The value for the SIZE attribute.
cmaxlength	The value for the MAXLENGTH attribute.
cvalue	The value for the VALUE attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<INPUT TYPE="password" NAME="cname" SIZE="csize" MAXLENGTH="cmaxlength"
VALUE="cvalue" cattributes>
```

FORMRADIO Procedure

This procedure generates the <INPUT> tag with TYPE="radio", which creates a radio button on the HTML form. Within a set of radio buttons, the user selects only one. Each radio button in the same set has the same name, but different values. The selected radio button generates a name/value pair.

Syntax

```
HTP.FORMRADIO(
   cname          IN       VARCHAR2,
   cvalue         IN       VARCHAR2,
   cchecked       IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–34 FORMRADIO Procedure Parameters

Parameter	Description
cname	The value for the NAME attribute.
cvalue	The value for the VALUE attribute.
cchecked	If the value for this parameter is not NULL, the CHECKED attribute is added to the tag.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<INPUT TYPE="radio" NAME="cname" VALUE="cvalue" CHECKED cattributes>
```

FORMRESET Procedure

This procedure generates the <INPUT> tag with TYPE="reset" which creates a button that, when selected, resets the form fields to their initial values.

Syntax

```
HTP.FORMRESET(
   cvalue         IN       VARCHAR2   DEFAULT 'Reset',
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–35 FORMRESET Procedure Parameters

Parameter	Description
cvalue	The value for the VALUE attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

<INPUT TYPE="*reset*" VALUE="*cvalue*" *cattributes*>

FORMSELECTCLOSE Procedure

This procedure generates the </SELECT> tag which marks the end of a Select form element. A Select form element is a listbox where the user selects one or more values. You mark the beginning of Select form element by means of the FORMSELECTOPEN Procedure.The values are inserted using FORMSELECTOPTION Procedure.

Syntax

```
HTP.FORMSELECTCLOSE;
```

Examples

This procedure generates

```
</SELECT>
```

as shown under Examples of the FORMSELECTOPEN Procedure.

FORMSELECTOPEN Procedure

This procedure generates the <SELECT> tags which creates a Select form element. A Select form element is a listbox where the user selects one or more values. You mark the end of Select form element by means of the FORMSELECTCLOSE Procedure.The values are inserted using FORMSELECTOPTION Procedure.

Syntax

```
FORMSELECTOPEN(
   cname          IN       VARCHAR2,
   cprompt        IN       VARCHAR2   DEFAULT NULL,
   nsize          IN       INTEGER    DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–36 FORMSELECTOPEN Procedure Parameters

Parameter	Description
cname	The value for the NAME attribute.
cprompt	The string preceding the list box.
nsize	The value for the SIZE attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
cprompt <SELECT NAME="cname" SIZE="nsize" cattributes>
</SELECT>
```

so that

```
HTP.FORMSELECTOPEN('greatest_player';
    'Pick the greatest player:');
HTP.FORMSELECTOPTION('Messier');
HTP.FORMSELECTOPTION('Howe');
HTP.FORMSELECTOPTION('Gretzky');.
HTP.FORMSELECTCLOSE;
```

generates

```
Pick the greatest player:
<SELECT NAME="greatest_player">
<OPTION>Messier
<OPTION>Howe
<OPTION>Gretzky
</SELECT>
```

FORMSELECTOPTION Procedure

This procedure generates the <OPTION> tag which represents one choice in a Select element.

Syntax

```
HTP.FORMSELECTOPTION(
   cvalue         IN       VARCHAR2,
   cselected      IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–37 FORMSELECTOPTION Procedure Parameters

Parameter	Description
cvalue	The text for the option.
cvalue	If the value for this parameter is not NULL, the SELECTED attribute is added to the tag.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<OPTION SELECTED cattributes>cvalue
```

as shown under Examples of the FORMSELECTOPEN Procedure.

FORMSUBMIT Procedure

This procedure generates the <INPUT> tag with TYPE="submit" which creates a button that, when clicked, submits the form. If the button has a NAME attribute, the button contributes a name/value pair to the submitted data.

Syntax

```
HTP.FORMSUBMIT(
   cname          IN       VARCHAR2   DEFAULT NULL,
   cvalue        IN       VARCHAR2   DEFAULT 'Submit',
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–38 FORMSUBMIT Procedure Parameters

Parameter	Description
cname	The value for the NAME attribute.
cvalue	The value for the VALUE attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<INPUT TYPE="submit" NAME="cname" VALUE="cvalue" cattributes>
```

FORMTEXT Procedure

This procedure generates the <INPUT> tag with TYPE="text", which creates a field for a single line of text.

Syntax

```
HTP.FORMTEXT(
   cname          IN       VARCHAR2,
   csize          IN       VARCHAR2   DEFAULT NULL,
   cmaxlength     IN       VARCHAR2   DEFAULT NULL,
   cvalue         IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–39 FORMTEXT Procedure Parameters

Parameter	Description
cname	The value for the NAME attribute.
csize	The value for the SIZE attribute.
cmaxlength	The value for the MAXLENGTH attribute.
cvalue	The value for the VALUE attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<INPUT TYPE="text" NAME="cname" SIZE="csize" MAXLENGTH="cmaxlength" VALUE="cvalue"
cattributes>
```

FORMTEXTAREA Procedure

This procedure generates the <TEXTAREA> tag, which creates a text field that has no predefined text in the text area. This field enables entering several lines of text. The same operation is performed by the FORMTEXTAREA2 Procedure which in addition has the cwrap parameter that lets you specify a wrap style.

Syntax

```
HTP.FORMTEXTAREA(
   cname          IN       VARCHAR2,
   nrows          IN       INTEGER,
   ncolumns       IN       INTEGER,
   calign      , IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–40 FORMTEXTAREA Procedure Parameters

Parameter	Description
cname	The value for the NAME attribute.
nrows	The value for the ROWS attribute.This is an integer.
ncolumns	The value for the COLS attribute.This is an integer.
calign	The value for the ALIGN attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolumns" ALIGN="calign"
cattributes></TEXTAREA>
```

FORMTEXTAREA2 Procedure

This procedure generates the <TEXTAREA> tag, which creates a text field that has no predefined text in the text area. This field enables entering several lines of text.The same operation is performed by the FORMTEXTAREA Procedure except that in that case you cannot specify a wrap style.

Syntax

```
HTP.FORMTEXTAREA2(
   cname          IN       VARCHAR2,
   nrows          IN       INTEGER,
   ncolumns       IN       INTEGER,
   calign         IN       VARCHAR2   DEFAULT NULL,
   cwrap          IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–41 FORMTEXTAREA2 Procedure Parameters

Parameter	Description
cname	The value for the NAME attribute.
nrows	The value for the ROWS attribute.This is an integer.
ncolumns	The value for the COLS attribute.This is an integer.
calign	The value for the ALIGN attribute.
cwrap	The value for the WRAP attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolumns" ALIGN="calign" WRAP="cwrap"
cattributes></TEXTAREA>
```

FORMTEXTAREACLOSE Procedure

This procedure generates the `</TEXTAREA>` tag which ends a text area form element. You open a text area element by means of either FORMTEXTAREAOPEN Procedure or FORMTEXTAREAOPEN2 Procedure.

Syntax

```
HTP.FORMTEXTAREACLOSE;
```

Examples

This procedure generates

```
</TEXTAREA>
```

FORMTEXTAREAOPEN Procedure

This procedure generates the <TEXTAREA> which marks the beginning of a text area form element. The same operation is performed by the FORMTEXTAREAOPEN2 Procedure which in addition has the cwrap parameter that lets you specify a wrap style. You mark the end of a text area form element by means of the FORMTEXTAREACLOSE Procedure.

Syntax

```
HTP.FORMTEXTAREAOPEN(
   cname          IN       VARCHAR2,
   nrows          IN       INTEGER,
   ncolumns       IN       INTEGER,
   calign         IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–42 FORMTEXTAREAOPEN Procedure Parameters

Parameter	Description
cname	The value for the NAME attribute.
nrows	The value for the ROWS attribute.This is an integer.
ncolumns	The value for the COLS attribute.This is an integer.
calign	The value for the ALIGN attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolumns" ALIGN="calign" cattributes>
```

FORMTEXTAREAOPEN2 Procedure

This procedure generates the <TEXTAREA> which marks the beginning of a text area form element. The same operation is performed by the FORMTEXTAREAOPEN Procedure except that in that case you cannot specify a wrap style. You mark the end of a text area form element by means of the FORMTEXTAREACLOSE Procedure.

Syntax

```
HTP.FORMTEXTAREAOPEN2(
   cname          IN       VARCHAR2,
   nrows          IN       INTEGER,
   ncolumns       IN       INTEGER,
   calign         IN       VARCHAR2   DEFAULT NULL,
   cwrap          IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–43 FORMTEXTAREAOPEN2 Procedure Parameters

Parameter	Description
cname	The value for the NAME attribute.
nrows	The value for the ROWS attribute.This is an integer.
ncolumns	The value for the COLS attribute.This is an integer.
calign	The value for the ALIGN attribute.
cwrap	The value for the WRAP attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolumns" ALIGN="calign" WRAP = "cwrap"
cattributes>
```

FRAME Procedure

This procedure generates the <FRAME> tag which defines the characteristics of a frame created by a <FRAMESET> tag.

Syntax

```
HTP.FRAME(
   csrc           IN       VARCHAR2,
   cname          IN       VARCHAR2   DEFAULT NULL,
   cmarginwidth   IN       VARCHAR2   DEFAULT NULL,
   cmarginheight  IN       VARCHAR2   DEFAULT NULL,
   cscrolling     IN       VARCHAR2   DEFAULT NULL,
   cnoresize      IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–44 FRAME Procedure Parameters

Parameter	Description
csrc	The URL to display in the frame.
cname	The value for the NAME attribute.
cmarginwidth	The value for the MARGINWIDTH attribute.
cscrolling	The value for the SCROLLING attribute.
cnoresize	If the value for this parameter is not NULL, the NORESIZE attribute is added to the tag.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<FRAME SRC="csrc" NAME="cname" MARGINWIDTH="cmarginwidth"
MARGINHEIGHT="cmarginheight" SCROLLING="cscrolling" NORESIZE cattributes>
```

FRAMESETCLOSE Procedure

This procedure generates the `</FRAMESET>` tag which ends a frameset section. You mark the beginning of a frameset section by means of the FRAMESETOPEN Procedure.

Syntax

```
HTP.FRAMESETCLOSE;
```

Examples

This procedure generates

```
</FRAMESET>
```

FRAMESETOPEN Procedure

This procedure generates the <FRAMESET> tag which define a frameset section. You mark the end of a frameset section by means of the FRAMESETCLOSE Procedure.

Syntax

```
HTP.FRAMESETOPEN(
   crows          IN       VARCHAR2   DEFAULT NULL,
   ccols          IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–45 FRAMESETOPEN Procedure Parameters

Parameter	Description
crows	The value for the ROWS attribute.
ccols	The value for the COLS attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<FRAMESET ROWS="crows" COLS="ccols" cattributes>
```

HEADCLOSE Procedure

This procedure generates the </HEAD> tag which marks the end of an HTML document head section. You mark the beginning of an HTML document head section by means of the HEADOPEN Procedure.

Syntax

```
HTP.HEADCLOSE;
```

Examples

This procedure generates

```
</HEAD>
```

HEADER Procedure

This procedure generates opening heading tags (<H1> to <H6>) and their corresponding closing tags (</H1> to </H6>).

Syntax

```
HTP.HEADER(
   nsize          IN       INTEGER,
   cheader        IN       VARCHAR2,
   calign         IN       VARCHAR2   DEFAULT NULL,
   cnowrap        IN       VARCHAR2   DEFAULT NULL,
   cclear         IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–46 HEADER Procedure Parameters

Parameter	Description
nsize	The the heading level. This is an integer between 1 and 6.
cheader	The text to display in the heading.
calign	The value for the ALIGN attribute.
cnowrap	The value for the NOWRAP attribute.
cclear	The value for the CLEAR attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

```
HTP.header (1,'Overview');
```

produces:

```
<H1>Overview</H1>
```

HEADOPEN Procedure

This procedure generates the <HEAD> tag which marks the beginning of the HTML document head section. You mark the end of an HTML document head section by means of the HEADCLOSE Procedure.

Syntax

```
HTP.HEADOPEN;
```

Examples

This procedure generates

```
<HEAD>
```

HR Procedure

This procedure generates the <HR> tag, which generates a line in the HTML document.This subprogram performs the same operation as the LINE Procedure.

Syntax

```
HTP.HR(
   cclear          IN       VARCHAR2   DEFAULT NULL,
   csrc            IN       VARCHAR2   DEFAULT NULL,
   cattributes     IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–47 HR Procedure Parameters

Parameter	Description
cclear	The value for the CLEAR attribute.
csrc	The value for the SRC attribute which specifies a custom image as the source of the line.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<HR CLEAR="cclear" SRC="csrc" cattributes>
```

HTMLCLOSE Procedure

This procedure generates the `</HTML>` tag which marks the end of an HTML document. You use the HTMLOPEN Procedure to mark the beginning of an HTML document.

Syntax

```
HTP.HTMLCLOSE;
```

Examples

This procedure generates

```
</HTML>
```

HTMLOPEN Procedure

This procedure generates the <HTML> tag which marks the beginning of an HTML document. You use the HTMLCLOSE Procedure to mark the end of the an HTML document.

Syntax

```
HTP.HTMLOPEN;
```

Examples

This procedure generates

```
<HTML>
```

IMG Procedure

This procedure generates the <IMG> tag which directs the browser to load an image onto the HTML page. The IMG2 Procedure performs the same operation but additionally uses the cusemap parameter.

Syntax

```
HTP.IMG(
   curl           IN       VARCHAR2   DEFAULT NULL,
   calign         IN       VARCHAR2   DEFAULT NULL,
   calt           IN       VARCHAR2   DEFAULT NULL,
   cismap         IN       VARCHAR2
DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–48 IMG Procedure Parameters

Parameter	Description
curl	The value for the SRC attribute.
calign	The value for the ALIGN attribute.
calt	The value for the ALT attribute which specifies alternative text to display if the browser does not support images.
cismap	If the value for this parameter is not NULL, the ISMAP attribute is added to the tag. The attribute indicates that the image is an imagemap.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<IMG SRC="curl" ALIGN="calign" ALT="calt" ISMAP cattributes>
```

IMG2 Procedure

This procedure generates the <IMG> tag, which directs the browser to load an image onto the HTML page. The IMG Procedure performs the same operation but does not use the `cusemap` parameter.

Syntax

```
HTP.IMG2(
   curl           IN       VARCHAR2   DEFAULT NULL,
   calign         IN       VARCHAR2   DEFAULT NULL,
   calt           IN       VARCHAR2   DEFAULT NULL,
   cismap         IN       VARCHAR2
DEFAULT NULL,
   cusemap        IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–49 IMG2 Procedure Parameters

Parameter	Description
`curl`	The value for the SRC attribute.
`calign`	The value for the `ALIGN` attribute.
`calt`	The value for the `ALT` attribute which specifies alternative text to display if the browser does not support images.
`cismap`	If the value for this parameter is not `NULL`, the `ISMAP` attribute is added to the tag. The attribute indicates that the image is an imagemap.
`cusemap`	The value for the `USEMAP` attribute which specifies a client-side image map.
`cattributes`	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<IMG SRC="curl" ALIGN="calign" ALT="calt" ISMAP USEMAP="cusemap" cattributes>
```

ISINDEX Procedure

This procedure creates a single entry field with a prompting text, such as "*enter value,*" then sends that value to the URL of the page or program.

Syntax

```
HTP.ISINDEX(
   cprompt        IN       VARCHAR2    DEFAULT NULL,
   curl           IN       VARCHAR2    DEFAULT NULL);
```

Parameters

Table 139–50 ISINDEX Procedure Parameters

Parameter	Description
`cprompt`	The value for the `PROMPT` attribute.
`curl`	The value for the `HREF` attribute.

Examples

This procedure generates

```
<ISINDEX PROMPT="cprompt" HREF="curl">
```

ITALIC Procedure

This procedure generates the <I> and </I> tags which direct the browser to render the text in italics.

Syntax

```
HTP.ITALIC(
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–51 ITALIC Procedure Parameters

Parameter	Description
ctext	The text to be rendered in italics.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<I cattributes>ctext</I>
```

KBD Procedure

This procedure generates the <KBD> and </KBD> tags which direct the browser to render the text in monospace font. This subprogram performs the same operation as the KEYBOARD Procedure.

Syntax

```
HTP.KBD(
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–52 KBD Procedure Parameters

Parameter	Description
ctext	The text to be rendered in monospace.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<KBD cattributes>ctext</KBD>
```

KEYBOARD Procedure

This procedure generates the <KBD> and </KBD> tags, which direct the browser to render the text in monospace font. This subprogram performs the same operation as the KBD Procedure.

Syntax

```
HTP.KEYBOARD(
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–53 KEYBOARD Procedure Parameters

Parameter	Description
`ctext`	The text to be rendered in monospace.
`cattributes`	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<KBD cattributes>ctext</KBD>
```

LINE Procedure

This procedure generates the <HR> tag, which generates a line in the HTML document. This subprogram performs the same operation as the HR Procedure.

Syntax

```
HTP.LINE(
   cclear          IN       VARCHAR2   DEFAULT NULL,
   csrc            IN       VARCHAR2   DEFAULT NULL,
   cattributes     IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–54 LINE Procedure Parameters

Parameter	Description
cclear	The value for the CLEAR attribute.
csrc	The value for the SRC attribute which specifies a custom image as the source of the line.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<HR CLEAR="cclear" SRC="csrc" cattributes>
```

LINKREL Procedure

This procedure generates the <LINK> tag with the REL attribute which delineates the relationship described by the hypertext link from the anchor to the target. This is only used when the HREF attribute is present. This is the opposite of LINKREV Procedure. This tag indicates a relationship between documents but does not create a link. To create a link, use the ANCHOR Procedure.

Syntax

```
HTP.LINKREL(
   crel           IN       VARCHAR2,
   curl           IN       VARCHAR2,
   ctitle         IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–55 LINKREL Procedure Parameters

Parameter	Description
crel	The value for the REL attribute.
curl	The value for the URL attribute.
ctitle	The value for the TITLE attribute.

Examples

This procedure generates

```
<LINK REL="crel" HREF="curl" TITLE="ctitle">
```

LINKREV Procedure

This procedure generates the <LINK> tag with the REV attribute which delineates the relationship described by the hypertext link from the target to the anchor. This is the opposite of the LINKREL Procedure. This tag indicates a relationship between documents, but does not create a link. To create a link, use the ANCHOR Procedure.

Syntax

```
HTP.LINKREV(
   crev           IN       VARCHAR2,
   curl           IN       VARCHAR2,
   ctitle         IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–56 LINKREV Procedure Parameters

Parameter	Description
crel	The value for the REV attribute.
curl	The value for the URL attribute.
ctitle	The value for the TITLE attribute.

Examples

This procedure generates

```
<LINK REV="crev" HREF="curl" TITLE="ctitle">
```

LISTHEADER Procedure

This procedure generates the <LH> and </LH> tags which print an HTML tag at the beginning of the list.

Syntax

```
HTP.LISTHEADER(
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–57 LISTHEADER Procedure Parameters

Parameter	Description
ctext	The text to place between <LH> and </LH>.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<LH cattributes>ctext</LH>
```

LISTINGCLOSE Procedure

This procedure generates the `</LISTING>` tags which marks the end of a section of fixed-width text in the body of an HTML page. To mark the beginning of a section of fixed-width text in the body of an HTML page, use the LISTINGOPEN Procedure.

Syntax

```
HTP.LISTINGCLOSE;
```

Examples

This procedure generates

```
</LISTING>
```

LISTINGOPEN Procedure

This procedure generates the <LISTING> tag which marks the beginning of a section of fixed-width text in the body of an HTML page. To mark the end of a section of fixed-width text in the body of an HTML page, use the LISTINGCLOSE Procedure.

Syntax

```
HTP.LISTINGOPEN;
```

Examples

This procedure generates

```
<LISTING>
```

LISTITEM Procedure

This procedure generates the <LI> tag, which indicates a list item.

Syntax

```
HTP.LISTITEM(
   ctext          IN       VARCHAR2   DEFAULT NULL,
   cclear         IN       VARCHAR2   DEFAULT NULL,
   cdingbat       IN       VARCHAR2   DEFAULT NULL,
   csrc           IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–58 LISTITEM Procedure Parameters

Parameter	Description
ctext	The text for the list item.
cclear	The value for the CLEAR attribute.
cdingbat	The value for the DINGBAT attribute.
csrc	The value for the SRC attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<LI CLEAR="cclear" DINGBAT="cdingbat" SRC="csrc" cattributes>ctext
```

MAILTO Procedure

This procedure generates the <A> tag with the HREF set to 'mailto' prepended to the mail address argument.

Syntax

```
HTP.MAILTO(
   caddress       IN       VARCHAR2,
   ctext          IN       VARCHAR2,
   cname          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–59 MAILTO Procedure Parameters

Parameter	Description
caddress	The email address of the recipient.
ctext	The clickable portion of the link.
cname	The value for the NAME attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<A HREF="mailto:caddress" NAME="cname" cattributes>ctext</A>
```

so that

```
HTP.mailto('pres@white_house.gov','Send Email to the President');
```

generates:

```
<A HREF="mailto:pres@white_house.gov">Send Email to the President</A>
```

MAPCLOSE Procedure

This procedure generates the </MAP> tag which marks the end of a set of regions in a client-side image map. To mark the beginning of a set of regions in a client-side image map, use the MAPOPEN Procedure.

Syntax

```
HTP.MAPCLOSE;
```

Examples

This procedure generates

```
</MAP>
```

MAPOPEN Procedure

This procedure generates the <MAP> tag which mark the beginning of a set of regions in a client-side image map. To mark the end of a set of regions in a client-side image map, use the MAPCLOSE Procedure.

Syntax

```
HTP.MAPOPEN(
   cname          IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–60 MAPOPEN Procedure Parameters

Parameter	Description
cname	The value for the NAME attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<MAP NAME="cname" cattributes>
```

MENULISTCLOSE Procedure

This procedure generates the </MENU> tag which ends a list that presents one line for each item. To begin a list of this kind, use the MENULISTOPEN Procedure. The items in the list appear more compact than an unordered list. The LISTITEM Procedure defines the list items in a menu list.

Syntax

```
HTP.MENULISTCLOSE;
```

Examples

This procedure generates

```
</MENU>
```

MENULISTOPEN Procedure

This procedure generates the <MENU> tag which create a list that presents one line for each item. To end a list of this kind, use the MENULISTCLOSE Procedure.The items in the list appear more compact than an unordered list. The LISTITEM Procedure defines the list items in a menu list.

Syntax

```
HTP.MENULISTOPEN;
```

Examples

This procedure generates

```
<MENU>
```

META Procedure

This procedure generates the <META> tag, which embeds meta-information about the document and also specifies values for HTTP headers. For example, you can specify the expiration date, keywords, and author name.

Syntax

```
HTP.META(
   chttp_equiv    IN       VARCHAR2,
   cname          IN       VARCHAR2,
   ccontent       IN       VARCHAR2);
```

Parameters

Table 139–61 META Procedure Parameters

Parameter	Description
chttp_equiv	The value for the CHTTP_EQUIV attribute.
cname	The value for the NAME attribute.
ccontent	The value for the CONTENT attribute.

Examples

This procedure generates

```
<META HTTP-EQUIV="chttp_equiv" NAME ="cname" CONTENT="ccontent">
```

so that

```
HTP.meta ('Refresh', NULL, 120);
```

generates

```
<META HTTP-EQUIV="Refresh" CONTENT=120>
```

On some Web browsers, this causes the current URL to be reloaded automatically every 120 seconds.

NL Procedure

This procedure generates the <BR> tag which begins a new line of text. It performs the same operation as the BR Procedure.

Syntax

```
HTP.NL(
   cclear          IN       VARCHAR2   DEFAULT NULL,
   cattributes     IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–62 NL Procedure Parameters

Parameter	Description
cclear	The value for the CLEAR attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<BR CLEAR="cclear" cattributes>
```

NOBR Procedure

This procedure generates the <NOBR> and </NOBR> tags which turn off line-breaking in a section of text.

Syntax

```
HTP.NOBR(
ctext        IN        VARCHAR2);
```

Parameters

Table 139–63 NOBR Procedure Parameters

Parameter	Description
ctext	The text that is to be rendered on one line.

Examples

This procedure generates

```
<NOBR>ctext</NOBR>
```

NOFRAMESCLOSE Procedure

This procedure generates the </NOFRAMES> tag which marks the end of a no-frames section. To mark the beginning of a no-frames section, use the FRAMESETOPEN Procedure. See also FRAME Procedure, FRAMESETOPEN Procedure and FRAMESETCLOSE Procedure.

Syntax

```
HTP.NOFRAMESCLOSE;
```

Examples

This procedure generates

```
</NOFRAMES>
```

NOFRAMESOPEN Procedure

This procedure generates the <NOFRAMES> tag which mark the beginning of a no-frames section. To mark the end of a no-frames section, use the FRAMESETCLOSE Procedure. See also FRAME Procedure, FRAMESETOPEN Procedure and FRAMESETCLOSE Procedure.

Syntax

```
HTP.NOFRAMESOPEN;
```

Examples

This procedure generates

```
<NOFRAMES>
```

OLISTCLOSE Procedure

This procedure generates the </OL> tag which defines the end of an ordered list. An ordered list presents a list of numbered items. To mark the beginning of a list of this kind, use the OLISTOPEN Procedure. Numbered items are added using LISTITEM Procedure.

Syntax

```
HTP.OLISTCLOSE;
```

Examples

This procedure generates

```
</OL>
```

OLISTOPEN Procedure

This procedure generates the <OL> tag which marks the beginning of an ordered list. An ordered list presents a list of numbered items. To mark the end of a list of this kind, use the OLISTCLOSE Procedure. Numbered items are added using LISTITEM Procedure.

Syntax

```
HTP.OLISTOPEN(
   cclear          IN       VARCHAR2   DEFAULT NULL,
   cwrap           IN       VARCHAR2   DEFAULT NULL,
   cattributes     IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–64 OLISTOPEN Procedure Parameters

Parameter	Description
cclear	The value for the CLEAR attribute.
cwrap	The value for the WRAP attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<OL CLEAR="cclear" WRAP="cwrap" cattributes>
```

PARA Procedure

This procedure generates the <P> tag which indicates that the text that comes after the tag is to be formatted as a paragraph. You can add attributes to the tag by means of the PARAGRAPH Procedure.

Syntax

```
HTP.PARA;
```

Examples

This procedure generates

```
<P>
```

PARAGRAPH Procedure

You can use this procedure to add attributes to the <P> tag created by the PARA Procedure.

Syntax

```
HTP.PARAGRAPH(
   calign         IN       VARCHAR2   DEFAULT NULL,
   cnowrap        IN       VARCHAR2   DEFAULT NULL,
   cclear         IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–65 PARAGRAPH Procedure Parameters

Parameter	Description
calign	The value for the ALIGN attribute.
cnowrap	If the value for this parameter is not NULL, the NOWRAP attribute is added to the tag.
cclear	The value for the CLEAR attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<P ALIGN="calign" NOWRAP CLEAR="cclear" cattributes>
```

PARAM Procedure

This procedure generates the <PARAM> tag which specifies parameter values for Java applets. The values can reference HTML variables. To invoke a Java applet from a Web page, use APPLETOPEN Procedure to begin the invocation. Use one PARAM Procedure for each desired name-value pair, and use APPLETCLOSE Procedure to end the applet invocation.

Syntax

```
HTP.PARAM(
   cname          IN       VARCHAR2
   cvalue         IN       VARCHAR2);
```

Parameters

Table 139–66 PARAM Procedure Parameters

Parameter	Description
cname	The value for the NAME attribute.
cvalue	The value for the VALUE attribute.

Examples

This procedure generates

```
<PARAM NAME=cname VALUE="cvalue">
```

PLAINTEXT Procedure

This procedure generates the <PLAINTEXT> and </PLAINTEXT> tags which direct the browser to render the text they surround in fixed-width type.

Syntax

```
HTP.PLAINTEXT(
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–67 PLAINTEXT Procedure Parameters

Parameter	Description
ctext	The text to be rendered in fixed-width font.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<PLAINTEXT cattributes>ctext</PLAINTEXT>
```

PRECLOSE Procedure

This procedure generates the </PRE> tag which marks the end of a section of preformatted text in the body of the HTML page. To mark the beginning of a section of preformatted text in the body of the HTML page, use the PREOPEN Procedure.

Syntax

```
HTP.PRECLOSE;
```

Examples

This procedure generates

```
</PRE>
```

PREOPEN Procedure

This procedure generates the <PRE> tag which marks the beginning of a section of preformatted text in the body of the HTML page. To mark the end of a section of preformatted text in the body of the HTML page, use the PRECLOSE Procedure.

Syntax

```
HTP.PREOPEN(
   cclear          IN       VARCHAR2   DEFAULT NULL,
   cwidth          IN       VARCHAR2   DEFAULT NULL,
   cattributes     IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–68 PREOPEN Procedure Parameters

Parameter	Description
cclear	The value for the CLEAR attribute.
cwidth	The value for the WIDTH attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<PRE CLEAR="cclear" WIDTH="cwidth" cattributes>
```

PRINT Procedures

These procedures generate the specified parameter as a string terminated with the \n newline character. The PRN Procedures performs the same operation but does not terminate with a newline character.

Syntax

```
HTP.PRINT (
   cbuf      IN       VARCHAR2);

HTP.PRINT (
   dbuf      IN       DATE);

HTP.PRINT (
   nbuf      IN       NUMBER);
```

Parameters

Table 139–69 PRINT Procedure Parameters

Parameter	Description
cbuf	The string to generate terminated by a newline.
dbuf	The string to generate terminated by a newline.
nbuf	The string to generate terminated by a newline.

Usage Notes

- The \n character is not the same as <BR>. The \n character formats the HTML source but it does not affect how the browser renders the HTML source. Use <BR> to control how the browser renders the HTML source.
- These procedures do not have function equivalents.

PRINTS Procedure

This procedure generates a string and replaces the following characters with the corresponding escape sequence.

- `<` to `<`
- `>` to `>`
- `"` to `&quot;`
- `&` to `&`

If not replaced, the special characters are interpreted as HTML control characters and produce garbled output. This procedure an the PS Procedure perform the same operation as the PRN Procedures but with character substitution.

Syntax

```
HTP.PRINTS (
   ctext      IN       VARCHAR2);
```

Parameters

Table 139–70 PRINTS Procedure Parameters

Parameter	Description
`ctext`	The string where to perform character substitution.

Usage Notes

This procedure does not have an HTF function equivalent (see Operational Notes on page 139-3 for the HTF implementation).

PRN Procedures

These procedures generate the specified parameter as a string. Unlike the PRINT Procedures the string is not terminated with the `\n` newline character.

Syntax

```
HTP.PRN (
   cbuf       IN       VARCHAR2);

HTP.PRN (
   dbuf       IN       DATE);

HTP.PRN (
   nbuf       IN       NUMBER);
```

Parameters

Table 139–71 PRN Procedure Parameters

Parameter	Description
`cbuf`	The string to generate (not terminated by a newline).
`dbuf`	The string to generate (not terminated by a newline).
`nbuf`	The string to generate (not terminated by a newline).

Usage Notes

These procedures do not have function equivalents.

PS Procedure

This procedure generates a string and replaces the following characters with the corresponding escape sequence.

- < to <
- > to >
- " to "
- & to &

If not replaced, the special characters are interpreted as HTML control characters and produce garbled output. This procedure and the PRINTS Procedure perform the same operation as the PRN Procedures but with character substitution.

Syntax

```
HTP.PS (
   ctext      IN       VARCHAR2);
```

Parameters

Table 139–72 PS Procedure Parameters

Parameter	Description
ctext	The string where to perform character substitution.

Usage Notes

This procedure does not have an HTF function equivalent (see Operational Notes on page 139-3 for the HTF implementation).

S Procedure

This procedure generates the <S> and </S> tags which direct the browser to render the text they surround in strikethrough type. This performs the same operation as STRIKE Procedure.

Syntax

```
HTP.S (
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–73 S Procedure Parameters

Parameter	Description
ctext	The text to be rendered in strikethrough type.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<S cattributes>ctext</S>
```

SAMPLE Procedure

This procedure generates the <SAMP> and </SAMP> tags which direct the browser to render the text they surround in monospace font or however "sample" is defined stylistically.

Syntax

```
HTP.SAMPLE (
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–74 SAMPLE Procedure Parameters

Parameter	Description
ctext	The text to be rendered in monospace font.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<SAMP cattributes>ctext</SAMP>
```

SCRIPT Procedure

This procedure generates the <SCRIPT> and </SCRIPT> tags which contain a script written in languages such as JavaScript and VBscript.

Syntax

```
HTP.SCRIPT (
   cscript        IN       VARCHAR2,
   clanguage      IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–75 SCRIPT Procedure Parameters

Parameter	Description
cscript	The text of the script. This is the text that makes up the script itself, not the name of a file containing the script.
clanguage	The language in which the script is written. If this parameter is omitted, the user's browser determines the scripting language.

Examples

This procedure generates

```
<SCRIPT LANGUAGE=clanguage>cscript</SCRIPT>
```

so that

```
HTP.script ('Erupting_Volcano', 'Javascript');
```

generates

```
<SCRIPT LANGUAGE=Javascript>"script text here"</SCRIPT>
```

This causes the browser to run the script enclosed in the tags.

SMALL Procedure

This procedure generates the <SMALL> and </SMALL> tags, which direct the browser to render the text they surround using a small font.

Syntax

```
HTP.SMALL (
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–76 SMALL Procedure Parameters

Parameter	Description
ctext	The text to be rendered in small font.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<SMALL cattributes>ctext</SMALL>
```

STRIKE Procedure

This procedure generates the <STRIKE> and </STRIKE> tags which direct the browser to render the text they surround in strikethrough type. This performs the same operation as S Procedure.

Syntax

```
HTP.STRIKE (
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–77 STRIKE Procedure Parameters

Parameter	Description
ctext	The text to be rendered in strikethrough type.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<STRIKE cattributes>ctext</STRIKE>
```

STRONG Procedure

This procedure generates the <STRONG> and </STRONG> tags which direct the browser to render the text they surround in bold or however "strong" is defined.

Syntax

```
HTP.STRONG(
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–78 STRONG Procedure Parameters

Parameter	Description
ctext	The text to be emphasized.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<STRONG cattributes>ctext</STRONG>
```

STYLE Procedure

This procedure generates the <STYLE> and </STYLE> tags which include a style sheet in a Web page. You can get more information about style sheets at http://www.w3.org. This feature is not compatible with browsers that support only HTML versions 2.0 or earlier. Such browsers will ignore this tag.

Syntax

```
HTP.STYLE(
   cstyle          IN       VARCHAR2);
```

Parameters

Table 139–79 STYLE Procedure Parameters

Parameter	Description
cstyle	The the style information to include.

Examples

This procedure generates

```
<STYLE>cstyle</STYLE>
```

SUB Procedure

This procedure generates the _{and} tags which direct the browser to render the text they surround as subscript.

Syntax

```
HTP.SUB(
   ctext          IN       VARCHAR2,
   calign         in       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–80 SUB Procedure Parameters

Parameter	Description
ctext	The text to render in subscript.
calign	The value for the ALIGN attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<SUB ALIGN="calign" cattributes>ctext</SUB>
```

SUP Procedure

This procedure generates the ^{and} tags which direct the browser to render the text they surround as superscript.

Syntax

```
HTP.SUP(
   ctext          IN       VARCHAR2,
   calign         in       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–81 SUP Procedure Parameters

Parameter	Description
ctext	The text to render in superscript.
calign	The value for the ALIGN attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<SUP ALIGN="calign" cattributes>ctext</SUP>
```

TABLECAPTION Procedure

This procedure generates the <CAPTION> and </CAPTION> tags which place a caption in an HTML table.

Syntax

```
HTP.TABLECAPTION(
   ccaption       IN       VARCHAR2,
   calign         in       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–82 TABLECAPTION Procedure Parameters

Parameter	Description
ctext	The text for the caption.
calign	The value for the ALIGN attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<CAPTION ALIGN="calign" cattributes>ccaption</CAPTION>
```

TABLECLOSE Procedure

This procedure generates the </TABLE> tag which marks the end of an HTML table. To define the beginning of an HTML table, use the TABLEOPEN Procedure.

Syntax

```
HTP.TABLECLOSE;
```

Examples

This procedure generates

```
</TABLE>
```

TABLEDATA Procedure

This procedure generates the <TD> and </TD> tags which insert data into a cell of an HTML table.

Syntax

```
HTP.TABLEDATA(
   cvalue          IN       VARCHAR2   DEFAULT NULL,
   calign          IN       VARCHAR2   DEFAULT NULL,
   cdp             IN       VARCHAR2   DEFAULT NULL,
   cnowrap         IN       VARCHAR2   DEFAULT NULL,
   crowspan        IN       VARCHAR2   DEFAULT NULL,
   ccolspan        IN       VARCHAR2   DEFAULT NULL,
   cattributes     IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–83 TABLEDATA Procedure Parameters

Parameter	Description
cvalue	The data for the cell in the table.
calign	The value for the ALIGN attribute.
cdp	The value for the DP attribute.
cnowrap	If the value of this parameter is not NULL, the NOWRAP attribute is added to the tag.
ccolspan	The value for the COLSPAN attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<TD ALIGN="calign" DP="cdp" ROWSPAN="crowspan" COLSPAN="ccolspan" NOWRAP
cattributes>cvalue</TD>
```

TABLEHEADER Procedure

This procedure generates the <TH> and </TH> tags which insert a header cell in an HTML table. The <TH> tag is similar to the <TD> tag except that the text in this case the rows are usually rendered in bold type.

Syntax

```
HTP.TABLEHEADER(
   cvalue         IN       VARCHAR2   DEFAULT NULL,
   calign         IN       VARCHAR2   DEFAULT NULL,
   cdp            IN       VARCHAR2   DEFAULT NULL,
   cnowrap        IN       VARCHAR2   DEFAULT NULL,
   crowspan       IN       VARCHAR2   DEFAULT NULL,
   ccolspan       IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–84 TABLEHEADER Procedure Parameters

Parameter	Description
cvalue	The data for the cell in the table.
calign	The value for the ALIGN attribute.
cdp	The value for the DP attribute.
cnowrap	If the value of this parameter is not NULL, the NOWRAP attribute is added to the tag.
crispen	The value for the ROWSPAN attribute.
ccolspan	The value for the COLSPAN attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<TH ALIGN="calign" DP="cdp" ROWSPAN="crowspan" COLSPAN="ccolspan" NOWRAP
cattributes>cvalue</TH>
```

TABLEOPEN Procedure

This procedure generates the <TABLE> tag which marks the beginning of an HTML table. To define the end of an HTML table, use the TABLECLOSE Procedure.

Syntax

```
HTP.TABLEOPEN(
   cborder        IN       VARCHAR2   DEFAULT NULL
   calign         IN       VARCHAR2   DEFAULT NULL,
   cnowrap        IN       VARCHAR2   DEFAULT NULL,
   cclear         IN       VARCHAR2   DEFAULT NULL
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–85 TABLEOPEN Procedure Parameters

Parameter	Description
border	The value for the BORDER attribute.
calign	The value for the ALIGN attribute.
cnowrap	If the value of this parameter is not NULL, the NOWRAP attribute is added to the tag.
cclear	The value for the CLEAR attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<TABLE "cborder" NOWRAP ALIGN="calign" CLEAR="cclear" cattributes>
```

TABLEROWCLOSE Procedure

This procedure generates the </TR> tag which marks the end of a new row in an HTML table. To mark the beginning of a new row, use the TABLEROWOPEN Procedure.

Syntax

```
HTP.TABLEROWCLOSE;
```

Examples

This procedure generates

```
</TABLE>
```

TABLEROWOPEN Procedure

This procedure generates the <TR> tag which marks the beginning of a new row in an HTML table. To mark the end of a new row, use the TABLEROWCLOSE Procedure.

Syntax

```
HTP.TABLEROWOPEN(
   calign          IN       VARCHAR2   DEFAULT NULL,
   cvalign         IN       VARCHAR2   DEFAULT NULL,
   cdp             IN       VARCHAR2   DEFAULT NULL,
   cnowrap         IN       VARCHAR2   DEFAULT NULL,
   cattributes     IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–86 TABLEROWOPEN Procedure Parameters

Parameter	Description
calign	The value for the ALIGN attribute.
cvalign	The value for the VALIGN attribute.
cdp	The value for the DP attribute.
cnowrap	If the value of this parameter is not NULL, the NOWRAP attribute is added to the tag.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<<TR ALIGN="calign" VALIGN="cvalign" DP="cdp" NOWRAP cattributes>
```

TELETYPE Procedure

This procedure generates the <TT> and </TT> tags which direct the browser to render the text they surround in a fixed width typewriter font, for example, the courier font.

Syntax

```
HTP.TELETYPE(
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–87 TELETYPE Procedure Parameters

Parameter	Description
ctext	The text to render in a fixed width typewriter font.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<TT cattributes>ctext</TT>
```

TITLE Procedure

This procedure generates the <TITLE> and </TITLE> tags which specify the text to display in the titlebar of the browser window.

Syntax

```
HTP.TITLE(
   ctitle          IN       VARCHAR2);
```

Parameters

Table 139–88 TITLE Procedure Parameters

Parameter	Description
ctitle	The text to display in the titlebar of the browser window.

Examples

This procedure generates

```
<TITLE>ctitle</TITLE>
```

ULISTCLOSE Procedure

This procedure generates the </UL> tag which marks the end of an unordered list. An unordered list presents items with bullets. To mark the beginning of an unordered list, use the ULISTOPEN Procedure. Add list items with LISTITEM Procedure.

Syntax

```
HTP.ULISTCLOSE;
```

Examples

This procedure generates

```
</TABLE>
```

ULISTOPEN Procedure

This procedure generates the <UL> tag which marks the beginning of an unordered list. An unordered list presents items with bullets. To mark the end of an unordered list, use the ULISTCLOSE Procedure. Add list items with LISTITEM Procedure.

Syntax

```
HTP.ULISTOPEN(
   cclear          IN       VARCHAR2   DEFAULT NULL,
   cwrap           IN       VARCHAR2   DEFAULT NULL,
   cdingbat        IN       VARCHAR2   DEFAULT NULL,
   csrc            IN       VARCHAR2   DEFAULT NULL,
   cattributes     IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–89 ULISTOPEN Procedure Parameters

Parameter	Description
cclear	The value for the CLEAR attribute.
cwrap	The value for the WRAP attribute.
cdingbat	The value for the DINGBAT attribute.
csrc	The value for the SRC attribute.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<UL CLEAR="cclear" WRAP="cwrap" DINGBAT="cdingbat" SRC="csrc" cattributes>
```

UNDERLINE Procedure

This procedure generates the <U> and </U> tags, which direct the browser to render the text they surround with an underline.

Syntax

```
HTP.UNDERLINE(
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–90 UNDERLINE Procedure Parameters

Parameter	Description
ctext	The text to render with an underline.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

```
<U cattributes>ctext</U>
```

VARIABLE Procedure

This procedure generates the <VAR> and </VAR> tags which direct the browser to render the text they surround in italics or however "variable" is defined stylistically.

Syntax

```
HTP.VARIABLE(
   ctext          IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 139–91 VARIABLE Procedure Parameters

Parameter	Description
ctext	The text to render in italics.
cattributes	The other attributes to be included as-is in the tag.

Examples

This procedure generates

<VAR *cattributes*>*ctext*</VAR>

WBR Procedure

This procedure generates the <WBR> tag, which inserts a soft line break within a section of NOBR text.

Syntax

```
HTP.WBR;
```

Examples

This procedure generates

```
<WBR>
```

140

OWA_CACHE

The OWA_CACHE package provides an interface that enables the PL/SQL Gateway cache to improve the performance of PL/SQL web applications.

> **See Also:** For more information about implementation of this package:
>
> - *Oracle HTTP Server Administrator's Guide*
> - *Oracle Application Server mod_plsql User's Guide*

The chapter contains the following topics:

Using OWA_CACHE

Constants

- `system_level CONSTANT VARCHAR(6) := 'SYSTEM';`
- `user_level CONSTANT VARCHAR(4) := 'USER';`

Summary of OWA_CACHE Subprograms

Table 140–1 OWA_CACHE Package Subprograms

Subprogram	Description
DISABLE Procedure on page 140-5	Disables the cache for this particular request
GET_ETAG Function on page 140-6	Returns the tag associated with the cached content (used in the Validation technique model only)
GET_LEVEL Function on page 140-7	Returns the caching level (used in the Validation technique model only)
SET_CACHE Procedure on page 140-8	Sets up the cache headers for validation model cache type
SET_EXPIRES Procedure on page 140-9	Sets up the cache headers for expires model cache type
SET_NOT_MODIFIED Procedure on page 140-10	Sets up the headers for a not modified cache hit (used in the Validation technique model only)
SET_SURROGATE_CONTROL Procedure on page 140-11	Sets up the headers for a surrogate-control header for web cache

DISABLE Procedure

This procedure disables the cache for this particular request.

Syntax

```
OWA_CACHE.DISABLE;
```

GET_ETAG Function

This function returns the tag associated with the cached content. It is used in the Validation technique only.

Syntax

```
OWA_CACHE.GET_ETAG
  RETURN VARCHAR2;
```

Return Values

The tag for cache hit, otherwise `NULL`.

GET_LEVEL Function

This returns the caching level. It is used in the Validation technique model only.

Syntax

```
OWA_CACHE.GET_LEVEL
  RETURN VARCHAR2;
```

Return Values

The caching level string ('USER' or 'SYSTEM') for cache hit, otherwise NULL.

SET_CACHE Procedure

This sets up the cache headers for validation model cache type.

Syntax

```
OWA_CACHE.SET_CACHE(
   p_etag        IN       VARCHAR2,
   p_level       IN       VARCHAR2);
```

Parameters

Table 140–2 SET_CACHE Procedure Parameters

Parameter	Description
p_etag	The etag associated with this content
p_level	The caching level ('USER' or 'SYSTEM').

Exceptions

VALUE_ERROR is thrown if

- p_etag is greater than 55
- p_level is not 'USER' or 'SYSTEM'

SET_EXPIRES Procedure

This procedure sets up the cache headers for expires model cache type.

Syntax

```
OWA_CACHE.SET_EXPIRES(
   p_expires      IN       NUMBER,
   p_level        IN       VARCHAR2);
```

Parameters

Table 140–3 SET_EXPIRES Procedure Parameters

Parameter	Description
p_expires	The number of minutes this content is valid.
p_level	The caching level ('USER' or 'SYSTEM').

Exceptions

VALUE_ERROR is thrown if

- p_expires is negative or zero
- p_level is not 'USER' or 'SYSTEM'
- p_expires is > 525600 (1 year)

SET_NOT_MODIFIED Procedure

This procedure sets up the headers for a not-modified cache hit. It is used in the Validation technique only.

Syntax

```
OWA_CACHE.SET_NOT_MODIFIED;
```

Exceptions

`VALUE_ERROR` is thrown if If the `etag` was not passed in

SET_SURROGATE_CONTROL Procedure

This procedure sets the headers for a surrogate-control header for web cache

Syntax

```
OWA_CACHE.SET_SURROGATE_CONTROL(
   p_value        IN       VARCHAR2);
```

Parameters

Table 140–4 SET_SURROGATE_CONTROL Procedure Parameters

Parameter	Description
p_value	The value to be passed as the Surrogate-Control header .

Exceptions

VALUE_ERROR is thrown if If p_value is greater than 55 in length.

141

OWA_COOKIE

The OWA_COOKIE package provides an interface for sending and retrieving HTTP cookies from the client's browser.

> **See Also:** For more information about implementation of this package:
>
> - *Oracle HTTP Server Administrator's Guide*
> - *Oracle Application Server mod_plsql User's Guide*

The chapter contains the following topics:

Using OWA_COOKIE

Overview

Cookies are opaque strings sent to the browser to maintain state between HTTP calls. State can be maintained throughout the client's sessions, or longer if an expiration date is included. The system date is calculated with reference to the information specified in the OWA_CUSTOM package.

Types

This data type contains cookie name-value pairs. Since the HTTP standard allows cookie names to be overloaded (that is, multiple values can be associated with the same cookie name), there is a PL/SQL `RECORD` holding all values associated with a given cookie name.

```
TYPE vc_arr IS TABLE OF VARCHAR2(4000) INDEX BY BINARY_INTEGER.

TYPE COOKIE IS RECORD (
   name           VARCHAR2(4000),
   vals           vc_arr,
   num_vals       INTEGER);
```

Rules and Limits

All HTTP headers must be in English and the ASCII character set. If the headers are generated from the database, verify they are created in the English language.

Summary of OWA_COOKIE Subprograms

Table 141–1 OWA_COOKIE Package Subprograms

Subprogram	Description
GET Function on page 141-7	Gets the value of the specified cookie
GET_ALL Procedure on page 141-8	Gets all cookie name-value pairs
REMOVE Procedure on page 141-9	Removes the specified cookie
SEND procedure on page 141-10	Generates a "Set-Cookie" line in the HTTP header

GET Function

This function returns the values associated with the specified cookie. The values are returned in a OWA_COOKIE.COOKIE DATA TYPE.

Syntax

```
OWA_COOKIE.GET(
    name           IN       VARCHAR2)
  RETURN COOKIE;
```

Parameters

Table 141–2 GET Procedure Parameters

Parameter	Description
name	The name of the cookie.

Return Values

OWA_COOKIE.COOKIE DATA TYPE.

GET_ALL Procedure

This procedure returns all cookie names and their values from the client's browser. The values appear in the order in which they were sent from the browser.

Syntax

```
OWA_COOKIE.GET_ALL(
   names          OUT      vc_arr,
   vals           OUT      vc_arr,
   num_vals       OUT      INTEGER);
```

Parameters

Table 141–3 GET_ALL Procedure Parameters

Parameter	Description
names	The names of the cookies.
vals	The values of the cookies.
num_vals	The number of cookie-value pairs.

REMOVE Procedure

This procedure forces a cookie to expire immediately by setting the "expires" field of a Set-Cookie line in the HTTP header to "01-Jan-1990". This procedure must be called within the context of an HTTP header.

Syntax

```
OWA_COOKIE.REMOVE(
   name           IN       VARCHAR2,
   val            IN       VARCHAR2,
   path           IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 141–4 REMOVE Procedure Parameters

Parameter	Description
name	The name of the cookie to expire.
val	The value of the cookie.
path	[Currently unused]

SEND procedure

This procedure generates a Set-Cookie line, which transmits a cookie to the client. This procedure must occur in the context of an HTTP header.

Syntax

```
OWA_COOKIE.SEND(
   name           in       varchar2,
   value          in       varchar2,
   expires        in       date       DEFAULT NULL,
   path           in       varchar2   DEFAULT NULL,
   domain         in       varchar2   DEFAULT NULL,
   secure         in       varchar2   DEFAULT NULL);
```

Parameters

Table 141–5 SEND Procedure Parameters

Parameter	Description
`name`	The name of the cookie.
`value`	The value of the cookie.
`expires`	The date at which the cookie will expire
`path`	The value for the path field.
`domain`	The value for the domain field.
`secure`	If the value of this parameter is not `NULL`, the "secure" field is added to the line.

142

OWA_CUSTOM

The OWA_CUSTOM package provides a Global PLSQL Agent Authorization callback function. It is used when PLSQL Agent's authorization scheme is set to GLOBAL or CUSTOM when there is no overriding OWA_CUSTOM package.

> **See Also:** For more information about implementation of this package:
>
> - *Oracle HTTP Server Administrator's Guide*
> - *Oracle Application Server mod_plsql User's Guide*

The chapter contains the following topics:

Using OWA_CUSTOM

- Constants

Constants

- `dbms_server_timezone CONSTANT VARCHAR2(3) := 'PST';`
- `dbms_server_gmtdiff CONSTANT NUMBER := NULL;`

Summary of OWA_CUSTOM Subprograms

Table 142–1 OWA_CUSTOM Package Subprograms

Subprogram	Description
AUTHORIZE Function on page 142-5	Provides a Global PLSQL Agent Authorization callback function

AUTHORIZE Function

This function is used when PLSQL Agent's authorization scheme is set to `GLOBAL` or `CUSTOM` when there is no overriding `OWA_CUSTOM` package.

Syntax

```
OWA_CUSTOM.AUTHORIZE
  RETURN BOOLEAN;
```

143

OWA_IMAGE

The `OWA_IMAGE` package provides an interface to access the coordinates where a user clicked on an image.

> **See Also:** For more information about implementation of this package:
>
> - *Oracle HTTP Server Administrator's Guide*
> - *Oracle Application Server mod_plsql User's Guide*

The chapter contains the following topics:

Using OWA_IMAGE

Overview

Use this package when you have any image map whose destination links invoke the PL/SQL Gateway.

Types

This data type (`point`) contain the `X` and `Y` values of a coordinate, and so provides the coordinates of a user's click on an imagemap. It is defined as:

```
TYPE POINT IS TABLE OF VARCHAR2(32767) INDEX BY BINARY_INTEGER
```

Variables

This package variable (`null_point`) of `TYPE POINT` is used to default point parameters. Both the X and the Y fields of this variable are `NULL`.

Examples

```
CREATE OR REPLACE PROCEDURE process_image
    (my_img in OWA_IMAGE.POINT)
    AS
    x integer := OWA_IMAGE.GET_X(my_img);
    y integer := OWA_IMAGE.GET_Y(my_img);
BEGIN
    /* process the coordinate */
END
```

Summary of OWA_IMAGE Subprograms

Table 143–1 OWA_IMAGE Package Subprograms

Subprogram	Description
GET_X Function on page 143-8	Gets the X value of a point type
GET_Y Function on page 143-9	Gets the Y value of a point type

GET_X Function

This function returns the X coordinate of the point where the user clicked on an image map.

Syntax

```
OWA_IMAGE.GET_X(
   p         IN        point)
 RETURN INTEGER;
```

Parameters

Table 143–2 GET_X Procedure Parameters

Parameter	Description
p	The point where the user clicked.

Return Values

The X coordinate as an integer.

GET_Y Function

This function returns the Y coordinate of the point where the user clicked on an image map.

Syntax

```
OWA_IMAGE.GET_Y(
   p        IN        point)
 RETURN INTEGER;
```

Parameters

Table 143–3 GET_Y Procedure Parameters

Parameter	Description
p	The point where the user clicked.

Return Values

The Y coordinate as an integer.

144

OWA_OPT_LOCK

The OWA_OPT_LOCK package contains subprograms that impose optimistic locking strategies so as to prevent lost updates.

> **See Also:** For more information about implementation of this package:
>
> - *Oracle HTTP Server Administrator's Guide*
> - *Oracle Application Server mod_plsql User's Guide*

This chapter contains the following topics:

Using OWA_OPT_LOCK

Overview

The OWA_OPT_LOCK package contains subprograms that impose optimistic locking strategies, so as to prevent lost updates.

It checks if the row that the user is interested in updating has been changed by someone else in the meantime.

The PL/SQL Gateway cannot use conventional database locking schemes because HTTP is a stateless protocol. The `OWA_OPT_LOCK` package gives you two ways of dealing with the lost update problem:

- The hidden fields method stores the previous values in hidden fields in the HTML page. When the user requests an update, the PL/SQL Gateway checks these values against the current state of the database. The update operation is performed only if the values match. To use this method, call the owa_opt_lock.store_values procedure.
- The checksum method stores a checksum rather than the values themselves. To use this method, call the owa_opt_lock.checksum function.

These methods are optimistic. They do not prevent other users from performing updates, but they do reject the current update if an intervening update has occurred.

Types

This data type is a PL/SQL table intended to hold ROWIDs.

```
TYPE VCARRAY IS TABLE OF VARCHAR2(2000) INDEX BY BINARY_INTEGER
```

Note that this is different from the `OWA_TEXT.VC_ARR DATA TYPE`.

Summary of OWA_OPT_LOCK Subprograms

Table 144–1 OWA_CACHE Package Subprograms

Subprogram	Description
CHECKSUM Functions on page 144-6	Returns the checksum value
GET_ROWID Function on page 144-7	Returns the ROWID value
STORE_VALUES Procedure on page 144-8	Stores unmodified values in hidden fields for later verification
VERIFY_VALUES Function on page 144-9	Verifies the stored values against modified values

CHECKSUM Functions

This function returns a `checksum` value for a specified string, or for a row in a table. For a row in a table, the function calculates the `checksum` value based on the values of the columns in the row. This function comes in two versions.

The first version returns a `checksum` based on the specified string. This is a "pure" 32-bit `checksum` executed by the database and based on the Internet 1 protocol.

The second version returns a `checksum` based on the values of a row in a table. This is a "impure" 32-bit checksum based on the Internet 1 protocol.

Syntax

```
OWA_OPT_LOCK.CHECKSUM(
   p_buff        IN       VARCHAR2)
 RETURN NUMBER;

OWA_OPT_LOCK.CHECKSUM(
   p_owner       IN      VARCHAR2,
   p_tname       IN       VARCHAR2,
   p_rowid       IN       ROWID)
  RETURN NUMBER;
```

Parameters

Table 144–2 CHECKSUM Procedure Parameters

Parameter	Description
`p_buff`	The nstring where you want to calculate the `checksum`.
`p_owner`	The owner of the table.
`p_tname`	The table name.
`p_rowid`	The row in `p_tname` where you want to calculate the `checksum` value. Use the GET_ROWID Function to convert `VCARRAY` values to proper rowids.

GET_ROWID Function

This function returns the ROWID data type from the specified OWA_OPT_LOCK.VCARRAY DATA TYPE.

Syntax

```
OWA_OPT_LOCK.GET_ROWID(
   p_old_values      IN      vcarray)
 RETURN ROWID;
```

Parameters

Table 144–3 GET_ROWID Procedure Parameters

Parameter	Description
p_old_values	This parameter is usually passed in from an HTML form.

STORE_VALUES Procedure

This procedure stores the column values of the row that you want to update later. The values are stored in hidden HTML form elements.

Syntax

```
OWA_OPT_LOCK.STORE_VALUES(
   p_owner        IN       VARCHAR2,
   p_tname        IN       VARCHAR2,
   p_rowid        IN       ROWID);
```

Parameters

Table 144–4 STORE_VALUES Procedure Parameters

Parameter	Description
p_owner	The owner of the table.
p_tname	The name of the table.
p_rowid	The row where you want to store values.

Usage Notes

Before updating the row, compare these values with the current row values to ensure that the values in the row have not been changed. If the values have changed, you can warn the users and let them decide if the update should take place.

The procedure generates series of hidden form elements:

- One hidden form element is created for the table owner. The name of the element is "*old*_p_tname", where p_tname is the name of the table. The value of the element is the owner name.
- One hidden form element is created for the table name. The name of the element is "*old*_p_tname", where p_tname is the name of the table. The value of the element is the table name.
- One element is created for each column in the row. The name of the element is "*old*_p_tname", where p_tname is the name of the table. The value of the element is the column value.

See also the VERIFY_VALUES Function.

VERIFY_VALUES Function

This function verifies whether values in the specified row have been updated since the last query. Use this function with the STORE_VALUES Procedure.

Syntax

```
OWA_OPT_LOCK.VERIFY_VALUES(
   p_old_values IN vcarray)
 RETURN BOOLEAN;
```

Parameters

Table 144–5 VERIFY_VALUES Procedure Parameters

Parameter	Description
p_old_values	A PL/SQL table containing the following information: ▪ p_old_values(1) specifies the owner of the table. ▪ p_old_values(2) specifies the table. ▪ p_old_values(3) specifies the rowid of the row to verify. The remaining indexes contain values for the columns in the table. Typically, this parameter is passed in from the HTML form, where you have previously called the STORE_VALUES Procedure to store the row values on hidden form elements.

Return Values

TRUE if no other update has been performed, otherwise FALSE.

145

OWA_PATTERN

The OWA_PATTERN package provides an interface to locate text patterns within strings and replace the matched string with another string.

> **See Also:** For more information about implementation of this package:
>
> - *Oracle HTTP Server Administrator's Guide*
> - *Oracle Application Server mod_plsql User's Guide*

The chapter contains the following topics:

Using OWA_PATTERN

Types

You can use a pattern as both an input and output parameter. Thus, you can pass the same regular expression to `OWA_PATTERN` function calls, and it only has to be parsed once.

- `OWA_PATTERN.PATTERN`

Operational Notes

The OWA_PATTERN subprograms are overloaded. Specifically, there are six versions of MATCH, and four each of AMATCH and CHANGE. The subprograms use the following parameters:

- line - This is the target to be examined for a match. It can be more than one line of text or a owa_text.multi_line data type.
- pat - This is the pattern that the subprograms attempt to locate in line. The pattern can contain regular expressions. In the owa_pattern.change function and procedure, this parameter is called from_str.
- flags - This specifies whether the search is case-sensitive or if substitutions are done globally.

Use regular expressions with the subprograms in this package. You Specify a regular expression by creating the string you want to match interspersed with various wildcard tokens and quantifiers.

- Wildcards
- Quantifiers
- Flags

Wildcards

Wildcard tokens match something other than themselves:

Table 145–1 Wildcard tokens recognized by OWA_PATTERN package

Token	Description
^	Matches newline or the beginning of the target
$	Matches newline or the end of the target
\n	Matches newline
.	Matches any character except newline
\t	Matches tab
\d	Matches digits [0-9]
\D	Matches non-digits [not 0-9]
\w	Matches word characters (0-9, a-z, A-Z, or _)
\W	Matches non-word characters (not 0-9, a-z, A-Z, or _)
\s	Matches whitespace characters (blank, tab, or newline).
\S	Matches non-whitespace characters (not blank, tab, or newline)
\b	Matches "word" boundaries (between \w and \W)
\x<HEX>	Matches the value in the current character set of the two hexadecimal digits
\<OCT>	Matches the value in the current character set of the two or three octal digits
\	Followed by any character not covered by another case matches that character
&	Applies only to CHANGE. This causes the string that matched the regular expression to be included in the string that replaces it. This differs from the other tokens in that it specifies how a target is changed rather than how it is matched. This is explained further under CHANGE Functions and Procedures.

Quantifiers

Any tokens except & can have their meaning extended by any of the following quantifiers. You can also apply these quantifiers to literals:

Table 145–2 Quantifiers

Quantifier	Description
?	0 or 1 occurrence(s)
*	0 or more occurrences
+	1 or more occurrence(s)
{n}	Exactly *n* occurrences
(n,}	At least *n* occurrences
{n,m}	At least *n*, but not more than *m*, occurrences

Flags

In addition to targets and regular expressions, the OWA_PATTERN functions and procedures use flags to affect how they are interpreted.

Table 145–3 Flags

Flag	Description
i	This indicates a case-insensitive search.
g	This applies only to CHANGE. It indicates a global replace. That is, all portions of the target that match the regular expression are replaced.

Summary of OWA_PATTERN Subprograms

Table 145–4 OWA_CACHE Package Subprograms

Subprogram	Description
AMATCH Function on page 145-7	Determines if a string contains the specified pattern. It lets you specify where in the string the match has to occur
CHANGE Functions and Procedures on page 145-9	Replaces a pattern within a string. If you call it as a function it returns the number of times the regular expression was found and replaced
GETPAT Procedure on page 145-11	Generates a pattern data type from a `VARCHAR2` type
MATCH Function on page 145-12	Determines if a string contains the specified pattern

AMATCH Function

This function specifies if a pattern occurs in a particular location in a string. There are four versions to this function:

- The first and second versions of the function do not save the matched tokens (these are saved in the `backrefs` parameters in the third and fourth versions). The difference between the first and second versions is the `pat` parameter, which can be a `VARCHAR2` or a pattern data type.
- The third and fourth versions of the function save the matched tokens in the `backrefs` parameter. The difference between the third and fourth versions is the `pat` parameter, which can be a VARCHAR2 or a pattern data type.

> **Note:** If multiple overlapping strings match the regular expression, this function takes the longest match.

Syntax

```
OWA_PATTERN.AMATCH(
   line           IN       VARCHAR2,
   from_loc       IN       INTEGER,
   pat            IN       VARCHAR2,
   flags          IN       VARCHAR2   DEFAULT NULL)
 RETURN INTEGER;

OWA_PATTERN.AMATCH(
   line           IN       VARCHAR2,
   from_loc       IN       INTEGER,
   pat            IN OUT   PATTERN,
   flags          IN       VARCHAR2   DEFAULT NULL)
 RETURN INTEGER;

OWA_PATTERN.AMATCH(
   line           IN       VARCHAR2
   from_loc       IN       INTEGER
   pat            in       varchar2
   backrefs       OUT      owa_text.vc_arr
   flags          IN       VARCHAR2   DEFAULT NULL)
 RETURN INTEGER;

OWA_PATTERN.AMATCH(
   line           IN       VARCHAR2
   from_loc       IN       INTEGER
   pat            IN OUT   PATTERN
   backrefs       OUT      owa_text.vc_arr
   flags          IN       VARCHAR2   DEFAULT NULL)
 RETURN INTEGER;
```

Parameters

Table 145–5 AMATCH Procedure Parameters

Parameter	Description
`line`	The text to search in.

Table 145–5 (Cont.) AMATCH Procedure Parameters

Parameter	Description
from_loc	The location (in number of characters) in `line` where the search is to begin.
`pat`	The string to match. It can contain regular expressions. This can be either a `VARCHAR2` or a pattern. If it is a pattern, the output value of this parameter is the pattern matched.
backrefs	The text that is matched. Each token that is matched is placed in a cell in the `OWA_TEXT.VC_ARR DATA TYPE` PL/SQL table.
flags	Whether or not the search is case-sensitive. If the value of this parameter is "i", the search is case-insensitive. Otherwise the search is case-sensitive.

Return Values

The index of the character after the end of the match, counting from the beginning of `line`. If there was no match, the function returns `0`.

CHANGE Functions and Procedures

This function or procedure searches and replaces a string or `multi_line` data type. If multiple overlapping strings match the regular expression, this subprogram takes the longest match.

Syntax

```
OWA_PATTERN.CHANGE(
   line           IN OUT    VARCHAR2,
   from_str       IN        VARCHAR2,
   to_str         IN        VARCHAR2,
   flags          IN        VARCHAR2   DEFAULT NULL)
 RETURN INTEGER;

OWA_PATTERN.CHANGE(
   line           IN OUT   VARCHAR2,
   from_str       IN       VARCHAR2,
   to_str         IN       VARCHAR2,
   flags          IN       VARCHAR2   DEFAULT NULL);

owa_pattern.change(
   mline          IN OUT   owa_text.multi_line,
   from_str       IN       VARCHAR2,
   to_str         IN       VARCHAR2,
   flags          IN       VARCHAR2   DEFAULT NULL)
 RETURN INTEGER;

OWA_PATTERN.CHANGE(
   mline          IN OUT   owa_text.multi_line,
   from_str       IN       VARCHAR2,
   to_str         IN       VARCHAR2,
   flags          IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 145–6 CHANGE Procedure Parameters

Parameter	Description
`line`	The text to search in. The output value of this parameter is the altered string.
mline	The text to search in. This is a `owa_text.multi_line data type`. The output value of this parameter is the altered string.
`from_str`	The regular expression to replace.
to_str	The substitution pattern.
flags	Whether or not the search is case-sensitive, and whether or not changes are to be made globally. If `"i"` is specified, the search is case-insensitive. If `"g"` is specified, changes are made to all matches. Otherwise, the function stops after the first substitution is made.

Return Values

As a function, it returns the number of substitutions made. If the flag "g" is not used, this number can only be 0 or 1 and only the first match is replaced. The flag "g" specifies to replace all matches with the regular expression.

Examples

```
OWA_PATTERN.CHANGE('Cats in pajamas', 'C.+in', '& red ')
```

The regular expression matches the substring "`Cats in`". It then replaces this string with "& red". The ampersand character "`&`" indicates "`Cats in`" because that is what matched the regular expression. Thus, this procedure replaces the string "`Cats in pajamas`" with "`Cats in red`" If you call this as a function instead of a procedure, the value returned is 1, indicating that a single substitution has been made.

Example 2:

```
CREATE OR REPLACE PROCEDURE test_pattern as theline VARCHAR2(256);
num_found     INTEGER;
   BEGIN
       theline := 'what is the goal?';
       num_found := OWA_PATTERN.CHANGE(theline, 'goal', 'idea', 'g');
       HTP.PRINT(num_found); -- num_found is 1
       HTP.PRINT(theline); -- theline is 'what is the idea?'
   END;
/
SHOW ERRORS
```

GETPAT Procedure

This procedure converts a VARCHAR2 string into an OWA_PATTERN.PATTERN DATA TYPE.

Syntax

```
OWA_PATTERN.GETPAT(
   arg      IN          VARCHAR2,
   pat      IN OUT      pattern);
```

Parameters

Table 145–7 GETPAT Procedure Parameters

Parameter	Description
arg	The string to convert.
pat	the OWA_PATTERN.PATTERN DATA TYPE initialized with arg.

MATCH Function

This function determines if a string contains the specified pattern. The pattern can contain regular expressions. If multiple overlapping strings can match the regular expression, this function takes the longest match.

Syntax

```
owa_pattern.match(
   line           IN       VARCHAR2,
   pat            IN       VARCHAR2,
   flags          IN       VARCHAR2   DEFAULT NULL)
 RETURN BOOLEAN;

owa_pattern.match(
   line           IN       VARCHAR2,
   pat            IN OUT   PATTERN,
   flags          IN       VARCHAR2   DEFAULT NULL)
 RETURN BOOLEAN;

owa_pattern.match(
   line           IN       VARCHAR2,
   pat            IN       VARCHAR2,
   backrefs       OUT      owa_text.vc_arr,
   flags          IN       VARCHAR2   DEFAULT NULL)
 RETURN BOOLEAN;

OWA_PATTERN.MATCH(
   line           IN       VARCHAR2,
   pat            IN OUT   PATTERN,
   backrefs       OUT      owa_text.vc_arr,
   flags          IN       VARCHAR2   DEFAULT NULL)
 RETURN BOOLEAN;

owa_pattern.match(
   mline          IN       owa_text.multi_line,
   pat            IN       VARCHAR2,
   rlist          OUT      owa_text.row_list,
   flags          IN       VARCHAR2   DEFAULT NULL)
 RETURN BOOLEAN;

OWA_PATTERN.MATCH(
   mline          IN       owa_text.multi_line,
   pat            IN OUT   pattern,
   rlist          OUT      owa_text.row_list,
   flags          IN       VARCHAR2   DEFAULT NULL)
 RETURN BOOLEAN;
```

Parameters

Table 145–8 CHANGE Procedure Parameters

Parameter	Description
`line`	The line to search in.
`mline`	The text to search in. This is a `owa_text.multi_line data type`..

Table 145–8 (Cont.) CHANGE Procedure Parameters

Parameter	Description
pat	The pattern to match. This is either a VARCHAR2 or a OWA_PATTERN.PATTERN DATA TYPE. It it is a pattern, the output value of this parameter is the pattern matched.
backrefs	The text that is matched. Each token that is matched is placed in a cell in the OWA_TEXT.VC_ARR DATA TYPE PL/SQL table. This parameter is a row_list that holds each string in the target that was matched by a sequence of tokens in the regular expression.
rlist	An output parameter containing a list of matches.
flags	Whether or not the search is case-sensitive. If the value of this parameter is "i", the search is case-insensitive. Otherwise the search is case-sensitive.

Return Values

TRUE if a match was found, FALSE otherwise.

Examples

KAZOO is the target where it is searching for the zoo.* regular expression. The period indicates any character other than newline, and the asterisk matches 0 or more of the preceding characters. In this case, it matches any character other than the newline.

Therefore, this regular expression specifies that a matching target consists of zoo, followed by any set of characters neither ending in nor including a newline (which does not match the period). The i flag indicates to ignore case in the search. In this case, the function returns TRUE, which indicates that a match had been found.

```
boolean foundMatch;
foundMatch := owa_pattern.match('KAZOO', 'zoo.*', 'i');
```

The following example searches for the string "goal" followed by any number of characters in sometext. If found,

```
sometext  VARCHAR2(256);
pat       VARCHAR2(256);

sometext    := 'what is the goal?'
pat         := 'goal.*';
IF OWA_PATTERN.MATCH(sometext, pat)
   THEN
      HTP.PRINT('Match found');
   ELSE
     HTP.PRINT('Match not found');
END IF;
```

Operational Notes

- The regular expression in this function can be either a VARCHAR2 or an OWA_PATTERN.PATTERN DATA TYPE. Create AN OWA_PATTERN.PATTERN DATA TYPE from a string using the OWA_PATTERN.GETPAT procedure.
- Create a MULTI_LINE DATA TYPE from a long string using the OWA_TEXT.STREAM2MULTI procedure. If a multi_line is used, the rlist parameter specifies a list of chunks where matches were found.

- If the line is a string and not a `multi_line`, you can add an optional output parameter called `backrefs`. This parameter is a `row_list` that holds each string in the target that was matched by a sequence of tokens in the regular expression.

146

OWA_SEC

The OWA_SEC package provides an interface for custom authentication.

> **See Also:** For more information about implementation of this package:
>
> - *Oracle HTTP Server Administrator's Guide*
> - *Oracle Application Server mod_plsql User's Guide*

The chapter contains the following topics:

Using OWA_SEC

Operational Notes

Parameters that have default values are optional.

Summary of OWA_SEC Subprograms

Table 146–1 OWA_SEC Package Subprograms

Subprogram	Description
GET_CLIENT_HOSTNAME Function on page 146-5	Returns the client's hostname
GET_CLIENT_IP Function on page 146-6	Returns the client's IP address
GET_PASSWORD Function on page 146-7	Returns the password that the user entered
GET_USER_ID Function on page 146-8	Returns the username that the user entered
SET_AUTHORIZATION Procedure on page 146-9	Enables the PL/SQL application to use custom authentication
SET_PROTECTION_REALM Procedure on page 146-10	Defines the realm that the page is in

GET_CLIENT_HOSTNAME Function

This function returns the hostname of the client.

Syntax

```
OWA_SEC.GET_CLIENT_HOSTNAME
 RETURN VARCHAR2;
```

Return Values

The hostname.

GET_CLIENT_IP Function

This function returns the IP address of the client.

Syntax

```
OWA_SEC.GET_CLIENT_IP
  RETURN OWA_UTIL.IP_ADDRESS;
```

Return Values

The IP address. The `owa_util.ip_address data type` is a PL/SQL table where the first four elements contain the four numbers of the IP address. For example, if the IP address is `123.45.67.89` and the variable `ipaddr` is of the owa_util.ip_address data type, the variable would contain the following values:

```
ipaddr(1) = 123
ipaddr(2) = 45
ipaddr(3) = 67
ipaddr(4) = 89
```

GET_PASSWORD Function

This function returns the password that the user used to log in.

Syntax

```
OWA_SEC.GET_PASSWORD
  RETURN VARCHAR2;
```

Return Values

The password.

Usage Notes

For security reasons, this function returns a true value only when custom authentication is used. If you call this function when you are not using custom authentication, the function returns an undefined value. Thus, the database passwords are not exposed.

GET_USER_ID Function

This function returns the username that the user used to log in.

Syntax

```
OWA_SEC.GET_USER_ID
  RETURN VARCHAR2;
```

Return Values

The username.

SET_AUTHORIZATION Procedure

This procedure, called in the initialization portion of the OWA_CUSTOM package, sets the authorization scheme for the PL/SQL Gateway. This implements your `authorize` function, which authorizes the user before his requested procedure is run. The placement of the `authorize` function depends on the scheme you select.

Syntax

```
OWA_SEC.SET_AUTHORIZATION(
    scheme          IN       INTEGER);
```

Parameters

Table 146–2 SET_AUTHORIZATION Procedure Parameters

Parameter	Description

`scheme`	The authorization scheme. It is one of the following schemes for `SET_AUTHORIZATION`:
	■ `OWA_SEC.NO_CHECK` - Specifies that the PL/SQL application is not to do any custom authentication. This is the default.
	■ `OWA_SEC.GLOBAL` - Defines an authorize function that is called for all users and all procedures. This is the `OWA_CUSTOM`.AUTHORIZE Function in the "`sys`" schema.
	■ `OWA_SEC.PER_PACKAGE` - Define an `authorize` function that is called when procedures in a package or anonymous procedures are called. If the procedures are in a package, the package.`AUTHORIZE` function in the user's schema is called to authorize the user. If the procedures are not in a package, then the anonymous `authorize` function in the user's schema is called.
	■ `OWA_SEC.CUSTOM` - Implements different authorize functions for each user. The function `OWA_CUSTOM`.AUTHORIZE Function in the user's schema is called to authorize the user. If the user's schema does not contain an `OWA_CUSTOM`.AUTHORIZE Function, the PL/SQL Gateway looks for it in the "sys" schema.
	The custom `authorize` function has the following signature:
	`FUNCTION AUTHORIZE`
	`  RETURN BOOLEAN;`
	If the function returns `TRUE`, authentication succeeded. If it returns `FALSE`, authentication failed. If the `authorize` function is not defined, the Gateway returns an error and fails.

SET_PROTECTION_REALM Procedure

This procedure sets the realm of the page that is returned to the user. The user enters a username and login that already exist in the realm.

Syntax

```
OWA_SEC.SET_PROTECTION_REALM(
   realm      IN      VARCHAR2);
```

Parameters

Table 146–3 SET_PROTECTION_REALM Procedure Parameters

Parameter	Description
`realm`	The realm where the page belongs. This string is displayed to the user.

147

OWA_TEXT

The OWA_TEXT package contains subprograms used by OWA_PATTERN for manipulating strings. They are externalized so you can use them directly.

> **See Also:** For more information about implementation of this package:
>
> - *Oracle HTTP Server Administrator's Guide*
> - *Oracle Application Server mod_plsql User's Guide*

The chapter contains the following topics:

Using OWA_TEXT

- Types

Types

MULTI_LINE DATA TYPE

This data type is a PL/SQL record that holds large amounts of text. The rows field, of type OWA_TEXT.VC_ARR DATA TYPE, contains the text data in the record.

```
TYPE multi_line IS RECORD (
   rows           vc_arr,
   num_rows       INTEGER,
   partial_row    BOOLEAN);
```

ROW_LIST DATA TYPE

This is the data type for holding data to be processed.

```
TYPE row_list IS RECORD (
   rows           int_arr,
   num_rows       INTEGER);
```

```
int_arr IS DEFINED AS:
```
TYPE int_arr IS TABLE OF INTEGER INDEX BY BINARY_INTEGER;

VC_ARR DATA TYPE

This is a component of the MULTI_LINE DATA TYPE and is used for holding large amounts of text.

```
TYPE vc_arr IS TABLE OF VARCHAR2(32767) INDEX BY BINARY_INTEGER;
```

Summary of OWA_TEXT Subprograms

Table 147–1 OWA_TEXT Package Subprograms

Subprogram	Description
ADD2MULTI Procedure on page 147-5	Adds text to an existing `multi_line` type
NEW_ROW_LIST Function and Procedure on page 147-6	Creates a new `row_list`
PRINT_MULTI Procedure on page 147-7	Prints out the contents of a `multi_list`
PRINT_ROW_LIST Procedure on page 147-8	Prints out the contents of a `row_list`
STREAM2MULTI Procedure on page 147-9	Converts a varchar2 to a `multi_line` type

ADD2MULTI Procedure

This procedure adds content to an existing MULTI_LINE DATA TYPE.

Syntax

```
OWA_TEXT.ADD2MULTI(
   stream          IN       VARCHAR2,
   mline           IN OUT   multi_line,
  continue      IN      BOOLEAN   DEFAULT TRUE);
```

Parameters

Table 147–2 ADD2MULTI Procedure Parameters

Parameter	Description
`stream`	The text to add.
`mline`	The `OWA_TEXT.MULTI_LINE DATA TYPE`. The output of this parameter contains *stream*.
`continue`	If `TRUE`, the procedure appends *stream* within the previous final row (assuming it is less than 32K). If `FALSE`, the procedure places *stream* in a new row.

NEW_ROW_LIST Function and Procedure

This function or procedure creates a new OWA_TEXT.ROW_LIST DATA TYPE. The function version uses no parameters and returns a new empty row_list. The procedure version creates the row_list data type as an output parameter.

Syntax

```
OWA_TEXT.NEW_ROW_LIST
  RETURN ROW_LIST;

OWA_TEXT.NEW_ROW_LIST(
  rlist    OUT     row_list);
```

Parameters

Table 147–3 NEW_ROW_LIST Procedure Parameters

Parameter	Description
rlist	This is an output parameter containing the new row_list data type

Return Values

The function version returns the new row_list data type.

PRINT_MULTI Procedure

This procedure uses the PRINT Procedures or the PRN Procedures to print the "rows" field of the OWA_TEXT.MULTI_LINE DATA TYPE.

Syntax

```
OWA_TEXT.PRINT_MULTI(
   mline       IN       multi_line);
```

Parameters

Table 147–4 PRINT_MULTI Procedure Parameters

Parameter	Description
mline	The multi_line data type to print.

Return Values

The contents of the multi_line.

PRINT_ROW_LIST Procedure

This procedure uses the PRINT Procedures or the PRN Procedures to print the "rows" field of the `OWA_TEXT.ROW_LIST DATA TYPE`.

Syntax

```
OWA_TEXT.PRINT_ROW_LIST(
   rlist       IN       multi_line);
```

Parameters

Table 147–5 PRINT_ROW_LIST Procedure Parameters

Parameter	Description
`rlist`	The row_list data type to print.

Return Values

The contents of the `row_list`.

STREAM2MULTI Procedure

This procedure converts a string to a `multi_line` data type.

Syntax

```
OWA_TEXT.STREAM2MULTI(
   stream          IN       VARCHAR2
   mline           OUT      multi_line);
```

Parameters

Table 147–6 STREAM2MULTI Procedure Parameters

Parameter	Description
`stream`	The string to convert.
`mline`	The stream in `OWA_TEXT.MULTI_LINE DATA TYPE` format

148

OWA_UTIL

The `OWA_UTIL` package contains utility subprograms for performing operations such as getting the value of CGI environment variables, printing the data that is returned to the client, and printing the results of a query in an HTML table.

> **See Also:** For more information about implementation of this package:
>
> - *Oracle HTTP Server Administrator's Guide*
> - *Oracle Application Server mod_plsql User's Guide*

This chapter contains the following topics:

Using OWA_UTIL

Overview

The `OWA_UTIL` package contains three types of utility subprograms.

- Dynamic SQL Utilities enable you to produce pages with dynamically generated SQL code.
- HTML utilities enable you to retrieve the values of CGI environment variables and perform URL redirects.
- Date utilities enable correct date-handling. Date values are simple strings in HTML, but are treated as a data type by the Oracle database.

Types

DATETYPE Data Type

The TODATE Function converts an item of this type to the type `DATE`, which is understood and properly handled as data by the database. The procedure CHOOSE_DATE Procedure enables the user to select the desired date.

```
TYPE dateType IS TABLE OF VARCHAR2(10) INDEX BY BINARY_INTEGER;
```

IDENT_ARR Data Type

This data type is used for an array.

```
TYPE ident_arr IS TABLE OF VARCHAR2(30) INDEX BY BINARY_INTEGER;
```

IP_ADDRESS Data Type

This data type is used by the GET_CLIENT_IP Function in the "OWA_SEC" package on on page 146-1.

```
TYPE ip_address IS TABLE OF INTEGER INDEX BY BINARY_INTEGER;
```

Summary of OWA_UTIL Subprograms

Table 148–1 OWA_UTIL Package Subprograms

Subprogram	Description
BIND_VARIABLES Function on page 148-6	prepares a SQL query and binds variables to it
CALENDARPRINT Procedures on page 148-7	prints a calendar
CELLSPRINT Procedures on page 148-8	prints the contents of a query in an HTML table
CHOOSE_DATE Procedure on page 148-10	generates HTML form elements that allow the user to select a date
GET_CGI_ENV Function on page 148-11	returns the value of the specified CGI environment variable
GET_OWA_SERVICE_PATH Function on page 148-12	returns the full virtual path for the PL/SQL Gateway
GET_PROCEDURE Function on page 148-13	returns the name of the procedure that is invoked by the PL/SQL Gateway
HTTP_HEADER_CLOSE Procedure on page 148-14	closes the HTTP header
LISTPRINT Procedure on page 148-15	generates a HTML form element that contains data from a query
MIME_HEADER Procedure on page 148-16	generates the Content-type line in the HTTP header
PRINT_CGI_ENV Procedure on page 148-17	generates a list of all CGI environment variables and their values
REDIRECT_URL Procedure on page 148-18	generates the Location line in the HTTP header
SHOWPAGE Procedure on page 148-19	prints a page generated by the HTP and HTF packages in SQL*Plus
SHOWSOURCE Procedure on page 148-20	prints the source for the specified subprogram
SIGNATURE procedure on page 148-21	prints a line that says that the page is generated by the PL/SQL Agent
STATUS_LINE Procedure on page 148-22	generates the Status line in the HTTP header
TABLEPRINT Function on page 148-23	prints the data from a table in the database as an HTML table
TODATE Function on page 148-26	converts dateType data to the standard PL/SQL date type
WHO_CALLED_ME Procedure on page 148-27	returns information on the caller of the procedure.

BIND_VARIABLES Function

This function prepares a SQL query by binding variables to it, and stores the output in an opened cursor. Use this function as a parameter to a procedure sending a dynamically generated query. Specify up to 25 bind variables.

Syntax

```
OWA_UTIL.BIND_VARIABLES(
   theQuery       IN       VARCHAR2   DEFAULT NULL,
   bv1Name        IN       VARCHAR2   DEFAULT NULL,
   bv1Value       IN       VARCHAR2   DEFAULT NULL,
   bv2Name        IN       VARCHAR2   DEFAULT NULL,
   bv2Value       IN       VARCHAR2   DEFAULT NULL,
   bv3Name        IN       VARCHAR2   DEFAULT NULL,
   bv3Value       IN       VARCHAR2   DEFAULT NULL,
        ...
   bv25Name       IN       VARCHAR2   DEFAULT NULL,
   bv25Value      IN       VARCHAR2   DEFAULT NULL)
 RETURN INTEGER;
```

Parameters

Table 148–2 BIND_VARIABLES Function Parameters

Parameter	Description
`theQuery`	The SQL query statement which must be a SELECT statement
`bv1Name`	The name of the variable
`bv1Value`	The value of the variable

Return Values

An integer identifying the opened cursor.

CALENDARPRINT Procedures

These procedures creates a calendar in HTML with a visible border. Each date in the calendar can contain any number of hypertext links.

This procedure has 2 versions.

- Version 1 uses a hard-coded query stored in a varchar2 string.
- Version 2 uses a dynamic query prepared with the BIND_VARIABLES Function.

Syntax

```
OWA_UTIL.CALENDARPRINT(
   p_query        IN       VARCHAR2,
   p_mf_only      IN       VARCHAR2   DEFAULT 'N');

OWA_UTIL.CALENDARPRINT(
   p_cursor       IN       INTEGER,
   p_mf_only      IN       VARCHAR2   DEFAULT 'N');
```

Parameters

Table 148–3 CALENDARPRINT Procedure Parameters

Parameter	Description
p_query	A PL/SQL query.
p_cursor	A PL/SQL cursor containing the same format as p_query.
p_mf_only	If "N" (the default), the generated calendar includes Sunday through Saturday. Otherwise, it includes Monday through Friday only.

Usage Notes

Design your query as follows:

- The first column is a DATE. This correlates the information produced by the query with the calendar output generated by the procedure.
- The query output must be sorted on this column using ORDER BY.
- The second column contains the text, if any, that you want printed for that date.
- The third column contains the destination for generated links. Each item in the second column becomes a hypertext link to the destination given in this column. If this column is omitted, the items in the second column are simple text, not links.

CELLSPRINT Procedures

This procedure generates an HTML table from the output of a SQL query. SQL atomic data items are mapped to HTML cells and SQL rows to HTML rows. You must write the code to begin and end the HTML table. There are nine versions of this procedure:

- The first version passes the results of a query into an index table. Perform the query and CELLSPRINT does the formatting. To have more control in generating an HTML table from the output of an SQL query, use the FORMAT_CELL Function in the "HTF" package on page 134-1.
- The second and third versions display rows (up to the specified maximum) returned by the query or cursor.
- The fourth and fifth versions exclude a specified number of rows from the HTML table. Use the fourth and fifth versions to scroll through result sets by saving the last row seen in a hidden form element.
- The sixth through ninth versions are the same as the first four versions, except that they return a row count output parameter.

Syntax

```
OWA_UTIL.CELLSPRINT(
   p_colCnt          IN    INTEGER,
   p_resultTbl       IN    vc_arr,
  p_format_numbers IN    VARCHAR2   DEFAULT NULL);

OWA_UTIL.CELLSPRINT(
   p_theQuery        IN    VARCHAR2,
   p_max_rows        IN    NUMBER     DEFAULT 100,
   p_format_numbers  IN    VARCHAR2   DEFAULT NULL);

OWA_UTIL.CELLSPRINT(
   p_theCursor       IN    INTEGER,
   p_max_rows        IN    NUMBER     DEFAULT 100,
   p_format_numbers  iN    VARCHAR2   DEFAULT NULL);

OWA_UTIL.CELLSPRINT(
   p_theQuery        IN    VARCHAR2,
   p_max_rows        IN    NUMBER     DEFAULT 100,
   p_format_numbers  IN    VARCHAR2   DEFAULT NULL,
   p_skip_rec        IN    NUMBER      DEFAULT 0,
   p_more_data       OUT   BOOLEAN);

OWA_UTIL.CELLSPRINT(
   p_theCursor       IN    INTEGER,
   p_max_rows        IN    NUMBER     DEFAULT 100,
   p_format_numbers  IN    VARCHAR2   DEFAULT NULL,
   p_skip_rec        IN    NUMBER     DEFAULT 0,
   p_more_data       OUT   BOOLEAN);

OWA_UTIL.CELLSPRINT(
   p_theQuery        IN    VARCHAR2,
   p_max_rows        IN    NUMBER     DEFAULT 100,
   p_format_numbers  IN    VARCHAR2   DEFAULT NULL,
   p_reccnt          OUT   NUMBER);

OWA_UTIL.CELLSPRINT(
```

```
   p_theCursor       IN    INTEGER,
   p_max_rows        IN    NUMBER     DEFAULT 100,
   p_format_numbers  IN    VARCHAR2   DEFAULT NULL,
   p_reccnt          OUT   NUMBER);

OWA_UTIL.CELLSPRINT(
   p_theQuery        IN    VARCHAR2,
   p_max_rows        IN    NUMBER     DEFAULT 100,
   p_format_numbers  IN    VARCHAR2   DEFAULT NULL,
   p_skip_rec        IN    NUMBER     DEFAULT 0,
   p_more_data       OUT   BOOLEAN
   p_reccnt          OUT   NUMBER);

OWA_UTIL.CELLSPRINT(
   p_theCursor       IN    INTEGER,
   p_max_rows        IN    NUMBER     DEFAULT 100,
   p_format_numbers  IN    VARCHAR2   DEFAULT NULL,
   p_skip_rec        IN    NUMBER     DEFAULT 0,
   p_more_data       OUT   BOOLEAN,
   p_reccnt          OUT   NUMBER);
```

Parameters

Table 148–4 CELLSPRINT Procedure Parameters

Parameter	Description
p_query	A PL/SQL query.
p_colCnt	The number of columns in the table.
p_theQuery	A SQL SELECT statement.
p_theCursor	A cursor ID. This can be the return value from the BIND_VARIABLES Function.
p_max_rows	The maximum number of rows to print.
p_format_numbers	If the value of this parameter is not NULL, number fields are right justified and rounded to two decimal places.
p_skip_rec	The number of rows to exclude from the HTML table.
p_more_data	TRUE if there are more rows in the query or cursor, FALSE otherwise.
p_reccnt	The number of rows that have been returned by the query. This value does not include skipped rows (if any).
p_resultTbl	The index table which will contain the result of the query. Each entry in the query will correspond to one column value.

Examples

This function generates

```
<tr><td>QueryResultItem</td><td>QueryResultItem</td></tr>...
```

CHOOSE_DATE Procedure

This procedure generates three HTML form elements that allow the user to select the day, the month, and the year.

Syntax

```
OWA_UTIL.CHOOSE_DATE(
   p_name         IN       VARCHAR2,
   p_date         IN       DATE       DEFAULT SYSDATE);
```

Parameters

Table 148–5 CHOOSE_DATE Procedure Parameters

Parameter	Description
p_name	The name of the form elements.
p_date	The initial date that is selected when the HTML page is displayed.

Usage Notes

- The parameter in the procedure that receives the data from these elements must be a GET_CGI_ENV Function.
- Use the TODATE Function to convert the GET_CGI_ENV Function value to the standard Oracle DATE data type.

Examples

```
<SELECT NAME="p_name" SIZE="1">
<OPTION value="01">1
     ...
<OPTION value="31">31
</SELECT>
-
<SELECT NAME="p_name" SIZE="1">
<OPTION value="01">JAN
     ...
<OPTION value="12">DEC
</SELECT>
-
<SELECT NAME="p_name" SIZE="1">
<OPTION value="1992">1992
     ...
<OPTION value="2002">2002
</SELECT>
```

GET_CGI_ENV Function

This function returns the value of the specified CGI environment variable.

Syntax

```
OWA_UTIL.GET_CGI_ENV(
   param_name       IN      VARCHAR2)
 RETURN VARCHAR2;
```

Parameters

Table 148–6 GET_CGI_ENV Function Parameters

Parameter	Description
`param_name`	The name of the CGI environment variable. It is case-insensitive.

Return Values

The value of the specified CGI environment variable. If the variable is not defined, the function returns `NULL`.

GET_OWA_SERVICE_PATH Function

This function returns the full virtual path of the PL/SQL Gateway that is handling the request.

Syntax

```
OWA_UTIL.GET_OWA_SERVICE_PATH
 RETURN VARCHAR2;
```

Return Values

A virtual path of the PL/SQL Gateway that is handling the request.

GET_PROCEDURE Function

This function returns the name of the procedure that is being invoked by the PL/SQL Gateway.

Syntax

```
OWA_UTIL.GET_PROCEDURE
 RETURN VARCHAR2;
```

Return Values

The name of a procedure, including the package name if the procedure is defined in a package.

HTTP_HEADER_CLOSE Procedure

This procedure generates a newline character to close the HTTP header.

Syntax

```
OWA_UTIL.HTTP_HEADER_CLOSE;
```

Return Values

A newline character, which closes the HTTP header.

Usage Notes

- Use this procedure if you have not closed the header by using the `bclose_header` parameter in calls such as MIME_HEADER Procedure, REDIRECT_URL Procedure, or STATUS_LINE Procedure
- The HTTP header must be closed before any `HTP.PRINT` or `HTP.PRN` calls.

LISTPRINT Procedure

This procedure generates an HTML selection list form element from the output of a SQL query. There are two versions of this procedure.

- The first version contains a hard-coded SQL query.
- The second version uses a dynamic query prepared with the BIND_VARIABLES Function.

Syntax

```
OWA_UTIL.LISTPRINT(
   p_theQuery     IN       VARCHAR2,
   p_cname        IN       VARCHAR2,
   p_nsize        IN       NUMBER,
   p_multiple     IN       BOOLEAN   DEFAULT FALSE);

OWA_UTIL.LISTPRINT(
   p_theCursor    IN    INTEGER,
   p_cname        IN       VARCHAR2,
   p_nsize        IN       NUMBER,
   p_multiple     IN       BOOLEAN   DEFAULT FALSE);
```

Parameters

Table 148–7 LISTPRINT Procedure Parameters

Parameter	Description
p_theQuery	The SQL query.
p_theCursor	The cursor ID. This can be the return value from the BIND_VARIABLES Function.
p_cname	The name of the HTML form element.
p_nsize	The size of the form element (this controls how many items the user can see without scrolling).
p_multiple	Whether multiple selection is permitted.

Usage Notes

The columns in the output of the query are handled in the following manner:

- The first column specifies the values that are sent back. These values are for the VALUE attribute of the OPTION tag.
- The second column specifies the values that the user sees.
- The third column specifies whether or not the row is marked as SELECTED in the OPTION tag. If the value is not NULL, the row is selected.

Examples

```
<SELECT NAME="p_cname" SIZE="p_nsize">
<OPTION SELECTED value='value_from_the_first_column'>value_from_the_second_column
<OPTION SELECTED value='value_from_the_first_column'>value_from_the_second_column
     ...
</SELECT>
```

MIME_HEADER Procedure

This procedure changes the default MIME header that the script returns. This procedure must come before any HTP.PRINT or HTP.PRN calls to direct the script not to use the default MIME header.

Syntax

```
OWA_UTIL.MIME_HEADER(
   ccontent_type    IN       VARCHAR2   DEFAULT 'text/html',
   bclose_header    IN       BOOLEAN    DEFAULT TRUE,
   ccharset         IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 148–8 MIME_HEADER Procedure Parameters

Parameter	Description
ccontent_type	The MIME type to generate
bclose_header	Whether or not to close the HTTP header. If TRUE, two newlines are sent, which closes the HTTP header. Otherwise, one newline is sent, and the HTTP header remains open.
ccharset	The character set to use.The character set only makes sense if the MIME type is of type 'text'. Therefore, the character set is only tagged on to the Content-Type header only if the MIME type passed in is of type 'text'. Any other MIME type, such as 'image', will not have any character set tagged on.

Examples

```
Content-type: <ccontent_type>; charset=<ccharset>
```

so that

```
owa_util.mime_header('text/plain', false, 'ISO-8859-4')
```

generates

```
Content-type: text/plain; charset=ISO-8859-4\n
```

PRINT_CGI_ENV Procedure

This procedure generates all the CGI environment variables and their values made available by the PL/SQL Gateway to the stored procedure.

Syntax

```
OWA_UTIL.PRINT_CGI_ENV;
```

Examples

This procedure generates a list in the following format:

```
cgi_env_var_name = value\n
```

REDIRECT_URL Procedure

This procedure specifies that the application server is to visit the specified URL. The URL may specify either a web page to return or a program to execute.

Syntax

```
OWA_UTIL.REDIRECT_URL(
   curl           IN       VARCHAR2
   bclose_header  IN       BOOLEAN    DEFAULT TRUE);
```

Parameters

Table 148–9 REDIRECT_URL Function Parameters

Parameter	Description
curl	The URL to visit.
bclose_header	Whether or not to close the HTTP header. If TRUE, two newlines are sent, which closes the HTTP header. Otherwise, one newline is sent, and the HTTP header remains open.

Usage Notes

This procedure must come before any HTP procedure or HTF function call.

Examples

This procedure generates

```
Location: <curl>\n\n
```

SHOWPAGE Procedure

This procedure prints out the HTML output of a procedure in SQL*Plus. The procedure must use the HTP or HTF packages to generate the HTML page, and this procedure must be issued after the HTP or HTF page-generating subprogram has been called and before any other HTP or HTF subprograms are directly or indirectly called.

Syntax

```
OWA_UTIL.SHOWPAGE;
```

Usage Notes

- This method is useful for generating pages filled with static data.
- This procedure uses the DBMS_OUTPUT package and is limited to 255 characters for each line and an overall buffer size of 1,000,000 bytes.

Examples

The output of htp procedure is displayed in SQL*Plus, SQL*DBA, or Oracle Server Manager. For example:

```
SQL> set serveroutput on
SQL> spool gretzky.html
SQL> execute hockey.pass("Gretzky")
SQL> execute owa_util.showpage
SQL> exit
```

This would generate an HTML page that could be accessed from Web browsers.

SHOWSOURCE Procedure

This procedure prints the source of the specified procedure, function, or package. If a procedure or function which belongs to a package is specified, then the entire package is displayed.

Syntax

```
OWA_UTIL.SHOWSOURCE (
   cname      IN     VARCHAR2);
```

Parameters

Table 148–10 SHOWSOURCE Procedure Parameters

Parameter	Description
cname	The function or procedure whose source you want to show.

SIGNATURE procedure

This procedure generates an HTML line followed by a signature line on the HTML document. If a parameter is specified, the procedure also generates a hypertext link to view the PL/SQL source for that procedure. The link calls the SHOWSOURCE Procedure.

Syntax

```
OWA_UTIL.SIGNATURE;

OWA_UTIL.SIGNATURE (
   cname        IN        VARCHAR2);
```

Parameters

Table 148–11 SIGNATURE Procedure Parameters

Parameter	Description
cname	The function or procedure whose source you want to show.

Examples

Without a parameter, the procedure generates a line that looks like the following:

```
This page was produced by the PL/SQL Agent on August 9, 2001 09:30.
```

With a parameter, the procedure generates a signature line in the HTML document that looks like the following:

```
This page was produced by the PL/SQL Agent on 8/09/01 09:30
View PL/SQL Source
```

STATUS_LINE Procedure

This procedure sends a standard HTTP status code to the client. This procedure must come before any `htp.print` or `htp.prn` calls so that the status code is returned as part of the header, rather than as "content data".

Syntax

```
OWA_UTIL.STATUS_LINE(
   nstatus        IN       INTEGER,
   creason        IN       VARCHAR2   DEFAULT NULL,
   bclose_header  IN       BOOLEAN    DEFAULT TRUE);
```

Parameters

Table 148–12 STATUS_LINE Procedure Parameters

Parameter	Description
`nstatus`	The status code.
`creason`	The string for the status code.
`bclose_header`	Whether or not to close the HTTP header. If `TRUE`, two newlines are sent, which closes the HTTP header. Otherwise, one newline is sent, and the HTTP header remains open.

Examples

This procedure generates

`Status: <`*`nstatus`*`> <`*`creason`*`>\n\n`

TABLEPRINT Function

This function generates either preformatted tables or HTML tables (depending on the capabilities of the user's browser) from database tables.

Syntax

```
OWA_UTIL.TABLEPRINT(
   ctable         IN       VARCHAR2,
   cattributes    IN       VARCHAR2   DEFAULT NULL,
   ntable_type    IN       INTEGER    DEFAULT HTML_TABLE,
   ccolumns       IN       VARCHAR2   DEFAULT '*',
   cclauses       IN
  VARCHAR2   DEFAULT NULL,
   ccol_aliases   IN       VARCHAR2   DEFAULT NULL,
   nrow_min       IN       NUMBER     DEFAULT 0,
   nrow_max       IN       NUMBER     DEFAULT NULL)
 RETURN BOOLEAN;
```

Parameters

Table 148–13 TABLEPRINT Function Parameters

Parameter	Description
ctable	The database table.
cattributes	Other attributes to be included as-is in the tag.
ntable_type	How to generate the table. Specify "HTML_TABLE" to generate the table using <TABLE> tags or "PRE_TABLE" to generate the table using the <PRE> tags.
ccolumns	A comma-delimited list of columns from ctable to include in the generated table.
cclauses	WHERE or ORDER BY clauses, which specify which rows to retrieve from the database table, and how to order them.
ccol_aliases	A comma-delimited list of headings for the generated table.
nrow_min	The first row, of those retrieved, to display.
nrow_max	The last row, of those retrieved, to display.

Return Values

Returns TRUE if there are more rows beyond the nrow_max requested, FALSE otherwise.

Usage Notes

- RAW columns are supported, but LONG RAW columns are not. References to LONG RAW columns will print the result 'Not Printable'.
- Note that in this function, cattributes is the second rather than the last parameter.

Examples

For browsers that do not support HTML tables, create the following procedure:

```
CREATE OR REPLACE PROCEDURE showemps IS
```

```
  ignore_more BOOLEAN;
BEGIN
  ignore_more := OWA_UTIL.TABLEPRINT('emp', 'BORDER', OWA_UTIL.PRE_TABLE);
END;
```

Requesting a URL such as

```
http://myhost:7777/pls/hr/showemps
```

returns to the following to the client:

```
<PRE>
---------------------------------------------------------------------
| EMPNO |ENAME |JOB      |MGR   |HIREDATE   | SAL | COMM | DEPTNO |
---------------------------------------------------------------------
| 7369| SMITH  | CLERK   | 7902 | 17-DEC-80 | 800 |      | 20 |
| 7499| ALLEN  | SALESMAN| 7698 | 20-FEB-81 | 1600 | 300 | 30 |
| 7521| WARD   | SALESMAN| 7698 | 22-FEB-81 | 1250 | 500 | 30 |
| 7566| JONES  | MANAGER | 7839 | 02-APR-81 | 2975 |     | 20 |
| 7654| MARTIN | SALESMAN| 7698 | 28-SEP-81 | 1250 | 1400| 30 |
| 7698| BLAKE  | MANAGER | 7839 | 01-MAY-81 | 2850 |     | 30 |
| 7782| CLARK  | MANAGER | 7839 | 09-JUN-81 | 2450 |     | 10 |
| 7788| SCOTT  | ANALYST | 7566 | 09-DEC-82 | 3000 |     | 20 |
| 7839| KING   | PRESIDENT |    | 17-NOV-81 | 5000 |     | 10 |
| 7844| TURNER | SALESMAN| 7698 | 08-SEP-81 | 1500 | 0   | 30 |
| 7876| ADAMS  | CLERK   | 7788 | 12-JAN-83 | 1100 |     | 20 |
| 7900| JAMES  | CLERK   | 7698 | 03-DEC-81 | 950  |     | 30 |
| 7902| FORD   | ANALYST | 7566 | 03-DEC-81 | 3000 |     | 20 |
| 7934| MILLER | CLERK   | 7782 | 23-JAN-82 | 1300 |     | 10 |
---------------------------------------------------------------------
</PRE>
```

To view the employees in department 10, and only their employee ids, names, and salaries, create the following procedure:

```
CREATE OR REPLACE PROCEDURE showemps_10 IS
 ignore_more BOOLEAN;
begin
 ignore_more := OWA_UTIL.TABLEPRINT
    ('EMP', 'BORDER', OWA_UTIL.PRE_TABLE,
     'empno, ename, sal', 'WHERE deptno=10 ORDER BY empno',
     'Employee Number, Name, Salary');
END;
```

A request for a URL like

```
http://myhost:7777/pls/hr/showemps_10
```

would return the following to the client:

```
<PRE>
-------------------------------------
| Employee Number |Name    | Salary |
-------------------------------------
| 7782            | CLARK  | 2450   |
| 7839            | KING   | 5000   |
| 7934            | MILLER | 1300   |
-------------------------------------
</PRE>
```

For browsers that support HTML tables, to view the department table in an HTML table, create the following procedure:

```
CREATE OR REPLACE PROCEDURE showdept IS
 ignore_more BOOLEAN;
BEGIN
 ignore_more := oWA_UTIL.TABLEPRINT('dept', 'BORDER');
END;
```

A request for a URL like

```
http://myhost:7777/pls/hr/showdept
```

would return the following to the client:

```
<TABLE BORDER>
<TR>
<TH>DEPTNO</TH>
<TH>DNAME</TH>
<TH>LOC</TH>
</TR>
<TR>
<TD ALIGN="LEFT">10</TD>
<TD ALIGN="LEFT">ACCOUNTING</TD>
<TD ALIGN="LEFT">NEW YORK</TD>
</TR>
<TR>
<TD ALIGN="LEFT">20</TD>
<TD ALIGN="LEFT">RESEARCH</TD>
<TD ALIGN="LEFT">DALLAS</TD>
</TR>
<TR>
<TD ALIGN="LEFT">30</TD>
<TD ALIGN="LEFT">SALES</TD>
<TD ALIGN="LEFT">CHICAGO</TD>
</TR>
<TR>
<TD ALIGN="LEFT">40</TD>
<TD ALIGN="LEFT">OPERATIONS</TD>
<TD ALIGN="LEFT">BOSTON</TD>
</TR>
</TABLE>
```

A Web browser would format this to look like the following table:

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO

TODATE Function

This function converts the DATETYPE Data Type to the standard Oracle `DATE` type.

Syntax

```
OWA_UTIL.TODATE(
   p_dateArray      IN     dateType)
 RETURN DATE;
```

Parameters

Table 148–14 TODATE Function Parameters

Parameter	Description
`p_dateArray`	The value to convert.

WHO_CALLED_ME Procedure

This procedure returns information (in the form of output parameters) about the PL/SQL code unit that invoked it.

Syntax

```
OWA_UTIL.WHO_CALLED_ME(
   owner          OUT      VARCHAR2,
   name           OUT      VARCHAR2,
   lineno         OUT      NUMBER,
   caller_t       OUT      VARCHAR2);
```

Parameters

Table 148–15 WHO_CALLED_ME Procedure Parameters

Parameter	Description
owner	The owner of the program unit.
name	The name of the program unit. This is the name of the package, if the calling program unit is wrapped in a package, or the name of the procedure or function if the calling program unit is a standalone procedure or function. If the calling program unit is part of an anonymous block, this is NULL.
lineno	The line number within the program unit where the call was made.
caller_t	The type of program unit that made the call. The possibilities are: package body, anonymous block, procedure, and function. Procedure and function are only for standalone procedures and functions.

149

SDO_CS

The SDO_CS package contains functions and procedures for working with coordinate systems. You can perform explicit coordinate transformations on a single geometry or an entire layer of geometries (that is, all geometries in a specified column in a table).

Documentation of SDO_CS

For a complete description of this package within the context of Oracle Spatial, see `SDO_CS` in the *Oracle Spatial User's Guide and Reference*.

150

SDO_GCDR

The SDO_GCDR package contains the Oracle Spatial geocoding subprograms, which let you geocode unformatted postal addresses.

Documentation of SDO_GCDR

For a complete description of this package within the context of Oracle Spatial, see SDO_GCDR in *Oracle Spatial User's Guide and Reference.*

151

SDO_GEOM

The SDO_GEOM package contains the geometry functions, which can be grouped into the following categories:

- Relationship (True/False) between two objects: RELATE, WITHIN_DISTANCE
- Validation: VALIDATE_GEOMETRY, VALIDATE_LAYER
- Single-object operations: SDO_ARC_DENSIFY, SDO_AREA, SDO_BUFFER, SDO_CENTROID, SDO_CONVEXHULL, SDO_LENGTH, SDO_MBR, SDO_POINTONSURFACE
- Two-object operations: SDO_DISTANCE, SDO_DIFFERENCE, SDO_INTERSECTION, SDO_UNION, SDO_XOR

This chapter contains the following topic:

Documentation of SDO_GEOM

For a complete description of this package within the context of Oracle Spatial, see `SDO_GEOM` in the *Oracle Spatial User's Guide and Reference.*

152

SDO_GEOR

The SDO_GEOR package contains functions and procedures for the Oracle Spatial GeoRaster feature, which lets you store, index, query, analyze, and deliver raster image data and its associated spatial vector geometry data and metadata.

Documentation of SDO_GEOR

For complete description of this package within the context of Oracle Spatial, see `SDO_GEOR` in the *Oracle Spatial GeoRaster*.

153

SDO_GEOR_UTL

The SDO_GEOR_UTL package contains utility functions and procedures for the Oracle Spatial GeoRaster feature, including those related to using triggers with GeoRaster data.

Documentation of SDO_GEOR_UTL

For complete description of this package within the context of Oracle Spatial, see `SDO_GEOR_UTL` in the *Oracle Spatial GeoRaster*.

154

SDO_LRS

The SDO_LRS package contains functions that create, modify, query, and convert linear referencing elements.

- Documentation of SDO_LRS

Documentation of SDO_LRS

For a complete description of this package within the context of Oracle Spatial, see `SDO_LRS` in the *Oracle Spatial User's Guide and Reference*.

155

SDO_MIGRATE

The SDO_MIGRATE package lets you upgrade geometry tables from previous releases of Oracle Spatial.

Documentation of SDO_MIGRATE

For a complete description of this package within the context of Oracle Spatial, see `SDO_MIGRATE` in the *Oracle Spatial User's Guide and Reference*.

156

SDO_NET

The SDO_NET package contains functions and procedures for working with data modeled as nodes and links in a network.

- Documentation of SDO_NET

Documentation of SDO_NET

For a complete description of this package within the context of Oracle Spatial, see `SDO_NET` in the *Oracle Spatial Topology and Network Data Models*.

157

SDO_NET_MEM

The SDO_NET_MEM package contains functions and procedures for performing editing and analysis operations on network data using a network memory object.

Documentation of SDO_NET_MEM

For a complete description of this package within the context of Oracle Spatial, see `SDO_NET_MEM` in the *Oracle Spatial Topology and Network Data Models*.

158

SDO_SAM

The SDO_SAM package contains functions and procedures for spatial analysis and data mining.

Documentation of SDO_SAM

For a complete description of this package within the context of Oracle Spatial, see `SDO_SAM` in the *Oracle Spatial User's Guide and Reference*.

159

SDO_TOPO

The SDO_TOPO package contains subprograms for creating and managing Oracle Spatial topologies.

Documentation of SDO_TOPO

For a complete description of this package within the context of Oracle Spatial, see `SDO_TOPO` in the *Oracle Spatial Topology and Network Data Models*.

160

SDO_TOPO_MAP

The SDO_TOPO_MAP package contains subprograms for editing Oracle Spatial topologies using a cache (TopoMap object).

Documentation of SDO_TOPO_MAP

For a complete description of this package within the context of Oracle Spatial, see `SDO_TOPO_MAP` in the *Oracle Spatial Topology and Network Data Models*.

161

SDO_TUNE

The SDO_TUNE package contains Spatial tuning functions and procedures.

Documentation of SDO_TUNE

For complete description of this package within the context of Oracle Spatial, see `SDO_TUNE` in the *Oracle Spatial User's Guide and Reference.*

162

SDO_UTIL

The SDO_UTIL package contains the utility functions and procedures for Oracle Spatial.

- Documentation of SDO_UTIL

Documentation of SDO_UTIL

For complete description of this package within the context of Oracle Spatial, see `SDO_UTIL` in the *Oracle Spatial User's Guide and Reference.*

163

UTL_COLL

The UTL_COLL package lets PL/SQL programs use collection locators to query and update.

This chapter contains the following topics:

Summary of UTL_COLL Subprograms

Table 163–1 UTL_COLL Package Subprograms

Subprogram	Description
IS_LOCATOR Function on page 163-3	Determines whether a collection item is actually a locator or not

IS_LOCATOR Function

This function determines whether a collection item is actually a locator or not.

Syntax

```
UTL_COLL.IS_LOCATOR (
   coln IN STANDARD)
  RETURNS BOOLEAN;
```

Pragmas

Asserts `WNDS`, `WNPS` and `RNPS` pragmas

Parameters

Table 163–2 IS_LOCATOR Function Parameters

Parameter	Description
`coln`	Nested table or varray item.

Return Values

Table 163–3 IS_LOCATOR Function Return Values

Return Value	Description
`1`	Collection item is indeed a locator.
`0`	Collection item is not a locator.

Examples

```
CREATE OR REPLACE TYPE list_t as TABLE OF VARCHAR2(20);
/

CREATE OR REPLACE TYPE phone_book_t AS OBJECT (
  pno  number,
  ph   list_t );
/

CREATE TABLE phone_book OF phone_book_t
      NESTED TABLE ph STORE AS nt_ph;
CREATE TABLE phone_book1 OF phone_book_t
      NESTED TABLE ph STORE AS nt_ph_1 RETURN LOCATOR;

INSERT INTO phone_book VALUES(1, list_t('650-633-5707','650-323-0953'));
INSERT INTO phone_book1 VALUES(1, list_t('415-555-1212'));

CREATE OR REPLACE PROCEDURE chk_coll IS
  plist list_t;
  plist1 list_t;
BEGIN
  SELECT ph INTO plist FROM phone_book WHERE pno=1;

  SELECT ph INTO plist1 FROM phone_book1 WHERE pno=1;

  IF (UTL_COLL.IS_LOCATOR(plist)) THEN
    DBMS_OUTPUT.PUT_LINE('plist is a locator');
```

```
  ELSE
    DBMS_OUTPUT.PUT_LINE('plist is not a locator');
  END IF;

  IF (UTL_COLL.IS_LOCATOR(plist1)) THEN
    DBMS_OUTPUT.PUT_LINE('plist1 is a locator');
  ELSE
    DBMS_OUTPUT.PUT_LINE('plist1 is not a locator');
  END IF;

END chk_coll;

SET SERVEROUTPUT ON
EXECUTE chk_coll;
```

164

UTL_COMPRESS

The `UTL_COMPRESS` package provides a set of data compression utilities.

This chapter contains the following topics:

Using UTL_COMPRESS

Constants

Define max number of handles for piecewise operations:

```
UTLCOMP_MAX_HANDLE   CONSTANT  PLS_INTEGER := 5;
```

Exceptions

Table 164–1 UTL_COMPRESS Exceptions

Exception	Description
BUFFER_TOO_SMALL	The compressed representation is too big.
DATA_ERROR	The input or output data stream was found to be an invalid format.
INVALID_ARGUMENT	One of the arguments was an invalid type or value.
INVALID_HANDLE	Invalid handle for piecewise compress or uncompress.
STREAM_ERROR	An error occurred during compression or uncompression of the data stream

Operational Notes

- It is the caller's responsibility to free the temporary `LOB` returned by the `LZ*` functions with `DBMS_LOB.FREETEMPORARY` call.
- A `BFILE` passed into `LZ_COMPRESS*` or l`Z_UNCOMPRESS*` has to be opened by `DBMS_LOB.FILEOPEN`.
- Under special circumstances (especially if the input has already been compressed) the output produced by one of the `UTL_COMPRESS` subprograms may be the same size, or even slightly larger than, the input.
- The output of the `UTL_COMPRESS` compressed data is compatible with `gzip`(with `-n` option)`/gunzip` on a single file.

Summary of UTL_COMPRESS Subprograms

Table 164–2 UTL_COMPRESS Package Subprograms

Subprogram	Description
ISOPEN Function on page 164-7	Checks to see if the handle to a piecewise (un)compress context is open or closed
LZ_COMPRESS Functions and Procedures on page 164-8	Compresses data using Lempel-Ziv compression algorithm
LZ_COMPRESS_ADD Procedure on page 164-10	Adds a piece of compressed data
LZ_COMPRESS_CLOSE on page 164-11	Closes and finishes piecewise compress operation
LZ_COMPRESS_OPEN on page 164-12	Initializes a piecewise context that maintains the compress state and data
LZ_UNCOMPRESS Functions and Procedures on page 164-13	Accepts compressed input, verifies it to be a valid and uncompresses it
LZ_UNCOMPRESS_EXTRACT Procedure on page 164-14	Extracts a piece of uncompressed data
LZ_UNCOMPRESS_OPEN Function on page 164-15	Initializes a piecewise context that maintains the uncompress state and data
LZ_UNCOMPRESS_CLOSE Procedure on page 164-16	Closes and finishes the piecewise uncompress

ISOPEN Function

This function checks to see if the handle to a piecewise (un)compress context is open or closed.

Syntax

```
UTL_COMPRESS.ISOPEN(
   handle in binary_integer)
 RETURN BOOLEAN;
```

Parameters

Table 164–3 ISOPEN Function Parameters

Parameter	Description
handle	The handle to a piecewise uncompress context.

Return Values

TRUE if the given piecewise handle is opened, otherwise FALSE.

Examples

```
IF (UTL_COMPRESS.ISOPEN(myhandle) = TRUE) then
   UTL_COMPRESS.LZ_COMPRESS_CLOSE(myhandle, lob_1);
END IF;
```

Alternatively:

```
IF (UTL_COMPRESS.ISOPEN(myhandle) = TRUE) THEN
   UTL_COMPRESS.LZ_UNCOMPRESS_CLOSE(myhandle);
END IF;
```

LZ_COMPRESS Functions and Procedures

These functions and procedures compress data using Lempel-Ziv compression algorithm.

Syntax

This function accept a RAW as input, compress it and return the compressed RAW result and metadata:

```
UTL_COMPRESS.LZ_COMPRESS (
   src       IN           RAW,
   quality   IN           BINARY_INTEGER DEFAULT 6)
 RETURN RAW;
```

This function accept a BLOB as input, compress it and returns a temporary BLOB for the compressed data:

```
UTL_COMPRESS.LZ_COMPRESS (
  src       IN           BLOB,
  quality   IN           BINARY_INTEGER DEFAULT 6)
 RETURN BLOB;
```

This procedure returns the compressed data into the existing BLOB(dst) which is trimmed to the compressed data size:

```
UTL_COMPRESS.LZ_COMPRESS (
  src      IN            BLOB,
  dst      IN OUT NOCOPY BLOB,
  quality  IN            BINARY_INTEGER DEFAULT 6);
```

This function returns a temporary BLOB for the compressed data:

```
UTL_COMPRESS.LZ_COMPRESS (
   src     IN            BFILE,
   quality IN            BINARY_INTEGER DEFAULT 6)
 RETURN BLOB;
```

This procedure will return the compressed data into the existing BLOB(dst) which is trimmed to the compressed data size:

```
UTL_COMPRESS.LZ_COMPRESS (
   src     IN            BFILE,
   dst     IN OUT NOCOPY BLOB,
   quality IN            BINARY_INTEGER DEFAULT 6);
```

Parameters

Table 164–4 LZ_COMPRESS Function and Procedures Parameters

Parameter	Description
src	Data (RAW, BLOB or BFILE) to be compressed.
dst	Destination for compressed data
quality	An integer in the range 1 to 9, 1=fast compression, 9=best compression, default=6

Usage Notes

- `quality` is an optional compression tuning value. It allows the `UTL_COMPRESS` user to choose between speed and compression quality, meaning the percentage of reduction in size. A faster compression speed will result in less compression of the data. A slower compression speed will result in more compression of the data. Valid values are [1..9], with 1=fastest and 9=slowest. The default 'quality' value is 6.

LZ_COMPRESS_ADD Procedure

This procedure adds a piece of compressed data.

Syntax

```
UTL_COMPRESS.LZ_COMPRESS_ADD (
   handle IN             BINARY_INTEGER,
   dst    IN OUT NOCOPY  BLOB,
   src    IN             RAW);
```

Parameters

Table 164–5 LZ_COMPRESS_ADD Procedure Parameters

Parameter	Description
`handle`	The handle to a piecewise compress context.
`dst`	The opened `LOB` from `LZ_COMPRESS_OPEN` to store compressed data.
`src`	The input data to be compressed.

Exceptions

- `invalid_handle` - out of range invalid or unopened handle.
- `invalid_argument` - `NULL` handle, src, dst, or invalid `dst`.

LZ_COMPRESS_CLOSE

This procedure closes and finishes piecewise compress operation.

Syntax

```
UTL_COMPRESS.LZ_COMPRESS_CLOSE (
   handle IN             BINARY_INTEGER,
   dst    IN OUT NOCOPY  BLOB);
```

Parameters

Table 164–6 LZ_COMPRESS_CLOSE Procedure Parameters

Parameter	Description
`handle`	The handle to a piecewise compress context.
`dst`	The opened `LOB` from `LZ_COMPRESS_OPEN` to store compressed data.

Exceptions

- `invalid_handle` - out of range invalid or uninitialized handle.
- `invalid_argument` - NULL handle, `dst`, or invalid `dst`.

LZ_COMPRESS_OPEN

This function initializes a piecewise context that maintains the compress state and data.

Syntax

```
UTL_COMPRESS.LZ_COMPRESS_OPEN (
   dst       IN OUT NOCOPY BLOB,
   quality   IN            BINARY_INTEGER DEFAULT 6)
 RETURN BINARY_INTEGER;
```

Parameters

Table 164–7 LZ_COMPRESS_OPEN Function Parameters

Parameter	Description
`dst`	User supplied LOB to store compressed data.
`quality`	Speed versus efficiency of resulting compressed output. ■ Valid values are the range 1..9, with a default value of 6. ■ 1=fastest compression, 9=slowest compression and best compressed file size.

Return Values

A handle to an initialized piecewise compress context.

Exceptions

- `invalid_handle` - invalid handle, too many open handles.
- `invalid_argument` - `NULL dst` or invalid quality specified.

Usage Notes

Close the opened handle with `LZ_COMPRESS_CLOSE`

- once the piecewise compress is completed
- in the event of an exception in the middle of process

because lack of doing so will cause these handles to leak.

LZ_UNCOMPRESS Functions and Procedures

This procedure accepts as input a RAW, BLOB or BFILE compressed string, verifies it to be a valid compressed value, uncompresses it using Lempel-Ziv compression algorithm, and returns the uncompressed RAW or BLOB result.

Syntax

This function returns uncompressed data as RAW:

```
UTL_COMPRESS.LZ_UNCOMPRESS(
   src  IN  RAW)
  RETURN RAW;
```

This function returns uncompressed data as a temporary BLOB:

```
UTL_COMPRESS.LZ_UNCOMPRESS(
   src  IN  BLOB)
  RETURN BLOB;
```

This procedure returns the uncompressed data into the existing BLOB(dst), which will be trimmed to the uncompressed data size:

```
UTL_COMPRESS.LZ_UNCOMPRESS(
   src  IN  BLOB,
   dst  IN OUT NOCOPY BLOB);
```

This function returns a temporary BLOB for the uncompressed data:

```
UTL_COMPRESS.LZ_UNCOMPRESS(
   src  IN BFILE)
 RETURN BLOB;
```

This procedure returns the uncompressed data into the existing BLOB(dst). The original dst data will be overwritten.

```
UTL_COMPRESS.LZ_UNCOMPRESS(
   src  IN  BFILE,
   dst  IN OUT NOCOPY BLOB);
```

Parameters

Table 164–8 LZ_UNCOMPRESS Function and Procedures Parameters

Parameter	Description
src	Compressed data.
dst	Destination for uncompressed data.

LZ_UNCOMPRESS_EXTRACT Procedure

This procedure extracts a piece of uncompressed data.

Syntax

```
UTL_COMPRESS.LZ_UNCOMPRESS_EXTRACT(
   handle  IN          BINARY_INTEGER,
   dst     OUT NOCOPY  RAW);
```

Parameters

Table 164–9 LZ_UNCOMPRESS_EXTRACT Function Parameters

Parameter	Description
`handle`	The handle to a piecewise uncompress context.
`dst`	The uncompressed data.

Exceptions

- `no_data_found` - finished uncompress.
- `invalid_handle` - out of range invalid or uninitialized handle.
- `invalid_argument` - NULL handle.

LZ_UNCOMPRESS_OPEN Function

This function initializes a piecewise context that maintains the uncompress state and data.

Syntax

```
UTL_COMPRESS.LZ_UNCOMPRESS_OPEN(
   src  IN  BLOB)
  RETURN BINARY_INTEGER;
```

Parameters

Table 164–10 LZ_UNCOMPRESS_OPEN Function Parameters

Parameter	Description
src	The input data to be uncompressed.

Return Values

A handle to an initialized piecewise compress context.

Exceptions

- invalid_handle - invalid handle, too many open handles.
- invalid_argument - NULL src.

Usage Notes

Close the opened handle with LZ_UNCOMPRESS_CLOSE

- once the piecewise uncompress is completed
- in the event of an exception in the middle of process

because lack of doing so will cause these handles to leak.

LZ_UNCOMPRESS_CLOSE Procedure

This procedure closes and finishes the piecewise uncompress.

Syntax

```
UTL_COMPRESS.LZ_UNCOMPRESS_CLOSE(
   handle  IN   BINARY_INTEGER);
```

Parameters

Table 164–11 LZ_UNCOMPRESS_CLOSE Procedure Parameters

Parameter	Description
`handle`	The handle to a piecewise uncompress context.

Exceptions

- `invalid_handle` - out of range invalid or uninitialized handle.
- `invalid_argument` - `NULL` handle.

165

UTL_DBWS

The `UTL_DBWS` package provides database web services.

This chapter contains the following topics:

Using UTL_DBWS

Supported Keys and Default Settings for Standard Call Properties

Table 165–1 Supported Keys and Default Settings for Standard Call Properties

Key	Explanation of Value, Default value
'USERNAME'	User name for authentication.
'PASSWORD'	Password for authentication.
'ENCODINGSTYLE_URI'	Encoding style specified as a namespace URI. The default value is the SOAP 1.1 encoding http://schemas.xmlsoap.org/soap/encoding/.
'OPERATION_STYLE'	Standard property for operation style. Set to 'RPC' if the operation style is RPC, 'DOCUMENT' if the operation style is document.
'SESSION_MAINTAIN'	This boolean property is used by a service client to indicate whether or not it wants to participate in a session with a service endpoint.If this property is set to 'TRUE', the service client indicates that it wants the session to be maintained. If set to 'FALSE', the session is not maintained. The default value for this property is 'FALSE'
'SOAPACTION_USE'	This boolean property indicates whether or not SOAPAction is to be used. The default value of this property is 'FALSE'.
'SOAPACTION_URI'	Indicates the SOAPAction URI if the SOAPACTION_USE property is set to 'TRUE'

Summary of UTL_DBWS Subprograms

Table 165–2 UTL_DBWS Subprograms

Subprogram	Description
CREATE_CALL Function on page 165-5	Creates a Call instance
CREATE_SERVICE Function on page 165-6	Creates a Service instance
GET_IN_PARAMETER_ TYPES Function on page 165-7	Lists the XML type of the input parameters of the Call that is returned
GET_OUT_PARAMETER_ TYPES Function on page 165-8	Lists the XML type of the output parameters of the Call that is returned
GET_OUTPUT_VALUES Function on page 165-9	Obtains the output arguments after a Call invocation
GET_PORTS Function on page 165-10	Lists the qualified names of all of the ports in a service
GET_PROPERTY Function on page 165-11	Returns the value of a particular property on a Call
GET_RETURN_TYPE Function on page 165-12	Lists the XML type that is returned by the given Call
GET_SERVICES Function on page 165-13	Lists the qualified names of the services defined in a WDSL document
INVOKE Function on page 165-14	Invokes a specific operation using a synchronous request-response interaction mode
RELEASE_ALL_SERVICES Procedure on page 165-15	Releases all Service instances
RELEASE_CALL Procedure on page 165-16	Releases a particular Call instance
RELEASE_SERVICE Procedure on page 165-17	Releases a particular Service instance
REMOVE_PROPERTY Procedure on page 165-18	Clears the value of a particular property on a Call
SET_PROPERTY Procedure on page 165-19	Sets the value of a particular property on a Call

CREATE_CALL Function

This procedure creates a Call instance.

Syntax

```
UTL_DBWS.CREATE_CALL (
   service_handle    SERVICE,
   port_name         QNAME,
   operation_name    QNAME)
 RETURN CALL;
```

Parameters

Table 165–3 CREATE_CALL Function Parameters

Parameter	Description
service_handle	The Service instance to be called.
port_name	The qualified name for the port. Use the first port if this is NULL.
operation_name	The qualified name for the operation.

Return Values

Table 165–4 CREATE_CALL Return Values

Parameter	Description
CALL	Returns a handle to the Call instance.

CREATE_SERVICE Function

This procedure creates a Service instance.

Syntax

```
UTL_DBWS.CREATE_SERVICE(
   wsdl_document_location  URITYPE,
   service_name            QNAME)
  RETURN SERVICE;
```

Parameters

Table 165–5 CREATE_SERVICE Function Parameters

Parameter	Description
`wsdl_document_location`	The URL for the WSDL document location for the service
`service_name`	The qualified name for the service. Use the first service if this is `NULL`.

Return Values

Table 165–6 CREATE_SERVICE Return Values

Parameter	Description
`SERVICE`	Returns a handle to the Service instance.

GET_IN_PARAMETER_TYPES Function

This procedure lists the XML type of the input parameters of the Call that is returned.

Syntax

```
UTL_DBWS.GET_IN_PARAMETER_TYPES(
   call_handle    CALL)
 RETURN QNAME_LIST;
```

Parameters

Table 165–7 GET_IN_PARAMETER_TYPES Function Parameters

Parameter	Description
call_handle	The Service instance whose input types are returned.

Return Values

Table 165–8 GET_IN_PARAMETER_TYPES Function Return Values

Parameter	Description
QNAME_LIST	The list of the XML type of the input parameters of the Call that is returned.

GET_OUT_PARAMETER_TYPES Function

This procedure lists the XML type of the output parameters of the Call that is returned.

Syntax

```
UTL_DBWS.GET_OUT_PARAMETER_TYPES(
   call_handle    CALL)
 RETURN QNAME_LIST;
```

Parameters

Table 165–9 GET_OUT_PARAMETER_TYPES Function Parameters

Parameter	Description
call_handle	The Service instance whose output types are returned.

Return Values

Table 165–10 GET_OUT_PARAMETER_TYPES Function Return Values

Parameter	Description
QNAME_LIST	The list of the XML type of the input parameters of the Call that is returned.

GET_OUTPUT_VALUES Function

This procedure obtains the output arguments after a Call invocation.

Syntax

```
UTL_DBWS.GET_OUTPUT_VALUES(
   call_handle    CALL)
 RETURN ANYDATA_LIST;
```

Parameters

Table 165–11 GET_OUTPUT_VALUES Function Parameters

Parameter	Description
call_handle	The instance of the Call.

Return Values

Table 165–12 GET_OUTPUT_VALUES Function Return Values

Parameter	Description
ANYDATA_LIST	Returns the output arguments in order.

GET_PORTS Function

This procedure lists the qualified names of all of the ports in a service.

Syntax

```
UTL_DBWS.GET_PORTS(
   service_handle    SERVICE)
 RETURN QNAME_LIST;
```

Parameters

Table 165–13 GET_PORTS Function Parameters

Parameter	Description
service_handle	The service instance whose ports are returned

Return Values

Table 165–14 GET_PORTS Function Return Values

Parameter	Description
QNAME_LIST	Returns a list of the qualified names of all ports in a service

GET_PROPERTY Function

This procedure returns the value of a particular property on a Call.

Syntax

```
UTL_DBWS.GET_PROPERTY(
   call_handle    CALL,
   key            VARCHAR2)
  RETURN value VARCHAR2;
```

Parameters

Table 165–15 GET_PROPERTY Function Parameters

Parameter	Description
`call_handle`	The the instance of the Call.
`key`	The key for the property (see Using UTL_DBWS on page 165-2)

Return Values

Table 165–16 GET_PROPERTY Function Return Values

Parameter	Description
`value`	Returns the value of a particular property on a Call.

GET_RETURN_TYPE Function

This procedure lists the XML type that is returned by the given Call.

Syntax

```
UTL_DBWS.GET_RETURN_TYPE (
   call_handle    CALL)
 RETURN QNAME;
```

Parameters

Table 165–17 GET_RETURN_TYPE Function Parameters

Parameter	Description
call_handle	The Service instance whose return type is returned.

Return Values

Table 165–18 GET_RETURN_TYPE Function Return Values

Parameter	Description
QNAME	The type that is returned.

GET_SERVICES Function

This function lists the qualified names of the services defined in a WDSL document.

Syntax

```
UTL_DBWS.GET_SERVICES(
   wsdl_document_location  URITYPE)
 RETURN QNAME_LIST;
```

Parameters

Table 165–19 GET_SERVICES Function Parameters

Parameter	Description
wsdl_document_ location	The Service instance whose return type is returned.

Return Values

Table 165–20 GET_SERVICES Function Return Values

Parameter	Description
QNAME_LIST	A list of the qualified names of the services defined in the WSDL document.

INVOKE Function

This procedure invokes a specific operation using a synchronous request-response interaction mode.

Syntax

```
UTL_DBWS.INVOKE(
   call_handle    CALL,
   input_params   ANYDATA_LIST)
 RETURN ANYDATA;
```

Parameters

Table 165–21 INVOKE Function Parameters

Parameter	Description
`call_handle`	The Service instance whose return type is returned.
`input_params`	The input parameters for this invocation.

Return Values

Table 165–22 INVOKE Function Return Values

Parameter	Description
`ANYDATA`	Returns the return value or NULL.

RELEASE_ALL_SERVICES Procedure

This procedure releases all Service instances.

Syntax

```
UTL_DBWS.RELEASE_ALL_SERVICES;
```

RELEASE_CALL Procedure

This procedure releases a particular Call instance.

Syntax

```
UTL_DBWS.RELEASE_CALL(
   call_handle    CALL);
```

Parameters

Table 165–23 RELEASE_CALL Procedure Parameters

Parameter	Description
`call_handle`	The call instance that is to be released.

RELEASE_SERVICE Procedure

This procedure releases a particular Service instance.

Syntax

```
UTL_DBWS.RELEASE_SERVICE (
   service_handle     SERVICE);
```

Parameters

Table 165–24 RELEASE_SERVICE Procedure Parameters

Parameter	Description
service_handle	The call instance that is to be released.

Usage Notes

This will implicitly release all Call instances that have been created for this Service instance.

REMOVE_PROPERTY Procedure

This procedure clears the value of a particular property on a Call.

Syntax

```
UTL_DBWS.REMOVE_PROPERTY(
   call_handle   CALL,
   key           VARCHAR2);
```

Parameters

Table 165–25 REMOVE_PROPERTY Procedure Parameters

Parameter	Description
`call_handle`	The call instance.
`key`	The key for the property (see Using UTL_DBWS on page 165-2).

SET_PROPERTY Procedure

This procedure sets the value of a particular property on a Call.

Syntax

```
UTL_DBWS.SET_PROPERTY(
   call_handle    CALL,
   key            VARCHAR2,
   value          VARCHAR2);
```

Parameters

Table 165–26 SET_PROPERTY Function Parameters

Parameter	Description
`call_handle`	The instance of the Call.
`key`	The key for the property (see Using UTL_DBWS on page 165-2).
`value`	The value for the property.

166

UTL_ENCODE

The UTL_ENCODE package provides functions that encode RAW data into a standard encoded format so that the data can be transported between hosts. You can use UTL_ENCODE functions to encode the body of email text. The package also contains the decode counterpart functions of the encode functions. The functions follow published standards for encoding to accommodate non-Oracle utilities on the sending or receiving ends.

This chapter contains the following topic:

Summary of UTL_ENCODE Subprograms

Table 166–1 UTL_ENCODE Package Subprograms

Subprogram	Description
BASE64_DECODE Function on page 166-3	Reads the base 64-encoded `RAW` input string and decodes it to its original `RAW` value
BASE64_ENCODE Function on page 166-4	Encodes the binary representation of the `RAW` value into base 64 elements and returns it in the form of a `RAW` string
MIMEHEADER_DECODE Function on page 166-5	Decodes a string from mime header format
MIMEHEADER_ENCODE Function on page 166-6	Encodes a string into mime header format
QUOTED_PRINTABLE_ DECODE Function on page 166-7	Reads the `varchar2` quoted printable format input string and decodes it to the corresponding `RAW` string
QUOTED_PRINTABLE_ ENCODE Function on page 166-8	Reads the `RAW` input string and encodes it to the corresponding quoted printable format string
TEXT_DECODE Function on page 166-9	Decodes a character set sensitive text string
TEXT_ENCODE Function on page 166-10	Encodes a character set sensitive text string
UUDECODE Function on page 166-11	Reads the `RAW` uuencode format input string and decodes it to the corresponding `RAW` string
UUENCODE Function on page 166-12	Reads the `RAW` input string and encodes it to the corresponding uuencode format string

BASE64_DECODE Function

This function reads the base 64-encoded RAW input string and decodes it to its original RAW value.

Syntax

```
UTL_ENCODE.BASE64_DECODE (
   r  IN RAW)
RETURN RAW;
```

Pragmas

```
pragma RESTRICT_REFERENCES(base64_decode, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 166–2 BASE64_DECODE Function Parameters

Parameter	Description
r	The RAW string containing base 64-encoded data. There are no defaults or optional parameters.

Return Values

Table 166–3 BASE64_DECODE Function Return Values

Return	Description
RAW	Contains the decoded string

BASE64_ENCODE Function

This function encodes the binary representation of the RAW value into base 64 elements and returns it in the form of a RAW string.

Syntax

```
UTL_ENCODE.BASE64_ENCODE (
   r  IN RAW)
RETURN RAW;
```

Pragmas

```
pragma RESTRICT_REFERENCES(base64_encode, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 166–4 BASE64_ENCODE Function Parameters

Parameter	Description
r	The RAW value to be encoded. There are no defaults or optional parameters.

Return Values

Table 166–5 BASE64_ENCODE Function Return Values

Return	Description
RAW	Contains the encoded base 64 elements

MIMEHEADER_DECODE Function

This function accepts as input an "encoded word" of the form:

```
=?<charset>?<encoding>?<encoded text>?=
=?ISO-8859-1?Q?Here is some encoded text?=
```

The <encoded text> is encapsulated in mime header tags which give the MIMEHEADER_DECODE function information about how to decode the string. The mime header metadata tags are stripped from the input string and the <encoded text> is converted to the base database character set as follows:

- If this is a UTF16 platform, convert the encoded text from UTF16 to ASCII
- If this is an EBCDIC platform, convert the encoded text from EBCDIC to ASCII
- If this is an ASCII or UTF8 platform, no conversion needed

The string is decoded using either quoted-printable or base64 decoding, as specified by the <encoding> metadata tag in the encoded word. The resulting converted and decoded text is returned to the caller as a VARCHAR2 string.

Syntax

```
UTL_ENCODE.MIMEHEADER_DECODE (
   buf    IN   VARCHAR2 CHARACTER SET ANY_CS)
  RETURN data VARCHAR2 CHARACTER SET buf%CHARSET;
```

Parameters

Table 166–6 MIMEHEADER_DECODE Function Parameters

Parameter	Description
buf	The encoded text data with mime header format tags.

Return Values

Table 166–7 MIMEHEADER_DECODE Function Return Values

Return	Description
data	The encoded text data with mime header format tags

Examples

```
v2:=utl_encode.mimeheader_decode('=?ISO-8859-1?Q?Here is some encoded text?=');
```

MIMEHEADER_ENCODE Function

This function accepts as input an "encoded word" of the form:

```
=?<charset>?<encoding>?<encoded text>?=
=?ISO-8859-1?Q?Here is some text?=
```

The `buf` input parameter is the text to be encoded and becomes the `<encoded text>`.

The <encoding> value is either "Q" or "B" for quoted-printable encode or base64 encoding respectively. The ENCODING input parameter accepts as valid values `UTL_ENCODE.QUOTED_PRINTABLE` or `UTL_ENCODE.BASE64` or `NULL`. If `NULL`, quoted-printable encoding is selected as a default value.

The `<charset>` value is specified as the input parameter `encode_charset`. If `NULL`, the database character set is selected as a default value.

The mimeheader encoding process includes conversion of the `buf` input string to the character set specified by the `encode_charset` parameter. The converted string is encoded to either quoted-printable or base64 encoded format. The mime header tags are appended and prepended.

Finally, the string is converted to the base character set of the database:

- If this is a UTF16 platform, convert the encoded text to UTF16
- If this is an EBCDIC platform, convert the encoded text to EBCDIC
- If this is an ASCII or UTF8 platform, no conversion needed.

Syntax

```
UTL_ENCODE.MIMEHEADER_ENCODE (
   buf            IN  VARCHAR2 CHARACTER SET ANY_CS,
   encode_charset IN  VARCHAR2 DEFAULT NULL,
   encoding       IN  PLS_INTEGER DEFAULT NULL)
 RETURN string VARCHAR2 CHARACTER SET buf%CHARSET;
```

Parameters

Table 166–8 MIMEHEADER_ENCODE Function Parameters

Parameter	Description
`buf`	The text data.
`encode_charset`	The target character set.
`encoding`	The encoding format. Valid values are `UTL_ENCODE.BASE64`, `UTL_ENCODE.QUOTED_PRINTABLE` and `NULL`

Return Values

Table 166–9 MIMEHEADER_ENCODE Function Return Values

Return	Description
`string`	A `VARCHAR2` encoded string with mime header format tags.

QUOTED_PRINTABLE_DECODE Function

This function reads the varchar2 quoted printable format input string and decodes it to the corresponding RAW string.

Syntax

```
UTL_ENCODE.QUOTED_PRINTABLE_DECODE (
   r  IN RAW)
RETURN RAW;
```

Pragmas

```
pragma RESTRICT_REFERENCES(quoted_printable_decode, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 166–10 QUOTED_PRINTABLE_DECODE Function Parameters

Parameters	Description
r	The RAW string containing a quoted printable data string. There are no defaults or optional parameters.

Return Values

Table 166–11 QUOTED_PRINTABLE_DECODE Function Return Values

Return	Description
RAW	The decoded string

QUOTED_PRINTABLE_ENCODE Function

This function reads the RAW input string and encodes it to the corresponding quoted printable format string.

Syntax

```
UTL_ENCODE.QUOTED_PRINTABLE_ENCODE (
   r  IN RAW)
RETURN RAW;
```

Pragmas

```
pragma RESTRICT_REFERENCES(quoted_printable_encode, WNDS, RNDS,WNPS, RNPS);
```

Parameters

Table 166–12 QUOTED_PRINTABLE_ENCODE Function Parameters

Parameter	Description
r	The RAW string. There are no defaults or optional parameters.

Return Values

Table 166–13 QUOTED_PRINTABLE_ENCODE Function Return Values

Return	Description
RAW	Contains the quoted printable string

TEXT_DECODE Function

This function converts the input text to the target character set as specified by the encode_charset parameter, if not NULL. The encoded text is converted to the base character set of database, as follows:

- If this is a UTF16 platform, convert the encoded text from UTF16 to ASCII
- If this is an EBCDIC platform, convert the encoded text from EBCDIC to ASCII
- If this is an ASCII or UTF8 platform, no conversion needed

You can decode from either quoted-printable or base64 format, with regard to each encoding parameter. If NULL, quoted-printable is selected as a default decoding format. If encode_charset is not NULL, you convert the string from the specified character set to the database character set. The resulting decoded and converted text string is returned to the caller.

Syntax

```
UTL_ENCODE.TEXT_DECODE(
   buf            IN  VARCHAR2 CHARACTER SET ANY_CS,
   encode_charset IN  VARCHAR2 DEFAULT NULL,
   encoding       IN  PLS_INTEGER DEFAULT NULL)
 RETURN string VARCHAR2 CHARACTER SET buf%CHARSET;
```

Parameters

Table 166–14 TEXT_DECODE Function Parameters

Parameter	Description
buf	The encoded text data.
encode_charset	The source character set.
encoding	The encoding format. Valid values are UTL_ENCODE.BASE64, UTL_ENCODE.QUOTED_PRINTABLE and NULL.

Return Values

Table 166–15 QUOTED_PRINTABLE_ENCODE Function Return Values

Return	Description
string	A VARCHAR2 decoded text string.

Examples

```
v2:=UTL_ENCODE.TEXT_DECODE(
     'Here is some text',
      WE8ISO8859P1,
      UTL_ENCODE.BASE64);
```

TEXT_ENCODE Function

This function converts the input text to the target character set as specified by the `encode_charset` parameter, if not `NULL`. The text is encoded to either base64 or quoted-printable format, as specified by the `encoding` parameter. Quoted-printable is selected as a default if ENCODING is NULL.

The encoded text is converted to the base character set of the database:

- If this is a UTF16 platform, convert the encoded text to UTF16
- If this is an EBCDIC platform, convert the encoded text to EBCDIC
- If this is an ASCII or UTF8 platform, no conversion needed

The resulting encoded and converted text string is returned to the caller.

Syntax

```
UTL_ENCODE.TEXT_ENCODE (
   buf            IN  VARCHAR2 CHARACTER SET ANY_CS,
   encode_charset IN  VARCHAR2 DEFAULT NULL,
   encoding       IN  PLS_INTEGER DEFAULT NULL)
 RETURN string VARCHAR2 CHARACTER SET buf%CHARSET;
```

Parameters

Table 166–16 TEXT_ENCODE Function Parameters

Parameter	Description
`buf`	The text data.
`encode_charset`	The target character set.
`encoding`	The encoding format. Valid values are `UTL_ENCODE.BASE64`, `UTL_ENCODE.QUOTED_PRINTABLE` and `NULL`

Return Values

Table 166–17 TEXT_ENCODE Function Return Values

Return	Description
`string`	A `VARCHAR2` encoded string with mime header format tags.

Examples

```
v2:=utl_encode.text_encode(
   'Here is some text',
   'WE8ISO8859P1',
    UTL_ENCODE.BASE64);
```

UUDECODE Function

This function reads the RAW uuencode format input string and decodes it to the corresponding RAW string. See "UUENCODE Function" on page 166-12 for discussion of the cumulative nature of UUENCODE and UUDECODE for data streams.

Syntax

```
UTL_ENCODE.UUDECODE (
   r  IN RAW)
RETURN RAW;
```

Pragmas

```
pragma RESTRICT_REFERENCES(uudecode, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 166–18 UUDECODE Function Parameters

Parameter	Description
r	The RAW string containing the uuencoded data string. There are no defaults or optional parameters.

Return Values

Table 166–19 UUDECODE Function Return Values

Return	Description
RAW	The decoded RAW string

UUENCODE Function

This function reads the RAW input string and encodes it to the corresponding uuencode format string. The output of this function is cumulative, in that it can be used to encode large data streams, by splitting the data stream into acceptably sized RAW values, encoded, and concatenated into a single encoded string.

Syntax

```
UTL_ENCODE.UUENCODE (
   r          IN RAW,
   type       IN PLS_INTEGER DEFAULT 1,
   filename   IN VARCHAR2 DEFAULT NULL,
   permission IN VARCHAR2 DEFAULT NULL) RETURN RAW;
```

Pragmas

```
pragma RESTRICT_REFERENCES(uuencode, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 166–20 UUENCODE Function Parameters

Parameter	Description
r	RAW string
type	Optional number parameter containing the type of uuencoded output. Options: complete—a defined PL/SQL constant with a value of 1. (default) header_piece ...middle_piece ...end_piece
filename	Optional varchar2 parameter containing the uuencode filename; the default is uuencode.txt
permission	Optional varchar2 parameter containing the permission mode; the default is 0 (a text string zero).

Return Values

Table 166–21 UUENCODE Function Return Values

Return	Description
RAW	Contains the uuencode format string

167

UTL_FILE

With the `UTL_FILE` package, PL/SQL programs can read and write operating system text files. `UTL_FILE` provides a restricted version of operating system stream file I/O.

This chapter contains the following topics:

Using UTL_FILE

Security Model

UTL_FILE is available for both client-side and server-side PL/SQL. Both the client (text I/O) and server implementations are subject to server-side file system permission checking.

In the past, accessible directories for the UTL_FILE functions were specified in the initialization file using the UTL_FILE_DIR parameter. However, UTL_FILE_DIR access is not recommended. It is recommended that you use the CREATE DIRECTORY feature, which replaces UTL_FILE_DIR. Directory objects offer more flexibility and granular control to the UTL_FILE application administrator, can be maintained dynamically (that is, without shutting down the database), and are consistent with other Oracle tools. CREATE DIRECTORY privilege is granted only to SYS and SYSTEM by default.

Note: Use the CREATE DIRECTORY feature instead of UTL_FILE_DIR for directory access verification.

On UNIX systems, the owner of a file created by the FOPEN function is the owner of the shadow process running the instance. Normally, this owner is ORACLE. Files created using FOPEN are always writable and readable using the UTL_FILE subprograms, but non privileged users who need to read these files outside of PL/SQL may need access from a system administrator.

Caution:

- **The privileges needed to access files in a directory object are operating system specific.** UTL_FILE **directory object privileges give you read and write access to all files within the specified directory.**
- **Attempting to apply invalid options will give rise to unpredictable results.**

Types

The contents of `FILE_TYPE` are private to the `UTL_FILE` package. You should not reference or change components of this record.

```
TYPE file_type IS RECORD (
   id       BINARY_INTEGER,
   datatype BINARY_INTEGER);
```

Operational Notes

The file location and file name parameters are supplied to the FOPEN function as separate strings, so that the file location can be checked against the list of accessible directories as specified by the ALL_DIRECTORIES view of accessible directory objects. Together, the file location and name must represent a legal filename on the system, and the directory must be accessible. A subdirectory of an accessible directory is not necessarily also accessible; it too must be specified using a complete path name matching an ALL_DIRECTORIES object.

UTL_FILE implicitly interprets line terminators on read requests, thereby affecting the number of bytes returned on a GET_LINE call. For example, the len parameter of UTL_FILE.GET_LINE specifies the requested number of bytes of character data. The number of bytes actually returned to the user will be the lesser of:

- The GET_LINE len parameter, or
- The number of bytes until the next line terminator character, or
- The max_linesize parameter specified by UTL_FILE.FOPEN

The FOPEN max_linesize parameter must be a number in the range 1 and 32767. If unspecified, Oracle supplies a default value of 1024. The GET_LINE len parameter must be a number in the range 1 and 32767. If unspecified, Oracle supplies the default value of max_linesize. If max_linesize and len are defined to be different values, then the lesser value takes precedence.

UTL_FILE.GET_RAW ignores line terminators and returns the actual number of bytes requested by the GET_RAW len parameter.

When data encoded in one character set is read and Globalization Support is told (such as by means of NLS_LANG) that it is encoded in another character set, the result is indeterminate. If NLS_LANG is set, it should be the same as the database character set.

Rules and Limits

Operating system-specific parameters, such as C-shell environment variables under UNIX, cannot be used in the file location or file name parameters.

UTL_FILE I/O capabilities are similar to standard operating system stream file I/O (OPEN, GET, PUT, CLOSE) capabilities, but with some limitations. For example, you call the FOPEN function to return a file handle, which you use in subsequent calls to GET_LINE or PUT to perform stream I/O to a file. When file I/O is done, you call FCLOSE to complete any output and free resources associated with the file.

Note: The UTL_FILE package is similar to the client-side TEXT_IO package currently provided by Oracle Procedure Builder. Restrictions for a server implementation require some API differences between UTL_FILE and TEXT_IO. In PL/SQL file I/O, errors are returned using PL/SQL exceptions.

Exceptions

Table 167–1 UTL_FILE Package Exceptions

Exception Name	Description
INVALID_PATH	File location is invalid.
INVALID_MODE	The open_mode parameter in FOPEN is invalid.
INVALID_FILEHANDLE	File handle is invalid.
INVALID_OPERATION	File could not be opened or operated on as requested.
READ_ERROR	Operating system error occurred during the read operation.
WRITE_ERROR	Operating system error occurred during the write operation.
INTERNAL_ERROR	Unspecified PL/SQL error
CHARSETMISMATCH	A file is opened using FOPEN_NCHAR, but later I/O operations use nonchar functions such as PUTF or GET_LINE.
FILE_OPEN	The requested operation failed because the file is open.
INVALID_ MAXLINESIZE	The MAX_LINESIZE value for FOPEN() is invalid; it should be within the range 1 to 32767.
INVALID_FILENAME	The filename parameter is invalid.
ACCESS_DENIED	Permission to access to the file location is denied.
INVALID_OFFSET	Causes of the INVALID_OFFSET exception: ■ ABSOLUTE_OFFSET = NULL and RELATIVE_OFFSET = NULL, or ■ ABSOLUTE_OFFSET < 0, or ■ Either offset caused a seek past the end of the file
DELETE_FAILED	The requested file delete operation failed.
RENAME_FAILED	The requested file rename operation failed.

Procedures in UTL_FILE can also raise predefined PL/SQL exceptions such as NO_DATA_FOUND or VALUE_ERROR.

Examples

Example 1

Note: The examples are UNIX-specific.

Given the following:

```
SQL> CREATE DIRECTORY log_dir AS '/appl/gl/log';
SQL> GRANT READ ON DIRECTORY log_dir TO DBA;

SQL> CREATE DIRECTORY out_dir AS '/appl/gl/user'';
SQL> GRANT READ ON DIRECTORY user_dir TO PUBLIC;
```

The following file locations and filenames are valid and accessible as follows:

File Location	Filename	Accessible By
/appl/gl/log	L12345.log	Users with DBA privilege
/appl/gl/user	u12345.tmp	All users

The following file locations and filenames are invalid:

File Location	Filename	Invalid Because
/appl/gl/log/backup	L12345.log	# subdirectories are not accessible
/APPL/gl/log	L12345.log	# directory strings must follow case sensitivity rules as required by the O/S
/appl/gl/log	backup/L1234.log	# filenames may not include portions of directory paths
/user/tmp	L12345.log	# no corresponding CREATE DIRECTORY command has been issued

Example 2

```
DECLARE
  V1 VARCHAR2(32767);
  F1 UTL_FILE.FILE_TYPE;
BEGIN
  -- In this example MAX_LINESIZE is less than GET_LINE's length request
  -- so the number of bytes returned will be 256 or less if a line terminator is
seen.
  F1 := UTL_FILE.FOPEN('MYDIR','MYFILE','R',256);
  UTL_FILE.GET_LINE(F1,V1,32767);
  UTL_FILE.FCLOSE(F1);

  -- In this example, FOPEN's MAX_LINESIZE is NULL and defaults to 1024,
  -- so the number of bytes returned will be 1024 or less if a line terminator is
seen.
  F1 := UTL_FILE.FOPEN('MYDIR','MYFILE','R');
  UTL_FILE.GET_LINE(F1,V1,32767);
  UTL_FILE.FCLOSE(F1);
```

```
  -- In this example, GET_LINE doesn't specify a number of bytes, so it defaults
to
  -- the same value as FOPEN's MAX_LINESIZE which is NULL in this case and
defaults to 1024.
  -- So the number of bytes returned will be 1024 or less if a line terminator is
seen.
  F1 := UTL_FILE.FOPEN('MYDIR','MYFILE','R');
  UTL_FILE.GET_LINE(F1,V1);
  UTL_FILE.FCLOSE(F1);
END;
```

Summary of UTL_FILE Subprograms

Table 167–2 UTL_FILE Subprograms

Subprogram	Description
FCLOSE Procedure on page 167-12	Closes a file
FCLOSE_ALL Procedure on page 167-13	Closes all open file handles
FCOPY Procedure on page 167-14	Copies a contiguous portion of a file to a newly created file
FFLUSH Procedure on page 167-15	Physically writes all pending output to a file
FGETATTR Procedure on page 167-16	Reads and returns the attributes of a disk file
FGETPOS Function on page 167-17	Returns the current relative offset position within a file, in bytes
FOPEN Function on page 167-18	Opens a file for input or output
FOPEN_NCHAR Function on page 167-20	Opens a file in Unicode for input or output
FREMOVE Procedure on page 167-21	Deletes a disk file, assuming that you have sufficient privileges
FRENAME Procedure on page 167-22	Renames an existing file to a new name, similar to the UNIX `mv` function
FSEEK Procedure on page 167-23	Adjusts the file pointer forward or backward within the file by the number of bytes specified
GET_LINE Procedure on page 167-24	Reads text from an open file
GET_LINE_NCHAR Procedure on page 167-25	Reads text in Unicode from an open file
GET_RAW Function on page 167-26	Reads a `RAW` string value from a file and adjusts the file pointer ahead by the number of bytes read
IS_OPEN Function on page 167-27	Determines if a file handle refers to an open file
NEW_LINE Procedure on page 167-28	Writes one or more operating system-specific line terminators to a file
PUT Procedure on page 167-29	Writes a string to a file
PUT_LINE Procedure on page 167-30	Writes a line to a file, and so appends an operating system-specific line terminator
PUT_LINE_NCHAR Procedure on page 167-31	Writes a Unicode line to a file
PUT_NCHAR Procedure on page 167-32	Writes a Unicode string to a file
PUTF Procedure on page 167-33	A `PUT` procedure with formatting

Table 167–2 (Cont.) UTL_FILE Subprograms

Subprogram	Description
PUTF_NCHAR Procedure on page 167-35	A `PUT_NCHAR` procedure with formatting, and writes a Unicode string to a file, with formatting
PUT_RAW Function on page 167-36	Accepts as input a `RAW` data value and writes the value to the output buffer

FCLOSE Procedure

This procedure closes an open file identified by a file handle.

Syntax

```
UTL_FILE.FCLOSE (
   file IN OUT FILE_TYPE);
```

Parameters

Table 167–3 FCLOSE Procedure Parameters

Parameter	Description
`file`	Active file handle returned by an `FOPEN` or `FOPEN_NCHAR` call.

Usage Notes

If there is buffered data yet to be written when `FCLOSE` runs, then you may receive a `WRITE_ERROR` exception when closing a file.

Exceptions

```
WRITE_ERROR
INVALID_FILEHANDLE
```

FCLOSE_ALL Procedure

This procedure closes all open file handles for the session. This should be used as an emergency cleanup procedure, for example, when a PL/SQL program exits on an exception.

Syntax

```
UTL_FILE.FCLOSE_ALL;
```

Usage Notes

Note: `FCLOSE_ALL` does not alter the state of the open file handles held by the user. This means that an `IS_OPEN` test on a file handle after an `FCLOSE_ALL` call still returns `TRUE`, even though the file has been closed. No further read or write operations can be performed on a file that was open before an `FCLOSE_ALL`.

Exceptions

`WRITE_ERROR`

FCOPY Procedure

This procedure copies a contiguous portion of a file to a newly created file. By default, the whole file is copied if the `start_line` and `end_line` parameters are omitted. The source file is opened in read mode. The destination file is opened in write mode. A starting and ending line number can optionally be specified to select a portion from the center of the source file for copying.

Syntax

```
UTL_FILE.FCOPY (
   location   IN VARCHAR2,
   filename   IN VARCHAR2,
   dest_dir   IN VARCHAR2,
   dest_file  IN VARCHAR2,
   start_line IN PLS_INTEGER DEFAULT 1,
   end_line   IN PLS_INTEGER DEFAULT NULL);
```

Parameters

Table 167–4 FCOPY Procedure Parameters

Parameters	Description
`location`	The directory location of the source file, a `DIRECTORY_NAME` from the `ALL_DIRECTORIES` view (case sensitive)
`filename`	The source file to be copied
`dest_dir`	The destination directory where the destination file is created.
`dest_file`	The destination file created from the source file.
`start_line`	The line number at which to begin copying. The default is `1` for the first line.
`end_line`	The line number at which to stop copying. The default is `NULL`, signifying end of file.

FFLUSH Procedure

FFLUSH physically writes pending data to the file identified by the file handle. Normally, data being written to a file is buffered. The FFLUSH procedure forces the buffered data to be written to the file. The data must be terminated with a newline character.

Flushing is useful when the file must be read while still open. For example, debugging messages can be flushed to the file so that they can be read immediately.

Syntax

```
UTL_FILE.FFLUSH (
   file  IN FILE_TYPE);
invalid_maxlinesize  EXCEPTION;
```

Parameters

Table 167–5 FFLUSH Procedure Parameters

Parameters	Description
file	Active file handle returned by an FOPEN or FOPEN_NCHAR call.

Exceptions

```
INVALID_FILEHANDLE
INVALID_OPERATION
WRITE_ERROR
```

FGETATTR Procedure

This procedure reads and returns the attributes of a disk file.

Syntax

```
UTL_FILE.FGETATTR(
   location    IN VARCHAR2,
   filename    IN VARCHAR2,
   exists      OUT BOOLEAN,
   file_length OUT NUMBER,
   blocksize   OUT NUMBER);
```

Parameters

Table 167–6 FGETATTR Procedure Parameters

Parameters	Description
`location`	Directory location of the source file, a `DIRECTORY_NAME` from the `ALL_DIRECTORIES` view (case sensitive).
`filename`	The name of the file to be examined.
`exists`	A `BOOLEAN` for whether or not the file exists.
`file_length`	The length of the file in bytes. `NULL` if file does not exist.
`blocksize`	The file system block size in bytes. `NULL` if the file does not exist.

FGETPOS Function

This function returns the current relative offset position within a file, in bytes.

Syntax

```
UTL_FILE.FGETPOS (
   fileid IN file_type)
 RETURN PLS_INTEGER;
```

Parameters

Table 167–7 FGETPOS Parameters

Parameters	Description
`fileid`	The directory location of the source file

Return Values

`FGETPOS` returns the relative offset position for an open file, in bytes. It raises an exception if the file is not open. It returns `0` for the beginning of the file.

FOPEN Function

This function opens a file. You can specify the maximum line size and have a maximum of 50 files open simultaneously. See also FOPEN_NCHAR Function on page 167-20.

Syntax

```
UTL_FILE.FOPEN (
   location     IN VARCHAR2,
   filename     IN VARCHAR2,
   open_mode    IN VARCHAR2,
   max_linesize IN BINARY_INTEGER)
  RETURN file_type;
```

Parameters

Table 167–8 FOPEN Function Parameters

Parameter	Description
location	Directory location of file. This string is a directory object name and is case sensitive. The default is uppercase. Read privileges must be granted on this directory object for the UTL_FILE user to run FOPEN.
filename	File name, including extension (file type), without directory path. If a directory path is given as a part of the filename, it is ignored by FOPEN. On Unix, the filename cannot end with /.
open_mode	Specifies how the file is opened. Modes include: r -- read text w -- write text a -- append text rb -- read byte mode wb -- write byte mode ab -- append byte mode If you try to open a file specifying 'a' or 'ab' for open_mode but the file does not exist, the file is created in write mode.
max_linesize	Maximum number of characters for each line, including the newline character, for this file (minimum value 1, maximum value 32767). If unspecified, Oracle supplies a default value of 1024.

Return Values

FOPEN returns a file handle, which must be passed to all subsequent procedures that operate on that file. The specific contents of the file handle are private to the UTL_FILE package, and individual components should not be referenced or changed by the UTL_FILE user.

Table 167–9 FOPEN Function Return Values

Return	Description
file_type	Handle to open file.

Usage Notes

The file location and file name parameters must be supplied to the FOPEN function as quoted strings so that the file location can be checked against the list of accessible directories as specified by the ALL_DIRECTORIES view of accessible directory objects.

Exceptions

```
INVALID_PATH: File location or name was invalid.
INVALID_MODE: The open_mode string was invalid.
INVALID_OPERATION: File could not be opened as requested.
INVALID_MAXLINESIZE: Specified max_linesize is too large or too small.
```

FOPEN_NCHAR Function

This function opens a file in Unicode for input or output, with the maximum line size specified. You can have a maximum of 50 files open simultaneously. With this function, you can read or write a text file in Unicode instead of in the database charset. See also FOPEN Function on page 167-18.

Syntax

```
UTL_FILE.FOPEN_NCHAR (
   location     IN VARCHAR2,
   filename     IN VARCHAR2,
   open_mode    IN VARCHAR2,
   max_linesize IN BINARY_INTEGER)
RETURN file_type;
```

Parameters

Table 167–10 FOPEN_NCHAR Function Parameters

Parameter	Description
`location`	Directory location of file.
`filename`	File name (including extension).
`open_mode`	Open mode (r,w,a,rb,wb,ab).
`max_linesize`	Maximum number of characters for each line, including the newline character, for this file. (minimum value 1, maximum value 32767).

FREMOVE Procedure

This procedure deletes a disk file, assuming that you have sufficient privileges.

Syntax

```
UTL_FILE.FREMOVE (
   location IN VARCHAR2,
   filename IN VARCHAR2);
```

Parameters

Table 167–11 FREMOVE Procedure Parameters

Parameters	Description
`location`	The directory location of the file, a `DIRECTORY_NAME` from `ALL_DIRECTORIES` (case sensitive)
`filename`	The name of the file to be deleted

Usage Notes

The `FREMOVE` procedure does not verify privileges before deleting a file. The O/S verifies file and directory permissions. An exception is returned on failure.

FRENAME Procedure

This procedure renames an existing file to a new name, similar to the UNIX `mv` function.

Syntax

```
UTL_FILE.FRENAME (
   location  IN VARCHAR2,
   filename  IN VARCHAR2,
   dest_dir  IN VARCHAR2,
   dest_file IN VARCHAR2,
   overwrite IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 167–12 FRENAME Procedure Parameters

Parameters	Description
`location`	The directory location of the source file, a `DIRECTORY_NAME` from the `ALL_DIRECTORIES` view (case sensitive).
`filename`	The source file to be renamed.
`dest_dir`	The destination directory of the destination file, a `DIRECTORY_NAME` from the `ALL_DIRECTORIES` view (case sensitive).
`dest_file`	The new name of the file.
`overwrite`	The default is `FALSE`.

Usage Notes

Permission on both the source and destination directories must be granted. You can use the `overwrite` parameter to specify whether or not to overwrite a file if one exists in the destination directory. The default is `FALSE` for no overwrite.

FSEEK Procedure

This procedure adjusts the file pointer forward or backward within the file by the number of bytes specified.

Syntax

```
UTL_FILE.FSEEK (
   fid             IN utl_file.file_type,
   absolute_offset IN PL_INTEGER DEFAULT NULL,
   relative_offset IN PLS_INTEGER DEFAULT NULL);
```

Parameters

Table 167–13 FSEEK Procedure Parameters

Parameters	Description
`fid`	The file ID.
`absolute_offset`	The absolute location to which to seek; default = `NULL`
`relative_offset`	The number of bytes to seek forward or backward; positive = forward, negative integer = backward, zero = current position, default = `NULL`

Usage Notes

Using `FSEEK`, you can read previous lines in the file without first closing and reopening the file. You must know the number of bytes by which you want to navigate.

If `relative_offset`, the procedure seeks forward. If `relative_offset` > 0, or backward, if `relative_offset` < 0, the procedure seeks through the file by the number of `relative_offset` bytes specified.

If the beginning of the file is reached before the number of bytes specified, then the file pointer is placed at the beginning of the file. If the end of the file is reached before the number of bytes specified, then an `INVALID_OFFSET` error is raised.

If `absolute_offset`, the procedure seeks to an absolute location specified in bytes.

GET_LINE Procedure

This procedure reads text from the open file identified by the file handle and places the text in the output buffer parameter. Text is read up to, but not including, the line terminator, or up to the end of the `len` parameter. It cannot exceed the `max_linesize` specified in `FOPEN`.

Syntax

```
UTL_FILE.GET_LINE (
   file        IN  FILE_TYPE,
   buffer      OUT VARCHAR2,
   len         IN  PLS_INTEGER DEFAULT NULL);
```

Parameters

Table 167–14 GET_LINE Procedure Parameters

Parameters	Description
`file`	Active file handle returned by an `FOPEN` call. The file must be open for reading (mode `r`); otherwise an `INVALID_OPERATION` exception is raised.
`buffer`	Data buffer to receive the line read from the file.
`len`	The number of bytes read from the file. Default is `NULL`. If `NULL`, Oracle supplies the value of max_linesize.

Usage Notes

If the line does not fit in the buffer, a `VALUE_ERROR` exception is raised. If no text was read due to end of file, the `NO_DATA_FOUND` exception is raised. If the file is opened for byte mode operations, the `INVALID_OPERATION` exception is raised.

Because the line terminator character is not read into the buffer, reading blank lines returns empty strings.

The maximum size of the `buffer` parameter is 32767 bytes unless you specify a smaller size in `FOPEN`.

If unspecified, Oracle supplies a default value of 1024. See also "GET_LINE_NCHAR Procedure" on page 167-25.

Exceptions

```
INVALID_FILEHANDLE
INVALID_OPERATION
READ_ERROR
NO_DATA_FOUND
VALUE_ERROR
```

GET_LINE_NCHAR Procedure

This procedure reads text from the open file identified by the file handle and places the text in the output buffer parameter. With this function, you can read a text file in Unicode instead of in the database charset. See also GET_LINE Procedure on page 167-24.

Syntax

```
UTL_FILE.GET_LINE_NCHAR (
   file        IN  FILE_TYPE,
   buffer      OUT NVARCHAR2,
   len         IN  PLS_INTEGER DEFAULT NULL);
```

Parameters

Table 167–15 GET_LINE_NCHAR Procedure Parameters

Parameters	Description
file	Active file handle returned by an FOPEN_NCHAR call. The file must be open for reading (mode r). If the file is opened by FOPEN instead of FOPEN_NCHAR, a CHARSETMISMATCH exception is raised.
buffer	Data buffer to receive the line read from the file.
len	The number of bytes read from the file. Default is NULL. If NULL, Oracle supplies the value of max_linesize.

GET_RAW Function

This function reads a RAW string value from a file and adjusts the file pointer ahead by the number of bytes read. UTL_FILE.GET_RAW ignores line terminators and returns the actual number of bytes requested by the GET_RAW len parameter.

Syntax

```
UTL_FILE.GET_RAW (
   fid  IN  utl_file.file_type,
   r    OUT NOCOPY RAW,
   len  IN  PLS_INTEGER DEFAULT NULL);
```

Parameters

Table 167–16 GET_RAW Procedure Parameters

Parameters	Description
fid	The file ID.
r	The RAW data.
len	The number of bytes read from the file. Default is NULL. If NULL, len is assumed to be the maximum length of RAW.

IS_OPEN Function

This function tests a file handle to see if it identifies an open file. IS_OPEN reports only whether a file handle represents a file that has been opened, but not yet closed. It does not guarantee that there will be no operating system errors when you attempt to use the file handle.

Syntax

```
UTL_FILE.IS_OPEN (
   file  IN FILE_TYPE)
  RETURN BOOLEAN;
```

Parameters

Table 167–17 IS_OPEN Function Parameters

Parameter	Description
file	Active file handle returned by an FOPEN or FOPEN_NCHAR call.

Return Values

TRUE or FALSE

NEW_LINE Procedure

This procedure writes one or more line terminators to the file identified by the input file handle. This procedure is separate from PUT because the line terminator is a platform-specific character or sequence of characters.

Syntax

```
UTL_FILE.NEW_LINE (
   file     IN FILE_TYPE,
   lines    IN NATURAL := 1);
```

Parameters

Table 167–18 NEW_LINE Procedure Parameters

Parameters	Description
file	Active file handle returned by an FOPEN or FOPEN_NCHAR call.
lines	Number of line terminators to be written to the file.

Exceptions

```
INVALID_FILEHANDLE
INVALID_OPERATION
WRITE_ERROR
```

PUT Procedure

PUT writes the text string stored in the buffer parameter to the open file identified by the file handle. The file must be open for write operations. No line terminator is appended by PUT; use NEW_LINE to terminate the line or use PUT_LINE to write a complete line with a line terminator. See also "PUT_NCHAR Procedure" on page 167-32.

Syntax

```
UTL_FILE.PUT (
   file      IN FILE_TYPE,
   buffer    IN VARCHAR2);
```

Parameters

Table 167–19 PUT Procedure Parameters

Parameters	Description
file	Active file handle returned by an FOPEN_NCHAR call. The file must be open for writing.
buffer	Buffer that contains the text to be written to the file. You must have opened the file using mode w or mode a; otherwise, an INVALID_OPERATION exception is raised.

Usage Notes

The maximum size of the buffer parameter is 32767 bytes unless you specify a smaller size in FOPEN. If unspecified, Oracle supplies a default value of 1024. The sum of all sequential PUT calls cannot exceed 32767 without intermediate buffer flushes.

Exceptions

```
INVALID_FILEHANDLE
INVALID_OPERATION
WRITE_ERROR
```

PUT_LINE Procedure

This procedure writes the text string stored in the buffer parameter to the open file identified by the file handle. The file must be open for write operations. PUT_LINE terminates the line with the platform-specific line terminator character or characters.

See also "PUT_LINE_NCHAR Procedure" on page 167-31.

Syntax

```
UTL_FILE.PUT_LINE (
   file      IN FILE_TYPE,
   buffer    IN VARCHAR2,
   autoflush IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 167–20 PUT_LINE Procedure Parameters

Parameters	Description
file	Active file handle returned by an FOPEN call.
buffer	Text buffer that contains the lines to be written to the file.
autoflush	Flushes the buffer to disk after the WRITE.

Usage Notes

The maximum size of the buffer parameter is 32767 bytes unless you specify a smaller size in FOPEN. If unspecified, Oracle supplies a default value of 1024. The sum of all sequential PUT calls cannot exceed 32767 without intermediate buffer flushes.

Exceptions

```
INVALID_FILEHANDLE
INVALID_OPERATION
WRITE_ERROR
```

PUT_LINE_NCHAR Procedure

This procedure writes the text string stored in the buffer parameter to the open file identified by the file handle. With this function, you can write a text file in Unicode instead of in the database charset. See also "PUT_LINE Procedure" on page 167-30.

Syntax

```
UTL_FILE.PUT_LINE_NCHAR (
   file    IN FILE_TYPE,
   buffer  IN NVARCHAR2);
```

Parameters

Table 167–21 PUT_LINE_NCHAR Procedure Parameters

Parameters	Description
`file`	Active file handle returned by an `FOPEN_NCHAR` call. The file must be open for writing.
`buffer`	Text buffer that contains the lines to be written to the file.

Usage Notes

The maximum size of the `buffer` parameter is 32767 bytes unless you specify a smaller size in `FOPEN`. If unspecified, Oracle supplies a default value of 1024. The sum of all sequential `PUT` calls cannot exceed 32767 without intermediate buffer flushes.

PUT_NCHAR Procedure

This procedure writes the text string stored in the buffer parameter to the open file identified by the file handle. With this function, you can write a text file in Unicode instead of in the database charset. See also "PUT Procedure" on page 167-29.

Syntax

```
UTL_FILE.PUT_NCHAR (
   file      IN FILE_TYPE,
   buffer    IN NVARCHAR2);
```

Parameters

Table 167–22 PUT_NCHAR Procedure Parameters

Parameters	Description
`file`	Active file handle returned by an `FOPEN_NCHAR` call. If the file is opened by `FOPEN` instead of `FOPEN_NCHAR`, a `CHARSETMISMATCH` exception is raised.
`buffer`	Buffer that contains the text to be written to the file. You must have opened the file using mode `w` or mode `a`; otherwise, an `INVALID_OPERATION` exception is raised.

Usage Notes

The maximum size of the `buffer` parameter is 32767 bytes unless you specify a smaller size in `FOPEN`. If unspecified, Oracle supplies a default value of 1024. The sum of all sequential `PUT` calls cannot exceed 32767 without intermediate buffer flushes.

PUTF Procedure

This procedure is a formatted PUT procedure. It works like a limited printf(). See also "PUTF_NCHAR Procedure" on page 167-35.

Syntax

```
UTL_FILE.PUTF (
   file    IN FILE_TYPE,
   format  IN VARCHAR2,
   [arg1   IN VARCHAR2  DEFAULT NULL,
   . . .
   arg5    IN VARCHAR2  DEFAULT NULL]);
```

Parameters

Table 167–23 PUTF Procedure Parameters

Parameters	Description
file	Active file handle returned by an FOPEN call.
format	Format string that can contain text as well as the formatting characters \n and %s.
arg1..arg5	From one to five operational argument strings. Argument strings are substituted, in order, for the %s formatters in the format string. If there are more formatters in the format parameter string than there are arguments, then an empty string is substituted for each %s for which there is no argument.

Usage Notes

The format string can contain any text, but the character sequences %s and \n have special meaning.

Character Sequence	Meaning
%s	Substitute this sequence with the string value of the next argument in the argument list.
\n	Substitute with the appropriate platform-specific line terminator.

Examples

The following example writes the lines:

```
Hello, world!
I come from Zork with greetings for all earthlings.
```

```
my_world  varchar2(4) := 'Zork';
...
PUTF(my_handle, 'Hello, world!\nI come from %s with %s.\n',
                my_world,
                'greetings for all earthlings');
```

If there are more %s formatters in the format parameter than there are arguments, then an empty string is substituted for each %s for which there is no matching argument.

Exceptions

```
INVALID_FILEHANDLE
INVALID_OPERATION
WRITE_ERROR
```

PUTF_NCHAR Procedure

This procedure is a formatted PUT_NCHAR procedure. Using PUTF_NCHAR, you can write a text file in Unicode instead of in the database charset.

Syntax

```
UTL_FILE.PUTF_NCHAR (
   file    IN FILE_TYPE,
   format  IN NVARCHAR2,
   [arg1   IN NVARCHAR2  DEFAULT NULL,
   . . .
   arg5    IN NVARCHAR2  DEFAULT NULL]);
```

Parameters

Table 167–24 PUTF_NCHAR Procedure Parameters

Parameters	Description
file	Active file handle returned by an FOPEN_NCHAR call. The file must be open for reading (mode r). If the file is opened by FOPEN instead of FOPEN_NCHAR, a CHARSETMISMATCH exception is raised.
format	Format string that can contain text as well as the formatting characters \n and %s.
arg1..arg5	From one to five operational argument strings. Argument strings are substituted, in order, for the %s formatters in the format string. If there are more formatters in the format parameter string than there are arguments, then an empty string is substituted for each %s for which there is no argument.

Usage Notes

The maximum size of the buffer parameter is 32767 bytes unless you specify a smaller size in FOPEN. If unspecified, Oracle supplies a default value of 1024. The sum of all sequential PUT calls cannot exceed 32767 without intermediate buffer flushes.

PUT_RAW Function

This function accepts as input a RAW data value and writes the value to the output buffer.

Syntax

```
UTL_FILE.PUT_RAW (
   fid       IN utl_file.file_type,
   r         IN RAW,
   autoflush IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 167–25 PUT_RAW Procedure Parameters

Parameters	Description
fid	The file ID.
r	The RAW data written to the buffer.
autoflush	If TRUE, performs a flush after writing the value to the output buffer; default is FALSE.

Usage Notes

You can request an automatic flush of the buffer by setting the third argument to TRUE.

The maximum size of the buffer parameter is 32767 bytes unless you specify a smaller size in FOPEN. If unspecified, Oracle supplies a default value of 1024. The sum of all sequential PUT calls cannot exceed 32767 without intermediate buffer flushes.

168

UTL_HTTP

The `UTL_HTTP` package makes Hypertext Transfer Protocol (HTTP) callouts from SQL and PL/SQL. You can use it to access data on the Internet over HTTP.

When the package fetches data from a Web site using HTTPS, it requires Oracle Wallet Manager to set up an Oracle wallet. Non-HTTPS fetches do not require an Oracle wallet.

See Also:

- Chapter 179, "UTL_URL"
- Chapter 177, "UTL_SMTP"
- *Oracle Database Advanced Security Administrator's Guide* for more information on Wallet Manager

This chapter contains the following topics:

Using UTL_HTTP

This section contains topics which relate to using the `UTL_HTTP` package.

Overview

With the UTL_HTTP package, you can write PL/SQL programs that communicate with Web (HTTP) servers. And UTL_HTTP contains a function that can be used in SQL queries.

The package also supports HTTP over the Secured Socket Layer protocol (SSL), also known as HTTPS, directly or through an HTTP proxy.

Other Internet-related data-access protocols (such as the File Transfer Protocol (FTP) or the Gopher protocol) are also supported using an HTTP proxy server that supports those protocols.

Constants

The UTL_HTTP package uses the constants shown in following tables.

- UTL_HTTP Constants - HTTP Versions
- UTL_HTTP Constants - Default Ports
- UTL_HTTP Constants - HTTP 1.1 Status Codes

Table 168–1 UTL_HTTP Constants - HTTP Versions

Name	Type	Value	Description
HTTP_VERSION_1_0	VARCHAR2(10)	'HTTP /1.0'	Denotes HTTP version 1.0 that can be used in the function BEGIN_REQUEST.
HTTP_VERSION_1	VARCHAR2(10)	'HTTP /1.1'	Denotes HTTP version 1.1 that can be used in the function BEGIN_REQUEST.

Table 168–2 UTL_HTTP Constants - Default Ports

Name	Type	Value	Description
DEFAULT_HTTP_PORT	PLS_INTEGER	80	The default TCP/IP port (80) at which a Web server or proxy server listens
DEFAULT_HTTPS_PORT	PLS_INTEGER	443	The default TCP/IP port (443) at which an HTTPS Web server listens

Table 168–3 UTL_HTTP Constants - HTTP 1.1 Status Codes

Name	Type	Value	Description
HTTP_CONTINUE	PLS_INTEGER	100	The client should continue with its request. This interim response is used to inform the client that the initial part of the request has been received and has not yet been rejected by the server.
HTTP_SWITCHING_ PROTOCOLS	PLS_INTEGER	101	The server understands and is willing to comply with the client's request, via the Upgrade message header field, for a change in the application protocol being used on this connection. The server will switch protocols to those defined by the response's Upgrade header field immediately after the empty line which terminates the 101 response.
HTTP_OK	PLS_INTEGER	200	The request has succeeded. The information returned with the response is dependent on the method used in the request
HTTP_CREATED CONSTANT	PLS_INTEGER	201	The request has been fulfilled and resulted in a new resource being created.
HTTP_ACCEPTED	PLS_INTEGER	202	The request has been accepted for processing, but the processing has not been completed. The request might or might not eventually be acted upon, as it might be disallowed when processing actually takes place.

Table 168–3 (Cont.) UTL_HTTP Constants - HTTP 1.1 Status Codes

Name	Type	Value	Description
HTTP_NON_ AUTHORITATIVE_INFO	PLS_INTEGER	203	The returned metainformation in the entity-header is not the definitive set as available from the origin server, but is gathered from a local or a third-party copy.
HTTP_NO_CONTENT	PLS_INTEGER	204	The server has fulfilled the request but does not need to return an entity-body, and might want to return updated metainformation.
HTTP_RESET_CONTENT	PLS_INTEGER	205	The server has fulfilled the request and the user agent should reset the document view which caused the request to be sent. The response must not include an entity.
HTTP_PARTIAL_ CONTENT	PLS_INTEGER	206	The server has fulfilled the partial GET request for the resource.
HTTP_MULTIPLE_ CHOICES	PLS_INTEGER	300	The requested resource corresponds to any one of a set of representations, each with its own specific location, and agent-driven negotiation information is being provided so that the user (or user agent) can select a preferred representation and redirect its request to that location.
HTTP_MOVED_ PERMANENTLY	PLS_INTEGER	301	The requested resource has been assigned a new permanent URI and any future references to this resource should use one of the returned URIs.
HTTP_FOUND CONSTANT	PLS_INTEGER	302	The requested resource resides temporarily under a different URI.
HTTP_SEE_OTHER	PLS_INTEGER	303	The response to the request can be found under a different URI and should be retrieved using a GET method on that resource.
HTTP_NOT_MODIFIED	PLS_INTEGER	304	If the client has performed a conditional GET request and access is allowed, but the document has not been modified, the server responds with this status code.
HTTP_USE_PROXY	PLS_INTEGER	305	The requested resource must be accessed through the proxy given by the Location field. The Location field gives the URI of the proxy.
HTTP_TEMPORARY_ REDIRECT	PLS_INTEGER	307	The requested resource resides temporarily under a different URI.
HTTP_BAD_REQUEST	PLS_INTEGER	400	The request could not be understood by the server due to malformed syntax.
HTTP_UNAUTHORIZED	PLS_INTEGER	401	The request requires user authentication. The client may repeat the request with a suitable Authorization header field. If the request already included Authorization credentials, then the 401 response indicates that authorization has been refused for those credentials.
HTTP_PAYMENT_ REQUIRED	PLS_INTEGER	402	This code is reserved for future use.

Table 168–3 (Cont.) UTL_HTTP Constants - HTTP 1.1 Status Codes

Name	Type	Value	Description
HTTP_FORBIDDEN	PLS_INTEGER	403	The server understood the request, but is refusing to fulfill it.
HTTP_NOT_FOUND	PLS_INTEGER	404	The server has not found anything matching the Request-URI.
HTTP_NOT_ ACCEPTABLE	PLS_INTEGER	406	The resource identified by the request is only capable of generating response entities which have content characteristics not acceptable according to the accept headers sent in the request.
HTTP_PROXY_AUTH_ REQUIRED	PLS_INTEGER	407	This code is similar to 401 (Unauthorized), but indicates that the client must first authenticate itself with the proxy.
HTTP_REQUEST_TIME_ OUT	PLS_INTEGER	408	The client did not produce a request within the time that the server was prepared to wait.
HTTP_CONFLICT	PLS_INTEGER	409	The request could not be completed due to a conflict with the current state of the resource.
HTTP_GONE	PLS_INTEGER	410	The requested resource is no longer available at the server and no forwarding address is known.
HTTP_LENGTH_ REQUIRED	PLS_INTEGER	411	The server refuses to accept the request without a defined Content-Length.
HTTP_PRECONDITION_ FAILED	PLS_INTEGER	412	The precondition given in one or more of the request-header fields evaluated to false when it was tested on the server.
HTTP_REQUEST_ ENTITY_TOO_LARGE CONSTANT	PLS_INTEGER	413	The server is refusing to process a request because the request entity is larger than the server is willing or able to process.
HTTP_REQUEST_URI_ TOO_LARGE	PLS_INTEGER	414	The server is refusing to service the request because the Request-URI is longer than the server is willing to interpret.
HTTP_UNSUPPORTED_ MEDIA_TYPE	PLS_INTEGER	415	The server is refusing to service the request because the entity of the request is in a format not supported by the requested resource for the requested method.
HTTP_REQ_RANGE_ NOT_SATISFIABLE	PLS_INTEGER	416	A server returns a response with this status code if a request included a Range request-header field, and none of the range-specifier values in this field overlap the current extent of the selected resource, and the request did not include an If-Range request-header field.
HTTP_EXPECTATION_ FAILED	PLS_INTEGER	417	The expectation given in an Expect request-header field could not be met by this server, or, if the server is a proxy, the server has unambiguous evidence that the request could not be met by the next-hop server.

Table 168–3 (Cont.) UTL_HTTP Constants - HTTP 1.1 Status Codes

Name	Type	Value	Description
HTTP_NOT_IMPLEMENTED	PLS_INTEGER	501	The server does not support the functionality required to fulfill the request.
HTTP_BAD_GATEWAY	PLS_INTEGER	502	The server, while acting as a gateway or proxy, received an invalid response from the upstream server it accessed in attempting to fulfill the request
HTTP_SERVICE_UNAVAILABLE	PLS_INTEGER	503	The server is currently unable to handle the request due to a temporary overloading or maintenance of the server.
HTTP_GATEWAY_TIME_OUT	PLS_INTEGER	504	The server, while acting as a gateway or proxy, did not receive a timely response from the upstream server specified by the URI (for example, HTTP, FTP, LDAP) or some other auxiliary server (for example, DNS) it needed to access in attempting to complete the request.
HTTP_VERSION_NOT_SUPPORTED	PLS_INTEGER	505	The server does not support, or refuses to support, the HTTP protocol version that was used in the request message.

Datatypes

REQ Type

Use this PL/SQL record type to represent an HTTP request.

Syntax

```
TYPE req IS RECORD (
   url           VARCHAR2(32767),
   method        VARCHAR2(64),
   http_version  VARCHAR2(64));
```

Parameters

Table 168–4 REQ Type Parameters

Parameter	Description
`url`	The URL of the HTTP request. It is set after the request is created by `BEGIN_REQUEST`.
`method`	The method to be performed on the resource identified by the URL. It is set after the request is created by `BEGIN_REQUEST`.
`http_version`	The HTTP protocol version used to send the request. It is set after the request is created by `BEGIN_REQUEST`.

Usage Notes

The information returned in `REQ` from the interface `begin_request` is for read only. Changing the field values in the record has no effect on the request.

There are other fields in `REQ` record type whose names begin with the prefix `private_`. The fields are private and are intended for use by implementation of the `UTL_HTTP` package. You should not modify the fields.

RESP Type

This PL/SQL record type is used to represent an HTTP response.

Syntax

```
TYPE resp IS RECORD (
   status_code    PLS_INTEGER,
   reason_phrase  VARCHAR2(256),
   http_version   VARCHAR2(64));
```

Parameters

Table 168–5 RESP Type Parameters

Parameter	Description
status_code	The status code returned by the Web server. It is a 3-digit integer that indicates the results of the HTTP request as handled by the Web server. It is set after the response is processed by GET_RESPONSE.
reason_phrase	The short textual message returned by the Web server that describe the status code. It gives a brief description of the results of the HTTP request as handled by the Web server. It is set after the response is processed by GET_RESPONSE.
http_version	The HTTP protocol version used in the HTTP response. It is set after the response is processed by GET_RESPONSE.

Usage Notes

The information returned in RESP from the interface GET_RESPONSE is read-only. There are other fields in the RESP record type whose names begin with the prefix private_. The fields are private and are intended for use by implementation of the UTL_HTTP package. You should not modify the fields.

COOKIE and COOKIE_TABLE Types

The COOKIE type is the PL/SQL record type that represents an HTTP cookie. The COOKIE_TABLE type is a PL/SQL index-by-table type that represents a collection of HTTP cookies.

Syntax

```
TYPE cookie IS RECORD (
   name  VARCHAR2(256),
   value  VARCHAR2(1024),
   domain  VARCHAR2(256),
   expire  TIMESTAMP WITH TIME ZONE,
   path  VARCHAR2(1024),
   secure  BOOLEAN,
   version  PLS_INTEGER,
   comment  VARCHAR2(1024));

TYPE cookie_table IS TABLE OF cookie INDEX BY binary_integer;
```

Fields of COOKIE Record Type

Table 168–6 shows the fields for the COOKIE and COOKIE_TABLE record types.

Table 168–6 Fields of COOKIE and COOKIE_TABLE Type

Field	Description
name	The name of the HTTP cookie
value	The value of the cookie
domain	The domain for which the cookie is valid
expire	The time by which the cookie will expire
path	The subset of URLs to which the cookie applies

Table 168–6 (Cont.) Fields of COOKIE and COOKIE_TABLE Type

Field	Description
secure	Should the cookie be returned to the Web server using secured means only.
version	The version of the HTTP cookie specification the cookie conforms. This field is NULL for Netscape cookies.
comment	The comment that describes the intended use of the cookie. This field is NULL for Netscape cookies.

Usage Notes

PL/SQL programs do not usually examine or change the cookie information stored in the UTL_HTTP package. The cookies are maintained by the package transparently. They are maintained inside the UTL_HTTP package, and they last for the duration of the database session only. PL/SQL applications that require cookies to be maintained beyond the lifetime of a database session can read the cookies using GET_COOKIES, store them persistently in a database table, and re-store the cookies back in the package using ADD_COOKIES in the next database session. All the fields in the cookie record, except for the comment field, must be stored. Do not alter the cookie information, which can result in an application error in the Web server or compromise the security of the PL/SQL and the Web server applications. See "Retrieving and Restoring Cookies" on page 168-20.

CONNECTION Type

Use the PL/SQL record type to represent the remote hosts and TCP/IP ports of a network connection that is kept persistent after an HTTP request is completed, according to the HTTP 1.1 protocol specification. The persistent network connection may be reused by a subsequent HTTP request to the same host and port. The subsequent HTTP request may be completed faster because the network connection latency is avoided. connection_table is a PL/SQL table of connection.

For a direct HTTP persistent connection to a Web server, the host and port fields contain the host name and TCP/IP port number of the Web server. The proxy_host and proxy_port fields are not set. For an HTTP persistent connection that was previously used to connect to a Web server using a proxy, the proxy_host and proxy_port fields contain the host name and TCP/IP port number of the proxy server. The host and port fields are not set, which indicates that the persistent connection, while connected to a proxy server, is not bound to any particular target Web server. An HTTP persistent connection to a proxy server can be used to access any target Web server that is using a proxy.

The SSL field indicates if Secured Socket Layer (SSL) is being used in an HTTP persistent connection. An HTTPS request is an HTTP request made over SSL. For an HTTPS (SSL) persistent connection connected using a proxy, the host and port fields contain the host name and TCP/IP port number of the target HTTPS Web server and the fields will always be set. An HTTPS persistent connection to an HTTPS Web server using a proxy server can only be reused to make another request to the same target Web server.

Syntax

```
TYPE connection IS RECORD (
   host  VARCHAR2(256),
   port  PLS_INTEGER,
   proxy_host  VARCHAR2(256),
   proxy_port  PLS_INTEGER,
```

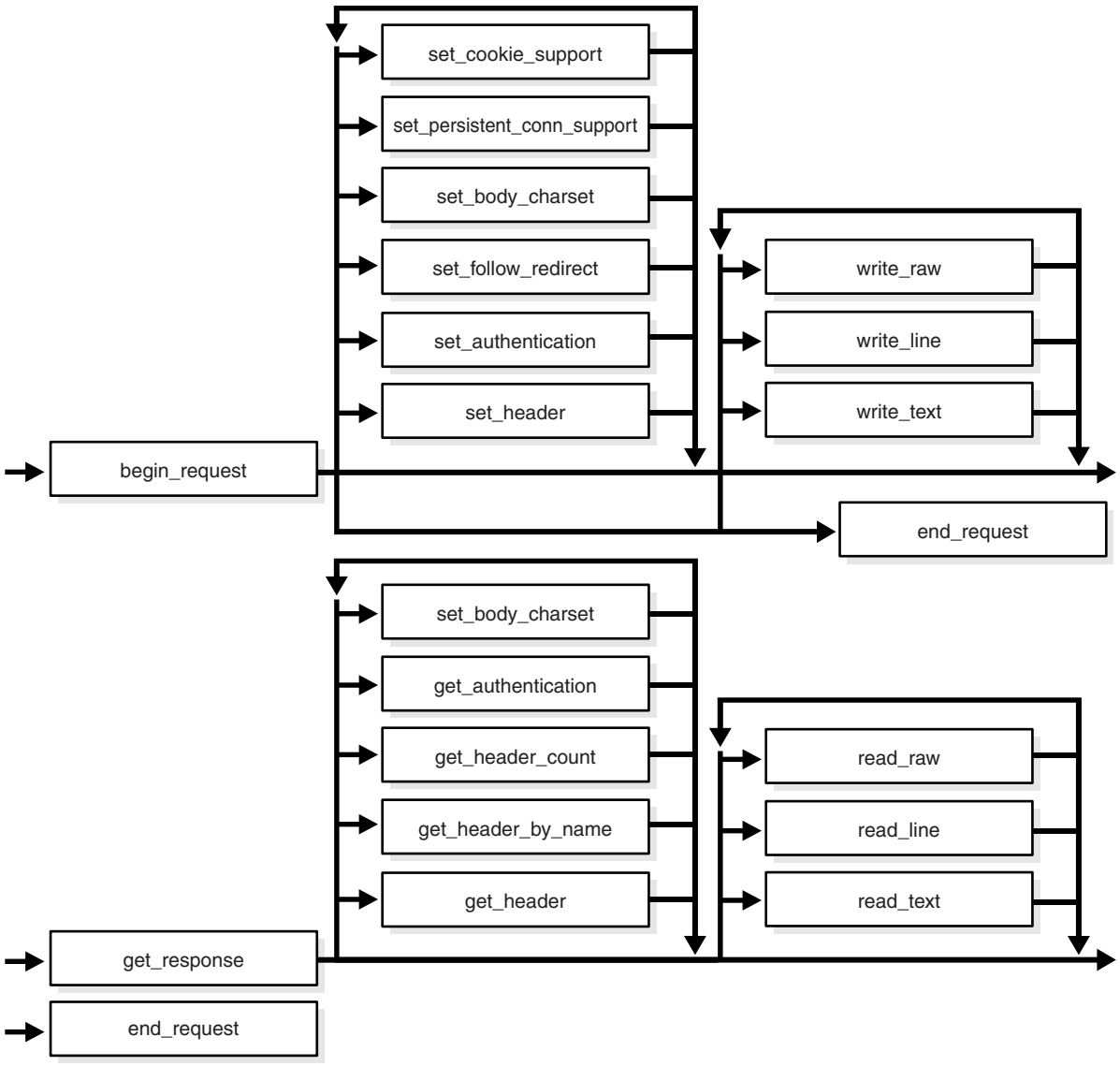
```
   ssl  BOOLEAN);

TYPE connection_table IS TABLE OF connection INDEX BY BINARY_INTEGER;
```

Operational Notes

Operational Flow

The `UTL_HTTP` package provides access to the HTTP protocol. The interfaces must be called in the order shown in Figure 168–1, or an exception will be raised.

Figure 168–1 Flow of the Core UTL_HTTP Package



The following can be called at any time:

- Non-protocol interfaces that manipulate cookies
  - GET_COOKIE_COUNT
  - GET_COOKIES
  - ADD_COOKIES
  - CLEAR_COOKIES
- Persistent connections
  - GET_PERSISTENT_CONN_COUNT
  - GET_PERSISTENT_CONNS
  - CLOSE_PERSISTENT_CONN
  - CLOSE_PERSISTENT_CONNS

- Interfaces that manipulate attributes and configurations of the UTL_HTTP package in the current session
  - SET_PROXY
  - GET_PROXY
  - SET_COOKIE_SUPPORT
  - GET_COOKIE_SUPPORT
  - SET_FOLLOW_REDIRECT
  - GET_FOLLOW_REDIRECT
  - SET_BODY_CHARSET
  - GET_BODY_CHARSET
  - SET_PERSISTENT_CONN_SUPPORT
  - GET_PERSISTENT_CONN_SUPPORT
  - SET_DETAILED_EXCP_SUPPORT
  - GET_DETAILED_EXCP_SUPPORT
  - SET_WALLET
  - SET_TRANSFER_TIMEOUT
  - GET_TRANSFER_TIMEOUT
- Interfaces that retrieve the last detailed exception code and message UTL_HTTP package in the current session
  - GET_DETAILED_SQLCODE
  - GET_DETAILED_SQLERRM

NOTE: Some of the request and response interfaces bear the same name as the interface that manipulates the attributes and configurations of the package in the current session. They are overloaded versions of the interface that manipulate a request or a response.

Simple HTTP Fetches

REQUEST and REQUEST_PIECES take a string uniform resource locator (URL), contact that site, and return the data (typically HTML) obtained from that site.

You should not expect REQUEST or REQUEST_PIECES to succeed in contacting a URL unless you can contact that URL by using a browser on the same machine (and with the same privileges, environment variables, and so on.)

If REQUEST or REQUEST_PIECES fails (for example, if it raises an exception, or if it returns an HTML-formatted error message, but you believe that the URL argument is correct), then try contacting that same URL with a browser to verify network availability from your machine. You may have a proxy server set in your browser that needs to be set with each REQUEST or REQUEST_PIECES call using the optional proxy parameter.

Note: UTL_HTTP can also use environment variables to specify its proxy behavior. For example, on UNIX, setting the environment variable http_proxy to a URL uses that service as the proxy server for HTTP requests. Setting the environment variable no_proxy to a domain name does not use the HTTP proxy server for URLs in that domain. When the UTL_HTTP package is executed in the Oracle database server, the environment variables are the ones that are set when the database instance is started.

See Also: Simple HTTP Fetches in a Single Call Subprograms on page 168-23

Session Settings

Session settings manipulate the configuration and default behavior of UTL_HTTP when HTTP requests are executed within a database user session. When a request is created, it inherits the default settings of the HTTP cookie support, follow-redirect, body character set, persistent-connection support, and transfer timeout of the current session. Those settings can be changed later by calling the request interface. When a response is created for a request, it inherits those settings from the request. Only the body character set can be changed later by calling the response interface.

See Also: Session Settings Subprograms on page 168-24

HTTP Requests

The HTTP Requests group of subprograms begin an HTTP request, manipulate attributes, and send the request information to the Web server. When a request is created, it inherits the default settings of the HTTP cookie support, follow-redirect, body character set, persistent-connection support, and transfer timeout of the current session. The settings can be changed by calling the request interface.

See Also: HTTP Requests Subprograms on page 168-25

HTTP Responses

The HTTP Responses group of subprograms manipulate an HTTP response obtained from GET_RESPONSE and receive response information from the Web server. When a response is created for a request, it inherits settings of the HTTP cookie support, follow-redirect, body character set, persistent-connection support, and transfer timeout from the request. Only the body character set can be changed by calling the response interface.

See Also: HTTP Responses Subprograms on page 168-26

HTTP Cookies

The UTL_HTTP package provides subprograms to manipulate HTTP cookies.

See Also: HTTP Cookies Subprograms on page 168-27

HTTP Persistent Connections

The UTL_HTTP package provides subprograms to manipulate persistent connections.

See Also: HTTP Persistent Connections Subprograms on page 168-28

Error Conditions

The `UTL_HTTP` package provides subprograms to retrieve error information.

See Also: Error Conditions Subprograms on page 168-29

Exceptions

Table 168–7 lists the exceptions that the UTL_HTTP package interface can raise. By default, UTL_HTTP raises the exception request_failed when a request fails to execute. If the package is set to raise a detailed exception by set_detailed_excp_support, the rest of the exceptions will be raised directly (except for the exception end_of_body, which will be raised by READ_TEXT, READ_LINE, and READ_RAW regardless of the setting).

Table 168–7 UTL_HTTP Exceptions

Exception	Error Code	Reason	Where Raised
REQUEST_FAILED	29273	The request fails to executes	Any HTTP request or response interface when detailed_exception is disabled
BAD_ARGUMENT	29261	The argument passed to the interface is bad	Any HTTP request or response interface when detailed_exception is enabled
BAD_URL	29262	The requested URL is badly formed	BEGIN_REQUEST, when detailed_exception is enabled
PROTOCOL_ERROR	29263	An HTTP protocol error occurs when communicating with the Web server	SET_HEADER, GET_RESPONSE, READ_RAW, READ_TEXT, and READ_LINE, when detailed_exception is enabled
UNKNOWN_SCHEME	29264	The scheme of the requested URL is unknown	BEGIN_REQUEST and GET_RESPONSE, when detailed_exception is enabled
HEADER_NOT_FOUND	29265	The header is not found	GET_HEADER, GET_HEADER_BY_NAME, when detailed_exception is enabled
END_OF_BODY	29266	The end of HTTP response body is reached	READ_RAW, READ_TEXT, and READ_LINE, when detailed_exception is enabled
ILLEGAL_CALL	29267	The call to UTL_HTTP is illegal at the current state of the HTTP request	SET_HEADER, SET_AUTHENTICATION, and SET_PERSISTENT_CONN_SUPPORT, when detailed_exception is enabled
HTTP_CLIENT_ERROR	29268	From GET_RESPONSE, the response status code indicates that a client error has occurred (status code in 4xx range). Or from begin_request, the HTTP proxy returns a status code in the 4xx range when making an HTTPS request through the proxy.	GET_RESPONSE, BEGIN_REQUEST when detailed_exception is enabled
HTTP_SERVER_ERROR	29269	From GET_RESPONSE, the response status code indicates that a client error has occurred (status code in 5xx range). Or from begin_request, the HTTP proxy returns a status code in the 5xx range when making an HTTPS request through the proxy.	GET_RESPONSE, BEGIN_REQUEST when detailed_exception is enabled

Table 168–7 (Cont.) UTL_HTTP Exceptions

Exception	Error Code	Reason	Where Raised
TOO_MANY_REQUESTS	29270	Too many requests or responses are open	BEGIN_REQUEST, when detailed_ exception is enabled
PARTIAL_ MULTIBYTE_ EXCEPTION	29275	No complete character is read and a partial multibyte character is found at the end of the response body	READ_TEXT and READ_LINE, when detailed_exception is enabled
TRANSFER_TIMEOUT	29276	No data is read and a read timeout occurred	READ_TEXT and READ_LINE, when detailed_exception is enabled

NOTE: The partial_multibyte_char and transfer_ timeout exceptions are duplicates of the same exceptions defined in UTL_TCP. They are defined in this package so that the use of this package does not require the knowledge of the UTL_TCP. As those exceptions are duplicates, an exception handle that catches the partial_multibyte_char and transfer_timeout exceptions in this package also catch the exceptions in the UTL_ TCP.

For REQUEST and REQUEST_PIECES, the request_failed exception is raised when any exception occurs and detailed_exception is disabled.

Examples

The following examples demonstrate how to use UTL_HTTP.

- General Usage
- Handling HTTP Authentication
- Retrieving and Restoring Cookies

General Usage

```
SET SERVEROUTPUT ON SIZE 40000

DECLARE
  req   UTL_HTTP.REQ;
  resp  UTL_HTTP.RESP;
  value VARCHAR2(1024);
BEGIN
  UTL_HTTP.SET_PROXY('proxy.my-company.com', 'corp.my-company.com');
  req := UTL_HTTP.BEGIN_REQUEST('http://www-hr.corp.my-company.com');
  UTL_HTTP.SET_HEADER(req, 'User-Agent', 'Mozilla/4.0');
  resp := UTL_HTTP.GET_RESPONSE(req);
  LOOP
    UTL_HTTP.READ_LINE(resp, value, TRUE);
    DBMS_OUTPUT.PUT_LINE(value);
  END LOOP;
  UTL_HTTP.END_RESPONSE(resp);
EXCEPTION
  WHEN UTL_HTTP.END_OF_BODY THEN
    UTL_HTTP.END_RESPONSE(resp);
END;
```

Retrieving HTTP Response Headers

```
SET SERVEROUTPUT ON SIZE 40000

DECLARE
  req   UTL_HTTP.REQ;
  resp  UTL_HTTP.RESP;
  name  VARCHAR2(256);
  value VARCHAR2(1024);
BEGIN
  UTL_HTTP.SET_PROXY('proxy.my-company.com', 'corp.my-company.com');
  req := UTL_HTTP.BEGIN_REQUEST('http://www-hr.corp.my-company.com');
  UTL_HTTP.SET_HEADER(req, 'User-Agent', 'Mozilla/4.0');
  resp := UTL_HTTP.GET_RESPONSE(req);
  DBMS_OUTPUT.PUT_LINE('HTTP response status code: ' || resp.status_code);
  DBMS_OUTPUT.PUT_LINE('HTTP response reason phrase: ' || resp.reason_phrase);
  FOR i IN 1..UTL_HTTP.GET_HEADER_COUNT(resp) LOOP
    UTL_HTTP.GET_HEADER(resp, i, name, value);
    DBMS_OUTPUT.PUT_LINE(name || ': ' || value);
  END LOOP;
  UTL_HTTP.END_RESPONSE(resp);
END;
```

Handling HTTP Authentication

```
SET serveroutput ON SIZE 40000

CREATE OR REPLACE PROCEDURE get_page (url      IN VARCHAR2,
```

```
                                        username IN VARCHAR2 DEFAULT NULL,
                                        password IN VARCHAR2 DEFAULT NULL,
                                        realm    IN VARCHAR2 DEFAULT NULL) AS
  req       UTL_HTTP.REQ;
  resp      UTL_HTTP.RESP;
  my_scheme VARCHAR2(256);
  my_realm  VARCHAR2(256);
  my_proxy  BOOLEAN;
BEGIN
  -- Turn off checking of status code. We will check it by ourselves.
  UTL_HTTP.HTTP_RESPONSE_ERROR_CHECK(FALSE);
  req := UTL_HTTP.BEGIN_REQUEST(url);
  IF (username IS NOT NULL) THEN
    UTL_HTTP.SET_AUTHENTICATION(req, username, password); -- Use HTTP Basic
Authen. Scheme
  END IF;
  resp := UTL_HTTP.GET_RESPONSE(req);
  IF (resp.status_code = UTL_HTTP.HTTP_UNAUTHORIZED) THEN
    UTL_HTTP.GET_AUTHENTICATION(resp, my_scheme, my_realm, my_proxy);
    IF (my_proxy) THEN
       DBMS_OUTPUT.PUT_LINE('Web proxy server is protected.');
       DBMS_OUTPUT.PUT('Please supplied the required ' || my_scheme || '
authentication username/password for realm ' || my_realm || ' for the proxy
server.');
    ELSE
      DBMS_OUTPUT.PUT_LINE('Web page ' || url || ' is protected.');
      DBMS_OUTPUT.PUT('Please supplied the required ' || my_scheme || '
authentication username/password for realm ' || my_realm || ' for the Web page.');
    END IF;
    UTL_HTTP.END_RESPONSE(resp);
    RETURN;
  END IF;
  FOR i IN 1..UTL_HTTP.GET_HEADER_COUNT(resp) LOOP
    UTL_HTTP.GET_HEADER(resp, i, name, value);
    DBMS_OUTPUT.PUT_LINE(name || ': ' || value);
  END LOOP;
  UTL_HTTP.END_RESPONSE(resp);
END;
```

Retrieving and Restoring Cookies

```
CREATE TABLE my_cookies (
    session_id  INTEGER,
    name        VARCHAR2(256),
    value       VARCHAR2(1024),
    domain      VARCHAR2(256),
    expire      DATE,
    path        VARCHAR2(1024),
    secure      VARCHAR2(1),
    version     INTEGER);

CREATE SEQUENCE session_id;
SET SERVEROUTPUT ON SIZE 40000

REM Retrieve cookies from UTL_HTTP
CREATE OR REPLACE FUNCTION save_cookies RETURN PLS_INTEGER AS
  cookies        UTL_HTTP.COOKIE_TABLE;
  my_session_id  PLS_INTEGER;
  secure         VARCHAR2(1);
BEGIN
```

```
  /* assume that some cookies have been set in previous HTTP requests. */
  UTL_HTTP.GET_COOKIES(cookies);
  SELECT session_id.nextval INTO my_session_id FROM DUAL;
  FOR i in 1..cookies.count LOOP
    IF (cookies(i).secure) THEN
      secure := 'Y';
    ELSE
      secure := 'N';
    END IF;
    INSERT INTO my_cookies
    VALUES (my_session_id, cookies(i).name, cookies(i).value,
            cookies(i).domain,
            cookies(i).expire, cookies(i).path, secure, cookies(i).version);
  END LOOP;
  RETURN my_session_id;
END;
/

REM Retrieve cookies from UTL_HTTP
CREATE OR REPLACE PROCEDURE restore_cookies (this_session_id IN PLS_INTEGER)
AS
  cookies        UTL_HTTP.COOKIE_TABLE;
  cookie         UTL_HTTP.COOKIE;
  i              PLS_INTEGER := 0;
  CURSOR c (c_session_id PLS_INTEGER) IS
    SELECT * FROM my_cookies WHERE session_id = c_session_id;
BEGIN
  FOR r IN c(this_session_id) LOOP
    i := i + 1;
    cookie.name     := r.name;
    cookie.value    := r.value;
    cookie.domain   := r.domain;
    cookie.expire   := r.expire;
    cookie.path     := r.path;
    IF (r.secure = 'Y') THEN
      cookie.secure := TRUE;
    ELSE
      cookie.secure := FALSE;
    END IF;
    cookie.version := r.version;
    cookies(i) := cookie;
  END LOOP;
  UTL_HTTP.CLEAR_COOKIES;
  UTL_HTTP.ADD_COOKIES(cookies);
END;
/
```

Subprogram Groups

`UTL_HTTP` subprograms are grouped by function:

- Simple HTTP Fetches in a Single Call Subprograms
- Session Settings Subprograms
- HTTP Requests Subprograms
- HTTP Responses Subprograms
- HTTP Cookies Subprograms
- HTTP Persistent Connections Subprograms
- Error Conditions Subprograms

Simple HTTP Fetches in a Single Call Subprograms

`REQUEST` and `REQUEST_PIECES` take a string uniform resource locator (URL), contact that site, and return the data (typically HTML) obtained from that site.

Table 168–8 UTL_HTTP Subprograms—Simple HTTP Fetches in a Single Call

Subprogram	Description
REQUEST Function on page 168-66	Returns up to the first 2000 bytes of the data retrieved from the given URL. This function can be used directly in SQL queries.
REQUEST_PIECES Function on page 168-68	Returns a PL/SQL table of 2000-byte pieces of the data retrieved from the given URL

Session Settings Subprograms

Table 168–9 UTL_HTTP Subprograms—Session Settings

Subprogram	Description
GET_BODY_CHARSET Procedure on page 168-44	Retrieves the default character set of the body of all future HTTP requests
GET_COOKIE_SUPPORT Procedure on page 168-46	Retrieves the current cookie support settings
GET_DETAILED_EXCP_SUPPORT Procedure on page 168-48	Checks if the `UTL_HTTP` package will raise a detailed exception or not
GET_FOLLOW_REDIRECT Procedure on page 168-51	Retrieves the follow-redirect setting in the current session
GET_PERSISTENT_CONN_SUPPORT Procedure on page 168-56	Checks if the persistent connection support is enabled and gets the maximum number of persistent connections in the current session
GET_PROXY Procedure on page 168-58	Retrieves the current proxy settings
GET_RESPONSE_ERROR_CHECK Procedure on page 168-60	Checks if the response error check is set or not
GET_TRANSFER_TIMEOUT Procedure on page 168-61	Retrieves the current network transfer timeout value
SET_TRANSFER_TIMEOUT Procedure on page 168-83	Sets the default character set of the body of all future HTTP requests when the media type is `text` and the character set is not specified in the `Content-Type` header
SET_COOKIE_SUPPORT Procedures on page 168-74	Sets whether or not future HTTP requests will support HTTP cookies; sets the maximum number of cookies maintained in the current database user session
SET_DETAILED_EXCP_SUPPORT Procedure on page 168-76	Sets the `UTL_HTTP` package to raise a detailed exception
SET_FOLLOW_REDIRECT Procedures on page 168-77	Sets the maximum number of times `UTL_HTTP` follows the HTTP redirect instruction in the HTTP responses to future requests in the `GET_RESPONSE` function
SET_PERSISTENT_CONN_SUPPORT Procedure on page 168-79	Sets whether or not future HTTP requests will support the HTTP 1.1 persistent connection; sets the maximum number of persistent connections maintained in the current database user session
SET_PROXY Procedure on page 168-81	Sets the proxy to be used for requests of HTTP or other protocols
SET_RESPONSE_ERROR_CHECK Procedure on page 168-82	Sets whether or not `GET_RESPONSE` raises an exception when the Web server returns a status code that indicates an error—a status code in the 4xx or 5xx ranges
SET_TRANSFER_TIMEOUT Procedure on page 168-83	Sets the timeout value for `UTL_HTTP` to read the HTTP response from the Web server or proxy server
SET_WALLET Procedure on page 168-84	Sets the Oracle Wallet used for all HTTP requests over Secured Socket Layer (SSL), that is, HTTPS

HTTP Requests Subprograms

Table 168–10 UTL_HTTP Subprograms—HTTP Requests

Subprogram	Description
BEGIN_REQUEST Function on page 168-35	Begins a new HTTP request.`UTL_HTTP` establishes the network connection to the target Web server or the proxy server and sends the HTTP request line.
END_REQUEST Procedure on page 168-41	Ends the HTTP request
SET_HEADER Procedure on page 168-78	Sets an HTTP request header. The request header is sent to the Web server as soon as it is set.
SET_AUTHENTICATION Procedure on page 168-71	Sets HTTP authentication information in the HTTP request header. The Web server needs this information to authorize the request.
SET_BODY_CHARSET Procedures on page 168-72	Sets the character set of the request body when the media type is `text` but the character set is not specified in the `Content-Type` header
SET_COOKIE_SUPPORT Procedures on page 168-74	Enables or disables support for the HTTP cookies in the request
SET_FOLLOW_REDIRECT Procedures on page 168-77	`Sets the maximum number of times UTL_HTTP` follows the HTTP redirect instruction in the HTTP response to this request in the GET_RESPONSE Function on page 168-59
SET_PERSISTENT_CONN_SUPPORT Procedure on page 168-79	Enables or disables support for the HTTP 1.1 persistent-connection in the request
SET_PROXY Procedure on page 168-81	Writes a text line in the HTTP request body and ends the line with new-line characters (CRLF as defined in `UTL_TCP`)
WRITE_RAW Procedure on page 168-86	Writes some binary data in the HTTP request body
WRITE_TEXT Procedure on page 168-87	Writes some text data in the HTTP request body

HTTP Responses Subprograms

Table 168–11 UTL_HTTP Subprograms—HTTP Responses

Subprogram	Description
END_RESPONSE Procedure on page 168-42	Ends the HTTP response. It completes the HTTP request and response.
GET_AUTHENTICATION Procedure on page 168-43	Retrieves the HTTP authentication information needed for the request to be accepted by the Web server as indicated in the HTTP response header
GET_HEADER Procedure on page 168-52	Returns the n^{th} HTTP response header name and value returned in the response
GET_HEADER_BY_NAME Procedure on page 168-53	Returns the HTTP response header value returned in the response given the name of the header
GET_HEADER_COUNT Function on page 168-54	Returns the number of HTTP response headers returned in the response
GET_RESPONSE Function on page 168-59	Reads the HTTP response. When the function returns, the status line and the HTTP response headers have been read and processed.
READ_LINE Procedure on page 168-62	Reads the HTTP response body in text form until the end of line is reached and returns the output in the caller-supplied buffer
READ_RAW Procedure on page 168-63	Reads the HTTP response body in binary form and returns the output in the caller-supplied buffer
READ_TEXT Procedure on page 168-64	Reads the HTTP response body in text form and returns the output in the caller-supplied buffer
SET_BODY_CHARSET Procedures on page 168-72	Sets the character set of the response body when the media type is "text" but the character set is not specified in the "Content-Type" header

HTTP Cookies Subprograms

Table 168–12 UTL_HTTP Subprograms—HTTP Cookies

Subprogram	Description
ADD_COOKIES Procedure on page 168-34	Adds the cookies maintained by UTL_HTTP
CLEAR_COOKIES Procedure on page 168-37	Clears all cookies maintained by the UTL_HTTP package
GET_COOKIE_COUNT Function on page 168-45	Returns the number of cookies currently maintained by the UTL_HTTP package set by all Web servers
GET_COOKIES Function on page 168-47	Returns all the cookies currently maintained by the UTL_HTTP package set by all Web servers

HTTP Persistent Connections Subprograms

Table 168–13 UTL_HTTP Subprograms—HTTP Persistent Connections

Subprogram	Description
CLOSE_PERSISTENT_CONN Procedure on page 168-38	Closes an HTTP persistent connection maintained by the `UTL_HTTP` package in the current database session
CLOSE_PERSISTENT_CONNS Procedure on page 168-39	Closes a group of HTTP persistent connections maintained by the `UTL_HTTP` package in the current database session
GET_PERSISTENT_CONN_COUNT Function on page 168-55	Returns the number of network connections currently kept persistent by the `UTL_HTTP` package to the Web servers
GET_PERSISTENT_CONNS Procedure on page 168-57	Returns all the network connections currently kept persistent by the `UTL_HTTP` package to the Web servers

Error Conditions Subprograms

Table 168–14 UTL_HTTP Subprograms—Error Conditions

Subprogram	Description
GET_DETAILED_SQLCODE Function on page 168-49	Retrieves the detailed SQLCODE of the last exception raised
GET_DETAILED_SQLERRM Function on page 168-50	Retrieves the detailed SQLERRM of the last exception raised

Summary of UTL_HTTP Subprograms

Table 168–15 UTL_HTTP Package Subprograms

Subprogram	Description	Group
ADD_COOKIES Procedure on page 168-34	Adds the cookies maintained by UTL_HTTP	HTTP Cookies Subprograms on page 168-27
BEGIN_REQUEST Function on page 168-35	Begins a new HTTP request. UTL_HTTP establishes the network connection to the target Web server or the proxy server and sends the HTTP request line	HTTP Requests Subprograms on page 168-25
CLEAR_COOKIES Procedure on page 168-37	Clears all cookies maintained by the UTL_HTTP package	HTTP Cookies Subprograms on page 168-27
CLOSE_PERSISTENT_CONN Procedure on page 168-38	Closes an HTTP persistent connection maintained by the UTL_HTTP package in the current database session	HTTP Persistent Connections Subprograms on page 168-28
CLOSE_PERSISTENT_CONNS Procedure on page 168-39	Closes a group of HTTP persistent connections maintained by the UTL_HTTP package in the current database session	HTTP Persistent Connections Subprograms on page 168-28
END_REQUEST Procedure on page 168-41	Ends the HTTP request	HTTP Requests Subprograms on page 168-25
END_RESPONSE Procedure on page 168-42	Ends the HTTP response. It completes the HTTP request and response	HTTP Responses Subprograms on page 168-26
GET_AUTHENTICATION Procedure on page 168-43	Retrieves the HTTP authentication information needed for the request to be accepted by the Web server as indicated in the HTTP response header	HTTP Responses Subprograms on page 168-26
GET_BODY_CHARSET Procedure on page 168-44	Retrieves the default character set of the body of all future HTTP requests	Session Settings Subprograms on page 168-24
GET_COOKIE_COUNT Function on page 168-45	Returns the number of cookies currently maintained by the UTL_HTTP package set by all Web servers	HTTP Cookies Subprograms on page 168-27
GET_COOKIE_SUPPORT Procedure on page 168-46	Retrieves the current cookie support settings	Session Settings Subprograms on page 168-24
GET_COOKIES Function on page 168-47	Returns all the cookies currently maintained by the UTL_HTTP package set by all Web servers	HTTP Cookies Subprograms on page 168-27
GET_DETAILED_EXCP_SUPPORT Procedure on page 168-48	Checks if the UTL_HTTP package will raise a detailed exception or not	Session Settings Subprograms on page 168-24
GET_DETAILED_SQLCODE Function on page 168-49	Retrieves the detailed SQLCODE of the last exception raised	Error Conditions Subprograms on page 168-29

Table 168–15 (Cont.) UTL_HTTP Package Subprograms

Subprogram	Description	Group
GET_DETAILED_SQLERRM Function on page 168-50	Retrieves the detailed SQLERRM of the last exception raised	Error Conditions Subprograms on page 168-29
GET_FOLLOW_REDIRECT Procedure on page 168-51	Retrieves the follow-redirect setting in the current session	Session Settings Subprograms on page 168-24
GET_HEADER Procedure on page 168-52	Returns the n^{th} HTTP response header name and value returned in the response	HTTP Responses Subprograms on page 168-26
GET_HEADER_BY_NAME Procedure on page 168-53	Returns the HTTP response header value returned in the response given the name of the header	HTTP Responses Subprograms on page 168-26
GET_HEADER_COUNT Function on page 168-54	Returns the number of HTTP response headers returned in the response	HTTP Responses on page 168-15 and HTTP Responses Subprograms on page 168-26
GET_PERSISTENT_CONN_COUNT Function on page 168-55	Returns the number of network connections currently kept persistent by the UTL_HTTP package to the Web servers	HTTP Persistent Connections Subprograms on page 168-28
GET_HEADER_COUNT Function on page 168-54	Sees whether or not future HTTP requests will support the HTTP 1.1 persistent connection; sets the maximum number of persistent connections maintained in the current database user session	Session Settings Subprograms on page 168-24
GET_PERSISTENT_CONN_SUPPORT Procedure on page 168-56	Checks if the persistent connection support is enabled and gets the maximum number of persistent connections in the current session (see Session Settings Subprograms on page 168-24)	HTTP Persistent Connections Subprograms on page 168-28
GET_PERSISTENT_CONNS Procedure on page 168-57	Returns all the network connections currently kept persistent by the UTL_HTTP package to the Web servers	HTTP Persistent Connections Subprograms on page 168-28
GET_PROXY Procedure on page 168-58	Retrieves the current proxy settings	Session Settings Subprograms on page 168-24
GET_RESPONSE Function on page 168-59	Reads the HTTP response. When the function returns, the status line and the HTTP response headers have been read and processed	HTTP Responses Subprograms on page 168-26
GET_RESPONSE_ERROR_CHECK Procedure on page 168-60	Checks if the response error check is set or no	Session Settings Subprograms on page 168-24
GET_TRANSFER_TIMEOUT Procedure on page 168-61	Retrieves the current network transfer timeout value	Session Settings Subprograms on page 168-24
READ_LINE Procedure on page 168-62	Reads the HTTP response body in text form until the end of line is reached and returns the output in the caller-supplied buffer	HTTP Responses Subprograms on page 168-26

Table 168–15 (Cont.) UTL_HTTP Package Subprograms

Subprogram	Description	Group
READ_RAW Procedure on page 168-63	Reads the HTTP response body in binary form and returns the output in the caller-supplied buffer	HTTP Responses Subprograms on page 168-26
READ_TEXT Procedure on page 168-64	Reads the HTTP response body in text form and returns the output in the caller-supplied buffer	HTTP Responses Subprograms on page 168-26
REQUEST Function on page 168-66	Returns up to the first 2000 bytes of the data retrieved from the given URL. This function can be used directly in SQL queries.	Simple HTTP Fetches in a Single Call Subprograms on page 168-23
REQUEST_PIECES Function on page 168-68	Returns a PL/SQL table of 2000-byte pieces of the data retrieved from the given URL	Simple HTTP Fetches in a Single Call Subprograms on page 168-23
SET_AUTHENTICATION Procedure on page 168-71	Sets HTTP authentication information in the HTTP request header. The Web server needs this information to authorize the request.	HTTP Requests Subprograms on page 168-25
SET_BODY_CHARSET Procedures on page 168-72	Sets the default character set of the body of all future HTTP requests when the media type is `text` and the character set is not specified in the `Content-Type` header	Session Settings Subprograms on page 168-24
SET_BODY_CHARSET Procedures on page 168-72	Sets the character set of the request body when the media type is `text` but the character set is not specified in the `Content-Type` header	HTTP Requests Subprograms on page 168-25
SET_BODY_CHARSET Procedures on page 168-72	Sets the character set of the response body when the media type is "text" but the character set is not specified in the "Content-Type" header	HTTP Responses Subprograms and Session Settings Subprograms on page 168-24
SET_COOKIE_SUPPORT Procedures on page 168-74	Enables or disables support for the HTTP cookies in the request	HTTP Requests Subprograms on page 168-25
SET_DETAILED_EXCP_ SUPPORT Procedure on page 168-76	Sets whether or not future HTTP requests will support HTTP cookies; sets the maximum number of cookies maintained in the current database user session	Session Settings Subprograms on page 168-24
SET_DETAILED_EXCP_ SUPPORT Procedure on page 168-76	Sets the `UTL_HTTP` package to raise a detailed exception	Session Settings Subprograms on page 168-24
SET_FOLLOW_REDIRECT Procedures on page 168-77	Sets the maximum number of times `UTL_HTTP` follows the HTTP redirect instruction in the HTTP response to this request in the `GET_RESPONSE` function	HTTP Requests Subprograms on page 168-25
SET_HEADER Procedure on page 168-78	Sets the maximum number of times `UTL_HTTP` follows the HTTP redirect instruction in the HTTP responses to future requests in the `GET_RESPONSE` function	Session Settings Subprograms on page 168-24

Table 168–15 (Cont.) UTL_HTTP Package Subprograms

Subprogram	Description	Group
SET_HEADER Procedure on page 168-78	Sets an HTTP request header. The request header is sent to the Web server as soon as it is set.	HTTP Requests Subprograms on page 168-25
SET_PERSISTENT_CONN_SUPPORT Procedure on page 168-79	Enables or disables support for the HTTP 1.1 persistent-connection in the request	HTTP Requests Subprograms on page 168-25
SET_PROXY Procedure on page 168-81	Sets the proxy to be used for requests of HTTP or other protocols	Session Settings on page 168-15 and Session Settings Subprograms on page 168-24
SET_RESPONSE_ERROR_CHECK Procedure on page 168-82	Sets whether or not `GET_RESPONSE` raises an exception when the Web server returns a status code that indicates an error—a status code in the 4xx or 5xx ranges	Session Settings Subprograms on page 168-24
SET_TRANSFER_TIMEOUT Procedure on page 168-83	Sets the timeout value for `UTL_HTTP` to read the HTTP response from the Web server or proxy server	Session Settings on page 168-15 and Session Settings Subprograms on page 168-24
SET_WALLET Procedure on page 168-84	Sets the Oracle Wallet used for all HTTP requests over Secured Socket Layer (SSL), that is, HTTPS	Session Settings Subprograms on page 168-24
WRITE_LINE Procedure on page 168-85	Writes a text line in the HTTP request body and ends the line with new-line characters (CRLF as defined in UTL_TCP	HTTP Requests Subprograms on page 168-25
WRITE_RAW Procedure on page 168-86	Writes some binary data in the HTTP request body	HTTP Requests Subprograms on page 168-25
WRITE_TEXT Procedure on page 168-87	Writes some text data in the HTTP request body	HTTP Requests Subprograms on page 168-25

ADD_COOKIES Procedure

This procedure adds the cookies maintained by `UTL_HTTP`.

> **See Also:** HTTP Cookies on page 168-15 and HTTP Cookies Subprograms on page 168-27

Syntax

```
UTL_HTTP.ADD_COOKIES (
   cookies  IN cookie_table);
```

Parameters

Table 168–16 ADD_COOKIES Procedure Parameters

Parameter	Description
`cookies`	The cookies to be added

Usage Notes

The cookies that the package currently maintains are not cleared before new cookies are added.

BEGIN_REQUEST Function

This functions begins a new HTTP request. UTL_HTTP establishes the network connection to the target Web server or the proxy server and sends the HTTP request line. The PL/SQL program continues the request by calling some other interface to complete the request. The URL may contain the username and password needed to authenticate the request to the server. The format is

```
scheme://[user[:password]@]host[:port]/[...]
```

See Also: HTTP Requests on page 168-15 and HTTP Requests Subprograms on page 168-25

Syntax

```
UTL_HTTP.BEGIN_REQUEST (
   url           IN VARCHAR2,
   method        IN VARCHAR2 DEFAULT 'GET',
   http_version  IN VARCHAR2 DEFAULT NULL)
RETURN req;
```

Parameters

Table 168–17 BEGIN_REQUEST Function Parameters

Parameter	Description
url	The URL of the HTTP request
method	The method performed on the resource identified by the URL
http_version	The HTTP protocol version that sends the request. The format of the protocol version is HTTP/major-version.minor-version, where major-version and minor-version are positive numbers. If this parameter is set to NULL, UTL_HTTP uses the latest HTTP protocol version that it supports to send the request. The latest version that the package supports is 1.1 and it can be upgraded to a later version. The default is NULL.

Usage Notes

- The URL passed as an argument to this function is not examined for illegal characters, such as spaces, according to URL specification RFC 2396. You should escape those characters with the UTL_URL package to return illegal and reserved characters. URLs should consist of US-ASCII characters only. See Chapter 179, "UTL_URL" for a list of legal characters in URLs. Note that URLs should consist of US-ASCII characters only. The use of non-US-ASCII characters in a URL is generally unsafe.
- UTL_HTTP.BEGIN_REQUEST can send a URL whose length is up to 32767 bytes. However, different web servers impose different limits on the length of the URL they can accept. This limit is often about 4000 bytes. If this limit is exceeded, the outcome will depend on the web server. For example, a web server might simply drop the HTTP connection without returning a response of any kind. If this happens, a subsequent invocation of UTL_HTTP.GET_RESPONSE will raise the UTL_HTTP.PROTOCOL_ERROR exception.

A URL will be long when its QUERY_STRING (that is, the information that follows the question mark (?)) is long. In general, it is better to send this parameterization in the body of the request using the POST method.

```
req := UTL_HTTP.BEGIN_REQUEST (url=>the_url, method=>'POST');
UTL_HTTP.SET_HEADER (r      =>  req,
                     name   =>  'Content-Type',
                     value  =>  'application/x-www-form-urlencoded');
UTL_HTTP.SET_HEADER (r      =>   req,
                     name   =>   'Content-Length',
                     value  =>'  <length of data posted in bytes>');
UTL_HTTP.WRITE_TEXT (r      =>   req,
                     data   =>   'p1 = value1&p2=value2...');
resp := UTL_HTTP.GET_RESPONSE
                     (r     =>   req);
...
```

The programmer must determine whether a particular web server may, or may not, accept data provided in this way.

- An Oracle wallet must be set before accessing Web servers over HTTPS. See the SET_WALLET procedure on how to set up an Oracle wallet.

CLEAR_COOKIES Procedure

This procedure clears all cookies maintained by the `UTL_HTTP` package.

> **See Also:** HTTP Cookies on page 168-15 and HTTP Cookies Subprograms on page 168-27

Syntax

```
UTL_HTTP.CLEAR_COOKIES;
```

CLOSE_PERSISTENT_CONN Procedure

This procedure closes an HTTP persistent connection maintained by the UTL_HTTP package in the current database session.

See Also: HTTP Persistent Connections on page 168-15 and HTTP Persistent Connections Subprograms on page 168-28

Syntax

```
UTL_HTTP.CLOSE_PERSISTENT_CONN (
   conn  IN connection);
```

Parameters

Table 168–18 CLOSE_PERSISTENT_CONN Procedure Parameters

Parameter	Description
conn	The HTTP persistent connection to close

CLOSE_PERSISTENT_CONNS Procedure

This procedure closes a group of HTTP persistent connections maintained by the UTL_HTTP package in the current database session. This procedure uses a pattern-match approach to decide which persistent connections to close.

To close a group of HTTP persistent connection that share a common property (for example, all connections to a particular host, or all SSL connections), set the particular parameters and leave the rest of the parameters NULL. If a particular parameter is set to NULL when this procedure is called, that parameter will not be used to decide which connections to close.

For example, the following call to the procedure closes all persistent connections to foobar:

```
UTL_HTTP.CLOSE_PERSISTENT_CONNS(host => 'foobar');
```

And the following call to the procedure closes all persistent connections through the proxy www-proxy at TCP/IP port 80:

```
UTL_HTTP.CLOSE_PERSISTENT_CONNS(proxy_host => 'foobar',
                                proxy_port => 80);
```

And the following call to the procedure closes all persistent connections:

```
UTL_HTTP.CLOSE_PERSISTENT_CONNS;
```

> **See Also:** HTTP Persistent Connections on page 168-15 and HTTP Persistent Connections Subprograms on page 168-28

Syntax

```
UTL_HTTP.CLOSE_PERSISTENT_CONNS (
   host        IN VARCHAR2 DEFAULT NULL,
   port        IN PLS_INTEGER DEFAULT NULL,
   proxy_host  IN VARCHAR2 DEFAULT NULL,
   proxy_port  IN PLS_INTEGER DEFAULT NULL,
   ssl         IN BOOLEAN DEFAULT NULL);
```

Parameters

Table 168–19 CLOSE_PERSISTENT_CONNS Procedure Parameters

Parameter	Description
host	The host for which persistent connections are to be closed
port	The port number for which persistent connections are to be closed
proxy_host	The proxy host for which persistent connections are to be closed
proxy_port	The proxy port for which persistent connections are to be closed
ssl	Close persistent SSL connection

Usage Notes

Connections to the same Web server at different TCP/IP ports are counted individually. The host names of the Web servers are identified as specified in the URL

of the original HTTP requests. Therefore, fully qualified host names with domain names will be counted differently from the host names without domain names.

Note that the use of a `NULL` value in a parameter when this procedure is called means that the caller does not care about its value when the package decides which persistent connection to close. If you want a `NULL` value in a parameter to match only a `NULL` value of the parameter of a persistent connection (which is when you want to close a specific persistent connection), you should use the `CLOSE_PERSISTENT_CONN` procedure that closes a specific persistent connection.

END_REQUEST Procedure

This procedure ends the HTTP request. To terminate the HTTP request without completing the request and waiting for the response, the program can call this procedure. Otherwise, the program should go through the normal sequence of beginning a request, getting the response, and closing the response. The network connection will always be closed and will not be reused.

See Also: HTTP Requests on page 168-15 and HTTP Requests Subprograms on page 168-25

Syntax

```
UTL_HTTP.END_REQUEST (
   r  IN OUT NOCOPY req);
```

Parameters

Table 168–20 END_REQUEST Procedure Parameters

Parameter	Description
r	The HTTP request

END_RESPONSE Procedure

This procedure ends the HTTP response. It completes the HTTP request and response. Unless HTTP 1.1 persistent connection is used in this request, the network connection is also closed.

> **See Also:** HTTP Responses on page 168-15 and HTTP Responses Subprograms on page 168-26

Syntax

```
UTL_HTTP.END_RESPONSE (
   r  IN OUT NOCOPY resp);
```

Parameters

Table 168–21 END_RESPONSE Procedure Parameters

Parameter	Description
r	The HTTP response

GET_AUTHENTICATION Procedure

This procedure retrieves the HTTP authentication information needed for the request to be accepted by the Web server as indicated in the HTTP response header.

> **See Also:** HTTP Responses on page 168-15 and HTTP Responses Subprograms on page 168-26

Syntax

```
UTL_HTTP.GET_AUTHENTICATION(
   r          IN OUT NOCOPY resp,
   scheme     OUT VARCHAR2,
   realm      OUT VARCHAR2,
   for_proxy  IN BOOLEAN  DEFAULT FALSE);
```

Parameters

Table 168–22 GET_AUTHENTICATION Procedure Parameters

Parameter	Description
r	The HTTP response
scheme	The scheme for the required HTTP authentication
realm	The realm for the required HTTP authentication
for_proxy	Returns the HTTP authentication information required for the access to the HTTP proxy server instead of the Web server? Default is FALSE

Usage Notes

When a Web client is unaware that a document is protected, at least two HTTP requests are required for the document to be retrieved. In the first HTTP request, the Web client makes the request without supplying required authentication information; so the request is denied. The Web client can determine the authentication information required for the request to be authorized by calling GET_AUTHENTICATION. The Web client makes the second request and supplies the required authentication information with SET_AUTHORIZATION. If the authentication information can be verified by the Web server, the request will succeed and the requested document is returned. Before making the request, if the Web client knows that authentication information is required, it can supply the required authentication information in the first request, thus saving an extra request.

GET_BODY_CHARSET Procedure

This procedure retrieves the default character set of the body of all future HTTP requests.

> **See Also:** Session Settings on page 168-15 and Session Settings Subprograms on page 168-24

Syntax

```
UTL_HTTP.GET_BODY_CHARSET (
   charset  OUT NOCOPY VARCHAR2);
```

Parameters

Table 168–23 GET_BODY_CHARSET Procedure Parameters

Parameter	Description
`charset`	The default character set of the body of all future HTTP requests

GET_COOKIE_COUNT Function

This function returns the number of cookies currently maintained by the `UTL_HTTP` package set by all Web servers.

> **See Also:** HTTP Cookies on page 168-15 and HTTP Cookies Subprograms on page 168-27

Syntax

```
UTL_HTTP.GET_COOKIE_COUNT
RETURN PLS_INTEGER;
```

GET_COOKIE_SUPPORT Procedure

This procedure retrieves the current cookie support settings.

> **See Also:** Session Settings on page 168-15 and Session Settings Subprograms on page 168-24

Syntax

```
UTL_HTTP.GET_COOKIE_SUPPORT (
   enable                OUT BOOLEAN,
   max_cookies           OUT PLS_INTEGER,
   max_cookies_per_site  OUT PLS_INTEGER);
```

Parameters

Table 168–24 GET_COOKIE SUPPORT Procedure Parameters

Parameter	Description
`enable`	Indicates whether future HTTP requests should support HTTP cookies (`TRUE`) or not (`FALSE`)
`max_cookies`	Indicates the maximum total number of cookies maintained in the current session
`max_cookies_per_site`	Indicates the maximum number of cookies maintained in the current session for each Web site

GET_COOKIES Function

This function returns all the cookies currently maintained by the UTL_HTTP package set by all Web servers.

> **See Also:** HTTP Cookies on page 168-15 and HTTP Cookies Subprograms on page 168-27

Syntax

```
UTL_HTTP.GET_COOKIES (
   cookies  IN OUT NOCOPY cookie_table);
```

Parameters

Table 168–25 GET_COOKIES Procedure Parameters

Parameter	Description
cookies	The cookies returned

GET_DETAILED_EXCP_SUPPORT Procedure

This procedure checks if the UTL_HTTP package will raise a detailed exception or not.

See Also: Session Settings on page 168-15 and Session Settings Subprograms on page 168-24

Syntax

```
UTL_HTTP.GET_DETAILED_EXCP_SUPPORT (
   enable  OUT BOOLEAN);
```

Parameters

Table 168–26 GET_DETAILED_EXCP_SUPPORT Procedure Parameters

Parameter	Description
enable	TRUE if UTL_HTTP raises a detailed exception; otherwise FALSE

GET_DETAILED_SQLCODE Function

This function retrieves the detailed `SQLCODE` of the last exception raised.

See Also: Error Conditions on page 168-16 and Error Conditions Subprograms on page 168-29

Syntax

```
UTL_HTTP.GET_DETAILED_SQLCODE
RETURN PLS_INTEGER;
```

GET_DETAILED_SQLERRM Function

This function retrieves the detailed `SQLERRM` of the last exception raised.

See Also: Error Conditions on page 168-16 and Error Conditions Subprograms on page 168-29

Syntax

```
UTL_HTTP.GET_DETAILED_SQLERRM
RETURN VARCHAR2;
```

GET_FOLLOW_REDIRECT Procedure

This procedure retrieves the follow-redirect setting in the current session

> **See Also:** Session Settings on page 168-15 and Session Settings Subprograms on page 168-24

Syntax

```
UTL_HTTP.GET_FOLLOW_REDIRECT (
   max_redirects  OUT PLS_INTEGER);
```

Parameters

Table 168–27 GET_FOLLOW_REDIRECT Procedure Parameters

Parameter	Description
`max_redirects`	The maximum number of redirections for all future HTTP requests

GET_HEADER Procedure

This procedure returns the n^{th} HTTP response header name and value returned in the response.

> **See Also:** HTTP Responses on page 168-15 and HTTP Responses Subprograms on page 168-26

Syntax

```
UTL_HTTP.GET_HEADER (
   r      IN OUT NOCOPY resp,
   n      IN PLS_INTEGER,
   name   OUT NOCOPY VARCHAR2,
   value  OUT NOCOPY VARCHAR2);
```

Parameters

Table 168–28 GET_HEADER Procedure Parameters

Parameter	Description
`r`	The HTTP response
`n`	The n^{th} header to return
`name`	The name of the HTTP response header
`value`	The value of the HTTP response header

Usage Notes

If the response body returned by the remote Web server is encoded in chunked transfer encoding format, the trailer headers that are returned at the end of the response body will be added to the response, and the response header count will be updated. You can retrieve the additional headers after the end of the response body is reached and before you end the response.

GET_HEADER_BY_NAME Procedure

This procedure returns the HTTP response header value returned in the response given the name of the header.

> **See Also:** HTTP Responses on page 168-15 and HTTP Responses Subprograms on page 168-26

Syntax

```
UTL_HTTP.GET_HEADER_BY_NAME(
   r      IN OUT NOCOPY resp,
   name   IN VARCHAR2,
   value  OUT NOCOPY VARCHAR2,
   n      IN PLS_INTEGER DEFAULT 1);
```

Parameters

Table 168–29 GET_HEADER_BY_NAME Procedure Parameters

Parameter	Description
r	The HTTP response
name	The name of the HTTP response header for which the value is to return
value	The value of the HTTP response header
n	The n^{th} occurrence of an HTTP response header by the specified name to return. The default is 1.

Usage Notes

If the response body returned by the remote Web server is encoded in chunked transfer encoding format, the trailer headers that are returned at the end of the response body will be added to the response, and the response header count will be updated. You can retrieve the additional headers after the end of the response body is reached and before you end the response.

GET_HEADER_COUNT Function

This function returns the number of HTTP response headers returned in the response.

> **See Also:** HTTP Responses on page 168-15 and HTTP Responses Subprograms on page 168-26

Syntax

```
UTL_HTTP.GET_HEADER_COUNT (
   r  IN OUT NOCOPY resp)
RETURN PLS_INTEGER;
```

Parameters

Table 168–30 GET_HEADER_COUNT Function Parameters

Parameter	Description
r	The HTTP response

Usage Notes

If the response body returned by the remote Web server is encoded in chunked transfer encoding format, the trailer headers that are returned at the end of the response body will be added to the response, and the response header count will be updated. You can retrieve the additional headers after the end of the response body is reached and before you end the response.

GET_PERSISTENT_CONN_COUNT Function

This function returns the number of network connections currently kept persistent by the `UTL_HTTP` package to the Web servers.

> **See Also:** HTTP Persistent Connections on page 168-15 and HTTP Persistent Connections Subprograms on page 168-28

Syntax

```
UTL_HTTP.GET_PERSISTENT_CONN_COUNT
RETURN PLS_INTEGER;
```

Usage Notes

Connections to the same Web server at different TCP/IP ports are counted individually. The host names of the Web servers are identified as specified in the URL of the original HTTP requests. Therefore, fully qualified host names with domain names will be counted differently from the host names without domain names.

GET_PERSISTENT_CONN_SUPPORT Procedure

This procedure checks:

- If the persistent connection support is enabled
- Gets the maximum number of persistent connections in the current session

See Also: Session Settings on page 168-15 and Session Settings Subprograms on page 168-24

Syntax

```
UTL_HTTP.GET_PERSISTENT_CONN_SUPPORT (
   enable     OUT BOOLEAN,
   max_conns  OUT PLS_INTEGER);
```

Parameters

Table 168–31 GET_PERSISTENT_CONN_SUPPORT Procedure Parameters

Parameter	Description
`enable`	`TRUE` if persistent connection support is enabled; otherwise `FALSE`
`max_conns`	the maximum number of persistent connections maintained in the current session

GET_PERSISTENT_CONNS Procedure

This procedure returns all the network connections currently kept persistent by the UTL_HTTP package to the Web servers.

> **See Also:** HTTP Persistent Connections on page 168-15 and HTTP Persistent Connections Subprograms on page 168-28

Syntax

```
UTL_HTTP.get_persistent_conns (
   connections  IN OUT NOCOPY connection_table);
```

Parameters

Table 168–32 GET_PERSISTENT_CONNS Procedure Parameters

Parameter	Description
connections	The network connections kept persistent

Usage Notes

Connections to the same Web server at different TCP/IP ports are counted individually. The host names of the Web servers are identified as specified in the URL of the original HTTP requests. Therefore, fully qualified host names with domain names will be counted differently from the host names without domain names.

GET_PROXY Procedure

This procedure retrieves the current proxy settings.

> **See Also:** Session Settings on page 168-15 and Session Settings Subprograms on page 168-24

Syntax

```
UTL_HTTP.GET_PROXY (
   proxy             OUT NOCOPY VARCHAR2,
   no_proxy_domains  OUT NOCOPY VARCHAR2);
```

Parameters

Table 168–33 GET_PROXY Procedure Parameters

Parameter	Description
`proxy`	The proxy (host and an optional port number) currently used by the `UTL_HTTP` package
`no_proxy_domains`	The list of hosts and domains for which no proxy is used for all requests

GET_RESPONSE Function

This function reads the HTTP response. When the function returns, the status line and the HTTP response headers have been read and processed. The status code, reason phrase, and the HTTP protocol version are stored in the response record. This function completes the HTTP headers section.

> **See Also:** HTTP Responses on page 168-15 and HTTP Responses Subprograms on page 168-26

Syntax

```
UTL_HTTP.GET_RESPONSE (
   r  IN OUT NOCOPY req)
RETURN resp;
```

Parameters

Table 168–34 GET_RESPONSE Procedure Parameters

Parameter	Description
r	The HTTP response

GET_RESPONSE_ERROR_CHECK Procedure

This procedure checks if the response error check is set or not.

> **See Also:** Session Settings on page 168-15 and Session Settings Subprograms on page 168-24

Syntax

```
UTL_HTTP.GET_RESPONSE_ERROR_CHECK (
   enable  OUT BOOLEAN);
```

Parameters

Table 168–35 GET_RESPONSE_ERROR_CHECK Procedure Parameters

Parameter	Description
`enable`	`TRUE` if the response error check is set; otherwise `FALSE`

GET_TRANSFER_TIMEOUT Procedure

This procedure retrieves the default timeout value for all future HTTP requests.

> **See Also:** Session Settings on page 168-15 and Session Settings Subprograms on page 168-24

Syntax

```
UTL_HTTP.GET_TRANSFER_TIMEOUT (
   timeout  OUT PLS_INTEGER);
```

Parameters

Table 168–36 GET_TRANSFER_TIMEOUT Procedure Parameters

Parameter	Description
`timeout`	The network transfer timeout value in seconds

READ_LINE Procedure

This procedure reads the HTTP response body in text form until the end of line is reached and returns the output in the caller-supplied buffer. The end of line is as defined in the function `read_line` of UTL_TCP. The `end_of_body` exception will be raised if the end of the HTTP response body is reached. Text data is automatically converted from the response body character set to the database character set.

> **See Also:** HTTP Responses on page 168-15 and HTTP Responses Subprograms on page 168-26

Syntax

```
UTL_HTTP.READ_LINE(
   r            IN OUT NOCOPY resp,
   data         OUT NOCOPY  VARCHAR2 CHARACTER SET ANY_CS,
   remove_crlf  IN  BOOLEAN DEFAULT FALSE);
```

Parameters

Table 168–37 READ_LINE Procedure Parameters

Parameter	Description
`r`	The HTTP response
`data`	The HTTP response body in text form
`remove_crlf`	Removes the newline characters if set to `TRUE`

Usage Notes

The `UTL_HTTP` package supports HTTP 1.1 chunked transfer-encoding. When the response body is returned in chunked transfer-encoding format as indicated in the response header, the package automatically decodes the chunks and returns the response body in de-chunked format.

If transfer timeout is set in the request of this response, `read_line` waits for each data packet to be ready to read until timeout occurs. If it occurs, this procedure stops reading and returns all the data read successfully. If no data is read successfully, the `transfer_timeout` exception is raised. The exception can be handled and the read operation can be retried later.

If a partial multibyte character is found at the end of the response body, `read_line` stops reading and returns all the complete multibyte characters read successfully. If no complete character is read successfully, the `partial_multibyte_char` exception is raised. The exception can be handled and the bytes of that partial multibyte character can be read as binary by the `read_raw` procedure. If a partial multibyte character is seen in the middle of the response body because the remaining bytes of the character have not arrived and read timeout occurs, the `transfer_timeout` exception is raised instead. The exception can be handled and the read operation can be retried later.

When the "Content-Type" response header specifies the character set of the response body and the character set is unknown or unsupported by Oracle, the "ORA-01482: unsupported character set" exception is raised if you try to read the response body as text. You can either read the response body as binary using the `READ_RAW` procedure, or set the character set of the response body explicitly using the `SET_BODY_CHARSET` procedure and read the response body as text again.

READ_RAW Procedure

This procedure reads the HTTP response body in binary form and returns the output in the caller-supplied buffer. The `end_of_body` exception will be raised if the end of the HTTP response body is reached.

> **See Also:** HTTP Responses on page 168-15 and HTTP Responses Subprograms on page 168-26

Syntax

```
UTL_HTTP.READ_RAW(
   r     IN OUT NOCOPY resp,
   data  OUT NOCOPY RAW,
   len   IN PLS_INTEGER DEFAULT NULL);
```

Parameters

Table 168–38 READ_RAW Procedure Parameters

Parameter	Description
`r`	The HTTP response
`data`	The HTTP response body in binary form
`len`	The number of bytes of data to read. If `len` is `NULL`, this procedure will read as much input as possible to fill the buffer allocated in `data`. The actual amount of data returned may be less than that specified if not much data is available before the end of the HTTP response body is reached or the `transfer_timeout` amount of time has elapsed. The default is `NULL`

Usage Notes

The `UTL_HTTP` package supports HTTP 1.1 chunked transfer-encoding. When the response body is returned in chunked transfer-encoding format as indicated in the response header, the package automatically decodes the chunks and returns the response body in de-chunked format.

If transfer timeout is set in the request of this response, `read_raw` waits for each data packet to be ready to read until timeout occurs. If it occurs, `read_raw` stops reading and returns all the data read successfully. If no data is read successfully, the `transfer_timeout` exception is raised. The exception can be handled and the read operation can be retried later.

READ_TEXT Procedure

This procedure reads the HTTP response body in text form and returns the output in the caller-supplied buffer. The `end_of_body` exception will be raised if the end of the HTTP response body is reached. Text data is automatically converted from the response body character set to the database character set.

See Also: HTTP Responses on page 168-15 and HTTP Responses Subprograms on page 168-26

Syntax

```
UTL_HTTP.READ_TEXT(
   r     IN OUT NOCOPY resp,
   data  OUT NOCOPY VARCHAR2 CHARACTER SET ANY_CS,
   len   IN PLS_INTEGER DEFAULT NULL);
```

Parameters

Table 168–39 READ_TEXT Procedure Parameters

Parameter	Description
`r`	The HTTP response
`data`	The HTTP response body in text form
`len`	The maximum number of characters of data to read. If `len` is NULL, this procedure will read as much input as possible to fill the buffer allocated in `data`. The actual amount of data returned may be less than that specified if little data is available before the end of the HTTP response body is reached or the `transfer_timeout` amount of time has elapsed. The default is NULL.

Usage Notes

The `UTL_HTTP` package supports HTTP 1.1 chunked transfer-encoding. When the response body is returned in chunked transfer-encoding format as indicated in the response header, the package automatically decodes the chunks and returns the response body in de-chunked format.

If transfer timeout is set in the request of this response, `read_text` waits for each data packet to be ready to read until timeout occurs. If it occurs, this procedure stops reading and returns all the data read successfully. If no data is read successfully, the `transfer_timeout` exception is raised. The exception can be handled and the read operation can be retried later.

If a partial multibyte character is found at the end of the response body, `read_text` stops reading and returns all the complete multibyte characters read successfully. If no complete character is read successfully, the `partial_multibyte_char` exception is raised. The exception can be handled and the bytes of that partial multibyte character can be read as binary by the `read_raw` procedure. If a partial multibyte character is seen in the middle of the response body because the remaining bytes of the character have not arrived and read timeout occurs, the `transfer_timeout` exception is raised instead. The exception can be handled and the read operation can be retried later.

When the "Content-Type" response header specifies the character set of the response body and the character set is unknown or unsupported by Oracle, the "`ORA-01482:`

unsupported character set" exception is raised if you try to read the response body as text. You can either read the response body as binary using the READ_RAW procedure, or set the character set of the response body explicitly using the SET_BODY_CHARSET procedure and read the response body as text again.

REQUEST Function

This function returns up to the first 2000 bytes of data retrieved from the given URL. This function can be used directly in SQL queries. The URL may contain the username and password needed to authenticate the request to the server. The format is

```
scheme://[user[:password]@]host[:port]/[...]
```

You can define a username/password for the proxy to be specified in the proxy string. The format is

```
[http://][user[:password]@]host[:port][/]
```

> **See Also:** Simple HTTP Fetches on page 168-14 and Simple HTTP Fetches in a Single Call Subprograms on page 168-23

Syntax

```
UTL_HTTP.REQUEST (
   url              IN VARCHAR2,
   proxy            IN VARCHAR2 DEFAULT NULL,
   wallet_path      IN VARCHAR2 DEFAULT NULL,
   wallet_password  IN VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

Pragmas

```
pragma restrict_references (request, wnds, rnds, wnps, rnps);
```

Parameters

Table 168–40 REQUEST Function Parameters

Parameter	Description
`url`	Uniform resource locator
`proxy`	(Optional) Specifies a proxy server to use when making the HTTP request. See `SET_PROXY` for the full format of the proxy setting.
`wallet_path`	(Optional) Specifies a client-side wallet. The client-side wallet contains the list of trusted certificate authorities required for HTTPS request. The format of `wallet_path` on a PC is, for example, `file:c:\WINNT\Profiles\<username>\WALLETS`, and in Unix is, for example, `file:/home/<username>/wallets` When the `UTL_HTTP` package is executed in the Oracle database server, the wallet is accessed from the database server. Therefore, the wallet path must be accessible from the database server. See `SET_WALLET` for a description on how to set up an Oracle wallet. Non-HTTPS requests do not require an Oracle wallet.
`wallet_password`	(Optional) Specifies the password required to open the wallet

Return Values

The return type is a string of length 2000 or less, which contains up to the first 2000 bytes of the HTML result returned from the HTTP request to the argument URL.

Exceptions

```
INIT_FAILED
REQUEST_FAILED
```

Usage Notes

The URL passed as an argument to this function is not examined for illegal characters, for example, spaces, according to URL specification RFC 2396. The caller should escape those characters with the UTL_URL package. See the comments of the package for the list of legal characters in URLs. Note that URLs should consist of US-ASCII characters only. The use of non-US-ASCII characters in a URL is generally unsafe.

Please see the documentation of the function `SET_WALLET` on the use of an Oracle wallet, which is required for accessing HTTPS Web servers.

Unless response error check is turned on, this function does not raise an exception when a 4xx or 5xx response is received from the Web server. Instead, it returns the formatted error message from the Web server:

```
<HTML>
<HEAD>
<TITLE>Error Message</TITLE>
</HEAD>
<BODY>
<H1>Fatal Error 500</H1>
Can't Access Document:  http://home.nothing.comm.
<P>
<B>Reason:</B> Can't locate remote host:  home.nothing.comm.
<P>
<P><HR>
<ADDRESS><A HREF="http://www.w3.org">
CERN-HTTPD3.0A</A></ADDRESS>
</BODY>
</HTML>
```

Examples

```
SQLPLUS> SELECT UTL_HTTP.REQUEST('http://www.my-company.com/') FROM DUAL;
UTL_HTTP.REQUEST('HTTP://WWW.MY-COMPANY.COM/')
<html>
<head><title>My Company Home Page</title>
<!--changed Jan. 16, 19
1 row selected.
```

If you are behind a firewall, include the `proxy` parameter. For example, from within the Oracle firewall, where there might be a proxy server named `www-proxy.my-company.com`:

```
SQLPLUS> SELECT
UTL_HTTP.REQUEST('http://www.my-company.com', 'www-proxy.us.my-company.com') FROM
DUAL;
```

REQUEST_PIECES Function

This function returns a PL/SQL table of 2000-byte pieces of the data retrieved from the given URL. You can define a username/password for the proxy to be specified in the proxy string. The format is

```
[http://][user[:password]@]host[:port][/]
```

> **See Also:** Simple HTTP Fetches on page 168-14 and Simple HTTP Fetches in a Single Call Subprograms on page 168-23

Syntax

```
TYPE html_pieces IS TABLE OF VARCHAR2(2000) INDEX BY BINARY_INTEGER;

UTL_HTTP.REQUEST_PIECES (
   url             IN VARCHAR2,
   max_pieces      IN NATURAL DEFAULT 32767,
   proxy           IN VARCHAR2 DEFAULT NULL,
   wallet_path     IN VARCHAR2 DEFAULT NULL,
   wallet_password IN VARCHAR2 DEFAULT NULL)
RETURN html_pieces;
```

Pragmas

```
PRAGMA RESTRICT_REFERENCES (request_pieces, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 168–41 REQUEST_PIECES Function Parameters

Parameter	Description
`url`	Uniform resource locator
`max_pieces`	(Optional) The maximum number of pieces (each 2000 characters in length, except for the last, which may be shorter), that `REQUEST_PIECES` should return. If provided, then that argument should be a positive integer.
`proxy`	(Optional) Specifies a proxy server to use when making the HTTP request. See `SET_PROXY` for the full format of the proxy setting.
`wallet_path`	(Optional) Specifies a client-side wallet. The client-side wallet contains the list of trusted certificate authorities required for HTTPS request. The format of wallet_path on a PC is, for example, `file:c:\WINNT\Profiles\<username>\WALLETS`, and in Unix is, for example, file:/home/<username>/wallets. When the `UTL_HTTP` package is executed in the Oracle database server, the wallet is accessed from the database server. Therefore, the wallet path must be accessible from the database server. See `SET_WALLET` for the description on how to set up an Oracle wallet. Non-HTTPS requests do not require an Oracle wallet.
`wallet_password`	(Optional) Specifies the password required to open the wallet

Return Values

REQUEST_PIECES returns a PL/SQL table of type UTL_HTTP.HTML_PIECES. Each element of that PL/SQL table is a string of maximum length 2000. The elements of the PL/SQL table returned by REQUEST_PIECES are successive pieces of the data obtained from the HTTP request to that URL.

Exceptions

```
INIT_FAILED
REQUEST_FAILED
```

Usage Notes

The URL passed as an argument to this function will not be examined for illegal characters, for example, spaces, according to URL specification RFC 2396. The caller should escape those characters with the UTL_URL package. See the comments of the package for the list of legal characters in URLs. Note that URLs should consist of US-ASCII characters only. The use of non-US-ASCII characters in a URL is generally unsafe.

Each entry of the PL/SQL table (the "pieces") returned by this function may not be filled to their fullest capacity. The function may start filling the data in the next piece before the previous "piece" is totally full.

Please see the documentation of the function SET_WALLET on the use of an Oracle wallet, which is required for accessing HTTPS Web servers.

Unless response error check is turned on, this function does not raise an exception when a 4xx or 5xx response is received from the Web server. Instead, it returns the formatted error message from the Web server:

```
<HTML>
<HEAD>
<TITLE>Error Message</TITLE>
</HEAD>
<BODY>
<H1>Fatal Error 500</H1>
Can't Access Document:  http://home.nothing.comm.
<P>
<B>Reason:</B> Can't locate remote host:  home.nothing.comm.
<P>
<P><HR>
<ADDRESS><A HREF="http://www.w3.org">
CERN-HTTPD3.0A</A></ADDRESS>
</BODY>
</HTML>
```

Examples

```
SET SERVEROUTPUT ON

DECLARE
   x   UTL_HTTP.HTML_PIECES;
   len PLS_INTEGER;
BEGIN
   x := UTL_HTTP.REQUEST_PIECES('http://www.oracle.com/', 100);
   DBMS_OUTPUT.PUT_LINE(x.count || ' pieces were retrieved.');
   DBMS_OUTPUT.PUT_LINE('with total length ');
   IF x.count < 1 THEN
      DBMS_OUTPUT.PUT_LINE('0');
```

```
  ELSE
   len := 0;
   FOR i in 1..x.count LOOP
      len := len + length(x(i));
   END LOOP;
   DBMS_OUTPUT.PUT_LINE(i);
  END IF;
END;
/
-- Output
Statement processed.
4 pieces were retrieved.
with total length
7687
```

SET_AUTHENTICATION Procedure

This procedure sets HTTP authentication information in the HTTP request header. The Web server needs this information to authorize the request.

> **See Also:** HTTP Requests on page 168-15 and HTTP Requests Subprograms on page 168-25

Syntax

```
UTL_HTTP.SET_AUTHENTICATION(
   r         IN OUT NOCOPY req,
   username  IN VARCHAR2,
   password  IN VARCHAR2,
   scheme    IN VARCHAR2 DEFAULT 'Basic',
   for_proxy IN BOOLEAN  DEFAULT FALSE);
```

Parameters

Table 168–42 SET_AUTHENTICATION Procedure Parameters

Parameter	Description
`r`	The HTTP request
`username`	The username for the HTTP authentication
`password`	The password for the HTTP authentication
`scheme`	The HTTP authentication scheme. The default, `BASIC`, denotes the HTTP Basic Authentication scheme.
`for_proxy`	Identifies if the HTTP authentication information is for access to the HTTP proxy server instead of the Web server. Default is `FALSE`.

Usage Notes

Only the HTTP Basic Authentication scheme is supported.

SET_BODY_CHARSET Procedures

This procedure is overloaded. The description of different functionality is located alongside the syntax declarations.

See Also:

- HTTP Responses on page 168-15 and HTTP Responses Subprograms on page 168-26
- Session Settings on page 168-15 and Session Settings Subprograms on page 168-24

Syntax

Sets the default character set of the body of all future HTTP requests when the media type is `text` and the character set is not specified in the `Content-Type` header. Following the HTTP protocol standard specification, if the media type of a request or a response is `text`, but the character set information is missing in the `Content-Type` header, the character set of the request or response body should default to `ISO-8859-1`. A response created for a request inherits the default body character set of the request instead of the body character set of the current session. The default body character set is ISO-8859-1 in a database user session. The default body character set setting affects only future requests and has no effect on existing requests. After a request is created, the body character set can be changed by using the other `SET_BODY_CHARSET` procedure that operates on a request:

```
UTL_HTTP.SET_BODY_CHARSET (
   charset  IN VARCHAR2 DEFAULT NULL);
```

Sets the character set of the request body when the media type is `text` but the character set is not specified in the `Content-Type` header. According to the HTTP protocol standard specification, if the media type of a request or a response is "text" but the character set information is missing in the `Content-Type` header, the character set of the request or response body should default to "`ISO-8859-1`". Use this procedure to change the default body character set a request inherits from the session default setting:

```
UTL_HTTP.SET_BODY_CHARSET(
   r        IN OUT NOCOPY req,
   charset  IN VARCHAR2 DEFAULT NULL);
```

Sets the character set of the response body when the media type is "text" but the character set is not specified in the "Content-Type" header. For each the HTTP protocol standard specification, if the media type of a request or a response is "text" but the character set information is missing in the "Content-Type" header, the character set of the request or response body should default to "ISO-8859-1". Use this procedure to change the default body character set a response inherits from the request:

```
UTL_HTTP.SET_BODY_CHARSET(
   r        IN OUT NOCOPY resp,
   charset  IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 168–43 SET_BODY_CHARSET Procedure Parameters

Parameter	Description
r	The HTTP response.
charset	The default character set of the response body. The character set can be in Oracle or Internet Assigned Numbers Authority (IANA) naming convention. If charset is NULL, the database character set is assumed.

SET_COOKIE_SUPPORT Procedures

This procedure is overloaded. The description of different functionality is located alongside the syntax declarations.

This procedure

See Also:

- HTTP Requests on page 168-15 andHTTP Requests Subprograms on page 168-25
- Session Settings on page 168-15 and Session Settings Subprograms on page 168-24

Syntax

Enables or disables support for the HTTP cookies in the request. Use this procedure to change the cookie support setting a request inherits from the session default setting:

```
UTL_HTTP.SET_COOKIE_SUPPORT(
   r       IN OUT NOCOPY REQ,
   enable  IN BOOLEAN DEFAULT TRUE);
```

Sets whether or not future HTTP requests will support HTTP cookies, and the maximum number of cookies maintained in the current database user session:

```
UTL_HTTP.SET_COOKIE_SUPPORT (
   enable       IN BOOLEAN,
   max_cookies  IN PLS_INTEGER DEFAULT 300,
   max_cookies_per_site  IN PLS_INTEGER DEFAULT 20);
```

Parameters

Table 168–44 SET_COOKIE_SUPPORT Procedure Parameters

Parameter	Description
`r`	The HTTP request
`enable`	Set enable to `TRUE` to enable HTTP cookie support; `FALSE` to disable
`max_cookies`	Sets the maximum total number of cookies maintained in the current session
`max_cookies_per_site`	Sets the maximum number of cookies maintained in the current session for each Web site

Usage Notes

If cookie support is enabled for an HTTP request, all cookies saved in the current session and applicable to the request are returned to the Web server in the request in accordance with HTTP cookie specification standards. Cookies set in the response to the request are saved in the current session for return to the Web server in the subsequent requests if cookie support is enabled for those requests. If the cookie support is disabled for an HTTP request, no cookies are returned to the Web server in the request and the cookies set in the response to the request are not saved in the current session, although the `Set-Cookie` HTTP headers can still be retrieved from the response.

Cookie support is enabled by default for all HTTP requests in a database user session. The default setting of the cookie support (enabled versus disabled) affects only the future requests and has no effect on the existing ones. After your request is created, the cookie support setting may be changed by using the other `SET_COOKIE_SUPPORT` procedure that operates on a request.

The default maximum number of cookies saved in the current session is 20 for each site and 300 total.

If you lower the maximum total number of cookies or the maximum number of cookies for each Web site, the oldest cookies will be purged first to reduce the number of cookies to the lowered maximum. HTTP cookies saved in the current session last for the duration of the database session only; there is no persistent storage for the cookies. Cookies saved in the current session are not cleared if you disable cookie support.

See "Examples" on page 168-19 for how to use `GET_COOKIES` and `ADD_COOKIES` to retrieve, save, and restore cookies.

SET_DETAILED_EXCP_SUPPORT Procedure

This procedure sets the UTL_HTTP package to raise a detailed exception. By default, UTL_HTTP raises the request_failed exception when an HTTP request fails. Use GET_DETAILED_SQLCODE and GET_DETAILED_SQLEERM for more detailed information about the error.

See Also: Session Settings on page 168-15 and Session Settings Subprograms on page 168-24

Syntax

```
UTL_HTTP.SET_DETAILED_EXCP_SUPPORT (
   enable  IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 168–45 SET_DETAILED_EXCP_SUPPORT Procedure Parameters

Parameter	Description
enable	Asks UTL_HTTP to raise a detailed exception directly if set to TRUE; otherwise FALSE

SET_FOLLOW_REDIRECT Procedures

This procedure sets the maximum number of times UTL_HTTP follows the HTTP redirect instruction in the HTTP response to this request, or future requests, in the GET_RESPONSE function.

See Also:

- HTTP Requests on page 168-15 and HTTP Requests Subprograms on page 168-25
- Session Settings on page 168-15 and Session Settings Subprograms on page 168-24

Syntax

Use this procedure to set the maximum number of redirections:

```
UTL_HTTP.SET_FOLLOW_REDIRECT (
   max_redirects  IN PLS_INTEGER DEFAULT 3);
```

Use this procedure to change the maximum number of redirections a request inherits from the session default setting:

```
UTL_HTTP.SET_FOLLOW_REDIRECT(
   r              IN OUT NOCOPY req,
   max_redirects  IN PLS_INTEGER DEFAULT 3);
```

Parameters

Table 168–46 SET_FOLLOW_REDIRECT Procedure Parameters

Parameter	Description
r	The HTTP request
max_redirects	The maximum number of redirects. Set to zero to disable redirects.

Usage Notes

If max_redirects is set to a positive number, the GET_RESPONSE Function will automatically follow the redirected URL for the HTTP response status code 301, 302, and 307 for the HTTP HEAD and GET methods, and 303 for all HTTP methods, and retry the HTTP request (the request method will be changed to HTTP GET for the status code 303) at the new location. It follows the redirection until the final, non-redirect location is reached, or an error occurs, or the maximum number of redirections has been reached (to prevent an infinite loop). The URL and method fields in the REQ record will be updated to the last redirected URL and the method used to access the URL. Set the maximum number of redirects to zero to disable automatic redirection.

While it is set not to follow redirect automatically in the current session, it is possible to specify individual HTTP requests to follow redirect instructions the function FOLLOW_REDIRECT and vice versa.

The default maximum number of redirections in a database user session is 3. The default value affects only future requests and has no effect on existing requests.

The SET_FOLLOW_REDIRECT procedure must be called before GET_RESPONSE for any redirection to take effect.

SET_HEADER Procedure

This procedure sets an HTTP request header. The request header is sent to the Web server as soon as it is set.

> **See Also:** HTTP Requests on page 168-15 and HTTP Requests Subprograms on page 168-25

Syntax

```
UTL_HTTP.SET_HEADER (
   r      IN OUT NOCOPY req,
   name   IN VARCHAR2,
   value  IN VARCHAR2);
```

Parameters

Table 168–47 SET_HEADER Procedure Parameters

Parameter	Description
r	The HTTP request
name	The name of the HTTP request header
value	The value of the HTTP request header

Usage Notes

Multiple HTTP headers with the same name are allowed in the HTTP protocol standard. Therefore, setting a header does not replace a prior header with the same name.

If the request is made using HTTP 1.1, UTL_HTTP sets the Host header automatically for you.

When you set the Content-Type header with this procedure, UTL_HTTP looks for the character set information in the header value. If the character set information is present, it is set as the character set of the request body. It can be overridden later by using the SET_BODY_CHARSET procedure.

When you set the Transfer-Encoding header with the value chunked, UTL_HTTP automatically encodes the request body written by the WRITE_TEXT, WRITE_LINE and WRITE_RAW procedures. Note that some HTTP-1.1-based Web servers or CGI programs do not support or accept the request body encoding in the HTTP 1.1 chunked transfer-encoding format.

SET_PERSISTENT_CONN_SUPPORT Procedure

This procedure enables or disables support for the HTTP 1.1 persistent-connection in the request.

> **See Also:** HTTP Requests on page 168-15 and HTTP Requests Subprograms on page 168-25

Syntax

```
UTL_HTTP.SET_PERSISTENT_CONN_SUPPORT(
   r       IN OUT NOCOPY req,
   enable  IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 168–48 SET_PERSISTENT_CONN_SUPPORT Procedure Parameters

Parameter	Description
r	The HTTP request
enable	TRUE to keep the network connection persistent. FALSE otherwise.

Usage Notes

If the persistent-connection support is enabled for an HTTP request, the package will keep the network connections to a Web server or the proxy server open in the package after the request is completed properly for a subsequent request to the same server to reuse for each HTTP 1.1 protocol specification. With the persistent connection support, subsequent HTTP requests may be completed faster because the network connection latency is avoided. If the persistent-connection support is disabled for a request, the package will always send the HTTP header "Connection: close" automatically in the HTTP request and close the network connection when the request is completed. This setting has no effect on HTTP requests that follows HTTP 1.0 protocol, for which the network connections will always be closed after the requests are completed.

When a request is being made, the package attempts to reuse an existing persistent connection to the target Web server (or proxy server) if one is available. If none is available, a new network connection will be initiated. The persistent-connection support setting for a request affects only whether the network connection should be closed after a request completes.

Use this procedure to change the persistent-connection support setting a request inherits from the session default setting.

Users should note that while the use of persistent connections in UTL_HTTP may reduce the time it takes to fetch multiple Web pages from the same server, it consumes precious system resources (network connections) in the database server. Also, excessive use of persistent connections may reduce the scalability of the database server when too many network connections are kept open in the database server. Network connections should be kept open only if they will be used immediately by subsequent requests and should be closed immediately when they are no longer needed. Set the default persistent connection support as disabled in the session, and enable persistent connection in individual HTTP requests as shown in "Examples" on page 168-80.

The default value of the maximum number of persistent connections in a database session is zero. To truly enable persistent connections, you must also set the maximum number of persistent connections to a positive value or no connections will be kept persistent.

Examples

Using SET_PERSISTENT_CONN_SUPPORT in HTTP Requests

```
DECLARE
  TYPE vc2_table IS TABLE OF VARCHAR2(256) INDEX BY BINARY_INTEGER;
  paths VC2_TABLE;

UTL_HTTP.fetch_pages(paths IN vc2_table) AS
    url_prefix   VARCHAR2(256) := 'http://www.my-company.com/';
    req          UTL_HTTP.REQ;
    resp         UTL_HTTP.RESP;
    data  VARCHAR2(1024);
  BEGIN
    FOR i IN 1..paths.count LOOP
      req := UTL_HTTP.BEGIN_REQUEST(url_prefix || paths(i));
      -- Use persistent connection except for the last request
      IF (i < paths.count) THEN
        UTL_HTTP.SET_PERSISTENT_CONN_SUPPORT(req, TRUE);
      END IF;
      resp := UTL_HTTP.GET_RESPONSE(req);
      BEGIN
        LOOP
          UTL_HTTP.READ_TEXT(resp, data);
          -- do something with the data
        END LOOP;
      EXCEPTION
        WHEN UTL_HTTP.END_OF_BODY THEN
          NULL;
      END;
      UTL_HTTP.END_RESPONSE(resp);
    END LOOP;
  END;

BEGIN
  UTL_HTTP.SET_PERSISTENT_CONN_SUPPORT(FALSE, 1);
  paths(1) := '...';
  paths(2) := '...';
  ...
  fetch_pages(paths);
END;
```

SET_PROXY Procedure

This procedure sets the proxy to be used for requests of the HTTP or other protocols, excluding those for hosts that belong to the domain specified in `no_proxy_domains`. `no_proxy_domains` is a comma-, semi-colon-, or space-separated list of domains or hosts for which HTTP requests should be sent directly to the destination HTTP server instead of going through a proxy server.

> **See Also:** Session Settings on page 168-15 and Session Settings Subprograms on page 168-24

Syntax

```
UTL_HTTP.SET_PROXY (
   proxy             IN VARCHAR2,
   no_proxy_domains  IN VARCHAR2);
```

Parameters

Table 168–49 SET_PROXY Procedure Parameters

Parameter	Description
`proxy`	The proxy (host and an optional port number) to be used by the `UTL_HTTP` package
`no_proxy_domains`	The list of hosts and domains for which no proxy should be used for all requests

Usage Notes

The proxy may include an optional TCP/IP port number at which the proxy server listens. The syntax is `[http://]host[:port][/]`, for example, `www-proxy.my-company.com:80`. If the port is not specified for the proxy, port 80 is assumed.

Optionally, a port number can be specified for each domain or host. If the port number is specified, the no-proxy restriction is only applied to the request at the port of the particular domain or host, for example, `corp.my-company.com`, `eng.my-company.com:80`. When `no_proxy_domains` is `NULL` and the proxy is set, all requests go through the proxy. When the proxy is not set, `UTL_HTTP` sends requests to the target Web servers directly.

You can define a username/password for the proxy to be specified in the proxy string. The format is

```
[http://][user[:password]@]host[:port][/]
```

If proxy settings are set when the database server instance is started, the proxy settings in the environment variables `http_proxy` and `no_proxy` are assumed. Proxy settings set by this procedure override the initial settings.

SET_RESPONSE_ERROR_CHECK Procedure

This procedure sets whether or not GET_RESPONSE raises an exception when the Web server returns a status code that indicates an error—a status code in the 4xx or 5xx ranges. For example, when the requested URL is not found in the destination Web server, a 404 (document not found) response status code is returned.

See Also: Session Settings on page 168-15 and Session Settings Subprograms on page 168-24

Syntax

```
UTL_HTTP.SET_RESPONSE_ERROR_CHECK (
   enable  IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 168–50 SET_RESPONSE_ERROR_CHECK Procedure Parameters

Parameter	Description
enable	TRUE to check for response errors; otherwise FALSE

Usage Notes

If the status code indicates an error—a 4xx or 5xx code—and this procedure is enabled, GET_RESPONSE will raise the HTTP_CLIENT_ERROR or HTTP_SERVER_ERROR exception. If SET_RESPONSE_ERROR_CHECK is set to FALSE, GET_RESPONSE will not raise an exception when the status code indicates an error.

Response error check is turned off by default.

The GET_RESPONSE function can raise other exceptions when SET_RESPONSE_ERROR_CHECK is set to FALSE.

SET_TRANSFER_TIMEOUT Procedure

This procedure sets the default time out value for all future HTTP requests that the UTL_HTTP package should attempt while reading the HTTP response from the Web server or proxy server. This time out value may be used to avoid the PL/SQL programs from being blocked by busy Web servers or heavy network traffic while retrieving Web pages from the Web servers.

> **See Also:** Session Settings on page 168-15 and Session Settings Subprograms on page 168-24

Syntax

```
UTL_HTTP.SET_TRANSFER_TIMEOUT (
   timeout  IN PLS_INTEGER DEFAULT 60);
```

Parameters

Table 168–51 SET_TRANSFER_TIMEOUT Procedure Parameters

Parameter	Description
timeout	The network transfer timeout value in seconds.

Usage Notes

The default value of the time out is 60 seconds.

SET_WALLET Procedure

This procedure sets the Oracle wallet used for all HTTP requests over Secured Socket Layer (SSL), namely HTTPS. When the UTL_HTTP package communicates with an HTTP server over SSL, the HTTP server presents its digital certificate, which is signed by a certificate authority, to the UTL_HTTP package for identification purpose. The Oracle wallet contains the list of certificate authorities that are trusted by the user of the UTL_HTTP package. An Oracle wallet is required to make an HTTPS request.

See Also: Session Settings on page 168-15 and Session Settings Subprograms on page 168-24

Syntax

```
UTL_HTTP.SET_WALLET (
   path      IN VARCHAR2,
   password  IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 168–52 SET_WALLET Procedure Parameters

Parameter	Description
path	The directory path that contains the Oracle wallet. The format is file:<directory-path>. The format of wallet_path on a PC is, for example, file:c:\WINNT\Profiles\<username>\WALLETS, and in Unix is, for example, file:/home/<username>/wallets. When the UTL_HTTP package is executed in the Oracle database server, the wallet is accessed from the database server. Therefore, the wallet path must be accessible from the database server.
password	The password needed to open the wallet. A second copy of a wallet in a wallet directory that may be opened without a password. That second copy of the wallet is read-only. If the password is NULL, the UTL_HTTP package will open the second, read-only copy of the wallet instead.

Usage Notes

To set up an Oracle wallet, use the Oracle Wallet Manager to create a wallet. In order for the HTTPS request to succeed, the certificate authority that signs the certificate of the remote HTTPS Web server must be a trust point set in the wallet.

When a wallet is created, it is populated with a set of well-known certificate authorities as trust points. If the certificate authority that signs the certificate of the remote HTTPS Web server is not among the trust points, or the certificate authority has new root certificates, you should obtain the root certificate of that certificate authority and install it as a trust point in the wallet using Oracle Wallet Manager

See Also: *Oracle Database Advanced Security Administrator's Guide* for more information on Wallet Manager

WRITE_LINE Procedure

This procedure writes a text line in the HTTP request body and ends the line with new-line characters (CRLF as defined in UTL_TCP). As soon as some data is sent as the HTTP request body, the HTTP request headers section is completed. Text data is automatically converted from the database character set to the request body character set.

See Also: HTTP Requests on page 168-15 and HTTP Requests Subprograms on page 168-25

Syntax

```
UTL_HTTP.WRITE_LINE(
   r     IN OUT NOCOPY req,
   data  IN VARCHAR2 CHARACTER SET ANY_CS);
```

Parameters

Table 168–53 WRITE_LINE Procedure Parameters

Parameter	Description
r	The HTTP request
data	The text line to send in the HTTP request body

Usage Notes

An HTTP client must always let the remote Web server know the length of the request body it is sending. If the amount of data is known beforehand, you can set the Content-Length header in the request, where the length of the content is measured in bytes instead of characters. If the length of the request body is not known beforehand, you can send the request body using the HTTP 1.1 chunked transfer-encoding format. The request body is sent in chunks, where the length of each chunk is sent before the chunk is sent. The UTL_HTTP package performs chunked transfer-encoding on the request body transparently when the Transfer-Encoding: chunked header is set. Note that some HTTP-1.1-based Web servers or CGI programs do not support or accept the request body encoding in the HTTP 1.1 chunked transfer-encoding format. See the SET_HEADER procedure for details.

If you send the Content-Length header, you should note that the length specified in the header should be the byte-length of the textual request body after it is converted from the database character set to the request body character set. When either one of the two character sets is a multibyte character set, the precise byte-length of the request body in the request body character set cannot be known beforehand. In this case, you can perform the character set conversion explicitly, determine the byte-length of the results, send the Content-Length header, and the results using the WRITE_RAW procedure to avoid the automatic character set conversion. Or, if the remove Web server or CGI programs allow, you can send the request body using the HTTP 1.1 chunked transfer-encoding format, where UTL_HTTP handles the length of the chunks transparently.

WRITE_RAW Procedure

This procedure writes some binary data in the HTTP request body. As soon as some data is sent as the HTTP request body, the HTTP request headers section is completed.

> **See Also:** HTTP Requests on page 168-15 and HTTP Requests Subprograms on page 168-25

Syntax

```
UTL_HTTP.WRITE_RAW(
   r     IN OUT NOCOPY REQ,
   data  IN            RAW);
```

Parameters

Table 168–54 WRITE_RAW Procedure Parameters

Parameter	Description
r	The HTTP request
data	The binary data to send in the HTTP request body

Usage Notes

An HTTP client must always let the remote Web server know the length of the request body it is sending. If the amount of data is known beforehand, you can set the Content-Length header in the request, where the length of the content is measured in bytes instead of characters. If the length of the request body is not known beforehand, you can send the request body using the HTTP 1.1 chunked transfer-encoding format. The request body is sent in chunks, where the length of each chunk is sent before the chunk is sent. UTL_HTTP performs chunked transfer-encoding on the request body transparently when the Transfer-Encoding:chunked header is set. Note that some HTTP-1.1-based Web servers or CGI programs do not support or accept the request body encoding in the HTTP 1.1 chunked transfer-encoding format. See the SET_HEADER procedure for details.

WRITE_TEXT Procedure

This procedure writes some text data in the HTTP request body. As soon as some data is sent as the HTTP request body, the HTTP request headers section is completed. Text data is automatically converted from the database character set to the request body character set.

> **See Also:** HTTP Requests on page 168-15 and HTTP Requests Subprograms on page 168-25

Syntax

```
UTL_HTTP.WRITE_TEXT(
   r     IN OUT NOCOPY REQ,
   data  IN            VARCHAR2 CHARACTER SET ANY_CS);
```

Parameters

Table 168–55 WRITE_TEXT Procedure Parameters

Parameter	Description
r	The HTTP request
data	The text data to send in the HTTP request body

Usage Notes

An HTTP client must always let the remote Web server know the length of the request body it is sending. If the amount of data is known beforehand, you can set the Content-Length header in the request, where the length of the content is measured in bytes instead of characters. If the length of the request body is not known beforehand, you can send the request body using the HTTP 1.1 chunked transfer-encoding format. The request body is sent in chunks, where the length of each chunk is sent before the chunk is sent. UTL_HTTP performs chunked transfer-encoding on the request body transparently when the `Transfer-Encoding: chunked` header is set. Note that some HTTP-1.1-based Web servers or CGI programs do not support or accept the request body encoding in the HTTP 1.1 chunked transfer-encoding format. See the SET_HEADER procedure for details.

If you send the Content-Length header, you should note that the length specified in the header should be the byte-length of the textual request body after it is converted from the database character set to the request body character set. When either one of the two character sets is a multibyte character set, the precise byte-length of the request body in the request body character set cannot be known beforehand. In this case, you can perform the character set conversion explicitly, determine the byte-length of the results, send the Content-Length header, and the results using the WRITE_RAW procedure to avoid the automatic character set conversion. Or, if the remove Web server or CGI programs allow, you can send the request body using the HTTP 1.1 chunked transfer-encoding format, where UTL_HTTP handles the length of the chunks transparently.

169

UTL_I18N

`UTL_I18N` is a set of services that provides additional globalization functionality for applications written in PL/SQL.

> **See Also:** *Oracle Database Globalization Support Guide*

The chapter contains the following topics:

Using UTL_I18N

This section contains topics which relate to using the `UTL_I18N` package.

Overview

The UTL_I18N PL/SQL package consists of the following categories of services:

- String conversion functions for various datatypes.
- Functions that convert a text string to character references and vice versa.
- Functions that map between Oracle, Java, and ISO languages and territories.
- Functions that map between Oracle, Internet Assigned Numbers Authority (IANA), and e-mail safe character sets.
- A function that returns the Oracle character set name from an Oracle language name.
- A function that performs script transliteration.
- Functions that return the ISO currency code, local time zones, and local languages supported for a given territory.
- Functions that return the most appropriate linguistic sort, a listing of all the applicable linguistic sorts, and the local territories supported for a given language.
- Functions that map between the Oracle full and short language names.
- A function that returns the language translation of a given language and territory name.
- A function that returns a listing of the most commonly used time zones.

Constants

The UTL_I18N package uses the constants shown in Table 169–1.

Table 169–1 UTL_I18N Constants

Constant	Type	Value	Description
GENERIC_CONTEXT	PLS_INTEGER	0	Returns the default character set for general cases.
MAIL_GENERIC	PLS_INTEGER	0	Map from an Oracle character set name to an email safe character set name on a non-Windows platform.
ORACLE_TO_IANA	PLS_INTEGER	0	Map from an Oracle character set name to an IANA character set name.
SHIFT_IN	PLS_INTEGER	0	Used with shift_status. Must be set the first time it is called in piecewise conversion.
IANA_TO_ORACLE	PLS_INTEGER	1	Map from an IANA character set name to an Oracle character set name.
MAIL_CONTEXT	PLS_INTEGER	1	The mapping is between an Oracle character set name and an email safe character set name.
MAIL_WINDOWS	PLS_INTEGER	1	Map from an Oracle character set name to an email safe character set name on a Windows platform.
SHIFT_OUT	PLS_INTEGER	1	
FWKATAKANA_HIRAGANA	VARCHAR2(30)	'fwkatakana_hiragana'	Converts only fullwidth Katakana characters to fullwidth Hiragana characters.
FWKATAKANA_HWKATAKANA	VARCHAR2(30)	'fwkatakana_hwkatakana'	Converts only fullwidth Katakana characters to halfwidth Katakana characters.
HIRAGANA_FWKATAKANA	VARCHAR2(30)	'hiragana_fwkatakana'	Converts only fullwidth Hiragana characters to fullwidth Katakana characters.
HIRAGANA_HWKATAKANA	VARCHAR2(30)	'hiragana_hwkatakana'	Converts only fullwidth Hiragana characters to halfwidth Katakana characters.
HWKATAKANA_FWKATAKANA	VARCHAR2(30)	'hwkatakana_fwkatakana'	Converts only halfwidth Katakana characters to fullwidth Katakana characters.
HWKATAKANA_HIRAGANA	VARCHAR2(30)	'hwkatakana_hiragana'	Converts only halfwidth Katakana characters to fullwidth Hiragana characters.
KANA_FWKATAKANA	VARCHAR2(30)	'kana_fwkatakana'	Converts any type of Kana character to a fullwidth Katakana character.

Table 169–1 (Cont.) UTL_I18N Constants

Constant	Type	Value	Description
KANA_HIRAGANA	VARCHAR2(30)	'kana_ hiragana'	Converts any type of Kana character to a fullwidth Hiragana character.
KANA_ HWKATAKANA	VARCHAR2(30)	'kana_ hwkatakana'	Converts any type of Kana character to a halfwidth Katakana character.

Summary of UTL_I18N Subprograms

Table 169–2 UTL_I18N Package Subprograms

Procedure	Description
ESCAPE_REFERENCE Function on page 169-8	Converts a given text string to its character reference counterparts, for characters that fall outside the document character set.
GET_COMMON_TIME_ZONES Function on page 169-9	Returns the list of common time zone IDs that are independent of the locales.
GET_DEFAULT_CHARSET Function on page 169-10	Returns the default Oracle character set name or the default e-mail safe character set name from an Oracle language name.
GET_DEFAULT_ISO_CURRENCY Function on page 169-11	Returns the default ISO 4217 currency code for the specified territory.
GET_DEFAULT_LINGUISTIC_SORT Function on page 169-12	Returns the default linguistic sort name for the specified language.
GET_LOCAL_LANGUAGES Function on page 169-13	Returns the local language names for the specified territory.
GET_LOCAL_LINGUISTIC_SORTS Function on page 169-14	Returns the local linguistic sort names for the specified language.
GET_LOCAL_TERRITORIES Function on page 169-15	Returns the local territory names for the specified language.
GET_LOCAL_TIME_ZONES Function on page 169-16	Returns the local time zone IDs for the specified territory.
GET_TRANSLATION Function on page 169-18	Returns the translation of the language and territory name in the specified translation language.
MAP_CHARSET Function on page 169-19	■ Maps an Oracle character set name to an IANA character set name. ■ Maps an IANA character set name to an Oracle character set name. ■ Maps an Oracle character set name to an e-mail safe character set name.
MAP_FROM_SHORT_LANGUAGE Function on page 169-21	Maps an Oracle short language name to an Oracle language name.
MAP_LANGUAGE_FROM_ISO Function on page 169-22	Returns an Oracle language name from an ISO locale name.
MAP_LOCALE_TO_ISO Function on page 169-23	Returns an ISO locale name from the Oracle language and territory name.
MAP_TERRITORY_FROM_ISO Function on page 169-24	Returns an Oracle territory name from an ISO locale name.
MAP_TO_SHORT_LANGUAGE Function on page 169-25	Maps an Oracle language name to an Oracle short language name.
RAW_TO_CHAR Functions on page 169-26	Converts `RAW` data that is not encoded in the database character set into a `VARCHAR2` string
RAW_TO_NCHAR Functions on page 169-28	Converts `RAW` data that is not encoded in the national character set into an `NVARCHAR2` string
STRING_TO_RAW Function on page 169-30	Converts a `VARCHAR2` or `NVARCHAR2` string to another character set. The result is returned as a `RAW` datatype.

Table 169–2 (Cont.) UTL_I18N Package Subprograms

Procedure	Description
TRANSLITERATE Function on page 169-31	Transliterates between Japanese hiragana and katakana.
UNESCAPE_REFERENCE Function on page 169-33	Converts an input string that contains character references to a text string.

ESCAPE_REFERENCE Function

This function converts a text string to its character reference counterparts for characters that fall outside the character set used by the current document. Character references are mainly used in HTML and XML documents to represent characters independently of the encoding of the document.

Character references may appear in two forms, numeric character references and character entity references. Numeric character references specify the Unicode code point value of a character, while character entity references use symbolic names to refer to the same character. For example, `&#xe5;` is the numeric character reference for the small letter "a" with a ring above, whereas `&aring;` is the character entity reference for the same character. Character entity references are also used to escape special characters, as an example, `<` represents the < (less than) sign. This is to avoid possible confusion with the beginning of a tag in Markup languages.

Syntax

```
UTL_I18N.ESCAPE_REFERENCE(
    str            IN VARCHAR2 CHARACTER SET ANY_CS,
    page_cs_name   IN VARCHAR2 DEFAULT NULL)
 RETURN VARCHAR2 CHARACTER SET str%CHARSET;
```

Parameters

Table 169–3 ESCAPE_REFERENCE Function Parameters

Parameter	Description
`str`	Specifies the input string
`page_cs_name`	Specifies the character set of the document. If `page_cs_name` is `NULL`, then the database character set is used for `CHAR` data and the national character set is used for `NCHAR` data.

Usage Notes

If the user specifies an invalid character set or a `NULL` string, then the function returns a `NULL` string.

Examples

```
UTL_I18N.ESCAPE_REFERENCE('hello < '||chr(229),'us7ascii')
```

This returns `'hello < &#xe5;'`.

GET_COMMON_TIME_ZONES Function

This function returns a listing of the most commonly used time zones. This list contains a subset of the time zones that are supported in the database.

Syntax

```
UTL_I18N.GET_COMMON_TIME_ZONES
 RETURN STRING_ARRAY;
```

Examples

Returns the list of the most commonly used time zones.

```
DECLARE
   retval UTL_I18N.STRING_ARRAY;
BEGIN
   retval := UTL_I18N.GET_COMMON_TIME_ZONES;
END;
/
```

GET_DEFAULT_CHARSET Function

This function returns the default Oracle character set name or the default e-mail safe character set name from an Oracle language name.

> **See Also:** "MAP_CHARSET Function" on page 169-19 for an explanation of an e-mail safe character set

Syntax

```
UTL_I18N.GET_DEFAULT_CHARSET(
   language  IN VARCHAR2,
   context   IN PLS_INTEGER DEFAULT GENERIC_CONTEXT,
   iswindows IN BOOLEAN DEFAULT FALSE)
 RETURN VARCHAR2;
```

Parameters

Table 169–4 GET_DEFAULT_CHARSET Function Parameters

Parameter	Description
`language`	Specifies a valid Oracle language
`context`	`GENERIC_CONTEXT | MAIL_CONTEXT` `GENERIC_CONTEXT`: Returns the default character set for general cases `MAIL_CONTEXT`: Returns the default e-mail safe character set name
`iswindows`	If `context` is set as `MAIL_CONTEXT`, then `iswindows` should be set to `TRUE` if the platform is Windows and `FALSE` if the platform is not Windows. The default is `FALSE`. `iswindows` has no effect if `context` is set as `GENERIC_CONTEXT`.

Usage Notes

If the user specifies an invalid language name or an invalid flag, then the function returns a `NULL` string.

Examples

GENERIC_CONTEXT, iswindows=FALSE

```
UTL_I18N.GET_DEFAULT_CHARSET('French', UTL_I18N.GENERIC_CONTEXT, FALSE)
```

This returns `'WE8ISO8859P1'`.

MAIL_CONTEXT, iswindows=TRUE

```
UTL_I18N.GET_DEFAULT_CHARSET('French', UTL_I18N.MAIL_CONTEXT, TRUE)
```

This returns 'WE8MSWIN1252'.

MAIL_CONTEXT, iswindows=FALSE

```
UTL_I18N.GET_DEFAULT_CHARSET('French', UTL_I18N.MAIL_CONTEXT, FALSE)
```

This returns 'WE8ISO8859P1'.

GET_DEFAULT_ISO_CURRENCY Function

This function returns the default ISO 4217 currency code for the specified territory.

Syntax

```
UTL_I18N.GET_DEFAULT_ISO_CURRENCY (
   territory    IN VARCHAR2 CHARACTER SET ANY_CS)
RETURN VARCHAR2;
```

Parameters

Table 169–5 GET_DEFAULT_ISO_CURRENCY Function Parameters

Parameter	Description
territory	Specifies a valid Oracle territory. It is case-insensitive.

Usage Notes

If the user specifies an invalid territory name, then the function returns a NULL string.

Examples

Displays the default ISO currency code for China.

```
DECLARE
   retval VARCHAR2(50);
BEGIN
   retval := UTL_I18N.GET_DEFAULT_ISO_CURRENCY('CHINA');
   DBMS_OUTPUT.PUT_LINE(retval);
END;
/
```

GET_DEFAULT_LINGUISTIC_SORT Function

This function returns the most commonly used Oracle linguistic sort for the specified language.

Syntax

```
UTL_I18N.GET_DEFAULT_LINGUISTIC_SORT (
   language  IN VARCHAR2 CHARACTER SET ANY_CS)
RETURN VARCHAR2;
```

Parameters

Table 169–6 GET_DEFAULT_LINGUISTIC_SORT Function Parameters

Parameter	Description
`language`	Specifies a valid Oracle language. It is case-insensitive.

Usage Notes

If the user specifies an invalid language name, then the function returns a `NULL` string.

Examples

Displays the name of the most appropriate linguistic sort name for the language used in the current SQL session.

```
DECLARE
   retval VARCHAR2(50);
BEGIN
  SELECT value INTO retval FROM nls_session_parameters
   WHERE parameter = 'NLS_LANGUAGE';
   retval := UTL_I18N.GET_DEFAULT_LINGUISTIC_SORT(retval);
   DBMS_OUTPUT.PUT_LINE(retval);
END;
/
```

GET_LOCAL_LANGUAGES Function

This function returns the local language names for the specified territory.

Syntax

```
UTL_I18N.GET_LOCAL_LANGUAGES (
   territory    IN VARCHAR2 CHARACTER SET ANY_CS)
RETURN STRING_ARRAY;
```

Parameters

Table 169–7 GET_LOCAL_LANGUAGES Function Parameters

Parameter	Description
territory	Specifies a valid Oracle territory. It is case-insensitive.

Usage Notes

If the user specifies an invalid territory name, then the function returns a NULL string.

Examples

Returns the list of local languages used in Belgium.

```
DECLARE
   retval UTL_I18N.STRING_ARRAY;
   cnt    INTEGER;
BEGIN
   retval   :=  UTL_I18N.GET_LOCAL_LANGUAGES('BELGIUM');
   DBMS_OUTPUT.PUT('Count = ');
   DBMS_OUTPUT.PUT_LINE(retval.LAST);
   cnt := retval.FIRST;
   WHILE cnt IS NOT NULL LOOP
      DBMS_OUTPUT.PUT_LINE(retval(cnt));
      cnt := retval.NEXT(cnt);
   END LOOP;
END;
/
...
Count = 2
DUTCH
FRENCH
```

GET_LOCAL_LINGUISTIC_SORTS Function

This function returns a list of the Oracle linguistic sort names that are appropriate for the specified language. A `BINARY` sort is included for all languages.

Syntax

```
UTL_I18N.GET_LOCAL_LINGUISTIC_SORTS (
   language  IN VARCHAR2 CHARACTER SET ANY_CS)
RETURN STRING_ARRAY;
```

Parameters

Table 169–8 GET_LOCAL_LINGUISTIC_SORTS Function Parameters

Parameter	Description
`language`	Specifies a valid Oracle language. It is case-insensitive.

Usage Notes

If the user specifies an invalid language name, then the function returns a `NULL` string.

Examples

Displays the local linguistic sort names for `JAPANESE`.

```
DECLARE
   retval UTL_I18N.STRING_ARRAY;
   cnt INTEGER;
BEGIN
  retval := UTL_I18N.GET_LOCAL_LINGUISTIC_SORTS('Japanese');
  DBMS_OUTPUT.PUT('Count = ');
  DBMS_OUTPUT.PUT_LINE(retval.COUNT);
  cnt := retval.FIRST;
   WHILE cnt IS NOT NULL LOOP
      DBMS_OUTPUT.PUT_LINE(retval(cnt));
      cnt := retval.NEXT(cnt);
    END LOOP;
END;
/

...
Count = 2
JAPANESE_M
BINARY
```

GET_LOCAL_TERRITORIES Function

This function returns the local territory names for the specified language.

Syntax

```
UTL_I18N.GET_LOCAL_TERRITORIES (
   language  IN VARCHAR2 CHARACTER SET ANY_CS)
 RETURN STRING_ARRAY;
```

Parameters

Table 169–9 GET_LOCAL_TERRITORIES Function Parameters

Parameter	Description
`language`	Specifies a valid Oracle language. It is case-insensitive.

Usage Notes

If the user specifies an invalid language name, then the function returns a `NULL` string.

Examples

Returns the list of Oracle territories that use German as one of their local languages.

```
DECLARE
   retval  UTL_I18N.STRING_ARRAY;
   cnt     INTEGER;
BEGIN
   retval  := UTL_I18N.GET_LCOAL_TERRITORIIES('GERMAN');
   DBMS_OUTPUT.PUT('Count = ');
   DBMS_OUTPUT.PUT_LINE(retval.LAST);
  cnt := retval.FIRST;
   WHILE cnt IS NOT NULL LOOP
      DBMS_OUTPUT.PUT_LINE(retval(cnt));
      cnt := retval.NEXT(cnt));
   END LOOP;
END;
/
...
Count = 4
GERMANY
AUSTRIA
LUXEMBOURG
SWITZERLAND
```

GET_LOCAL_TIME_ZONES Function

This function returns the local time zone IDs for the specified territory.

Syntax

```
UTL_I18N.GET_LOCAL_TIME_ZONES (
   territory      IN VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL)
RETURN STRING_ARRAY;
```

Parameters

Table 169–10 GET_LOCAL_TIME_ZONES Function Parameters

Parameter	Description
territory	Specifies a valid Oracle territory. It is case-insensitive.

Usage Notes

If the user specifies an invalid territory name, then the function returns a NULL string.

Examples

Creates a function that returns the list of time zones locally used in the territory AZERBAIJAN followed by the general common time zones. This is useful for when the user's territory is known and the application still allows the user to choose other time zones as a user's preference.

```
CREATE OR REPLACE FUNCTION get_time_zones
(territory IN VARCHAR2 CHARACTER SET ANY_CS)
RETURN utl_i18n.string_array
IS
  retval  utl_i18n.string_array;
  retval2 utl_i18n.string_array;
  stpos   INTEGER;
BEGIN
  retval  := utl_i18n.get_local_time_zones(
    territory);
  retval2 := utl_i18n.get_common_time_zones;
  stpos := retval.LAST + 1;
  retval(stpos) := '-----'; -- a separator
  FOR i IN retval2.FIRST..retval2.LAST LOOP
    stpos := stpos + 1;
    retval(stpos) := retval2(i);
  END LOOP;
  RETURN retval;
END;
/
```

Returns the list of local time zones for AZERBAIJAN followed by the common time zones with a separator string of five dashes (-----).

```
DECLARE
   retval UTL_I18N.STRING_ARRAY;
   cnt INTEGER;
BEGIN
   DBMS_OUTPUT.ENABLE(100000);
   retval UTL_I18N.GET_TIME_ZONES('AZERBAIJAN');
   cnt := retval.FIRST;
```

```
  WHILE cnt IS NOT NULL LOOP
    DBMS_OUTPUT.PUT_LINE(retval(cnt));
    cnt := retval.NEXT(cnt);
  END LOOP;
END;
/

Asia/Baku
-----
Pacific/Pago_Pago
Pacific/Honolulu
America/Anchorage
America/Vancouver
America/Los_Angeles
America/Tijuana
America/Edmonton
America/Denver
America/Phoenix
America/Mazatlan
America/Winnipeg
America/Regina
America/Chicago
America/Mexico_City
America/Guatemala
America/El_Salvador
America/Managua
America/Costa_Rica
America/Montreal
...
```

GET_TRANSLATION Function

This function returns the translation of the language and territory name in the specified translation language.

Syntax

```
UTL_I18N.GET_TRANSLATION (
   parameter        IN VARCHAR2 CHARCTER SET ANY_CS,
   trans_language   IN VARCHAR2 'AMERICAN',
   flag             IN PLS_INTEGER DEFAULT LANGUAGE_TRANS)
RETURN VARCHAR2 CHARACTER SET parameter%CHARSET;
```

Parameters

Table 169–11 GET_TRANSLATION Function Parameters

Parameter	Description
`parameter`	Specifies a valid language name, territory name, or a combined string in the form of *`language_territory`*. It is case-insensitive.
`trans_language`	Specifies a translation language name. For example, `ITALIAN` is for the Italian language. The default is `AMERICAN`, which indicates American English.
`flag`	Specifies the translation type: ■ `LANGUAGE_TRANS`: The function returns the language translation. ■ `TERRITORY_TRANS`: The function returns the territory translation. ■ `LANGUAGE_TERRITORY_TRANS`: The function returns the language and territory translation. The default translation type is `LANGUAGE_TRANS`.

Usage Notes

If `VARCHAR2` is used as a parameter type, the returned translation text can be corrupted due to the conversion to the database character set. Using `NVARCHAR2` as the parameter type will preserve the translation text because Unicode can encode all translated languages.

If the specified translation language is not available or an invalid name is provided, the default "American English" translations are returned. For example, Oracle does not provide `GUJARATI` translations, so the returned translation would be in American English.

Examples

The following returns the names of all the Oracle-supported languages in Italian.

```
DECLARE
   CURSOR c1 IS
     SELECT value FROM V$NLS_VALID_VALUES
     WHERE parameter = 'LANGUAGE'
  ORDER BY value;
  retval NVARCHAR2(100);
BEGIN
   FOR item IN c1 LOOP
     retval := UTL_I18N.GET_TRANSLATION (TO_NCHAR(item.value), 'italian');
   END LOOP;
END;
```

MAP_CHARSET Function

This function maps the following:

- An Oracle character set name to an IANA character set name.
- An IANA character set name to an Oracle character set name.
- An Oracle character set to an e-mail safe character set name.

Syntax

```
UTL_I18N.MAP_CHARSET(
   charset   IN VARCHAR2,
   context   IN PLS_INTEGER DEFAULT GENERIC_CONTEXT,
   flag      IN PLS_INTEGER DEFAULT ORACLE_TO_IANA)
RETURN VARCHAR2;
```

Parameters

Table 169–12 MAP_CHARSET Function Parameters

Parameter	Description
charset	Specifies the character set name to be mapped. The mapping is case-insensitive.
context	GENERIC_CONTEXT | MAIL_CONTEXT GENERIC_CONTEXT: The mapping is between an Oracle character set name and an IANA character set name. This is the default value. MAIL_CONTEXT: The mapping is between an Oracle character set name and an email safe character set name.
flag	■ ORACLE_TO_IANA | IANA_TO_ORACLE if GENERIC_CONTEXT is set ORACLE_TO_IANA: Map from an Oracle character set name to an IANA character set name. This is the default. IANA_TO_ORACLE: Map from an IANA character set name to an Oracle character set name. ■ MAIL_GENERIC | MAIL_WINDOWS if MAIL_CONTEXT is set MAIL_GENERIC: Map from an Oracle character set name to an email safe character set name on a non-Windows platform. MAIL_WINDOWS: Map from an Oracle character set name to an email safe character set name on a Windows platform.

Usage Notes

An e-mail safe character set is an Oracle character set that is commonly used by applications when they submit e-mail messages. The character set is usually used to convert contents in the database character set to e-mail safe contents. To specify the character set name in the mail header, you should use the corresponding IANA character set name obtained by calling the MAP_CHARSET function with the ORACLE_TO_IANA option, providing the e-mail safe character set name as input.

For example, no e-mail client recognizes message contents in the WE8DEC character set, whose corresponding IANA name is DEC-MCS. If WE8DEC is passed to the MAP_CHARSET function with the MAIL_CONTEXT option, then the function returns WE8ISO8859P1. Its corresponding IANA name, ISO-8859-1, is recognized by most e-mail clients.

The steps in this example are as follows:

1. Call the MAP_CHARSET function with the MAIL_CONTEXT | MAIL_GENERIC option with the database character set name, WE8DEC. The result is WE8ISO8859P1.
2. Convert the contents stored in the database to WE8ISO8859P1.
3. Call the MAP_CHARSET function with the ORACLE_TO_IANA | GENERIC_CONTEXT option with the e-mail safe character set, WE8ISO8859P1. The result is ISO-8859-1.
4. Specify ISO-8859-1 in the mail header when the e-mail message is submitted.

The function returns a character set name if a match is found. If no match is found or if the flag is invalid, then it returns NULL.

Note: Many Oracle character sets can map to one e-mail safe character set. There is no function that maps an e-mail safe character set to an Oracle character set name.

Examples

Generic Context

```
UTL_I18N.MAP_CHARSET('iso-8859-1',UTL_I18N.GENERIC_CONTEXT,UTL_I18N.IANA_TO_
ORACLE)
```

This returns 'WE8ISO8859P1'.

Context

```
UTL_I18N.MAP_CHARSET('WE8DEC', utl_i18n.mail_context,  utl_i18n.mail_generic)
```

This returns 'WE8ISO8859P1'.

> **See Also:** *Oracle Database Globalization Support Guide* for a list of valid Oracle character sets

MAP_FROM_SHORT_LANGUAGE Function

This function maps an Oracle short language name to an Oracle language name.

Syntax

```
UTL_I18N.MAP_FROM_SHORT_LANGUAGE (
   language           IN VARCHAR2 CHARACTER SET ANY_CS)
RETURN VARCHAR2;
```

Parameters

Table 169–13 MAP_FROM_SHORT_LANGUAGE Function Parameters

Parameter	Description
language	Specifies a valid short language name. It is case-insensitive.

Usage Notes

If the user specifies an invalid language name, then the function returns a NULL string.

Examples

Returns the default linguistic sort name for the customer with the ID of 9000. Note that the table customers is from the oe user in the Common Schema. Because the customer's language preference is stored using a short language name, you need to convert to a full language name by calling the GET_DEFAULT_LINGUISTIC_SORT procedure.

```
DECLARE
   short_n VARCHAR2(10);
   ling_n VARCHAR2(50);
BEGIN
   SELECT nls_language INTO short
   FROM customers WHERE customer_id = 9000;
   ling_n := UTL_I18N.GET_DEFAULT_LINGUISTIC_SORT (
   UTL_I18N.MAP_FROM_SHORT_LANGUAGE(short_n));
   DBMS_OUTPUT.PUT_LINE(ling_n);
END;
/
```

MAP_LANGUAGE_FROM_ISO Function

This function returns an Oracle language name from an ISO locale name.

Syntax

```
UTL_I18N.MAP_LANGUAGE_FROM_ISO(
  isolocale IN VARCHAR2)
RETURN VARCHAR2;
```

Parameters

Table 169–14 MAP_LANGUAGE_FROM_ISO Function Parameters

Parameter	Description
`isolocale`	Specifies the ISO locale. The mapping is case-insensitive.

Usage Notes

If the user specifies an invalid locale string, then the function returns a `NULL` string.

If the user specifies a locale string that includes only the language (for example, `en_` instead of `en_US`), then the function returns the default language name for the specified language (for example, `American`).

Examples

```
UTL_I18N.MAP_LANGUAGE_FROM_ISO('en_US')
```

This returns `'American'`.

> **See Also:** *Oracle Database Globalization Support Guide* for a list of valid Oracle languages

MAP_LOCALE_TO_ISO Function

This function returns an ISO locale name from an Oracle language name and an Oracle territory name. A valid string must include at least one of the following: a valid Oracle language name or a valid Oracle territory name.

Syntax

```
UTL_I18N.MAP_LOCALE_TO_ISO (
   ora_language   IN VARCHAR2,
   ora_territory  IN VARCHAR2)
 RETURN VARCHAR2;
```

Parameters

Table 169–15 MAP_LOCALE_TO_ISO Function Parameters

Parameter	Description
`ora_language`	Specifies an Oracle language name. It is case-insensitive.
ora_territory	Specifies an Oracle territory name. It is case-insensitive.

Usage Notes

If the user specifies an invalid string, then the function returns a `NULL` string.

Examples

```
UTL_I18N.MAP_LOCALE_TO_ISO('American','America')
```

This returns `'en_US'`.

> **See Also:** *Oracle Database Globalization Support Guide* for a list of valid Oracle languages and territories

MAP_TERRITORY_FROM_ISO Function

This function returns an Oracle territory name from an ISO locale.

Syntax

```
UTL_I18N.MAP_TERRITORY_FROM_ISO (
  isolocale IN VARCHAR2)
 RETURN VARCHAR2;
```

Parameters

Table 169–16 MAP_TERRITORY_FROM_ISO Function Parameters

Parameter	Description
`isolocale`	Specifies the ISO locale. The mapping is case-insensitive.

Usage Notes

If the user specifies an invalid locale string, then the function returns a `NULL` string.

If the user specifies a locale string that includes only the territory (for example, `_fr` instead of `fr_fr`), then the function returns the default territory name for the specified territory (for example, `French`).

Examples

```
UTL_I18N.MAP_TERRITORY_FROM_ISO('en_US')
```

This returns `'America'`.

> **See Also:** *Oracle Database Globalization Support Guide* for a list of valid Oracle territories

MAP_TO_SHORT_LANGUAGE Function

This function maps an Oracle language name to an Oracle short language name.

Syntax

```
UTL_I18N.MAP_TO_SHORT_LANGUAGE (
   language    IN VARCHAR2 CHARACTER SET ANY_CS)
RETURN VARCHAR2;
```

Parameters

Table 169–17 MAP_TO_SHORT_LANGUAGE Function Parameters

Parameter	Description
language	Specifies a valid full language name. It is case-insensitive.

Usage Notes

If the user specifies an invalid language name, then the function returns a NULL string.

Examples

Returns the short language name for the language.

```
DECLARE
  retval VARCHAR2(100);
BEGIN
  retval := UTL_I18N.MAP_TO_SHORT_LANGUAGE('american');
  DBMS_OUTPUT.PUT_LINE(retval);
END;
/

US
```

RAW_TO_CHAR Functions

This function converts RAW data from a valid Oracle character set to a VARCHAR2 string in the database character set.

The function is overloaded. The different forms of functionality are described along with the syntax declarations.

Syntax

Buffer Conversion:

```
UTL_I18N.RAW_TO_CHAR(
   data          IN RAW,
   src_charset   IN VARCHAR2 DEFAULT NULL)
 RETURN VARCHAR2;
```

Piecewise conversion converts raw data into character data piece by piece:

```
UTL_I18N.RAW_TO_CHAR (
   data            IN RAW,
   src_charset     IN VARCHAR2 DEFAULT NULL,
   scanned_length  OUT PLS_INTEGER,
   shift_status    IN OUT PLS_INTEGER)
RETURN VARCHAR2;
```

Parameters

Table 169–18 RAW_TO_CHAR Function Parameters

Parameter	Description
data	Specifies the RAW data to be converted to a VARCHAR2 string
src_charset	Specifies the character set that the RAW data was derived from. If src_charset is NULL, then the database character set is used.
scanned_length	Specifies the number of bytes of source data scanned
shift_status	Specifies the shift status at the end of the scan. The user must set it to SHIFT_IN the first time it is called in piecewise conversion. **Note:** ISO 2022 character sets use escape sequences instead of shift characters to indicate the encoding method. shift_status cannot hold the encoding method information that is provided by the escape sequences for the next function call. As a result, this function cannot be used to reconstruct ISO 2022 character from raw data in a piecewise way unless each unit of input can be guaranteed to be a closed string. A closed string begins and ends in a 7-bit escape state.

Usage Notes

If the user specifies an invalid character set, NULL data, or data whose length is 0, then the function returns a NULL string.

Examples

Buffer Conversion

```
UTL_I18N.RAW_TO_CHAR(hextoraw('616263646566C2AA'), 'utf8')
```

This returns the following string in the database character set:

```
'abcde'||chr(170)
```

Piecewise Conversion

```
UTL_I18N.RAW_TO_CHAR(hextoraw('616263646566C2AA'),'utf8',shf,slen)
```

This expression returns the following string in the database character set:

```
'abcde'||chr(170)
```

It also sets `shf` to `SHIFT_IN` and `slen` to `8`.

The following example converts data from the Internet piece by piece to the database character set.

```
rvalue RAW(1050);
  nvalue VARCHAR2(1024);
  conversion_state  PLS_INTEGER = 0;
  converted_len   PLS_INTEGER;
  rtemp  RAW(10) = '';
  conn   utl_tcp.connection;
  tlen PLS_INTEGER;

  ...
  conn := utl_tcp.open_connection ( remote_host => 'localhost',
                                    remote_port => 2000);
  LOOP
      tlen := utl_tcp.read_raw(conn, rvalue, 1024);
      rvalue := utl_raw.concat(rtemp, rvalue);
      nvalue := utl_i18n.raw_to_char(rvalue, 'JA16SJIS', converted_len,
conversion_stat);
      if (converted_len < utl_raw.length(rvalue) )
      then
        rtemp := utl_raw.substr(rvalue, converted_len+1);
      else
        rtemp := '';
      end if;
      /* do anything you want with nvalue */
      /* e.g htp.prn(nvalue); */
    END LOOP;
    utl_tcp.close_connection(conn);
  EXCEPTION
    WHEN utl_tcp.end_of_input THEN
      utl_tcp.close_connection(conn);
END;
```

RAW_TO_NCHAR Functions

This function converts RAW data from a valid Oracle character set to an NVARCHAR2 string in the national character set.

The function is overloaded. The different forms of functionality are described along with the syntax declarations.

Syntax

Buffer Conversion:

```
UTL_I18N.RAW_TO_NCHAR (
   data         IN RAW,
   src_charset  IN VARCHAR2 DEFAULT NULL)
 RETURN NVARCHAR2;
```

Piecewise conversion converts raw data into character data piece by piece:

```
UTL_I18N.RAW_TO_NCHAR (
   data            IN RAW,
   src_charset      IN VARCHAR2 DEFAULT NULL,
   scanned_length   OUT PLS_INTEGER,
   shift_status     IN OUT PLS_INTEGER)
 RETURN NVARCHAR2;
```

Parameters

Table 169–19 RAW_TO_NCHAR Function Parameters

Parameter	Description
data	Specifies the RAW data to be converted to an NVARCHAR2 string
src_charset	Specifies the character set that the RAW data was derived from. If src_charset is NULL, then the database character set is used.
scanned_length	Specifies the number of bytes of source data scanned
shift_status	Specifies the shift status at the end of the scan. The user must set it to SHIFT_IN the first time it is called in piecewise conversion. **Note:** ISO 2022 character sets use escape sequences instead of shift characters to indicate the encoding method. shift_status cannot hold the encoding method information that is provided by the escape sequences for the next function call. As a result, this function cannot be used to reconstruct ISO 2022 character from raw data in a piecewise way unless each unit of input can be guaranteed to be a closed string. A closed string begins and ends in a 7-bit escape state.

Usage Notes

If the user specifies an invalid character set, NULL data, or data whose length is 0, then the function returns a NULL string.

Examples

Buffer Conversion

```
UTL_I18N.RAW_TO_NCHAR(hextoraw('616263646566C2AA'),'utf8')
```

This returns the following string in the national character set:

```
'abcde'||chr(170)
```

Piecewise Conversion

```
UTL_I18N.RAW_TO_NCHAR(hextoraw('616263646566C2AA'),'utf8', shf, slen)
```

This expression returns the following string in the national character set:

```
'abcde'||chr(170)
```

It also sets `shf` to `SHIFT_IN` and `slen` to `8`.

The following example converts data from the Internet piece by piece to the national character set.

```
rvalue RAW(1050);
  nvalue NVARCHAR2(1024);
  converstion_state  PLS_INTEGER = 0;
  converted_len   PLS_INTEGER;
  rtemp  RAW(10) = '';
  conn   utl_tcp.connection;
  tlen PLS_INTEGER;

  ...
  conn := utl_tcp.open_connection ( remote_host => 'localhost',
                                    remote_port => 2000);
  LOOP
      tlen := utl_tcp.read_raw(conn, rvalue, 1024);
      rvalue := utl_raw.concat(rtemp, rvalue);
      nvalue := utl_i18n.raw_to_nchar(rvalue, 'JA16SJIS', converted_len,
conversion_stat);
      if (converted_len < utl_raw.length(rvalue) )
      then
        rtemp := utl_raw.substr(rvalue, converted_len+1);
      else
        rtemp := '';
      end if;
      /* do anything you want with nvalue */
      /* e.g htp.prn(nvalue); */
    END LOOP;
    utl_tcp.close_connection(conn);
  EXCEPTION
    WHEN utl_tcp.end_of_input THEN
      utl_tcp.close_connection(conn);
  END;
```

STRING_TO_RAW Function

This function converts a VARCHAR2 or NVARCHAR2 string to another valid Oracle character set and returns the result as RAW data.

Syntax

```
UTL_I18N.STRING_TO_RAW(
   data          IN VARCHAR2 CHARACTER SET ANY_CS,
   dst_charset   IN VARCHAR2 DEFAULT NULL)
RETURN RAW;
```

Parameters

Table 169–20 STRING_TO_RAW Function Parameters

Parameter	Description
data	Specifies the VARCHAR2 or NVARCHAR2 string to convert.
dst_charset	Specifies the destination character set. If dst_charset is NULL, then the database character set is used for CHAR data and the national character set is used for NCHAR data.

Usage Notes

If the user specifies an invalid character set, a NULL string, or a string whose length is 0, then the function returns a NULL string.

Examples

```
DECLARE
    r raw(50);
    s varchar2(20);
  BEGIN
    s:='abcdef'||chr(170);
    r:=utl_i18n.string_to_raw(s,'utf8');
    dbms_output.put_line(rawtohex(r));
  end;
/
```

This returns a hex value of '616263646566C2AA'.

TRANSLITERATE Function

This function performs script transliteration. In this release, the TRANSLITERATE function only supports Japanese Kana conversion.

Syntax

```
UTL_I18N.TRANSLITERATE (
  data IN VARCHAR2 CHARACTER SET ANY_CS,
  name IN VARCHAR2)
RETURN VARCHAR2 CHARACTER SET data%CHARSET;
```

Parameters

Table 169–21 TRANSLITERATE Function Parameters

Parameter	Description
data	Specifies the data to be converted. Either CHAR or NCHAR data type can be specified.
name	Specifies the transliteration name string. For a list of valid names, see Table 169–22.

Constants

These options specify Japanese Kana conversions.

Table 169–22 TRANSLITERATE Function Constants

Constant Name	Value	Description
KANA_FWKATAKANA	'kana_fwkatakana'	Converts any type of Kana character to a fullwidth Katakana character.
KANA_HWKATAKANA	'kana_hwkatakana'	Converts any type of Kana character to a halfwidth Katakana character.
KANA_HIRAGANA	'kana_hiragana'	Converts any type of Kana character to a fullwidth Hiragana character.
FWKATAKANA_HWKATAKANA	'fwkatakana_hwkatakana'	Converts only fullwidth Katakana characters to halfwidth Katakana characters.
FWKATAKANA_HIRAGANA	'fwkatakana_hiragana'	Converts only fullwidth Katakana characters to fullwidth Hiragana characters.
HWKATAKANA_FWKATAKANA	'hwkatakana_fwkatakana'	Converts only halfwidth Katakana characters to fullwidth Katakana characters.
HWKATAKANA_HIRAGANA	'hwkatakana_hiragana'	Converts only halfwidth Katakana characters to fullwidth Hiragana characters.
HIRAGANA_FWKATAKANA	'hiragana_fwkatakana'	Converts only fullwidth Hiragana characters to fullwidth Katakana characters.
HIRAGANA_HWKATAKANA	'hiragana_hwkatakana'	Converts only fullwidth Hiragana characters to halfwidth Katakana characters.

Usage Notes

The function returns the converted string.

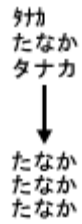
Examples

Given a table `japanese_emp`, containing an `NVARCHAR2` column `ename`, the following statement can be used to normalize all the kana names in `ename` to hiragana:

```
UPDATE japanese_emp
   SET ename = UTL_I18N.TRANSLITERATE (ename, 'kana_hiragana');
```

Figure shows how this output might look.

Figure 169–1 Loading Locale-Specific Data to the Database



The following statement normalizes one kana name to hiragana:

```
DECLARE
   Name  japanese_emp.ename%TYPE;
   Eno   CONSTANT  NUMBER(4) := 1;
BEGIN
   SELECT ename INTO name FROM japanese_emp WHERE enumber = eno;
   name := UTL_I18N.TRANSLITERATE(name, UTL_I18N.KANA_HIRAGANA);
   UPDATE japanese_emp SET ename = name WHERE enumber = eno;
EXCEPTION
   WHEN  UTL_I18N.UNSUPPORTED_TRANSLITERATION THEN
   DBMS_OUTPUT.PUT_LINE('transliteration not supported');
END;
/
```

UNESCAPE_REFERENCE Function

This function returns a string from an input string that contains character references. It decodes each character reference to the corresponding character value.

> **See Also:** "ESCAPE_REFERENCE Function" on page 169-8 for more information about escape sequences

Syntax

```
UTL_I18N.UNESCAPE_REFERENCE (
   str IN VARCHAR2 CHARACTER SET ANY_CS)
 RETURN VARCHAR2 CHARACTER SET str%CHARSET;
```

Parameters

Table 169–23 UNESCAPE_REFERENCE Function Parameters

Parameter	Description
`str`	Specifies the input string

Usage Notes

If the user specifies a `NULL` string or a string whose length is 0, then the function returns a `NULL` string. If the function fails, then it returns the original string.

Examples

```
UTL_I18N.UNESCAPE_REFERENCE('hello &lt; &#xe5;')
```

This returns `'hello <'||chr(229).`

170

UTL_INADDR

The `UTL_INADDR` package provides a PL/SQL procedures to support internet addressing. It provides an API to retrieve host names and IP addresses of local and remote hosts.

This chapter contains the following topics:

Using UTL_INADDR

- Exceptions
- Examples

Exceptions

Table 170–1 Exception from Internet Address Package

Exception	Description
UNKNOWN_HOST	The host is unknown.

Examples

Retrieve the local host name and IP address.

```
SET serveroutput on
BEGIN
  DBMS_OUTPUT.PUT_LINE(UTL_INADDR.GET_HOST_NAME);  -- get local host name
  DBMS_OUTPUT.PUT_LINE(UTL_INADDR.GET_HOST_ADDRESS);  -- get local IP addr
END;
/
```

Summary of UTL_INADDR Subprograms

Table 170–2 UTL_INADDR Package Subprograms

Subprogram	Description
GET_HOST_ADDRESS Function on page 170-6	Retrieves the IP address of the local or remote host given its name
GET_HOST_NAME Function on page 170-7	Retrieves the name of the local or remote host given its IP address

GET_HOST_ADDRESS Function

This function retrieves the IP address of the specified host.

Syntax

```
UTL_INADDR.GET_HOST_ADDRESS (
   host  IN VARCHAR2 DEFAULT NULL)
RETURN host_address VARCHAR2;
```

Parameters

Table 170–3 GET_HOST_ADDRESS Function Parameters

Parameter	Description
host	The name of the host to retrieve the IP address.

Return Values

Table 170–4 GET_HOST_ADDRESS Function Return Values

Parameter	Description
host_address	The IP address of the specified host, or that of the local host if host is NULL.

Exceptions

UNKNOWN_HOST: The specified IP address is unknown.

GET_HOST_NAME Function

This function retrieves the name of the local or remote host given its IP address.

Syntax

```
UTL_INADDR.GET_HOST_NAME (
   ip  IN VARCHAR2 DEFAULT NULL)
RETURN host_name VARCHAR2;
```

Parameters

Table 170–5 GET_HOST_NAME Function Parameters

Parameter	Description
ip	The IP address of the host used to determine its host name. If ip is not NULL, the official name of the host with its domain name is returned. If this is NULL, the name of the local host is returned and the name does not contain the domain to which the local host belongs.

Return Values

Table 170–6 GET_HOST_NAME Function Return Values

Parameter	Description
host_name	The name of the local or remote host of the specified IP address .

Exceptions

UNKNOWN_HOST: The specified IP address is unknown

171

UTL_LMS

`UTL_LMS` retrieves and formats error messages in different languages.

> **See Also:** *Oracle Database Globalization Support Guide*

This chapter contains the following topics:

Using UTL_LMS

This section contains topics which relate to using the `UTL_LMS` package.

- Security Model

Security Model

This package must be created as the user `SYS`.

Summary of UTL_LMS Subprograms

Table 171–1 UTL_LMS Package Subprograms

Function	Description
FORMAT_MESSAGE Function on page 171-5	Formats a retrieved error message
GET_MESSAGE Function on page 171-6	Retrieves an error message based on error number, product, facility, language, and message specified

FORMAT_MESSAGE Function

This function formats a message retrieved by the GET_MESSAGE function and returns the formatted message. If the function fails, then it returns a NULL result.

The following table shows special characters that can be used in the format string.

Special Character	Description
'%s'	Substitute the next string argument
'%d'	Substitute the next integer argument
'%%'	Represents the special character %

Syntax

```
UTL_LMS.FORMAT_MESSAGE (
   format IN VARCHAR2 CHARACTER SET ANY_CS,
   args   IN VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL)
 RETURN VARCHAR2 CHARACTER SET format%CHARSET;
```

Parameters

Table 171–2 FORMAT_MESSAGE Procedure Parameters

Parameter	Description
format	Specifies the string to format
args	Specifies the list of arguments

Examples

```
DECLARE
   s varchar2(200);
   i  pls_integer;
BEGIN
   i:= utl_lms.get_messsage(26052, 'rdbms', 'ora', 'french', s);
   dbms_output.put_line('before format, message is: '||s);
   dbms_output.put_line('formatted message is: '||utl_lms.format_message(s, 9,
'my_column_name');
END;
/
```

The following is an unformatted message:

```
Type %d non pris en charge pour l'expression SQL sur la colonne %s.
```

The following is the formatted message:

```
Type 9 non pris en charge pour l'expression SQL sur la colonne my_column_name.
```

GET_MESSAGE Function

This function retrieves an Oracle error message. The user can define user-specific error messages with the `lmsgen` utility.

It returns `0` when it is successful. It returns `-1` when it fails.

> **See Also:** *Oracle Database Globalization Support Guide* for more information about the `lmsgen` utility

Syntax

```
UTL_LMS.GET_MESSAGE (
   errnum    IN PLS_INTEGER,
   product   IN VARCHAR2,
   facility  IN VARCHAR2,
   language  IN VARCHAR2,
   message   OUT NOCOPY VARCHAR2CHARCTER SET ANY_CS)
RETURN PLS_INTEGER;
```

Parameters

Table 171–3 GET_MESSAGE Function Parameters

Parameter	Description
`errnum`	Specifies the error number. Example: `'972'` (for `ORA-00972`)
`product`	Specifies the product to which the error message applies Example: '`rdbms`'
`facility`	Specifies the error message prefix Example: `'ora'`
`language`	Specifies the language of the message. The parameter is case-insensitive. The default is `NULL`, which causes `GET_MESSAGE` to use the value of the `NLS_LANGUAGE` session parameter.
`message`	Specifies the output buffer for the retrieved message

Usage Notes

If the `language` parameter is set to `NULL`, then the value of the `NLS_LANGUAGE` session parameter is used as the default.

Examples

```
DECLARE
  s varchar2(200);
  i pls_integer;
BEGIN
   i:=utl_lms.get_message(601, 'rdbms', 'oci', 'french', s);
   dbms_output.put_line('OCI--00601 is: '||s);
END
/
```

The following output results:

```
OCI--00601 is: Echec du processus de nettoyage.
```

172

UTL_MAIL

The `UTL_MAIL` package is a utility for managing email which includes commonly used email features, such as attachments, CC, BCC, and return receipt.

This chapter contains the following topics:

Using UTL_MAIL

Security Model

UTL_MAIL is not installed by default because of the SMTP_OUT_SERVER configuration requirement and the security exposure this involves. In installing UTL_MAIL, you should take steps to prevent the port defined by SMTP_OUT_SERVER being swamped by data transmissions .

Operational Notes

You must both install UTL_MAIL and define the SMTP_OUT_SERVER.

- To install UTL_MAIL:

```
sqlplus sys/<pwd>
SQL> @$ORACLE_HOME/rdbms/admin/utlmail.sql
SQL> @$ORACLE_HOME/rdbms/admin/prvtmail.plb
```

- You define the SMTP_OUT_SERVER parameter in the init.ora rdbms initialization file. However, if SMTP_OUT_SERVER is not defined, this invokes a default of DB_DOMAIN which is guaranteed to be defined to perform appropriately.

Summary of UTL_MAIL Subprograms

Table 172–1 UTL_MAIL Package Subprograms

Subprogram	Description
SEND Procedure on page 172-6	Packages an email message into the appropriate format, locates SMTP information, and delivers the message to the SMTP server for forwarding to the recipients
SEND_ATTACH_RAW Procedure on page 172-7	Represents the `SEND` Procedure overloaded for `RAW` attachments
SEND_ATTACH_VARCHAR2 Procedure on page 172-8	Represents the `SEND` Procedure overloaded for `VARCHAR2` attachments

SEND Procedure

This procedure packages an email message into the appropriate format, locates SMTP information, and delivers the message to the SMTP server for forwarding to the recipients. It hides the SMTP API and exposes a one-line email facility for ease of use.

Syntax

```
UTL_MAIL.SEND (
   sender      IN    VARCHAR2 CHARACTER SET ANY_CS,
   recipients  IN    VARCHAR2 CHARACTER SET ANY_CS,
   cc          IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
   bcc         IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
   subject     IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
   message     IN    VARCHAR2 CHARACTER SET ANY_CS,
   mime_type   IN    VARCHAR2 DEFAULT 'text/plain; charset=us-ascii',
   priority    IN    PLS_INTEGER DEFAULT NULL);
```

Parameters

Table 172–2 SEND Procedure Parameters

Parameter	Description
`sender`	The email address of the sender.
`recipients`	The email addresses of the recipient(s), separated by commas.
`cc`	The email addresses of the CC recipient(s), separated by commas, default is `NULL`
`bcc`	The email addresses of the BCC recipient(s), separated by commas, default is `NULL`
`subject`	A string to be included as email subject string, default is `NULL`
`message`	A text message body.
`mime_type`	The mime type of the message, default is 'text/plain; charset=us-ascii'
`priority`	The message priority, default is `NULL`.

SEND_ATTACH_RAW Procedure

This procedure is the SEND Procedure overloaded for RAW attachments.

Syntax

```
UTL_MAIL.SEND_ATTACH_RAW (
   sender           IN    VARCHAR2 CHARACTER SET ANY_CS,
   recipients       IN    VARCHAR2 CHARACTER SET ANY_CS,
   cc               IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
   bcc              IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
   subject          IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
   message          IN    VARCHAR2 CHARACTER SET ANY_CS,
   mime_type        IN    VARCHAR2 DEFAULT 'text/plain; charset=us-ascii',
   priority         IN    PLS_INTEGER DEFAULT NULL
   attachment       IN    RAW,
   att_inline   IN    BOOLEAN DEFAULT TRUE,
   att_mime_type    IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT
                          'application/octet',
   att_filename     IN    VARCHAR2 DEFAULT NULL);
```

Parameters

Table 172–3 SEND_ATTACH_RAW Procedure Parameters

Parameter	Description
sender	The email address of the sender.
recipients	The email addresses of the recipient(s), separated by commas.
cc	The email addresses of the CC recipient(s), separated by commas, default is NULL.
bcc	The email addresses of the BCC recipient(s), separated by commas, default is NULL.
subject	A string to be included as email subject string, default is NULL.
message	A text message body.
mime_type	The mime type of the message, default is 'text/plain; charset=us-ascii'.
priority	The message priority, the default is NULL.
attachment	A RAW attachment.
att_inline	Specifies whether the attachment is viewable inline with the message body, default is TRUE.
att_mime_type	The mime type of the attachment, default is 'application/octet'.
att_filename	The string specifying a filename containing the attachment, default is NULL.

SEND_ATTACH_VARCHAR2 Procedure

This procedure is the SEND Procedure overloaded for VARCHAR2 attachments.

Syntax

```
UTL_MAIL.SEND_ATTACH_VARCHAR2 (
   sender            IN    VARCHAR2 CHARACTER SET ANY_CS,
   recipients        IN    VARCHAR2 CHARACTER SET ANY_CS,
   cc                IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
   bcc               IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
   subject           IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
   message           IN    VARCHAR2 CHARACTER SET ANY_CS,
   mime_type         IN    VARCHAR2 DEFAULT 'text/plain; charset=us-ascii',
   priority          IN    PLS_INTEGER DEFAULT NULL
   attachment        IN    RAW,
   att_inline        IN    BOOLEAN DEFAULT TRUE,
   att_mime_type     IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT
                          'application/octet',
   att_filename      IN    VARCHAR2 DEFAULT NULL);
```

Parameters

Table 172–4 SEND_ATTACH_VARCHAR2 Procedure Parameters

Parameter	Description
sender	The email address of the sender.
recipients	The email addresses of the recipient(s), separated by commas.
cc	The email addresses of the CC recipient(s), separated by commas, default is NULL.
bcc	The email addresses of the BCC recipient(s), separated by commas, default is NULL.
subject	A string to be included as email subject string, default is NULL.
message	A text message body.
mime_type	The mime type of the message, default is 'text/plain; charset=us-ascii'.
priority	The message priority, the default is NULL.
attachment	A text attachment.
att_inline	Specifies whether the attachment is inline, default TRUE.
att_mime_type	The mime type of the attachment, default is 'text/plain; charset=us-ascii'.
att_filename	The string specifying a filename containing the attachment, default is NULL.

173

UTL_NLA

The UTL_NLA package exposes a subset of the BLAS and LAPACK (Version 3.0) operations on vectors and matrices represented as VARRAYs.

This chapter contains the following topics:

Using UTL_NLA

This section contains topics which relate to using the `UTL_NLA` package.

Overview

The UTL_NLA package exposes a subset of the BLAS (Basic Linear Algebra Subprograms) and LAPACK (Linear Algebra PACKage)(Version 3.0) operations on vectors and matrices represented as VARRAYs.

Standards

For more information on the BLAS and LAPACK standards see

http://www.netlib.org/blas/
http://www.netlib.org/lapack/

Required Expertise

Users of this package are expected to have a sound grasp of linear algebra in general and of the BLAS and LAPACK libraries in particular.

Implementation

The mapping between BLAS and LAPACK procedures and their corresponding PL/SQL calls is one-to-one.

- All BLAS functions have the BLAS_ prefix (for example, theBLAS_ASUM Functions). The subroutines and functions in BLAS are mapped to PL/SQL procedures and functions, respectively.
- All LAPACK functions have the LAPACK_ prefix (for example, the LAPACK_ GBSV Procedures). The subroutines in LAPACK are mapped to PL/SQL procedures. Procedures that perform the same operation but differ only on the datatype of the arguments have the same overloaded names.

The mapping between BLAS and LAPACK procedure parameters and those of their corresponding PL/SQL subprograms is almost one-to-one.

- Also in the PL/SQL interface for LAPACK, all /work/ arguments have been removed. The UTL_NLA package manages the allocation and de-allocation of all work areas required by the libraries.
- A new optional parameter, pack, has been added to the end of each LAPACK procedure that specifies if the matrix has been linearized in the row-major or column-major (default) format.

Rules and Limits

Vectors and matrices are stored in `VARRAY`s with a maximum size of one million entries. Given this restriction, `UTL_NLA` vectors can be up to one million entries but matrices need to be of size RxC <= 1,000,000.

Subprogram Groups

BLAS Level 1 (Vector-Vector Operations) Subprograms

Table 173–1 BLAS Level 1 (Vector-Vector Operations) Subprograms

Subprogram	Description
BLAS_ASUM Functions on page 173-17	Computes the sum of the absolute values of the vector components
BLAS_AXPY Procedures on page 173-18	Copies `alpha*X + Y` into vector `Y`
BLAS_COPY Procedures on page 173-19	Copies the contents of vector `X` to vector `Y`
BLAS_DOT Functions on page 173-20	Returns the dot (scalar) product of two vectors `X` and `Y`
BLAS_IAMAX Functions on page 173-30	Computes the index of the first element of a vector that has the largest absolute value
BLAS_NRM2 Functions on page 173-31	Computes the vector 2-norm (Euclidean norm)
BLAS_ROT Procedures on page 173-32	Returns the plane rotation of points
BLAS_ROTG Procedures on page 173-33	Returns the Givens rotation of points
BLAS_SCAL Procedures on page 173-34	Scales a vector by a constant
BLAS_SWAP Procedures on page 173-43	Swaps the contents of two vectors each of size n

BLAS Level 2 (Matrix-Vector Operations) Subprograms

Table 173–2 BLAS Level 2 (Matrix-Vector Operations) Subprograms

Subprogram	Description
BLAS_GBMV Procedures on page 173-21	Performs the matrix-vector operation `y := alpha*A*x + beta*y` or `y := alpha*A'*x + beta*y` where `alpha` and `beta` are scalars, `x` and `y` are vectors and `A` is an `m` by `n` band matrix, with `kl` sub-diagonals and `ku` super-diagonals
BLAS_GEMV Procedures on page 173-26	Performs the matrix-vector operations `y := alpha*A*x + beta*y` or `y := alpha*A'*x + beta*y` where `alpha` and `beta` are scalars, `x` and `y` are vectors and `A` is an `m` by `n` matrix
BLAS_GER Procedures on page 173-28	Performs a rank 1 operation `A := alpha*x*y' + A` where `alpha` is a scalar, `x` is an `m` element vector, `y` is an n element vector and `A` is an `m` by `n` matrix
BLAS_SBMV Procedures on page 173-41	Performs a matrix-vector operation `y := alpha*A*x + beta*y` where `alpha` and `beta` are scalars, `x` and `y` are n element vectors and `A` is an `n` by `n` symmetric band matrix, with `k` super-diagonals
BLAS_SPMV Procedures on page 173-35	Performs a matrix-vector operation `y := alpha*A*x + beta*y` where `alpha` and `beta` are scalars, `x` and `y` are `n` element vectors and `A` is an `n` by `n` symmetric matrix, supplied in packed form
BLAS_SPR Procedures on page 173-37	Performs a symmetric rank 1 operation `A := alpha*x*x' + A` where `alpha` is a real scalar, `x` is an `n` element vector, and `A` is an `n` by `n` symmetric matrix, supplied in packed form
BLAS_SPR2 Procedures on page 173-39	Performs a symmetric rank 2 operation `A := alpha*x*y' + alpha*y*x' +A` where `alpha` is a scalar, `x` and `y` are `n` element vectors, and `A` is an `n` by `n` symmetric matrix, supplied in packed form
BLAS_SBMV Procedures on page 173-41	Performs a matrix-vector operation `y := alpha*A*x + beta*y` where `alpha` and `beta` are scalars, `x` and `y` are n element vectors and `A` is an `n` by `n` symmetric band matrix, with `k` super-diagonals
BLAS_SYMV Procedures on page 173-46	Performs a matrix-vector operation `y := alpha*A*x + beta*y` where `alpha` and `beta` are scalars, `x` and `y` are `n` element vectors and `A` is an `n` by `n` symmetric matrix
BLAS_SYR Procedures on page 173-48	Performs a symmetric rank 1 operation `A := alpha*x*x' + A` where `alpha` is a real scalar, `x` is an `n` element vector, and `A` is an `n` by `n` symmetric matrix
BLAS_SYR2 Procedures on page 173-50	Performs a symmetric rank 2 operation `A := alpha*x*y' + alpha*y*x' + A` where `alpha` is a scalar, `x` and `y` are `n` element vectors, and `A` is an `n` by `n` symmetric matrix
BLAS_TBMV Procedures on page 173-57	Performs a matrix-vector operation `x := A*x` or `A'*x = b` where `x` is an `n` element vector and `A` is an `n` by `n` unit, or non-unit, upper or lower triangular band matrix, with `(k + 1)` diagonals
BLAS_TBSV Procedures on page 173-59	Solves one of the systems of equation `A*x = b` or `A'*x = b` where `b` and `x` are `n` element vectors and `A` is an `n` by n unit, or non-unit, upper or lower triangular band matrix, with `(k + 1)` diagonals

Table 173–2 (Cont.) BLAS Level 2 (Matrix-Vector Operations) Subprograms

Subprogram	Description
BLAS_TPMV Procedures on page 173-61	Performs a matrix-vector operation `x := A*x` or `x := A'*x` where `x` is an `n` element vector and `A` is an `n` by `n` unit, or non-unit, upper or lower triangular matrix, supplied in packed form
BLAS_TPSV Procedures on page 173-63	Solves one of the systems of equation `A*x = b` or `A'*x = b` where `b` and `x` are `n` element vectors and `A` is an `n` by `n` unit, or non-unit, upper or lower triangular matrix, supplied in packed form
BLAS_TRMV Procedures on page 173-68	Performs a matrix-vector operation `x := A*x` or `x := A'*x` where `x` is an `n` element vector and `A` is an `n` by `n` unit, or non-unit, upper or lower triangular matrix
BLAS_TRSV Procedures on page 173-73	Solves one of the systems of equation `A*x = b` or `A'*x = b` where `b` and `x` are `n` element vectors and `A` is an n by `n` unit, or non-unit, upper or lower triangular matrix

BLAS Level 3 (Matrix-Matrix Operations) Subprograms

Table 173–3 BLAS Level 3 (Matrix-Matrix Operations) Subprograms

Subprogram	Description
BLAS_GEMM Procedures on page 173-26	Performs one of the matrix-vector operations `C := alpha*op(A)*op(B) + beta*C` where `op(X) is one of op(X) = X or op(X) = X'` where `alpha` and `beta` are scalars, and `A`, `B` and `C` are matrices, with `op(A)` an `m` by `k` matrix, `op(B)` a `k` by `n` matrix and `C` an `m` by `n` matrix
BLAS_SYMM Procedures on page 173-44	Performs one of the matrix-vector operations `C := alpha*A*B + beta*C` or `C := alpha*B*A + beta*C` where `alpha` and `beta` are scalars, `A` is a symmetric matrix, and `B` and `C` are `m` by `n` matrices
BLAS_SYR2K Procedures on page 173-52	Performs one of the symmetric rank2 k operations `C := alpha*A*B' + alpha*B*A' + beta*C` or `C := alpha*A'*B + alpha*B'*A + beta*C` where `alpha` and `beta` are scalars, `C` is an `n` by `n` symmetric matrix and `A` and `B` are `n` by `k` matrices in the first case and `k` by `n` matrices in the second case
BLAS_SYRK Procedures on page 173-55	Performs one of the symmetric rank `k` operations `C := alpha*A*A' + beta*C` or `C := alpha*A'*A + beta*C` where `alpha` and `beta` are scalars, `C` is an `n` by `n` symmetric matrix and `A` is an `n` by `k` matrix in the first case and a `k` by `n` matrix in the second case
BLAS_TRMM Procedures on page 173-65	Performs one of the matrix-vector operations `B := alpha*op(A)*B` or `B := alpha*B*op(A)` where `alpha` is a scalar, `B` is an `m` by `n` matrix, `A` is a unit, or non-unit, upper or lower triangular matrix and `op(A)` is one of two alternatives
BLAS_TRSM Procedures on page 173-70	Performs one of the matrix-vector operations `op(A)*X = alpha*B` or `X*op(A) = alpha*B` where `alpha` is a scalar, `X` and `B` are `m` by `n` matrices, `A` is a unit, or non-unit, upper or lower triangular matrix, `op(A)` is one of two alternatives. The matrix `X` is overwritten on `B`

LAPACK Driver Routines (Linear Equations) Subprograms

Table 173–4 LAPACK Driver Routines (Linear Equations) Subprograms

Subprogram	Description
LAPACK_GBSV Procedures on page 173-75	This procedure computes the solution to a real system of linear equations `a * x = b` where `a` is an `n` by `n` matrix and `x` and `b` are `n` by `nrhs` matrices. The `LU` decomposition with partial pivoting and row interchanges is used to factor `A`.
LAPACK_GESV Procedures on page 173-84	This procedure computes the solution to a real system of linear equations `a * x = b` where `a` is an `n` by `n` matrix and `x` and `b` are `n` by `nrhs` matrices. The `LU` decomposition with partial pivoting and row interchanges is used to factor `A`.
LAPACK_GTSV Procedures on page 173-92	This procedure solves the equation `a * x = b` where `a` is an `n` by `n` tridiagonal matrix, by Gaussian elimination with partial pivoting.
LAPACK_PBSV Procedures on page 173-94	This procedure computes the solution to a real system of linear equations `a * x = b` where `a` is an n by n symmetric positive definite band matrix and `x` and `b` are `n` by `nrhs` matrices. The Cholesky decomposition is used to factor `A`.
LAPACK_POSV Procedures on page 173-96	This procedure computes the solution to a real system of linear equations `a * x = b` where `a` is an n by n symmetric positive definite matrix and `x` and `b` are `n` by `nrhs` matrices. The Cholesky decomposition is used to factor `A`.
LAPACK_PPSV Procedures on page 173-98	This procedure computes the solution to a real system of linear equations `a * x = b` where `a` is an n by n symmetric positive definite matrix stored in packed format and `x` and `b` are `n` by `nrhs` matrices. The Cholesky decomposition is used to factor `A`.
LAPACK_PTSV Procedures on page 173-100	This procedure computes the solution to a real system of linear equations `a * x = b` where `a` is an n by n symmetric positive definite tridiagonal matrix, and `x` and `b` are `n` by `nrhs` matrices.
LAPACK_SPSV Procedures on page 173-110	This procedure computes the solution to a real system of linear equations `a * x = b` where `a` is an n by n symmetric matrix stored in packed format, and `x` and `b` are `n` by `nrhs` matrices. The diagonal pivoting method is used to factor `A`.
LAPACK_SYSV Procedures on page 173-120	This procedure computes the solution to a real system of linear equations `a * x = b` where `a` is an n by n symmetric matrix, and `x` and `b` are `n` by `nrhs` matrices. The diagonal pivoting method is used to factor `A`.

LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms

Table 173–5 LAPACK Driver Routines (LLS and Eigenvalue Problems)

Subprogram	Description
LAPACK_GEES Procedures on page 173-77	Computes for an n by n real nonsymmetric matrix A, the eigenvalues, the real Schur form T, and, optionally, the matrix of Schur vectors Z. This gives the Schur factorization A = Z*T*(Z**T).
LAPACK_GEEV Procedures on page 173-89	Computes for an n by n real nonsymmetric matrix A, the eigenvalues and, optionally, the left and/or right eigenvectors.
LAPACK_GELS Procedures on page 173-79	Solves overdetermined or underdetermined real linear systems involving an m by n matrix A, or its transpose, using a QR or LQ factorization of A. It is assumed that A has full rank.
LAPACK_GESDD Procedures on page 173-81	Computes the singular value decomposition (SVD) of a real m by n matrix A, optionally computing the left and right singular vectors. If singular vectors are desired, it uses a divide-and-conquer algorithm that makes mild assumptions about floating point arithmetic.
LAPACK_GESVD Procedures on page 173-86	Computes the singular value decomposition (SVD) of a real m by n matrix A, optionally computing the left and/or right singular vectors. The SVD is written A = U * SIGMA * transpose(V).
LAPACK_SBEV Procedures on page 173-102	Computes all the eigenvalues and, optionally, eigenvectors of a real symmetric band matrix A
LAPACK_SBEVD Procedures on page 173-104	Ccomputes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix A. If eigenvectors are desired, it uses a divide and conquer algorithm that makes mild assumptions about floating point arithmetic.
LAPACK_SPEV Procedures on page 173-106	Computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix A in packed storage
LAPACK_SPEVD Procedures on page 173-108	Computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix A in packed storage. If eigenvectors are desired, it uses a divide and conquer algorithm that makes mild assumptions about floating point arithmetic.
LAPACK_STEV Procedures on page 173-112	Computes all eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix A
LAPACK_STEVD Procedures on page 173-114	Computes all eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix A. If eigenvectors are desired, it uses a divide and conquer algorithm that makes mild assumptions about floating point arithmetic.
LAPACK_SYEV Procedures on page 173-116	Computes all eigenvalues and, optionally, eigenvectors of a real symmetric matrix A
LAPACK_SYEVD Procedures on page 173-118	Computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix A. If eigenvectors are desired, it uses a divide and conquer algorithm that makes mild assumptions about floating point arithmetic.

Summary of UTL_NLA Subprograms

Table 173–6 UTL_NLA Package Subprograms

Subprogram	Description	Group
BLAS_ASUM Functions on page 173-17	Computes the sum of the absolute values of the vector components	BLAS Level 1 (Vector-Vector Operations) Subprograms
BLAS_AXPY Procedures on page 173-18	Copies `alpha*X + Y` into vector `Y`	BLAS Level 1 (Vector-Vector Operations) Subprograms
BLAS_COPY Procedures on page 173-19	Copies the contents of vector `X` to vector `Y`	BLAS Level 1 (Vector-Vector Operations) Subprograms
BLAS_DOT Functions on page 173-20	Returns the dot (scalar) product of two vectors `X` and `Y`	BLAS Level 1 (Vector-Vector Operations) Subprograms
BLAS_GBMV Procedures on page 173-21	Performs the matrix-vector operation `y := alpha*A*x + beta*y` or `y := alpha*A'*x + beta*y` where `alpha` and `beta` are scalars, `x` and `y` are vectors and `A` is an `m` by `n` band matrix, with `kl` sub-diagonals and `ku` super-diagonals	BLAS Level 2 (Matrix-Vector Operations) Subprograms
BLAS_GEMM Procedures on page 173-26	Performs one of the matrix-vector operations where `alpha` and `beta` are scalars, and `A`, `B` and `C` are matrices, with `op(A)` an `m` by `k` matrix, `op(B)` a `k` by `n` matrix and `C` an `m` by `n` matrix	BLAS Level 3 (Matrix-Matrix Operations) Subprograms
BLAS_GEMV Procedures on page 173-26	Performs the matrix-vector operations `y := alpha*A*x + beta*y` or `y := alpha*A'*x + beta*y` where `alpha` and `beta` are scalars, `x` and `y` are vectors and `A` is an `m` by `n` matrix	BLAS Level 2 (Matrix-Vector Operations) Subprograms
BLAS_GER Procedures on page 173-28	Performs a rank 1 operation `A := alpha*x*y' + A` where `alpha` is a scalar, `x` is an `m` element vector, `y` is an n element vector and `A` is an `m` by `n` matrix	BLAS Level 2 (Matrix-Vector Operations) Subprograms
BLAS_IAMAX Functions on page 173-30	Computes the index of the first element of a vector that has the largest absolute value	BLAS Level 1 (Vector-Vector Operations) Subprograms
BLAS_NRM2 Functions on page 173-31	Computes the vector 2-norm (Euclidean norm)	BLAS Level 1 (Vector-Vector Operations) Subprograms
BLAS_ROT Procedures on page 173-32	Returns the plane rotation of points	BLAS Level 1 (Vector-Vector Operations) Subprograms
BLAS_ROTG Procedures on page 173-33	Returns the Givens rotation of points	BLAS Level 1 (Vector-Vector Operations) Subprograms

Table 173–6 (Cont.) UTL_NLA Package Subprograms

Subprogram	Description	Group
BLAS_SBMV Procedures on page 173-41	Performs a matrix-vector operation `y := alpha*A*x + beta*y` where `alpha` and `beta` are scalars, `x` and `y` are `n` element vectors and `A` is an `n` by `n` symmetric band matrix, with `k` super-diagonals	BLAS Level 2 (Matrix-Vector Operations) Subprograms
BLAS_SCAL Procedures on page 173-34	Scales a vector by a constant	BLAS Level 1 (Vector-Vector Operations) Subprograms
BLAS_SPMV Procedures on page 173-35	Performs a matrix-vector operation `y := alpha*A*x + beta*y` where `alpha` and `beta` are scalars, `x` and `y` are `n` element vectors and `A` is an `n` by `n` symmetric matrix, supplied in packed form	BLAS Level 2 (Matrix-Vector Operations) Subprograms
BLAS_SPR Procedures on page 173-37	Performs a symmetric rank 1 operation `A := alpha*x*x' + A` where `alpha` is a real scalar, `x` is an `n` element vector, and `A` is an `n` by `n` symmetric matrix, supplied in packed form	BLAS Level 2 (Matrix-Vector Operations) Subprograms
BLAS_SPR2 Procedures on page 173-39	Performs a symmetric rank 2 operation where `alpha` is a scalar, `x` and `y` are `n` element vectors, and `A` is an `n` by `n` symmetric matrix, supplied in packed form	BLAS Level 2 (Matrix-Vector Operations) Subprograms
BLAS_SWAP Procedures on page 173-43	Swaps the contents of two vectors each of size n	BLAS Level 1 (Vector-Vector Operations) Subprograms
BLAS_SYMM Procedures on page 173-44	Performs one of the matrix-vector operations where `alpha` and `beta` are scalars, `A` is a symmetric matrix, and `B` and `C` are `m` by `n` matrices	BLAS Level 3 (Matrix-Matrix Operations) Subprograms
BLAS_SYMV Procedures on page 173-46	Performs a matrix-vector operation where `alpha` and `beta` are scalars, `x` and `y` are `n` element vectors and `A` is an `n` by `n` symmetric matrix	BLAS Level 2 (Matrix-Vector Operations) Subprograms
BLAS_SYR Procedures on page 173-48	Performs a symmetric rank 1 operation where `alpha` is a real scalar, `x` is an `n` element vector, and `A` is an `n` by `n` symmetric matrix	BLAS Level 2 (Matrix-Vector Operations) Subprograms
BLAS_SYR2 Procedures on page 173-50	Performs a symmetric rank 2 operation where `alpha` is a scalar, `x` and `y` are `n` element vectors, and `A` is an `n` by `n` symmetric matrix	BLAS Level 2 (Matrix-Vector Operations) Subprograms
BLAS_SYR2K Procedures on page 173-52	Performs one of the symmetric rank2 k operations where `alpha` and `beta` are scalars, `C` is an `n` by `n` symmetric matrix and `A` and `B` are `n` by `k` matrices in the first case and `k` by `n` matrices in the second case	BLAS Level 3 (Matrix-Matrix Operations) Subprograms
BLAS_SYRK Procedures on page 173-55	Performs one of the symmetric rank `k` operations where `alpha` and `beta` are scalars, `C` is an `n` by `n` symmetric matrix and `A` is an `n` by `k` matrix in the first case and a `k` by `n` matrix in the second case	BLAS Level 3 (Matrix-Matrix Operations) Subprograms
BLAS_TBMV Procedures on page 173-57	Performs a matrix-vector operation where `x` is an `n` element vector and `A` is an `n` by `n` unit, or non-unit, upper or lower triangular band matrix, with `(k + 1)` diagonals	BLAS Level 2 (Matrix-Vector Operations) Subprograms

Table 173–6 (Cont.) UTL_NLA Package Subprograms

Subprogram	Description	Group
BLAS_TBSV Procedures on page 173-59	Solves one of the systems of equation where b and x are n element vectors and A is an n by n unit, or non-unit, upper or lower triangular band matrix, with (k + 1) diagonals	BLAS Level 2 (Matrix-Vector Operations) Subprograms
BLAS_TPMV Procedures on page 173-61	Performs a matrix-vector operation where x is an n element vector and A is an n by n unit, or non-unit, upper or lower triangular matrix, supplied in packed form	BLAS Level 2 (Matrix-Vector Operations) Subprograms
BLAS_TPSV Procedures on page 173-63	Solves one of the systems of equation where b and x are n element vectors and A is an n by n unit, or non-unit, upper or lower triangular matrix, supplied in packed form	BLAS Level 2 (Matrix-Vector Operations) Subprograms
BLAS_TRMM Procedures on page 173-65	Performs one of the matrix-vector operations wherealpha is a scalar, B is an m by n matrix, A is a unit, or non-unit, upper or lower triangular matrix and op(A) is one of two alternatives	BLAS Level 2 (Matrix-Vector Operations) Subprograms
BLAS_TRMV Procedures on page 173-68	Performs a matrix-vector operation where x is an n element vector and A is an n by n unit, or non-unit, upper or lower triangular matrix	BLAS Level 2 (Matrix-Vector Operations) Subprograms
BLAS_TRSM Procedures on page 173-70	Performs one of the matrix-vector operations op(A)*X = alpha*B or X*op(A) = alpha*B where alpha is a scalar, X and B are m by n matrices, A is a unit, or non-unit, upper or lower triangular matrix, op(A) is one of two alternatives. The matrix X is overwritten on B	BLAS Level 3 (Matrix-Matrix Operations) Subprograms
BLAS_TRSV Procedures on page 173-73	Solves one of the systems of equation where b and x are n element vectors and A is an n by n unit, or non-unit, upper or lower triangular matrix	BLAS Level 2 (Matrix-Vector Operations) Subprograms
LAPACK_GBSV Procedures on page 173-75	This procedure computes the solution to a real system of linear equations a * x = b where a is an n by n matrix and x and b are n by nrhs matrices. The LU decomposition with partial pivoting and row interchanges is used to factor A.	LAPACK Driver Routines (Linear Equations) Subprograms
LAPACK_GEES Procedures on page 173-77	Computes for an n by n real nonsymmetric matrix A, the eigenvalues, the real Schur form T, and, optionally, the matrix of Schur vectors Z. This gives the Schur factorization A = Z*T*(Z**T).	LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms
LAPACK_GEEV Procedures on page 173-89	Computes for an n by n real nonsymmetric matrix A, the eigenvalues and, optionally, the left and/or right eigenvectors.	LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms
LAPACK_GELS Procedures on page 173-79	Solves overdetermined or underdetermined real linear systems involving an m by n matrix A, or its transpose, using a QR or LQ factorization of A. It is assumed that A has full rank.	LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms
LAPACK_ GESDD Procedures on page 173-81	Computes the singular value decomposition (SVD) of a real m by n matrix A, optionally computing the left and right singular vectors. If singular vectors are desired, it uses a divide-and-conquer algorithm that makes mild assumptions about floating point arithmetic.	LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms

Table 173–6 (Cont.) UTL_NLA Package Subprograms

Subprogram	Description	Group
LAPACK_GESV Procedures on page 173-84	This procedure computes the solution to a real system of linear equations `a * x = b` where `a` is an `n` by `n` matrix and `x` and `b` are `n` by `nrhs` matrices. The `LU` decomposition with partial pivoting and row interchanges is used to factor `A`.	LAPACK Driver Routines (Linear Equations) Subprograms
LAPACK_ GESVD Procedures on page 173-86	Computes the singular value decomposition (SVD) of a real `m` by `n` matrix `A`, optionally computing the left and/or right singular vectors. The SVD is written `A = U * SIGMA * transpose(V)`.	LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms
LAPACK_GTSV Procedures on page 173-92	This procedure solves the equation `a * x = b` where `a` is an `n` by `n` tridiagonal matrix, by Gaussian elimination with partial pivoting.	LAPACK Driver Routines (Linear Equations) Subprograms
LAPACK_PBSV Procedures on page 173-94	This procedure computes the solution to a real system of linear equations `a * x = b` where `a` is an n by n symmetric positive definite band matrix and `x` and `b` are `n` by `nrhs` matrices. The Cholesky decomposition is used to factor `A`.	LAPACK Driver Routines (Linear Equations) Subprograms
LAPACK_POSV Procedures on page 173-96	This procedure computes the solution to a real system of linear equations `a * x = b` where `a` is an n by n symmetric positive definite matrix and `x` and `b` are `n` by `nrhs` matrices. The Cholesky decomposition is used to factor `A`.	LAPACK Driver Routines (Linear Equations) Subprograms
LAPACK_PPSV Procedures on page 173-98	This procedure computes the solution to a real system of linear equations `a * x = b` where `a` is an n by n symmetric positive definite matrix stored in packed format and `x` and `b` are `n` by `nrhs` matrices. The Cholesky decomposition is used to factor `A`.	LAPACK Driver Routines (Linear Equations) Subprograms
LAPACK_PTSV Procedures on page 173-100	This procedure computes the solution to a real system of linear equations `a * x = b` where `a` is an n by n symmetric positive definite tridiagonal matrix, and `x` and `b` are `n` by `nrhs` matrices.	LAPACK Driver Routines (Linear Equations) Subprograms
LAPACK_SBEV Procedures on page 173-102	Computes all the eigenvalues and, optionally, eigenvectors of a real symmetric band matrix `A`	LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms
LAPACK_ SBEVD Procedures on page 173-104	Ccomputes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix `A`. If eigenvectors are desired, it uses a divide and conquer algorithm that makes mild assumptions about floating point arithmetic.	LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms
LAPACK_SPEV Procedures on page 173-106	Computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix `A` in packed storage	LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms
LAPACK_ SPEVD Procedures on page 173-108	Computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix `A` in packed storage. If eigenvectors are desired, it uses a divide and conquer algorithm that makes mild assumptions about floating point arithmetic.	LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms

Table 173–6 (Cont.) UTL_NLA Package Subprograms

Subprogram	Description	Group
LAPACK_SPSV Procedures on page 173-110	This procedure computes the solution to a real system of linear equations `a * x = b` where `a` is an n by n symmetric matrix stored in packed format, and `x` and `b` are `n` by `nrhs` matrices. The diagonal pivoting method is used to factor `A`.	LAPACK Driver Routines (Linear Equations) Subprograms
LAPACK_STEV Procedures on page 173-112	Computes all eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix `A`	LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms
LAPACK_ STEVD Procedures on page 173-114	Computes all eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix `A`. If eigenvectors are desired, it uses a divide and conquer algorithm that makes mild assumptions about floating point arithmetic.	LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms
LAPACK_ SYEVD Procedures on page 173-118	Computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix `A`. If eigenvectors are desired, it uses a divide and conquer algorithm that makes mild assumptions about floating point arithmetic.	LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms
LAPACK_SYSV Procedures on page 173-120	This procedure computes the solution to a real system of linear equations `a * x = b` where `a` is an n by n symmetric matrix, and `x` and `b` are `n` by `nrhs` matrices. The diagonal pivoting method is used to factor `A`.	LAPACK Driver Routines (Linear Equations) Subprograms

BLAS_ASUM Functions

This procedure computes the sum of the absolute values of the vector components.

See Also: BLAS Level 1 (Vector-Vector Operations) Subprograms on page 173-6 for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_ASUM (
   n      IN       POSITIVEN,
   x      IN       UTL_NLA_ARRAY_DBL,
   incx   IN       POSITIVEN)
 RETURN BINARY_DOUBLE;

UTL_NLA.BLAS_ASUM (
   n      IN       POSITIVEN,
   alpha  IN       SCALAR_DOUBLE,
   x      IN       UTL_NLA_ARRAY_FLT)
 RETURN BINARY_FLOAT
```

Parameters

Table 173–7 BLAS_ASUM Function Parameters

Parameter	Description
n	Specifies the number of elements of the vectors x and y. n must be at least zero.
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least (1 + (n - 1)*abs(incx))
incx	Specifies the increment for the elements of x. incx must not be zero.

BLAS_AXPY Procedures

This procedure copies `alpha*X + Y` into vector `Y`.

> **See Also:** BLAS Level 1 (Vector-Vector Operations) Subprograms on page 173-6 for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_AXPY (
   n      IN       POSITIVEN,
   alpha  IN       SCALAR_DOUBLE,
   x      IN       UTL_NLA_ARRAY_DBL,
   incx   IN       POSITIVEN,
   y      IN OUT   UTL_NLA_ARRAY_DBL,
   incy   IN       POSITIVEN);

UTL_NLA.BLAS_AXPY (
   n      IN       POSITIVEN,
   alpha  IN       SCALAR_DOUBLE,
   x      IN       UTL_NLA_ARRAY_FLT,
   incx   IN       POSITIVEN,
   y      IN OUT   UTL_NLA_ARRAY_FLT,
   incy   IN       POSITIVEN);
```

Parameters

Table 173–8 BLAS_AXPY Procedure Parameters

Parameter	Description
`n`	Specifies the number of elements of the vectors `x` and `y`. `n` must be at least zero.
`alpha`	Specifies the scalar alpha.
`x`	`UTL_NLA_ARRAY_FLT/DBL` of `dimension` at least `( 1 + ( n - 1 )*abs( incx ) )`
`incx`	Specifies the increment for the elements of `x`. `incx` must not be zero.
`y`	`UTL_NLA_ARRAY_FLT/DBL` of `DIMENSION` at least `( 1 + ( n - 1 )*abs( incy) )`
`incy`	Specifies the increment for the elements of `y`. `incy` must not be zero.

BLAS_COPY Procedures

This procedure copies the contents of vector X to vector Y.

See Also: BLAS Level 1 (Vector-Vector Operations) Subprograms on page 173-6 for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_COPY (
   n     IN     POSITIVEN,
   x     IN     UTL_NLA_ARRAY_DBL,
   incx  IN     POSITIVEN,
   y     IN OUT UTL_NLA_ARRAY_DBL,
   incy  IN     POSITIVEN);

UTL_NLA.BLAS_COPY (
   n     IN     POSITIVEN,
   x     IN     UTL_NLA_ARRAY_FLT,
   incx  IN     POSITIVEN,
   y     IN OUT UTL_NLA_ARRAY_FLT,
   incy  IN     POSITIVEN);
```

Parameters

Table 173–9 BLAS_COPY Procedure Parameters

Parameter	Description
n	Specifies the number of elements of the vectors x and y. n must be at least zero.
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least (1 + (n - 1)*abs(incx))
incx	Specifies the increment for the elements of x. incx must not be zero.
y	UTL_NLA_ARRAY_FLT/DBL of dimension at least (1 + (n - 1)*abs(incy))
incy	Specifies the increment for the elements of y. incy must not be zero.

BLAS_DOT Functions

This function returns the dot (scalar) product of two vectors `X` and `Y`.

> **See Also:** BLAS Level 1 (Vector-Vector Operations) Subprograms on page 173-6 for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_DOT (
   n     IN   POSITIVEN,
   x     IN   UTL_NLA_ARRAY_DBL,
   incx  IN   POSITIVEN,
   y     IN   UTL_NLA_ARRAY_DBL,
   incy  IN   POSITIVEN)
  RETURN BINARY_DOUBLE;

UTL_NLA.BLAS_DOT (
   n     IN   POSITIVEN,
   x     IN   UTL_NLA_ARRAY_FLT,
   incx  IN   POSITIVEN,
   y     IN   UTL_NLA_ARRAY_FLT,
   incy  IN   POSITIVEN)
  RETURN BINARY_FLOAT;
```

Parameters

Table 173–10 BLAS_DOT Function Parameters

Parameter	Description
`n`	Specifies the number of elements of the vectors `x` and `y`. `n` must be at least zero.
`x`	`UTL_NLA_ARRAY_FLT/DBL` of dimension at least `( 1 + ( n - 1 )*abs( incx ) )`
`incx`	Specifies the increment for the elements of `x`. `incx` must not be zero.
`y`	`UTL_NLA_ARRAY_FLT/DBL` of dimension at least `( 1 + ( n - 1 )*abs( incy) )`
`incy`	Specifies the increment for the elements of `y`. `incy` must not be zero.

BLAS_GBMV Procedures

This procedure performs one of the matrix-vector operations

```
y := alpha*A*x + beta*y
```

or

```
y := alpha*A'*x + beta*y
```

where `alpha` and `beta` are scalars, `x` and `y` are vectors and `A` is an `m` by `n` band matrix, with `kl` sub-diagonals and `ku` super-diagonals.

See Also: BLAS Level 2 (Matrix-Vector Operations) Subprograms on page 173-7 for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_GEMV (
   trans  IN      flag,
   m      IN      POSITIVEN,
   n      IN      POSITIVEN,
   kl     IN      NATURALN,
   ku     IN      NATURALN,
   alpha  IN      SCALAR_DOUBLE,
   a      IN      UTL_NLA_ARRAY_DBL,
   lda    IN      POSITIVEN,
   x      IN      UTL_NLA_ARRAY_DBL,
   incx   IN      POSITIVEN,
   beta   IN      SCALAR_DOUBLE,
   y      IN OUT  UTL_NLA_ARRAY_DBL,
   incy   IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');

UTL_NLA.BLAS_GEMV (
   trans  IN      flag,
   m      IN      POSITIVEN,
   n      IN      POSITIVEN,
   kl     IN      NATURALN,
   ku     IN      NATURALN,
   alpha  IN      SCALAR_FLOAT,
   a      IN      UTL_NLA_ARRAY_FLT,
   lda    IN      POSITIVEN,
   x      IN      UTL_NLA_ARRAY_FLT,
   incx   IN      POSITIVEN,
   beta   IN      SCALAR_FLOAT,
   y      IN OUT  UTL_NLA_ARRAY_FLT,
   incy   IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');
```

Parameters

Table 173–11 BLAS_GBMV Procedure Parameters

Parameter	Description
trans	Specifies the operation to be performed: ■ trans = 'N' or 'n'y := alpha*A*x + beta*y ■ trans = 'T' or 't'y := alpha*A'*x + beta*y ■ trans = 'C' or 'c'y := alpha*A'*x + beta*y
m	Specifies the number of rows of the matrix A. m must be at least zero.
n	Specifies the number of columns of the matrix A. n must be at least zero.
kl	Specifies the number of sub-diagonals of the matrix A. kl must satisfy 0. le. kl.
ku	Specifies the number of super-diagonals of the matrix A. ku must satisfy 0 .le. ku.
alpha	SCALAR_FLOAT/DOUBLE. Specifies the scalar alpha.
a	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda,n). Before entry, the leading (kl + ku + 1) by n part of the array A must contain the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row (ku+1) of the array, the first super-diagonal starting at position 2 in row ku, the first sub-diagonal starting at position 1 in row (ku+2), and so on. Elements in the array A that do not correspond to elements in the band matrix (such as the top left ku by ku triangle) are not referenced.
lda	Specifies the first dimension of a as declared in the calling (sub) program. lda must be at least (kl+ku+1).
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least (1 + (n - 1)*abs(incx)) when trans = ''N' or 'n' and at least (1 + (m - 1)*abs(incx)) otherwise. Before entry, the incremented array X must contain the vector x.
incx	Specifies the increment for the elements of x. Must not be zero.
beta	SCALAR_FLOAT/DOUBLE. Specifies the scalar beta. When beta is supplied as zero then y need not be set on input.
y	UTL_NLA_ARRAY_FLT/DBL of dimension at least (1 + (m - 1)*abs(incy)) when trans = 'N' or 'n' and at least (1+(n-1)*abs(incy)) otherwise. Before entry with beta non-zero, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.
incy	Specifies the increment for the elements of y. Must not be zero.

Table 173–11 (Cont.) BLAS_GBMV Procedure Parameters

Parameter	Description
pack	(Optional) Flags the packing of the matrices: ■ 'C': column-major (default) ■ 'R': row-major

BLAS_GEMM Procedures

This procedure performs one of the matrix-matrix operations

```
C := alpha*op( A )*op( B ) + beta*C
```

where `op(X)` is one of

```
op(X) = X
```

or

```
op(X) = X'
```

where `alpha` and `beta` are scalars, and `A`, `B` and `C` are matrices, with `op(A)` an `m` by `k` matrix, `op(B)` a `k` by `n` matrix and `C` an `m` by `n` matrix.

> **See Also:** BLAS Level 3 (Matrix-Matrix Operations) Subprograms on page 173-9 for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_GEMM (
   transa IN      flag,
   transb IN      flag,
   m      IN      POSITIVEN,
   n      IN      POSITIVEN,
   k      IN      POSITIVEN,
   alpha  IN      SCALAR_DOUBLE,
   a      IN      UTL_NLA_ARRAY_DBL,
   lda    IN      POSITIVEN,
   b      IN      UTL_NLA_ARRAY_DBL,
   ldb    IN      POSITIVEN,
   beta   IN      SCALAR_DOUBLE,
   c      IN OUT  UTL_NLA_ARRAY_DBL,
   ldc    IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');

UTL_NLA.BLAS_GEMM (
   transa IN      flag,
   transb IN      flag,
   m      IN      POSITIVEN,
   n      IN      POSITIVEN,
   k      IN      POSITIVEN,
   alpha  IN      SCALAR_FLOAT,
   a      IN      UTL_NLA_ARRAY_FLT,
   lda    IN      POSITIVEN,
   b      IN      UTL_NLA_ARRAY_FLT,
   ldb    IN      POSITIVEN,
   beta   IN      SCALAR_FLOAT,
   c      IN OUT  UTL_NLA_ARRAY_FLT,
   ldc    IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');
```

Parameters

Table 173–12 BLAS_GEMM Procedure Parameters

Parameter	Description
transa	Specifies the form of op(A) to be used in the matrix multiplication as follows: ■ transa = 'N' or 'n': op(A) = 'A' ■ transa = 'T' or 't': op(A) = 'A' ■ transa = 'C' or 'c': op(A) = 'A'
transb	Specifies the form of op(B) to be used in the matrix multiplication as follows: ■ transb = 'N' or 'n': op(B) = B ■ transb = 'T' or 't': op(B) = B' ■ transb = 'C' or 'c': op(B) = B'
m	Specifies the number of rows of the matrix op(A) and of the matrix C. m must be at least zero.
n	Specifies the number of columns of the matrix op(B) and of the matrix C. n must be at least zero.
k	Specifies the rows of the matrix op(A) and the number of columns of the matrix op(B). k must be at least zero.
alpha	SCALAR_FLOAT/DOUBLE. Specifies the scalar alpha.
a	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, ka) where ka is k when transa = 'N' or 'n', and is m otherwise. Before entry with transa = 'N' or 'n', the leading m by k part of the array A must contain the matrix A, otherwise the leading k by m part of the array A must contain the matrix A.
lda	Specifies the first dimension of a as declared in the calling (sub) program. When transa = 'N' or 'n', lda must be at least max (1,k).
b	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, kb) where kb is n when transb = ''N' or 'n', and is k otherwise. Before entry with transb = 'N' or 'n', the leading k by n part of the array b must contain the matrix B, otherwise the leading n by k part of the arrayb must contain the matrix B.
ldb	Specifies the first dimension of b as declared in the calling (sub) program. When transb = 'N' or 'n', ldb must be at least max (1, n).
beta	SCALAR_FLOAT/DOUBLE. Specifies the scalar beta. When beta is supplied as zero then c need not be set on input.
c	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (ldc, n). Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be set on entry. On exit, the arrayC is overwritten by the m by n matrix (alpha*op(A)*op(B) + beta*C).
ldc	Specifies the first dimension of C as declared in the calling (sub) program. ldc must be at least max(1, m).
pack	(Optional) Flags the packing of the matrices: ■ 'C': column-major (default) ■ 'R': row-major

BLAS_GEMV Procedures

This procedure performs one of the matrix-vector operations

```
y := alpha*A*x + beta*y
```

or

```
y := alpha*A'*x + beta*y
```

where `alpha` and `beta` are scalars, `x` and `y` are vectors and `A` is an `m` by `n` matrix.

> **See Also:** BLAS Level 2 (Matrix-Vector Operations) Subprograms on page 173-7 for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_GEMV (
   trans  IN      flag,
   m      IN      POSITIVEN,
   n      IN      POSITIVEN,
   alpha  IN      SCALAR_DOUBLE,
   a      IN      UTL_NLA_ARRAY_DBL,
   lda    IN      POSITIVEN,
   x      IN      UTL_NLA_ARRAY_DBL,
   incx   IN      POSITIVEN,
   beta   IN      SCALAR_DOUBLE,
   y      IN OUT  UTL_NLA_ARRAY_DBL,
   incy   IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');

UTL_NLA.BLAS_GEMV (
   trans  IN      flag,
   m      IN      POSITIVEN,
   n      IN      POSITIVEN,
   alpha  IN      SCALAR_FLOAT,
   a      IN      UTL_NLA_ARRAY_FLT,
   lda    IN      POSITIVEN,
   x      IN      UTL_NLA_ARRAY_FLT,
   incx   IN      POSITIVEN,
   beta   IN      SCALAR_FLOAT,
   y      IN OUT  UTL_NLA_ARRAY_FLT,
   incy   IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');
```

Parameters

Table 173–13 BLAS_GEMV Procedure Parameters

Parameter	Description
`trans`	Specifies the operation to be performed: ▪ `trans = 'N'` or `'n'`,`y := alpha*A*x + beta*y` ▪ `trans = 'T'` or `'t'y := alpha*A'*x + beta*y` ▪ `trans = 'C'` or `'c'y := alpha*A'*x + beta*y`
`m`	Specifies the number of rows of the matrix A. `m` must be at least zero.

Table 173–13 (Cont.) BLAS_GEMV Procedure Parameters

Parameter	Description
n	Specifies the number of columns of the matrix A. n must be at least zero.
alpha	SCALAR_FLOAT/DOUBLE. Specifies the scalar alpha.
a	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, n). Before entry, the leading m by n part of the array a must contain the matrix of coefficients.
lda	Specifies the first dimension of a as declared in the calling (sub) program. lda must be at least max(1, m).
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least (1 + (n - 1)*abs(incx)) when trans = ''N' or 'n' and at least (1+(m-1)*abs(incx)) otherwise. Before entry, the incremented array X must contain the vector x.
incx	Specifies the increment for the elements of x. Must not be zero.
beta	SCALAR_FLOAT/DOUBLE. Specifies the scalar beta. When beta is supplied as zero then y need not be set on input.
Y	UTL_NLA_ARRAY_FLT/DBL of dimension at least (1 + (m - 1)*abs(incy)) when trans = 'N' or 'n' and at least (1 + (n - 1)*abs(incy)) otherwise. Before entry with beta non-zero, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.
incy	Specifies the increment for the elements of y. Must not be zero.
pack	(Optional) Flags the packing of the matrices: ■ 'C': column-major (default) ■ 'R': row-major

BLAS_GER Procedures

This procedure performs the rank 1 operation

```
A := alpha*x*y' + A
```

where `alpha` is a scalar, `x` is an `m` element vector, `y` is an n element vector and `A` is an `m` by `n` matrix.

See Also: BLAS Level 2 (Matrix-Vector Operations) Subprograms on page 173-7 for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_GER (
   m      IN      POSITIVEN,
   n      IN      POSITIVEN,
   alpha  IN      SCALAR_DBL,
   x      IN OUT  UTL_NLA_ARRAY_DBL,
   incx   IN      POSITIVEN,
   y      IN      UTL_NLA_ARRAY_DBL,
   incy   IN      POSITIVEN,
   a      IN OUT  UTL_NLA_ARRAY_DBL,
   lda    IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');

UTL_NLA.BLAS_GER (
   m      IN      POSITIVEN,
   n      IN      POSITIVEN,
   alpha  IN      SCALAR_FLT,
   x      IN OUT  UTL_NLA_ARRAY_FLT,
   incx   IN      POSITIVEN,
   y      IN      UTL_NLA_ARRAY_FLT,
   incy   IN      POSITIVEN,
   a      IN OUT  UTL_NLA_ARRAY_FLT,
   lda    IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');
```

Parameters

Table 173–14 BLAS_GER Procedure Parameters

Parameter	Description
`m`	Specifies the number of rows of the matrix `A`. `m` must be at least zero.
`n`	Specifies the number of columns of the matrix `A`. `n` must be at least zero.
`alpha`	Specifies the scalar alpha.
`x`	`UTL_NLA_ARRAY_FLT/DBL` of dimension at least `( 1 + ( m - 1 )*abs( incx ) )` Before entry, the incremented array `X` must contain the m element vector `x`.
`incx`	Specifies the increment for the elements of `x`. incx must not be zero.

Table 173–14 (Cont.) BLAS_GER Procedure Parameters

Parameter	Description
y	UTL_NLA_ARRAY_FLT/DBL of dimension at least `( 1 + ( n - 1 )*abs( incy ) )` Before entry, the incremented array Y must contain the m element vector y.
incy	Specifies the increment for the elements of y. incx must not be zero.
a	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, n). Before entry, the leading m by n part of the array a must contain the matrix of coefficients. On exit, a is overwritten by the updated matrix.
lda	Specifies the first dimension of a as declared in the calling (sub) program. lda must be at least `max( 1, m )`
pack	(Optional) Flags the packing of the matrices: ■ 'C': column-major (default) ■ 'R': row-major

BLAS_IAMAX Functions

This function computes the index of first element of a vector that has the largest absolute value.

> **See Also:** BLAS Level 1 (Vector-Vector Operations) Subprograms on page 173-6 for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_IAMAX (
   n     IN   POSITIVEN,
   x     IN   UTL_NLA_ARRAY_DBL,
   incx  IN   POSITIVEN,
  RETURN POSITIVEN;

UTL_NLA.BLAS_IAMAX (
   n     IN   POSITIVEN,
   x     IN   UTL_NLA_ARRAY_FLT,
   incx  IN   POSITIVEN,
  RETURN POSITIVEN;
```

Parameters

Table 173–15 BLAS_IAMAX Function Parameters

Parameter	Description
n	Specifies the number of elements of the vectors x and y. n must be at least zero.
x	UTL_NLA_ARRAY_FLT/DBL of DIMENSION at least (1 + (n - 1)*abs(incx))
incx	Specifies the increment for the elements of x. incx must not be zero.

BLAS_NRM2 Functions

This function computes the vector 2-norm (Euclidean norm).

> **See Also:** BLAS Level 1 (Vector-Vector Operations) Subprograms on page 173-6 for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_NRM2 (
   n     IN   POSITIVEN,
   x     IN   UTL_NLA_ARRAY_DBL,
   incx  IN   POSITIVEN)
  RETURN BINARY_DOUBLE;

UTL_NLA.BLAS_NRM2 (
   n     IN   POSITIVEN,
   x     IN   UTL_NLA_ARRAY_FLT,
   incx  IN   POSITIVEN)
  RETURN BINARY_FLOAT;
```

Parameters

Table 173–16 BLAS_NRM2 Function Parameters

Parameter	Description
n	Specifies the number of elements of the vectors x and y. n must be at least zero.
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least (1 + (n - 1)*abs(incx))
incx	Specifies the increment for the elements of x. incx must not be zero.

BLAS_ROT Procedures

This procedure returns the plane rotation of points.

See Also: BLAS Level 1 (Vector-Vector Operations) Subprograms on page 173-6 for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_ROT (
   n      IN      POSITIVEN,
   x      IN OUT  UTL_NLA_ARRAY_DBL,
   incx   IN      POSITIVEN,
   y      IN OUT  UTL_NLA_ARRAY_DBL,
   incy   IN      POSITIVEN,
   c      IN      SCALAR_DOUBLE,
   s      IN      SCALAR_DOUBLE);

UTL_NLA.BLAS_ROT (
   n      IN      POSITIVEN,
   x      IN OUT  UTL_NLA_ARRAY_FLT,
   incx   IN      POSITIVEN,
   y      IN OUT  UTL_NLA_ARRAY_FLT,
   incy   IN      POSITIVEN,
   c      IN      SCALAR_DOUBLE,
   s      IN      SCALAR_DOUBLE);
```

Parameters

Table 173–17 BLAS_ROT Procedure Parameters

Parameter	Description
n	Specifies the number of elements of the vectors x and y. n must be at least zero.
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least (1+(n-1)* abs(incx))
incx	Specifies the increment for the elements of x. incx must not be zero.
y	UTL_NLA_ARRAY_FLT/DBL of DIMENSION at least (1+(n-1)*abs(incy))
incy	Specifies the increment for the elements of y. incy must not be zero.
c	SCALAR_FLOAT/DOUBLE.Specifies the scalar C.
s	SCALAR_FLOAT/DOUBLE.Specifies the scalar S.

BLAS_ROTG Procedures

This procedure returns the Givens rotation of points.

> **See Also:** BLAS Level 1 (Vector-Vector Operations) Subprograms on page 173-6 for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_ROTG (
   a   IN OUT   SCALAR_DOUBLE,
   b   IN OUT   SCALAR_DOUBLE,
   c   IN OUT   SCALAR_DOUBLE,
   s   IN OUT   SCALAR_DOUBLE);

UTL_NLA.BLAS_ROTG (
   a   IN OUT   SCALAR_FLOAT,
   b   IN OUT   SCALAR_FLOAT,
   c   IN OUT   SCALAR_FLOAT,
   s   IN OUT   SCALAR_FLOAT);
```

Parameters

Table 173–18 BLAS_ROT G Procedure Parameters

Parameter	Description
a	SCALAR_FLOAT/DOUBLE. Specifies the scalar A.
b	SCALAR_FLOAT/DOUBLE. Specifies the scalar B.
c	SCALAR_FLOAT/DOUBLE. Specifies the scalar C.
s	SCALAR_FLOAT/DOUBLE. Specifies the scalar S.

BLAS_SCAL Procedures

This procedure scales a vector by a constant.

See Also: BLAS Level 1 (Vector-Vector Operations) Subprograms on page 173-6 for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_SCAL (
   n      IN  POSITIVEN,
   alpha  IN  SCALAR_DOUBLE,
   x      IN  OUT UTL_NLA_ARRAY_DBL,
   incx   IN  POSITIVEN);

UTL_NLA.BLAS_SCAL (
   n      IN  POSITIVEN,
   alpha  IN  SCALAR_FLOAT,
   x      IN  OUT UTL_NLA_ARRAY_FLT,
   incx   IN  POSITIVEN);
```

Parameters

Table 173–19 BLAS_SCAL Procedure Parameters

Parameter	Description
n	Specifies the number of elements of the vectors x and y. n must be at least zero.
alpha	Specifies the scalar alpha.
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least (1+(n-1)*abs(incx))
incx	Specifies the increment for the elements of x. incx must not be zero.

BLAS_SPMV Procedures

This procedure performs the matrix-vector operation

```
y := alpha*A*x + beta*y
```

where `alpha` and `beta` are scalars, `x` and `y` are `n` element vectors and `A` is an `n` by `n` symmetric matrix, supplied in packed form.

> **See Also:** BLAS Level 2 (Matrix-Vector Operations) Subprograms on page 173-7 for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_SPMV (
   uplo   IN      flag,
   n      IN      POSITIVEN,
   alpha  IN      SCALAR_DOUBLE,
   ap     IN      UTL_NLA_ARRAY_DBL,
   x      IN      UTL_NLA_ARRAY_DBL,
   incx   IN      POSITIVEN,
   beta   IN      SCALAR_DOUBLE,
   y      IN OUT  UTL_NLA_ARRAY_DBL,
   incy   IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');

UTL_NLA.BLAS_SPMV (
   uplo   IN      flag,
   n      IN      POSITIVEN,
   alpha  IN      SCALAR_FLOAT,
   ap      IN     UTL_NLA_ARRAY_FLT,
   x      IN      UTL_NLA_ARRAY_FLT,
   incx   IN      POSITIVEN,
   beta   IN      SCALAR_FLOAT,
   y      IN OUT  UTL_NLA_ARRAY_FLT,
   incy   IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');
```

Parameters

Table 173–20 BLAS_SPMV Procedure Parameters

Parameter	Description
`uplo`	Specifies the upper or lower triangular part of the matrix A is supplied in the packed array `AP`: ▪ `uplo = 'U'` or `'u'`. The upper triangular part of `A` is supplied in `AP`. ▪ `uplo = 'L'` or `'l'`. The lower triangular part of `A` is supplied in `AP`.
`n`	Specifies the order of the matrix `A`. `n` must be at least zero.
`alpha`	`SCALAR_FLOAT/DOUBLE`. Specifies the scalar alpha.

Table 173–20 (Cont.) BLAS_SPMV Procedure Parameters

Parameter	Description
ap	UTL_NLA_ARRAY_FLT/DBL of dimension at least ((n*(n+1))/2) Before entry with uplo = 'U' or 'u', the array ap must contain the upper triangular part of the symmetric matrix packed sequentially, column by column, so that ap(1) contains a(1,1), ap(2) and ap(3) contain a(1,2) and a(2,2) respectively, and so on. Before entry with uplo = 'L' or 'l', the array ap must contain the lower triangular part of the symmetric matrix packed sequentially, column by column, so that ap(1) contains, ap(2) and ap(3) contain a(2,1) and a(3,1) respectively, and so on.
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least (1+(n-1)*abs(incx)) Before entry, the incremented array X must contain the n element vector x.
incx	Specifies the increment for the elements of x. Must not be zero.
beta	SCALAR_FLOAT/DOUBLE. Specifies the scalar beta. When beta is supplied as zero then Y need not be set on input.
y	UTL_NLA_ARRAY_FLT/DBL of dimension at leasT (1+(n-1)*abs(incy)) Before entry, the incremented array Y must contain the n element vector y. On exit, Y is overwritten by the updated vector y.
incy	Specifies the increment for the elements of y. Must not be zero.
pack	(Optional) Flags the packing of the matrices: ■ 'C': column-major (default) ■ 'R': row-major

BLAS_SPR Procedures

This procedure performs the rank 1 operation

```
A := alpha*x*x' + A
```

where alpha is a real scalar, x is an n element vector, and A is an n by n symmetric matrix, supplied in packed form.

See Also: BLAS Level 2 (Matrix-Vector Operations) Subprograms on page 173-7 for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_SPR (
   uplo   IN      flag,
   n      IN      POSITIVEN,
   alpha  IN      SCALAR_DBL,
   x      IN OUT  UTL_NLA_ARRAY_DBL,
   incx   IN      POSITIVEN,
   ap     IN OUT  UTL_NLA_ARRAY_DBL,
   pack   IN      flag DEFAULT 'C');

UTL_NLA.BLAS_SPR (
   uplo   IN      flag,
   n      IN      POSITIVEN,
   alpha  IN      SCALAR_FLT,
   x      IN OUT  UTL_NLA_ARRAY_FLT,
   incx   IN      POSITIVEN,
   ap     IN OUT  UTL_NLA_ARRAY_FLT,
   pack   IN      flag DEFAULT 'C');
```

Parameters

Table 173–21 BLAS_SPR Procedure Parameters

Parameter	Description
uplo	Specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array ap: ■ uplo = 'U' or 'u': The upper triangular part of A is supplied in ap. ■ uplo = 'L' or 'l': The lower triangular part of A is supplied in ap.
n	Specifies the order of the matrix A. n must be at least zero.
alpha	Specifies the scalar alpha.
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least (1+(n-1)*abs(incx)) Before entry, the incremented array X must contain the m element vector x.
incx	Specifies the increment for the elements of x. incx must not be zero.

Table 173–21 (Cont.) BLAS_SPR Procedure Parameters

Parameter	Description
ap	UTL_NLA_ARRAY_FLT/DBL of dimension at least ((n*(n +1))/2) Before entry with uplo = 'U' or 'u', the array ap must contain the upper triangular part of the symmetric matrix packed sequentially, column by column, so that ap(1) contains a(1,1), ap(2) and ap(3) contain a(1,2) and a(2,2) respectively, and so on. On exit, the array ap is overwritten by the upper triangular part of the updated matrix. Before entry with uplo = 'L' or 'l', the array ap must contain the lower triangular part of the symmetric matrix packed sequentially, column by column, so that ap(1) contains a(1,1), ap(2) and ap(3) contain a(2,1) and a(3,1) respectively, and so on. On exit, the array ap is overwritten by the lower triangular part of the updated matrix
pack	(Optional) Flags the packing of the matrices: ■ 'C': column-major (default) ■ 'R': row-major

BLAS_SPR2 Procedures

This procedure performs the rank 2 operation

```
A := alpha*x*y' + alpha*y*x' +A
```

where alpha is a scalar, x and y are n element vectors, and A is an n by n symmetric matrix, supplied in packed form.

See Also: BLAS Level 2 (Matrix-Vector Operations) Subprograms on page 173-7 for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_SPR2 (
   uplo   IN      flag,
   n      IN      POSITIVEN,
   alpha  IN      SCALAR_DBL,
   x      IN      UTL_NLA_ARRAY_DBL,
   incx   IN      POSITIVEN,
   y      IN      UTL_NLA_ARRAY_DBL,
   incy   IN      POSITIVEN,
   a      IN OUT  UTL_NLA_ARRAY_DBL,
   lda    IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');

UTL_NLA.BLAS_SPR2 (
   uplo   IN      flag,
   n      IN      POSITIVEN,
   alpha  IN      SCALAR_FLT,
   x      IN      UTL_NLA_ARRAY_FLT,
   incx   IN      POSITIVEN,
   y      IN      UTL_NLA_ARRAY_FLT,
   incy   IN      POSITIVEN,
   a      IN OUT  UTL_NLA_ARRAY_FLT,
   lda    IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');
```

Parameters

Table 173–22 BLAS_SPR2 Procedure Parameters

Parameter	Description
uplo	Specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array ap : ■ uplo = 'U' or 'u' : The upper triangular part of A is supplied in ap. ■ uplo = 'L' or 'l' : The lower triangular part of A is supplied in ap.
n	Specifies the order of the matrix A. n must be at least zero.
alpha	Specifies the scalar alpha.
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least (1+(n-1)*abs(incx)) Before entry, the incremented array X must contain the m element vector x.

Table 173–22 (Cont.) BLAS_SPR2 Procedure Parameters

Parameter	Description
incx	Specifies the increment for the elements of x. incx must not be zero.
y	UTL_NLA_ARRAY_FLT/DBL of dimension at least (1+(n-1)*abs(incy)) Before entry, the incremented array X must contain the m element vector y.
incy	Specifies the increment for the elements of y. incy must not be zero.
ap	UTL_NLA_ARRAY_FLT/DBL of dimension at least ((n*(n+1))/2) Before entry with uplo = 'U' or 'u', the array ap must contain the upper triangular part of the symmetric matrix packed sequentially, column by column, so that ap(1) contains ap(1) contains a(1,1), ap(2) and ap(3) contain a(1,2) and a(2,2) respectively, and so on. On exit, the array ap is overwritten by the upper triangular part of the updated matrix. Before entry with uplo = 'L' or 'l', the array ap must contain the lower triangular part of the symmetric matrix packed sequentially, column by column, so that ap(1) contains a(1,1), ap(2) and ap(3) contain a(2,1) and a(3,1) respectively, and so on. On exit, the array ap is overwritten by the lower triangular part of the updated matrix
lda	Specifies the first dimension of a as declared in the calling (sub) program. lda must be at least (k + 1).
pack	(Optional) Flags the packing of the matrices: ■ 'C': column-major (default) ■ 'R': row-major

BLAS_SBMV Procedures

This procedure performs the matrix-vector operation

```
y := alpha*A*x + beta*y
```

where `alpha` and `beta` are scalars, `x` and `y` are `n` element vectors and `A` is an `n` by `n` symmetric band matrix, with `k` super-diagonals.

See Also: BLAS Level 2 (Matrix-Vector Operations) Subprograms on page 173-7 for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_SBMV (
   uplo   IN      flag,
   n      IN      POSITIVEN,
   k      IN      NATURALN,
   alpha  IN      SCALAR_DOUBLE,
   a      IN      UTL_NLA_ARRAY_DBL,
   lda    IN      POSITIVEN,
   x      IN      UTL_NLA_ARRAY_DBL,
   incx   IN      POSITIVEN,
   beta   IN      SCALAR_DOUBLE,
   y      IN OUT  UTL_NLA_ARRAY_DBL,
   incy   IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');

UTL_NLA.BLAS_SBMV (
   uplo   IN      flag,
   n      IN      POSITIVEN,
   k      IN      NATURALN,
   alpha  IN      SCALAR_FLOAT,
   a      IN      UTL_NLA_ARRAY_FLT,
   lda    IN      POSITIVEN,
   x      IN      UTL_NLA_ARRAY_FLT,
   incx   IN      POSITIVEN,
   beta   IN      SCALAR_FLOAT,
   y      IN OUT  UTL_NLA_ARRAY_FLT,
   incy   IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');
```

Parameters

Table 173–23 BLAS_SBMV Procedure Parameters

Parameter	Description
`uplo`	Specifies whether the upper or lower triangular part of the band matrix `A` is being supplied: ▪ `uplo` = `'U'` or `'u'`. The upper triangular part of `A` is supplied. ▪ `uplo` = `'L'` or `'l'`. The lower triangular part of `A` is supplied.
`n`	Specifies the order of the matrix `A`. `n` must be at least zero.
`k`	Specifies the number of super-diagonals of the matrix `A`. `k` must satisfy `0 .le. k`.

Table 173–23 (Cont.) BLAS_SBMV Procedure Parameters

Parameter	Description
alpha	SCALAR_FLOAT/DOUBLE. Specifies the scalar alpha.
a	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda,n). Before entry with uplo = 'U' or 'u', the leading (k+1) by n part of the array A must contain the upper triangular band part of the symmetric matrix, supplied column by column, with the leading diagonal of the matrix in row (k+1) of the array, the first super-diagonal starting at position 2 in rowk, and so on. The top left k by k triangle of the array A is not referenced. Before entry with uplo = 'L' or 'l', the leading (k+1) by n part of the array A must contain the lower triangular band part of the symmetric matrix, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced. Unchanged on exit
lda	Specifies the first dimension of a as declared in the calling (sub) program. lda must be at least (k + 1).
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least (1+(n-1)*abs(incx)) Before entry, the incremented array X must contain the n element vector x.
incx	Specifies the increment for the elements of x. Must not be zero.
beta	SCALAR_FLOAT/DOUBLE. Specifies the scalar beta.
y	UTL_NLA_ARRAY_FLT/DBL of dimension at least (1+(n-1)*abs(incy)) Before entry, the incremented array Y must contain the n element vector y. On exit, Y is overwritten by the updated vector y.
incy	Specifies the increment for the elements of y. Must not be zero.
pack	(Optional) Flags the packing of the matrices: ■ 'C': column-major (default) ■ 'R': row-major

BLAS_SWAP Procedures

This procedure swaps the contents of two vectors each of size n.

Syntax

```
UTL_NLA.BLAS_SWAP (
   n     IN     POSITIVEN,
   x     IN OUT UTL_NLA_ARRAY_DBL,
   incx  IN     POSITIVEN,
   y     IN OUT UTL_NLA_ARRAY_DBL,
   incy  IN     POSITIVEN);

UTL_NLA.BLAS_SWAP (
   n     IN     POSITIVEN,
   x     IN OUT UTL_NLA_ARRAY_FLT,
   incx  IN     POSITIVEN,
   y     IN OUT UTL_NLA_ARRAY_FLT,
   incy  IN     POSITIVEN);
```

Parameters

Table 173–24 BLAS_SWAP Procedure Parameters

Parameter	Description
n	Specifies the number of elements of the vectors x and y. n must be at least zero.
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least (1+(n-1)*abs(incx))
incx	Specifies the increment for the elements of x. incx must not be zero.
y	UTL_NLA_ARRAY_FLT/DBL of DIMENSION at least (1+(n-1)*abs(incy))
incy	Specifies the increment for the elements of y. incy must not be zero.

BLAS_SYMM Procedures

This procedure performs one of the matrix-matrix operations

```
C := alpha*A*B + beta*C
```

or

```
C := alpha*B*A + beta*C
```

where `alpha` and `beta` are scalars, `A` is a symmetric matrix, and `B` and `C` are `m` by `n` matrices.

> **See Also:** BLAS Level 3 (Matrix-Matrix Operations) Subprograms on page 173-9 for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_SYMM (
   side   IN      flag,
   uplo   IN      flag,
   m      IN      POSITIVEN,
   n      IN      POSITIVEN,
   alpha  IN      SCALAR_DOUBLE,
   a      IN      UTL_NLA_ARRAY_DBL,
   lda    IN      POSITIVEN,
   b      IN      UTL_NLA_ARRAY_DBL,
   ldb    IN      POSITIVEN,
   beta   IN      SCALAR_DOUBLE,
   c      IN OUT  UTL_NLA_ARRAY_DBL,
   ldc    IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');

UTL_NLA.BLAS_SYMM (
   side   IN      flag,
   uplo   IN      flag,
   m      IN      POSITIVEN,
   n      IN      POSITIVEN,
   alpha  IN      SCALAR_FLOAT,
   a      IN      UTL_NLA_ARRAY_FLT,
   lda    IN      POSITIVEN,
   b      IN      UTL_NLA_ARRAY_FLT,
   ldb    IN      POSITIVEN,
   beta   IN      SCALAR_FLOAT,
   c      IN OUT  UTL_NLA_ARRAY_FLT,
   ldc    IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');
```

Parameters

Table 173–25 BLAS_SYMM Procedure Parameters

Parameter	Description
`side`	Specifies whether the symmetric matrix `A` appears on the left or right in the operation: ■ `side = 'L'` or `'l'` : `C := alpha*A*B + beta*C` ■ `side = 'R'` or `'r'` : `C := alpha*B*A + beta*C`

Table 173–25 (Cont.) BLAS_SYMM Procedure Parameters

Parameter	Description
uplo	Specifies whether the upper or lower triangular part of the array A is to be referenced: ■ uplo = 'U' or 'u' : Only the upper triangular part of the symmetric matrix is to be referenced. ■ uplo = 'L' or 'l' : Only the lower triangular part of the symmetric matrix is to be referenced.
m	Specifies the number of rows of the matrix C. m must be at least zero.
n	Specifies the number of columns of the matrix C. n must be at least zero.
alpha	SCALAR_FLOAT/DOUBLE. Specifies the scalar alpha.
a	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda,ka) where ka is m when side = 'L' or 'l', and is n otherwise. Before entry with side = 'L' or 'l', the leading m by m part of the array A must contain the symmetric matrix, such that when uplo = 'U' or 'u', the leading m by m upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced, and when uplo = 'L' or 'l', the leading m by m lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced. Before entry with side = 'R' or 'r', the n by n part of the array A must contain the symmetric matrix, such that when uplo = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced, and when uplo = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced.
lda	Specifies the first dimension of a as declared in the calling (sub) program. When side = 'L' or 'l', lda must be at least max(1,m), otherwise lda must be at least max(1,n).
b	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (ldb,n). Before entry, the leading m by n part of the array B must contain the matrix B.
ldb	Specifies the first dimension of b as declared in the calling (sub) program. ldb must be at least max(1,m).
beta	SCALAR_FLOAT/DOUBLE. Specifies the scalar beta. When beta is supplied as zero then c need not be set on input.
c	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (ldc,n). Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be set on entry. On exit, the array C is overwritten by the m by n updated matrix.
ldc	Specifies the first dimension of C as declared in the calling (sub) program. ldc must be at least max (1,m).
pack	(Optional) Flags the packing of the matrices: ■ 'C': column-major (default) ■ 'R': row-major

BLAS_SYMV Procedures

This procedure performs the matrix-vector operation

```
y := alpha*A*x + beta*y
```

where `alpha` and `beta` are scalars, `x` and `y` are `n` element vectors and `A` is an `n` by `n` symmetric matrix.

See Also: BLAS Level 2 (Matrix-Vector Operations) Subprograms on page 173-7 for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_SYMV (
   uplo   IN      flag,
   n      IN      POSITIVEN,
   alpha  IN      SCALAR_DOUBLE,
   a      IN      UTL_NLA_ARRAY_DBL,
   lda    IN      POSITIVEN,
   x      IN      UTL_NLA_ARRAY_DBL,
   incx   IN      POSITIVEN,
   beta   IN      SCALAR_DOUBLE,
   y      IN OUT  UTL_NLA_ARRAY_DBL,
   incy   IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');

UTL_NLA.BLAS_SYMV (
   uplo   IN      flag,
   n      IN      POSITIVEN,
   alpha  IN      SCALAR_FLOAT,
   a      IN      UTL_NLA_ARRAY_FLT,
   lda    IN      POSITIVEN,
   x      IN      UTL_NLA_ARRAY_FLT,
   incx   IN      POSITIVEN,
   beta   IN      SCALAR_FLOAT,
   y      IN OUT  UTL_NLA_ARRAY_FLT,
   incy   IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');
```

Parameters

Table 173–26 BLAS_SYMV Procedure Parameters

Parameter	Description
`uplo`	Specifies whether the upper or lower triangular part of the array `A` is to be referenced: ▪ `uplo` = `'U'` or `'u'`. Only the upper triangular part of `A` is to be referenced. ▪ `uplo` = `'L'` or `'l'`. Only the lower triangular part of `A` is to be referenced.
`n`	Specifies the order of the matrix `A`. `n` must be at least zero.
`alpha`	`SCALAR_FLOAT/DOUBLE`. Specifies the scalar alpha.

Table 173–26 (Cont.) BLAS_SYMV Procedure Parameters

Parameter	Description
a	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda,n). Before entry with uplo = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced. Before entry with uplo = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced.
lda	Specifies the first dimension of a as declared in the calling (sub) program. lda must be at least max(1,n).
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least (1+(n-1)*abs(incx)) Before entry, the incremented array X must contain the n element vector x.
incx	Specifies the increment for the elements of x. Must not be zero.
beta	SCALAR_FLOAT/DOUBLE. Specifies the scalar beta. When beta is supplied as zero then y need not be set on input.
y	UTL_NLA_ARRAY_FLT/DBL of dimension at least (1+(n-1)*abs(incy)) Before entry, the incremented array Y must contain the n element vector y. On exit, Y is overwritten by the updated vector y.
incy	Specifies the increment for the elements of y. Must not be zero.
pack	(Optional) Flags the packing of the matrices: ■ 'C': column-major (default) ■ 'R': row-major

BLAS_SYR Procedures

This procedure performs the rank 1 operation

```
A := alpha*x*x' + A
```

where `alpha` is a real scalar, `x` is an `n` element vector, and `A` is an `n` by `n` symmetric matrix.

See Also: BLAS Level 2 (Matrix-Vector Operations) Subprograms on page 173-7 for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_SYR (
   uplo   IN      flag,
   n      IN      POSITIVEN,
   alpha  IN      SCALAR_DBL,
   x      IN OUT  UTL_NLA_ARRAY_DBL,
   incx   IN      POSITIVEN,
   a      IN OUT  UTL_NLA_ARRAY_DBL,
   lda    IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');

UTL_NLA.BLAS_SYR (
   uplo   IN      flag,
   n      IN      POSITIVEN,
   alpha  IN      SCALAR_FLT,
   x      IN OUT  UTL_NLA_ARRAY_FLT,
   incx   IN      POSITIVEN,
   a      IN OUT  UTL_NLA_ARRAY_FLT,
   lda    IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');
```

Parameters

Table 173–27 BLAS_SYR Procedure Parameters

Parameter	Description
`uplo`	Specifies whether the upper or lower triangular part of the array `A` is to be referenced: ■ `uplo = 'U'` or `'u'` : Only the upper triangular part of `A` is to be referenced. ■ `uplo = 'L'` or `'l'` : Only the lower triangular part of A is to be referenced.
`n`	Specifies the order of the matrix `A`. `n` must be at least zero.
`alpha`	Specifies the scalar alpha.
`x`	`UTL_NLA_ARRAY_FLT/DBL` of dimension at least `(1+(n-1)*abs(incx))` Before entry, the incremented array `X` must contain the m element vector `x`.
`incx`	Specifies the increment for the elements of `x`. `incx` must not be zero.

Table 173–27 (Cont.) BLAS_SYR Procedure Parameters

Parameter	Description
a	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, n) Before entry with uplo = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced. On exit, the upper triangular part of the array A is overwritten by the upper triangular part of the updated matrix. Before entry with uplo = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced. On exit, the lower triangular part of the array A is overwritten by the lower triangular part of the updated matrix.
lda	Specifies the first dimension of a as declared in the calling (sub) program. lda must be at least max(1, n)
pack	(Optional) Flags the packing of the matrices: ■ 'C': column-major (default) ■ 'R': row-major

BLAS_SYR2 Procedures

This procedure performs the rank 2 operation

```
A := alpha*x*y' + alpha*y*x' + A
```

where `alpha` is a scalar, `x` and `y` are `n` element vectors, and `A` is an `n` by `n` symmetric matrix.

See Also: BLAS Level 2 (Matrix-Vector Operations) Subprograms on page 173-7 for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_SYR2 (
   uplo   IN      flag,
   n      IN      POSITIVEN,
   alpha  IN      SCALAR_DBL,
   x      IN      UTL_NLA_ARRAY_DBL,
   incx   IN      POSITIVEN,
   y      IN      UTL_NLA_ARRAY_DBL,
   incy   IN      POSITIVEN,
   a      IN OUT  UTL_NLA_ARRAY_DBL,
   lda    IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');

UTL_NLA.BLAS_SYR2 (
   uplo   IN      flag,
   n      IN      POSITIVEN,
   alpha  IN      SCALAR_FLT,
   x      IN      UTL_NLA_ARRAY_FLT,
   incx   IN      POSITIVEN,
   y      IN      UTL_NLA_ARRAY_FLT,
   incy   IN      POSITIVEN,
   a      IN OUT  UTL_NLA_ARRAY_FLT,
   lda    IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');
```

Parameters

Table 173–28 BLAS_SYR2 Procedure Parameters

Parameter	Description
`uplo`	Specifies whether the upper or lower triangular part of the array `A` is to be referenced: ▪ `uplo` = `'U'` or `'u'`: Only the upper triangular part of `A` is to be referenced. ▪ `uplo` = `'L'` or `'l'`: Only the lower triangular part of `A` is to be referenced.
`n`	Specifies the order of the matrix `A`. `n` must be at least zero.
`alpha`	Specifies the scalar alpha.
`x`	`UTL_NLA_ARRAY_FLT/DBL` of dimension at least `( 1 + ( n - 1 )*abs( incx ) )` Before entry, the incremented array `X` must contain the m element vector `x`.

Table 173–28 (Cont.) BLAS_SYR2 Procedure Parameters

Parameter	Description
incx	Specifies the increment for the elements of x. incx must not be zero.
y	UTL_NLA_ARRAY_FLT/DBL of dimension at least (1 + (n - 1)*abs(incy)) Before entry, the incremented array Y must contain the m element vector y.
incy	Specifies the increment for the elements of y. incy must not be zero.
a	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, n) With uplo = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced. On exit, the upper triangular part of the array A is overwritten by the upper triangular part of the updated matrix. With uplo = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced. On exit, the lower triangular part of the array A is overwritten by the lower triangular part of the updated matrix.
lda	Specifies the first dimension of a as declared in the calling (sub) program. lda must be at least max(1, n)
pack	(Optional) Flags the packing of the matrices: ■ 'C': column-major (default) ■ 'R': row-major

BLAS_SYR2K Procedures

This procedure performs one of the symmetric rank2 k operations

```
C := alpha*A*B' + alpha*B*A' + beta*C
```

or

```
C := alpha*A'*B + alpha*B'*A + beta*C
```

where `alpha` and `beta` are scalars, `C` is an `n` by `n` symmetric matrix and `A` and `B` are `n` by `k` matrices in the first case and `k` by `n` matrices in the second case.

> **See Also:** BLAS Level 3 (Matrix-Matrix Operations) Subprograms on page 173-9 for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_SYR2K (
   uplo   IN      flag,
   trans  IN      flag,
   n      IN      POSITIVEN,
   k      IN      POSITIVEN,
   alpha  IN      SCALAR_DOUBLE,
   a      IN      UTL_NLA_ARRAY_DBL,
   lda    IN      POSITIVEN,
   b      IN      UTL_NLA_ARRAY_DBL,
   ldb    IN      POSITIVEN,
   beta   IN      SCALAR_DOUBLE,
   c      IN OUT  UTL_NLA_ARRAY_DBL,
   ldc    IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');

UTL_NLA.BLAS_SYR2K (
   uplo   IN      flag,
   trans  IN      flag,
   n      IN      POSITIVEN,
   k      IN      POSITIVEN,
   alpha  IN      SCALAR_FLOAT,
   a      IN      UTL_NLA_ARRAY_FLT,
   lda    IN      POSITIVEN,
   b      IN OUT  UTL_NLA_ARRAY_FLT,
   ldb    IN      POSITIVEN,
   beta   IN      SCALAR_FLOAT,
   c      IN OUT  UTL_NLA_ARRAY_FLT,
   ldc    IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');
```

Parameters

Table 173–29 BLAS_SYR2K Procedure Parameters

Parameter	Description
uplo	Specifies whether the upper or lower triangular part of the array C is to be referenced: ■ uplo = 'U' or 'u' : Only the upper triangular part of C is to be referenced. ■ uplo = 'L' or 'l' : Only the lower triangular part of C is to be referenced.
trans	Specifies the operations to be performed: ■ trans = 'N' or 'n'C : C := alpha*A*B' + alpha*B*A' + beta*C ■ trans = 'T' or 't'C : C := alpha*A'*B + alpha*B'*A + beta*C ■ trans = 'C' or 'c'C : C := alpha*A'*B + alpha*B'*A + beta*C
n	Specifies the order of matrix C. n must be at least zero.
k	On entry with trans = 'N' or 'n', k specifies the number of columns of the matrices A and B. On entry with trans = 'T' or 't' or trans = 'C' or 'c', k specifies the number of rows of the matrices A and B. k must be at least zero.
alpha	SCALAR_FLOAT/DOUBLE. Specifies the scalar alpha.
a	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda,ka) where kb is k when trans = 'N' or 'n', and is n otherwise. Before entry with trans = 'N' or 'n', the leading n byk part of the array A must contain the matrix A, otherwise the leading k by n part of the array A must contain the matrix A.
lda	Specifies the first dimension of a as declared in the calling (sub) program. When trans = 'N' or 'n', lda must be at leastmax(1,n), otherwise lda must be at least max(1,k).
b	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda,kb) where kb is k when trans = 'N' or 'n', and is n otherwise. Before entry with trans = 'N' or 'n', the leading n byk part of the array B must contain the matrix B, otherwise the leading k by n part of the array B must contain the matrix B.
ldb	Specifies the first dimension of b as declared in the calling (sub) program. When trans = 'N' or 'n', ldb must be at least max (1,n), otherwise ldb must be at least max (1,k).
beta	SCALAR_FLOAT/DOUBLE. Specifies the scalar beta.

Table 173–29 (Cont.) BLAS_SYR2K Procedure Parameters

Parameter	Description
c	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (ldc,n). Before entry with uplo = 'U' or 'u', the leading n by n upper triangular part of the array C must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of the array C is overwritten by the upper triangular part of the updated matrix. Before entry with uplo = 'L' or 'l', the leading n by n lower triangular part of the array C must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of C is not referenced. On exit, the lower triangular part of the array C is overwritten by the lower triangular part of the updated matrix.
ldc	Specifies the first dimension of C as declared in the calling (sub) program. ldc must be at least max(1,n).
pack	(Optional) Flags the packing of the matrices: ■ 'C': column-major (default) ■ 'R': row-major

BLAS_SYRK Procedures

This procedure performs one of the symmetric rank k operations

```
C := alpha*A*A' + beta*C
```

or

```
C := alpha*A'*A + beta*C
```

where `alpha` and `beta` are scalars, `C` is an `n` by `n` symmetric matrix and`A` is an `n` by `k` matrix in the first case and a `k` by`n` matrix in the second case.

> **See Also:** BLAS Level 3 (Matrix-Matrix Operations) Subprograms on page 173-9 for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_SYRK (
   uplo   IN      flag,
   trans  IN      flag,
   n      IN      POSITIVEN,
   k      IN      POSITIVEN,
   alpha  IN      SCALAR_DOUBLE,
   a      IN      UTL_NLA_ARRAY_DBL,
   lda    IN      POSITIVEN,
   beta   IN      SCALAR_DOUBLE,
   c      IN OUT  UTL_NLA_ARRAY_DBL,
   ldc    IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');

UTL_NLA.BLAS_SYRK (
   uplo   IN      flag,
   trans  IN      flag,
   n      IN      POSITIVEN,
   k      IN      POSITIVEN,
   alpha  IN      SCALAR_FLOAT,
   a      IN      UTL_NLA_ARRAY_FLT,
   lda    IN      POSITIVEN,
   beta   IN      SCALAR_FLOAT,
   c      IN OUT  UTL_NLA_ARRAY_DBL,
   ldc    IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');
```

Parameters

Table 173–30 BLAS_SYRK Procedure Parameters

Parameter	Description
`uplo`	Specifies whether the upper or lower triangular part of the array `C` is to be referenced: ▪ `uplo = 'U'` or `'u'` : Only the upper triangular part of `C` is to be referenced. ▪ `uplo = 'L'` or `'l'` : Only the lower triangular part of `C` is to be referenced.

Table 173–30 (Cont.) BLAS_SYRK Procedure Parameters

Parameter	Description
trans	Specifies the operations to be performed: ■ trans = 'N' or 'n' : C := alpha*A*A' + beta*C ■ trans = 'T' or 't' : C := alpha*A'*A + beta*C ■ trans ='C' or 'c' : C := alpha*A'*A + beta*C
n	Specifies the order of matrix C. n must be at least zero.
k	On entry with trans = 'N' or 'n', k specifies the number of columns of the matrix A. On entry with trans = 'T' or 't' or trans = 'C' or 'c', k specifies the number of rows of the matrix A. k must be at least zero.
alpha	SCALAR_FLOAT/DOUBLE. Specifies the scalar alpha.
a	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda,ka) where ka is k when trans = 'N' or 'n', and is n otherwise. Before entry with trans = 'N' or 'n', the leading n by k part of the array A must contain the matrix A, otherwise the leading k by n part of the array A must contain the matrix A.
lda	Specifies the first dimension of a as declared in the calling (sub) program. When trans = 'N' or 'n', lda must be at least max(1,n), otherwise lda must be at least max(1,k).
beta	SCALAR_FLOAT/DOUBLE. Specifies the scalar beta.
c	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (ldc,n). Before entry with uplo = 'U' or 'u', the leading n by n upper triangular part of the array C must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of the array C is overwritten by the upper triangular part of the updated matrix. Before entry with uplo = 'L' or 'l', the leading n by n lower triangular part of the array C must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of C is not referenced. On exit, the lower triangular part of the array C is overwritten by the lower triangular part of the updated matrix.
ldc	Specifies the first dimension of C as declared in the calling (sub) program. ldc must be at least max(1,n).
pack	(Optional) Flags the packing of the matrices: ■ 'C': column-major (default) ■ 'R': row-major

BLAS_TBMV Procedures

This procedure performs the matrix-vector operations

```
x := A*x
```

or

```
x := A'*x
```

where x is an n element vector and A is an n by n unit, or non-unit, upper or lower triangular band matrix, with (k+1) diagonals.

See Also: BLAS Level 2 (Matrix-Vector Operations) Subprograms on page 173-7 for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_TBMV (
   uplo   IN      flag,
   trans  IN      flag,
   diag   IN      flag,
   n      IN      POSITIVEN,
   k      IN      NATURALN,
   a      IN      UTL_NLA_ARRAY_DBL,
   lda    IN      POSITIVEN,
   x      IN OUT  UTL_NLA_ARRAY_DBL,
   incx   IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');

UTL_NLA.BLAS_TBMV (
   uplo   IN      flag,
   trans  IN      flag,
   diag   IN      flag,
   n      IN      POSITIVEN,
   k      IN      NATURALN,
   a      IN      UTL_NLA_ARRAY_FLT,
   lda    IN      POSITIVEN,
   x      IN OUT  UTL_NLA_ARRAY_FLT,
   incx   IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');
```

Parameters

Table 173–31 BLAS_TBMV Procedure Parameters

Parameter	Description
uplo	Specifies whether the matrix is an upper or lower triangular matrix: ■ uplo = 'U' or 'u'. A is an upper triangular matrix. ■ uplo = 'L' or 'l'. A is a lower triangular matrix.
trans	Specifies the operation to be performed: ■ trans = 'N' or 'n'x := A*x ■ trans = 'T' or 't'x := A'*x ■ trans = 'C' or 'c'x := A'*x

Table 173–31 (Cont.) BLAS_TBMV Procedure Parameters

Parameter	Description
diag	Specifies whether or not A is unit triangular: ■ diag = 'U' or 'u'. A is assumed to be unit triangular. ■ diag = 'N' or 'n'. A is not assumed to be unit triangular.
n	Specifies the order of the matrix A. n must be at least zero.
k	Specifies whether or not A is unit triangular: ■ with uplo = 'U' or 'u', K specifies the number of super-diagonals of the matrix A. ■ with uplo = 'L' or 'l', K specifies the number of sub-diagonals of the matrix A. K must satisfy 0 .le. k.
a	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, n). Before entry with uplo = 'U' or 'u', the leading (k+1) by n part of the array A must contain the upper triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row (k+1) of the array, the first super-diagonal starting at position 2 in row k, and so on. The top left k by k triangle of the array A is not referenced. Before entry with uplo = 'L' or 'l', the leading (k+1) by n part of the array A must contain the lower triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced. Note that when diag = ''U' or 'u', the elements of the array A corresponding to the diagonal elements of the matrix are not referenced, but are assumed to be unity.
lda	Specifies the first dimension of a as declared in the calling (sub) program. lda must be at least (k+1).
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least (1+(n-1)*abs(incx)). Before entry, the incremented array X must contain the n element vector x. On exit, X is overwritten with the transformed vector x.
incx	Specifies the increment for the elements of x. Must not be zero.
pack	(Optional) Flags the packing of the matrices: ■ 'C': column-major (default) ■ 'R': row-major

BLAS_TBSV Procedures

This procedure solves one of the systems of equations

```
A*x = b
```

or

```
A'*x = b
```

where b and x are n element vectors and A is an n by n unit, or non-unit, upper or lower triangular band matrix, with (k+1) diagonals.

> **See Also:** BLAS Level 2 (Matrix-Vector Operations) Subprograms on page 173-7 for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_TBSV (
   uplo   IN      flag,
   trans  IN      flag,
   diag   IN      flag,
   n      IN      POSITIVEN,
   k      IN      NATURALN,
   a      IN      UTL_NLA_ARRAY_DBL,
   lda    IN      POSITIVEN,
   x      IN OUT  UTL_NLA_ARRAY_DBL,
   incx   IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');

UTL_NLA.BLAS_STBSV (
   uplo   IN      flag,
   trans  IN      flag,
   diag   IN      flag,
   n      IN      POSITIVEN,
   k      IN      NATURALN,
   a      IN      UTL_NLA_ARRAY_FLT,
   lda    IN      POSITIVEN,
   x      IN OUT  UTL_NLA_ARRAY_FLT,
   incx   IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');
```

Parameters

Table 173–32 BLAS_TBSV Procedure Parameters

Parameter	Description
uplo	Specifies whether the matrix is an upper or lower triangular matrix: ■ uplo = 'U' or 'u'. A is an upper triangular matrix. ■ uplo = 'L' or 'l'. A is a lower triangular matrix.
trans	Specifies the equations to be solved: ■ trans = 'N' or 'n : 'A*x = b ■ trans = 'T' or 't': A'*x = b ■ trans = 'C' or 'c': A'*x = b

Table 173–32 (Cont.) BLAS_TBSV Procedure Parameters

Parameter	Description
diag	Specifies whether or not A is unit triangular: ■ diag = 'U' or 'u' : A is assumed to be unit triangular. ■ diag = 'N' or 'n' : A is not assumed to be unit triangular.
n	Specifies the order of the matrix A. n must be at least zero.
k	Specifies whether or not A is unit triangular: ■ with uplo = 'U' or 'u', K specifies the number of super-diagonals of the matrix A. ■ with uplo = 'L' or 'l', K specifies the number of sub-diagonals of the matrix A. K must satisfy 0 .le. k.
a	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda,n). Before entry with uplo = 'U' or 'u', the leading (k+1) by n part of the array A must contain the upper triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row (k+1) of the array, the first super-diagonal starting at position 2 in row k, and so on. The top left k by k triangle of the array A is not referenced. Before entry with uplo = 'L' or 'l', the leading (k+1) by n part of the array A must contain the lower triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced. Note that when diag = 'U' or 'u', the elements of the array A corresponding to the diagonal elements of the matrix are not referenced, but are assumed to be unity.
lda	On entry, lda specifies the first dimension of A as declared in the calling (sub) program. lda must be at least (k+1).
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least (1 + (n - 1) *abs(incx)) Before entry, the incremented array X must contain the n element right-hand side vector b. On exit, X is overwritten with the solution vector x.
incx	Specifies the increment for the elements of x. incx must not be zero.
pack	(Optional) Flags the packing of the matrices: ■ 'C': column-major (default) ■ 'R': row-major

Usage Notes

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

BLAS_TPMV Procedures

This procedure performs the matrix-vector operations

```
x := A*x
```

or

```
x := A'*x
```

where x is an n element vector and A is an n by n unit, or non-unit, upper or lower triangular matrix, supplied in packed form.

> **See Also:** BLAS Level 2 (Matrix-Vector Operations) Subprograms on page 173-7 for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_TPMV (
   uplo   IN      flag,
   trans  IN      flag,
   diag   IN      flag,
   n      IN      POSITIVEN,
   ap     IN      UTL_NLA_ARRAY_DBL,
   x      IN OUT  UTL_NLA_ARRAY_DBL,
   incx   IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');

UTL_NLA.BLAS_TBMV (
   uplo   IN      flag,
   trans  IN      flag,
   diag   IN      flag,
   n      IN      POSITIVEN,
   ap     IN      UTL_NLA_ARRAY_FLT,
   x      IN OUT  UTL_NLA_ARRAY_FLT,
   incx   IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');
```

Parameters

Table 173–33 BLAS_TPMV Procedure Parameters

Parameter	Description
uplo	Specifies whether the matrix is an upper or lower triangular matrix: ■ uplo = 'U' or 'u'. A is an upper triangular matrix. ■ uplo = 'L' or 'l'. A is a lower triangular matrix.
trans	Specifies the operation to e performed: ■ trans = 'N' or 'n'x := A*x ■ trans = 'T' or 't'x := A'*x ■ trans = 'C' or 'c'x := A'*x
diag	Specifies whether or not A is unit triangular: ■ diag = 'U' or 'u'. A is assumed to be unit triangular. ■ diag = 'N' or 'n'. A is not assumed to be unit triangular.
n	Specifies the order of the matrix A. n must be at least zero.

Table 173–33 (Cont.) BLAS_TPMV Procedure Parameters

Parameter	Description
ap	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda,n). Before entry with uplo = 'U' or 'u', the leading (k+1) by n part of the array A must contain the upper triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row (k+1) of the array, the first super-diagonal starting at position 2 in row k, and so on. The top left k by k triangle of the array A is not referenced. Before entry with uplo = 'L' or 'l', the leading (k+1) by n part of the array A must contain the lower triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced. Note that when diag = 'U' or 'u', the elements of the array A corresponding to the diagonal elements of the matrix are not referenced, but are assumed to be unity.
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least (1+(n-1)*abs(incx)). Before entry, the incremented array X must contain the n element vector x. On exit, X is overwritten with the transformed vector x.
incx	Specifies the increment for the elements of x. Must not be zero.
pack	(Optional) Flags the packing of the matrices: ■ 'C': column-major (default) ■ 'R': row-major

BLAS_TPSV Procedures

This procedure solves one of the systems of equations

```
A*x = b
```

or

```
A'*x = b
```

where b and x are n element vectors and A is an n by n unit, or non-unit, upper or lower triangular matrix, supplied in packed form.

See Also: BLAS Level 2 (Matrix-Vector Operations) Subprograms on page 173-7 for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_TPSV (
   uplo   IN      flag,
   trans  IN      flag,
   diag   IN      flag,
   n      IN      POSITIVEN,
   ap     IN      UTL_NLA_ARRAY_DBL,
   x      IN OUT  UTL_NLA_ARRAY_DBL,
   incx   IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');

UTL_NLA.BLAS_TPSV (
   uplo   IN      flag,
   trans  IN      flag,
   diag   IN      flag,
   n      IN      POSITIVEN,
   ap     IN      UTL_NLA_ARRAY_FLT,
   x      IN OUT  UTL_NLA_ARRAY_FLT,
   incx   IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');
```

Parameters

Table 173–34 BLAS_TPSV Procedure Parameters

Parameter	Description
uplo	Specifies whether the matrix is an upper or lower triangular matrix: ■ uplo = 'U' or 'u' : A is an upper triangular matrix. ■ uplo = 'L' or 'l' : A is a lower triangular matrix.
trans	Specifies the operation to be performed: ■ trans = 'N' or 'n' : A*x = b ■ trans = 'T' or 't' : A'*x = b ■ trans = 'C' or 'c' : A'*x = b
diag	Specifies whether or not A is unit triangular: ■ diag = 'U' or 'u' : A is assumed to be unit triangular. ■ diag = 'N' or 'n' : 'A is not assumed to be unit triangular.

Table 173–34 (Cont.) BLAS_TPSV Procedure Parameters

Parameter	Description
n	Specifies the order of the matrix A. n must be at least zero.
ap	UTL_NLA_ARRAY_FLT/DBL of dimension at least ((n*(n+1))/2) Before entry with uplo = 'U' or 'u', the array ap must contain the upper triangular matrix packed sequentially, column by column, so that ap(1) contains a(1,1), ap(2) and ap(3) contain a(1,2) and a(2,2) respectively, and so on. Before entry with uplo = 'L' or 'l', the array ap must contain the lower triangular matrix packed sequentially, column by column, so that ap(1) contains a(1,1), ap(2) and ap(3) contain a(2,1) and a(3,1) respectively, and so on. Note that when diag = 'U' or 'u', the diagonal elements of A are not referenced, but are assumed to be unity.
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least (1 + (n - 1) *abs(incx)) Before entry, the incremented array X must contain the n element right-hand side vector b. On exit, X is overwritten with the solution vector x.
incx	Specifies the increment for the elements of x. incx must not be zero.
pack	(Optional) Flags the packing of the matrices: ■ 'C': column-major (default) ■ 'R': row-major

Usage Notes

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

BLAS_TRMM Procedures

This procedure performs one of the matrix-matrix operations

```
B := alpha*op( A )*B
```

or

```
B := alpha*B*op( A )
```

where `alpha` is a scalar, `B` is an `m` by `n` matrix, `A` is a unit, or non-unit, upper or lower triangular matrix and `op(A)` is one of

```
op( A ) = A
```

or

```
op( A ) = A'
```

See Also: BLAS Level 3 (Matrix-Matrix Operations) Subprograms on page 173-9 for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_TRMM (
   side   IN      flag,
   uplo   IN      flag,
   transa IN      flag,
   diag   IN      flag,
   m      IN      POSITIVEN,
   n      IN      POSITIVEN,
   alpha  IN      SCALAR_DOUBLE,
   a      IN      UTL_NLA_ARRAY_DBL,
   lda    IN      POSITIVEN,
   b      IN OUT  UTL_NLA_ARRAY_DBL,
   ldb    IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');

UTL_NLA.BLAS_TRMM (
   side   IN      flag,
   uplo   IN      flag,
   transa IN      flag,
   diag   IN      flag,
   m      IN      POSITIVEN,
   n      IN      POSITIVEN,
   alpha  IN      SCALAR_FLOAT,
   a      IN      UTL_NLA_ARRAY_FLT,
   lda    IN      POSITIVEN,
   b      IN OUT  UTL_NLA_ARRAY_FLT,
   ldb    IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');
```

Parameters

Table 173–35 BLAS_TRMM Procedure Parameters

Parameter	Description
side	Specifies whether the symmetric matrix A appears on the left or right in the operation: ■ side = 'L' or 'l' : B := alpha*op(A)*B ■ side = 'R' or 'r' : B := alpha*B*op(A)
uplo	Specifies whether the upper or lower triangular part of the array A is to be referenced: ■ uplo = 'U' or 'u' : A is an upper triangular matrix. ■ uplo = 'L' or 'l'' : A is a lower triangular matrix.
transa	Specifies the form of op(A) to be used in the matrix multiplication as follows: ■ transa = 'N' or 'n' : op(A) = A ■ transa = 'T' or 't' : op(A) = A' ■ transa ='C' or 'c' : op(A) = A'
diag	Specifies whether or not A is unit triangular: ■ diag = 'U' or 'u'. A is assumed to be unit triangular. ■ diag = 'N' or 'n'. A is not assumed to be unit triangular.
m	Specifies the number of rows of the B. m must be at least zero.
n	Specifies the number of columns of B. n must be at least zero.
alpha	SCALAR_FLOAT/DOUBLE. Specifies the scalar alpha. When alpha is zero then A is not referenced and B need not be set before entry.
a	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda,k) where k is m when side = 'L' or 'l', and is n when side = ''R' or 'r'. Before entry with uplo = 'U' or 'u' , the leading k by k upper triangular part of the array A must contain the upper triangular matrix, and the strictly lower triangular part of A is not referenced. Before entry with uplo = 'L' or 'l', the leading k byk lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced. Note that when diag = ''U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity.
lda	Specifies the first dimension of a as declared in the calling (sub) program. When side = 'L' or 'l', lda must be at least max(1,m), otherwise lda must be at least max(1,n).
b	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (ldb,n). Before entry, the leading m by n part of the array B must contain the matrix B, and on exit is overwritten by the transformed matrix.
ldb	Specifies the first dimension of b as declared in the calling (sub) program. ldb must be at least max(1,m).

Table 173–35 (Cont.) BLAS_TRMM Procedure Parameters

Parameter	Description
`pack`	(Optional) Flags the packing of the matrices: ■ 'C': column-major (default) ■ 'R': row-major

BLAS_TRMV Procedures

This procedure performs the matrix-vector operations

```
x := A*x
```

or

```
x := A'*x
```

where x is an n element vector and A is an n by n unit, or non-unit, upper or lower triangular matrix.

See Also: BLAS Level 2 (Matrix-Vector Operations) Subprograms on page 173-7 for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_TRMV (
   uplo   IN      flag,
   trans  IN      flag,
   diag   IN      flag,
   n      IN      POSITIVEN,
   a      IN      UTL_NLA_ARRAY_DBL,
   lda    IN      POSITIVEN,
   x      IN OUT  UTL_NLA_ARRAY_DBL,
   incx   IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');

UTL_NLA.BLAS_TRMV (
   uplo   IN      flag,
   trans  IN      flag,
   diag   IN      flag,
   n      IN      POSITIVEN,
   a      IN      UTL_NLA_ARRAY_FLT,
   lda    IN      POSITIVEN,
   x      IN OUT  UTL_NLA_ARRAY_FLT,
   incx   IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');
```

Parameters

Table 173–36 BLAS_TRMV Procedure Parameters

Parameter	Description
uplo	Specifies whether the matrix is an upper or lower triangular matrix: ■ uplo = 'U' or 'u'. A is an upper triangular matrix. ■ uplo = 'L' or 'l'. A is a lower triangular matrix.
trans	Specifies the operation to be performed: ■ trans = 'N' or 'n'x := A*x ■ trans= 'T' or 't'x := A'*x ■ trans = 'C' or 'c'x := A'*x

Table 173–36 (Cont.) BLAS_TRMV Procedure Parameters

Parameter	Description
diag	Specifies whether or not A is unit triangular: ■ diag = 'U' or 'u'. A is assumed to be unit triangular. ■ diag = 'N' or 'n'. A is not assumed to be unit triangular.
n	Specifies the order of the matrix A. n must be at least zero.
a	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, n). Before entry with uplo = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular matrix and the strictly lower triangular part of A is not referenced. Before entry with uplo = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced. Note that when diag = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity
lda	Specifies the first dimension of a as declared in the calling (sub) program. lda must be at least max(1,n).
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least (1+(n-1)*as(incx)). Before entry, the incremented array X must contain the n element vector x.
incx	Specifies the increment for the elements of x. Must not be zero.
pack	(Optional) Flags the packing of the matrices: ■ 'C': column-major (default) ■ 'R': row-major

BLAS_TRSM Procedures

This procedure performs one of the matrix-matrix operations

```
op( A )*X = alpha*B
```

or

```
X*op( A ) = alpha*B
```

where `alpha` is a scalar, `X` and `B` are `m` by `n` matrices, `A` is a unit, or non-unit, upper or lower triangular matrix and `op(A)` is one of

```
op( A ) = A
```

or

```
op( A ) = A'
```

The matrix `X` is overwritten on `B`.

> **See Also:** BLAS Level 3 (Matrix-Matrix Operations) Subprograms on page 173-9 for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_TRSM (
   side   IN      flag,
   uplo   IN      flag,
   transa IN      flag,
   diag   IN      flag,
   m      IN      POSITIVEN,
   n      IN      POSITIVEN,
   alpha  IN      SCALAR_DOUBLE,
   a      IN      UTL_NLA_ARRAY_DBL,
   lda    IN      POSITIVEN,
   b      IN OUT  UTL_NLA_ARRAY_DBL,
   ldb    IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');

UTL_NLA.BLAS_TRSM (
   side   IN      flag,
   uplo   IN      flag,
   transa IN      flag,
   diag   IN      flag,
   m      IN      POSITIVEN,
   n      IN      POSITIVEN,
   alpha  IN      SCALAR_FLOAT,
   a      IN      UTL_NLA_ARRAY_FLT,
   lda    IN      POSITIVEN,
   b      IN OUT  UTL_NLA_ARRAY_FLT,
   ldb    IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');
```

Parameters

Table 173–37 BLAS_TRSM Procedure Parameters

Parameter	Description
side	Specifies whether the symmetric matrix A appears on the left or right in the operation: ■ side = 'L' or 'l' : op(A)*X = alpha*B ■ side = 'R' or 'r' : X*op(A) = alpha*B
uplo	Specifies whether the upper or lower triangular part of the array A is to be referenced: ■ uplo = 'U' or 'u' : A is an upper triangular matrix. ■ uplo = 'L' or 'l' : A is a lower triangular matrix.
transa	Specifies the form of op(A) to be used in the matrix multiplication as follows: ■ transa = 'N' or 'n' : op(A) = A ■ transa = 'T' or 't' : op(A) = A' ■ transa = 'C' or 'c' : op(A) = A'
diag	Specifies whether or not A is unit triangular: ■ diag = 'U' or 'u'. A is assumed to be unit triangular. ■ diag = 'N' or 'n'. A is not assumed to be unit triangular.
m	Specifies the number of rows of the B. m must be at least zero.
n	Specifies the number of columns of B. n must be at least zero.
alpha	SCALAR_FLOAT/DOUBLE. Specifies the scalar alpha. When alpha is zero then A is not referenced and B need not be set before entry.
a	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, k) where k is m when side = 'L' or 'l', and is n when side = 'R' or 'r'. Before entry with uplo = 'U' or 'u', the leading k by k upper triangular part of the array A must contain the upper triangular matrix, and the strictly lower triangular part of A is not referenced. Before entry with uplo = 'L' or 'l', the leading k by k lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced. Note that when diag = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity.
lda	Specifies the first dimension of a as declared in the calling (sub) program. When side = 'L' or 'l', lda must be at least max(1, m), otherwise lda must be at least max (1, n).
b	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (ldb, n). Before entry, the leading m by n part of the array B must contain the matrix B, and on exit is overwritten by the solution matrix X.
ldb	Specifies the first dimension of b as declared in the calling (sub) program. ldb must be at least max(1, m).

Table 173–37 (Cont.) BLAS_TRSM Procedure Parameters

Parameter	Description
`pack`	(Optional) Flags the packing of the matrices: ■ 'C': column-major (default) ■ 'R': row-major

BLAS_TRSV Procedures

This procedure solves one of the systems of equations

```
A*x = b
```

or

```
A'*x = b
```

where b and x are n element vectors and A is an n by n unit, or non-unit, upper or lower triangular matrix.

See Also: BLAS Level 2 (Matrix-Vector Operations) Subprograms on page 173-7 for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_TRSV (
   uplo   IN      flag,
   trans  IN      flag,
   diag   IN      flag,
   n      IN      POSITIVEN,
   a      IN      UTL_NLA_ARRAY_DBL,
   lda    IN      POSITIVEN,
   x      IN OUT  UTL_NLA_ARRAY_DBL,
   incx   IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');

UTL_NLA.BLAS_TRSV (
   uplo   IN      flag,
   trans  IN      flag,
   diag   IN      flag,
   n      IN      POSITIVEN,
   a      IN      UTL_NLA_ARRAY_FLT,
   lda    IN      POSITIVEN,
   x      IN OUT  UTL_NLA_ARRAY_FLT,
   incx   IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');
```

Parameters

Table 173–38 BLAS_TRSV Procedure Parameters

Parameter	Description
uplo	Specifies whether the matrix is an upper or lower triangular matrix: ■ uplo = 'U' or 'u'. A is an upper triangular matrix. ■ uplo = 'L' or 'l'. A is a lower triangular matrix.
trans	Specifies the operation to be performed: ■ trans = 'N' or 'n'A*x = b ■ trans = 'T' or 't'A'*x = b ■ trans = 'C' or 'c'A'*x = b

Table 173–38 (Cont.) BLAS_TRSV Procedure Parameters

Parameter	Description
diag	Specifies whether or not A is unit triangular: ■ diag = 'U' or 'u'. A is assumed to be unit triangular. ■ diag = 'N' or 'n'. A is not assumed to be unit triangular.
n	Specifies the order of the matrix A. n must be at least zero.
a	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, n). Before entry with uplo = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular matrix and the strictly lower triangular part of A is not referenced. Before entry with uplo = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced. Note that when diag = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity.
lda	Specifies the first dimension of A as declared in the calling (sub) program. lda must be at least max(1, n).
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least (1 + (n - 1) * abs (incx)) Before entry, the incremented array X must contain the n element right-hand side vector b. On exit, X is overwritten with the solution vector x.
incx	Specifies the increment for the elements of x. Must not be zero.
pack	(Optional) Flags the packing of the matrices: ■ 'C': column-major (default) ■ 'R': row-major

Usage Notes

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

LAPACK_GBSV Procedures

This procedure computes the solution to a real system of linear equations

```
a * x = b
```

where a is a band matrix of order n with kl sub diagonals and ku superdiagonals, and x and b are n by nrhs matrices.

The LU decomposition with partial pivoting and row interchanges is used to factor A as

```
a = L * U
```

where L is a product of permutation and unit lower triangular matrices with kl sub diagonals, and U is upper triangular with kl+ku superdiagonals. The factored form of a is then used to solve the system of equations

```
a * x = b
```

> **See Also:** LAPACK Driver Routines (Linear Equations) Subprograms on page 173-10 for other subprograms in this group

Syntax

```
UTL_NLA.LAPACK_GBSV (
   n      IN      POSITIVEN,
   kl     IN      NATURALN,
   ku     IN      NATURALN,
   nrhs   IN      POSITIVEN,
   ab     IN OUT  UTL_NLA_ARRAY_DBL,
   ldab   IN      POSITIVEN,
   ipiv   IN OUT  UTL_NLA_ARRAY_INT,
   b      IN OUT  UTL_NLA_ARRAY_DBL,
   ldb    IN      POSITIVEN,
   info   OUT  INTEGER,
   pack   IN      flag DEFAULT 'C');

UTL_NLA.LAPACK_GBSV (
   n      IN      POSITIVEN,
   kl     IN      NATURALN,
   ku     IN      NATURALN,
   nrhs   IN      POSITIVEN,
   ab     IN OUT  UTL_NLA_ARRAY_FLT,
   ldab   IN      POSITIVEN,
   ipiv   IN OUT  UTL_NLA_ARRAY_INT,
   b      IN OUT  UTL_NLA_ARRAY_FLT,
   ldb    IN      POSITIVEN,
   info   OUT  INTEGER,
   pack   IN      flag DEFAULT 'C');
```

Parameters

Table 173–39 LAPACK_GBSV Procedure Parameters

Parameter	Description
n	The number of linear equations, equivalent to the order of the matrixa .n >= 0
kl	The number of sub diagonals within the band of a. kl >= 0.

Table 173–39 (Cont.) LAPACK_GBSV Procedure Parameters

Parameter	Description
ku	The number of superdiagonals within the band of a . ku >= 0.
nrhs	The number of right-hand sides, which is the number of columns of the matrix b. nrhs >= 0.
ab	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldab, n). On entry, the matrix a in band storage, in rows kl+1 to 2*kl+ku+1; rows 1 to kl of the array need not be set. The j-th column of A is stored in the j-th column of the array ab: ab(kl+ku+1+i-j,j) = a(i,j) for max(1,j-ku)<=i<=min(n,j+kl) On exit, details of the factorization: U is stored as an upper triangular band matrix with kl+ku superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows: kl+ku+2 to 2*kl+ku+1
ldab	The leading dimension of the array ab. ldab >= 2*kl+ku+1
ipiv	INTEGER array, DIMENSION (n). The pivot indices that define the permutation matrix P; row iof the matrix was interchanged with row ipiv(i).
b	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldb, nrhs). On entry, the n by nrhs matrix of right hand side matrix b. On exit, if info = 0 , the n by nrhs solution matrix X.
ldb	The leading dimension of the array b. ldb >= max(1,n)
info	■ = 0 : successful exit ■ < 0 : if info = -i , the i-th argument had an illegal value ■ > 0 : if info = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and the solution has not been computed
pack	(Optional) Flags the packing of the matrices: ■ 'C': column-major (default) ■ 'R': row-major

LAPACK_GEES Procedures

This procedure computes for an n by n real nonsymmetric matrix A, the eigenvalues, the real Schur form T, and, optionally, the matrix of Schur vectors Z. This gives the Schur factorization A = Z*T*(Z**T).

A matrix is in real Schur form if it is upper quasi-triangular with 1 by 1 and 2 by 2 blocks. 2 by 2 blocks will be standardized in the form

```
[  a  b  ]
[  c  a  ]
```

where b*c < 0. The eigenvalues of such a block are a +- sqrt(bc).

> **See Also:** LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms on page 173-11 for other subprograms in this group

Syntax

```
UTL_NLA.LAPACK_GEES (
   jobvs   IN       flag,
   n       IN       POSITIVEN,
   a       IN  OUT  UTL_NLA_ARRAY_DBL,
   lda     IN       POSITIVEN,
   wr      IN OUT   UTL_NLA_ARRAY_DBL,
   wi      IN OUT   UTL_NLA_ARRAY_DBL,
   vs      IN OUT   UTL_NLA_ARRAY_DBL,
   ldvs    IN       POSITIVEN,
   info    OUT      INTEGER,
   pack    IN       flag DEFAULT 'C');

UTL_NLA.LAPACK_GEES (
   jobvs   IN       flag,
   n       IN       POSITIVEN,
   a       IN OUT   UTL_NLA_ARRAY_FLT,
   lda     IN       POSITIVEN,
   wr      IN       OUT UTL_NLA_ARRAY_FLT,
   wi      IN       OUT UTL_NLA_ARRAY_FLT,
   vs      IN OUT   UTL_NLA_ARRAY_FLT,
   ldvs    IN       POSITIVEN,
   info    OUT      integer,
   pack    IN       flag DEFAULT 'C');
```

Parameters

Table 173–40 LAPACK_GEES Procedure Parameters

Parameter	Description
jobz	■ 'N': Schur vectors are not computed. ■ 'V': Schur vectors are computed.
n	The order of the matrix a. N >= 0.
a	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (lda, n). ■ On entry, the n by n matrix A. ■ On exit, A has been overwritten by its real Schur form T.
lda	The leading dimension of the array a. lda >= max(1,n).

Table 173–40 (Cont.) LAPACK_GEES Procedure Parameters

Parameter	Description
wr	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n). wr and wi contain the real and imaginary parts respectively of the computed eigenvalues in the same order that they appear on the diagonal of the output Schur form T. Complex conjugate pairs of eigenvalues will appear consecutively with the eigenvalue having the positive imaginary part first.
wi	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldz, n). wr and wi contain the real and imaginary parts respectively of the computed eigenvalues in the same order that they appear on the diagonal of the output Schur form T. Complex conjugate pairs of eigenvalues will appear consecutively with the eigenvalue having the positive imaginary part first.
vs	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n). ■ If jobvs = 'V', vs contains the orthogonal matrix Z of Schur vectors. ■ If jobvs = 'N', vs is not referenced.
ldvs	The leading dimension of the array vs. VS. ldvs >= 1. If jobvs = 'V', ldvs >= N
info	■ = 0 : successful exit ■ < 0 : if info = -i, the i-th argument had an illegal value ■ > 0 : if info = i, and i is <= N: the QR algorithm failed to compute all the eigenvalues. Elements 1:ILO-1 and i+1:N of wr and wi contain those eigenvalues which have converged. If jobvs = 'V', vs contains the matrix which reduces A to its partially converged Schur form.
pack	(Optional) Flags the packing of the matricies: ■ 'C': column-major (default) ■ 'R': row-major

LAPACK_GELS Procedures

This procedure solves overdetermined or underdetermined real linear systems involving an m by n matrix A, or its transpose, using a QR or LQ factorization of A. It is assumed that A has full rank.

The following options are provided:

1. If TRANS = 'N' and m >= n: find the least squares solution of an overdetermined system, that is, solve the least squares problem.

   ```
   minimize || B - A*X ||
   ```

2. If TRANS = 'N' and m < n: find the minimum norm solution of an underdetermined system A * X = B.
3. If TRANS = 'T' and m >= n: find the minimum norm solution of an undetermined system A**T * X = B.
4. If TRANS = 'T' and m < n: find the least squares solution of an overdetermined system, that is, solve the least squares problem minimize || B - A**T * X ||.

See Also: LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms on page 173-11 for other subprograms in this group

Syntax

```
UTL_NLA.LAPACK_GELS (
   trans    IN       flag,
   m        IN       POSITIVEN,
   n        IN       POSITIVEN,
   nrhs     IN       POSITIVEN,
   a        IN OUT   UTL_NLA_ARRAY_DBL,
   lda      IN       POSITIVEN,
   b        IN OUT   UTL_NLA_ARRAY_DBL,
   ldb      IN       POSITIVEN,
   info     OUT      INTEGER,
   pack     IN       flag DEFAULT 'C');

UTL_NLA.LAPACK_GELS (
  trans    IN       flag,
  m        IN       POSITIVEN,
  n        IN       POSITIVEN,
  nrhs     IN       POSITIVEN,
  a        IN OUT   UTL_NLA_ARRAY_FLT,
  lda      IN       POSITIVEN,
  b        IN OUT   UTL_NLA_ARRAY_FLT,
  ldb      IN       POSITIVEN,
  info     OUT      INTEGER,
  pack     IN       flag DEFAULT 'C');
```

Parameters

Table 173–41 LAPACK_GELS Procedure Parameters

Parameter	Description
trans	■ CHARACTER = 'N': The linear system involves A. ■ CHARACTER = 'T': The linear system involves A**T .

Table 173–41 (Cont.) LAPACK_GELS Procedure Parameters

Parameter	Description
m	The number of rows of the matrix a. M >= 0.
n	The number of columns of the matrix a. N >= 0.
nrhs	The number of right-hand sides, which is the number of columns of the matrix band x.nrhs >= 0.
a	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (lda, n). On entry, the matrix b of right hand side vectors, stored columnwise; b is m by nrhs if TRANS = 'N', or n by nrhsif trans = 'T'. On exit, if m >= n, a is overwritten by details of its QR factorization as returned by SGEQRF. If m < n, A is overwritten by details of its LQ factorization as returned by SGELQF.
lda	The leading dimension of the array A. lda >= max(1,m).
b	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldb, nrhs). On entry, the matrix b of right hand side vectors, stored columnwise. b is m bynrhs if trans = 'n', or n by nrhs if trans = 'T'. On exit, b is overwritten by the solution vectors, stored columnwise: ■ If trans = 'n' and m >= n, rows 1 to n of b contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of elements n+1 to m in that column. ■ If trans = 'n' and m < n, rows 1 to n of b contain the minimum norm solution vectors. ■ If trans = 'T' and m >= n, rows 1 to m of b contain the minimum norm solution vectors. ■ If trans = 'T' and m < n, rows 1 to m of b contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of elements m+1 to n in that column.
ldb	The leading dimension of the array b. ldb >= max(1,m,n)
info	■ = 0 : successful exit ■ < 0 : if info = -i, the i-th argument had an illegal value
pack	(Optional) Flags the packing of the matricies: ■ 'C': column-major (default) ■ 'R': row-major

LAPACK_GESDD Procedures

This procedures computes the singular value decomposition (SVD) of a real `m` by `n` matrix `A`, optionally computing the left and right singular vectors. If singular vectors are desired, it uses a divide-and-conquer algorithm that makes mild assumptions about floating point arithmetic.

The SVD is written

```
A = U * SIGMA * transpose(V)
```

where SIGMA is an `m` by `n` matrix which is zero except for its `min(m,n)` diagonal elements, `U` is an`m` by `m` orthogonal matrix, and `V` is an`n` by `n` orthogonal matrix. The diagonal elements of SIGMA are the singular values of `A`, they are real and non-negative, and are returned in descending order. The first `min(m,n)` columns of `U` and `V` are the left and right singular vectors of `A`.

Note that the routine returns `V**T`, not `V`.

> **See Also:** LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms on page 173-11 for other subprograms in this group

Syntax

```
UTL_NLA.LAPACK_GESDD (
   jobz   IN       flag,
   m      IN       POSITIVEN,
   n      IN       POSITIVEN,
   a      IN OUT   UTL_NLA_ARRAY_DBL,
   lda    IN       POSITIVEN,
   s      IN OUT   UTL_NLA_ARRAY_DBL,
   u      IN OUT   UTL_NLA_ARRAY_DBL,
   ldu    IN       POSITIVEN,
   vt     IN OUT   UTL_NLA_ARRAY_DBL,
   ldvt   IN       POSITIVEN,
   info   OUT      INTEGER,
   pack   IN       flag DEFAULT 'C');

UTL_NLA.LAPACK_GESDD (
   jobz   IN       flag,
   m      IN       POSITIVEN,
   n      IN       POSITIVEN,
   a      IN OUT   UTL_NLA_ARRAY_FLT,
   lda    IN       POSITIVEN,
   s      IN OUT   UTL_NLA_ARRAY_FLT,
   u      IN OUT   UTL_NLA_ARRAY_FLT,
   ldu    IN       POSITIVEN,
   vt     IN OUT   UTL_NLA_ARRAY_FLT,
   ldvt   IN       POSITIVEN,
   info   OUT      INTEGER,
   pack   IN       flag DEFAULT 'C');
```

Parameters

Table 173–42 LAPACK_GESDD Procedure Parameters

Parameter	Description
jobz	Specifies options for computing all or part of the matrix U: ■ 'A': All m columns of u and all n rows of V**T are returned in arrays u and vt. ■ 'S': The first min(m,n) columns of u and the first min(m,n) rows of V**T are returned in the arrays u and vt. ■ 'O': The first min(m,n) columns of u (the left singular vectors) are overwritten on the array a. jobu and jobvt cannot both be 'O ■ 'N': No columns of u (no left singular vectors) are computed.
m	The order of the matrix a. m >= 0.
n	The order of the matrix a. n >= 0.
a	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (lda, n). On entry, the n by n matrix A. On exit: ■ If jobz = 'O', a is overwritten with the first min(m,n) columns of u (the left singular vectors, stored columnwise). ■ If m >= n, a is overwritten with the first m rows of V**T (the right singular vectors, stored rowwise). ■ If jobz .ne. 'O', the contents of a are destroyed.
lda	The leading dimension of the array a. lda >= max(1,m).
s	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (min(m,n). The singular values of a, sorted so that S(i) >= S(i+1).
u	UTL_NLA_ARRAY_FLT/DBL. ucol = m if jobz = 'A' or jobz = 'O' and m < n; ucol = min(m,n) if jobz = 'S'. ■ If jobz = 'A' or jobz = 'O' and m < n, u contains the m by m orthogonal matrix u. ■ If jobz = 'S', u contains the first min(m,n) columns of u (the left singular vectors, stored columnwise). ■ If jobz = 'O' and m >= n, or jobz = 'n', u is not referenced.
ldu	The leading dimension of the array U. ldu >= 1. If jobz = 'S' or 'A',or jobz = 'O' and m < n, ldu >= m.
vt	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldvt, n). ■ If jobz = 'A' or jobz = 'O' and m >= n, vt contains the n by n orthogonal matrix V**T. ■ If jobz = 'S', vt contains the first min(m,n) rows of V**T (the right singular vectors, stored rowwise). ■ If jobz = 'O' and m < n, or jobz = 'N', vt is not referenced.

Table 173–42 (Cont.) LAPACK_GESDD Procedure Parameters

Parameter	Description
ldvt	The leading dimension of the array vt. ldvt >= 1. ■ If jobz = 'A', or jobz = 'O' and m >= n, ldvt >= n. ■ If jobz = 'S', ldvt >= min(m,n).
info	■ = 0 : successful exit ■ < 0 : If info = -i, the i-th argument had an illegal value ■ > 0 : SBDSDC did not converge, updating process failed.
pack	(Optional) Flags the packing of the matricies: ■ 'C': column-major (default) ■ 'R': row-major

LAPACK_GESV Procedures

This procedure computes the solution to a real system of linear equations

```
a * x = b
```

where `a` is an `n` by `n` matrix and `x` and `b` are `n` by `nrhs` matrices.

The `LU` decomposition with partial pivoting and row interchanges is used to factor `A` as

```
a = P * L * U
```

where `P` is a permutation matrix, `L` is unit lower triangular, and `U` is upper triangular. The factored form of `a` is then used to solve the system of equations

```
a * x = b
```

See Also: LAPACK Driver Routines (Linear Equations) Subprograms on page 173-10 for other subprograms in this group

Syntax

```
UTL_NLA.LAPACK_GESV (
   n      IN      POSITIVEN,
   nrhs   IN      POSITIVEN,
   a      IN OUT  UTL_NLA_ARRAY_DBL,
   lda    IN      POSITIVEN,
   ipiv   IN OUT  UTL_NLA_ARRAY_INT,
   b      IN OUT  UTL_NLA_ARRAY_DBL,
   ldb    IN      POSITIVEN,
   info      OUT  INTEGER,
   pack   IN      flag DEFAULT 'C');

UTL_NLA.LAPACK_GESV (
   n      IN      POSITIVEN,
   nrhs   IN      POSITIVEN,
   a      IN OUT  UTL_NLA_ARRAY_FLT,
   lda    IN      POSITIVEN,
   ipiv   IN OUT  UTL_NLA_ARRAY_INT,
   b      IN OUT  UTL_NLA_ARRAY_FLT,
   ldb    IN      POSITIVEN,
   info      OUT  INTEGER,
   pack   IN      flag DEFAULT 'C');
```

Parameters

Table 173–43 LAPACK_GESV Procedure Parameters

Parameter	Description
`n`	The number of linear equations, equivalent to the order of the matrix `a`. `n >= 0`
`nrhs`	The number of right-hand sides, which is the number of columns of the matrix `b`. `nrhs >= 0`.

Table 173–43 (Cont.) LAPACK_GESV Procedure Parameters

Parameter	Description
a	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (lda, n). On entry, the n by n coefficient matrix a. On exit, the factors L and U from the factorization a = P*L*U; the unit diagonal elements of L are not stored.
lda	The leading dimension of the array a. lda >= max(1,n)
ipiv	INTEGER array, DIMENSION (n). The pivot indices that define the permutation matrix P; row iof the matrix was interchanged with row ipiv(i).
b	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldb, nrhs). On entry, the n by nrhs matrix of right hand side matrix b. On exit, if info = 0 , the n by nrhs solution matrix X.
ldb	The leading dimension of the array b. ldb >= max(1,n)
info	■ = 0 : successful exit ■ < 0 : if info = -i , the i-th argument had an illegal value ■ > 0 : if info = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.
pack	(Optional) Flags the packing of the matrices: ■ 'C': column-major (default) ■ 'R': row-major

LAPACK_GESVD Procedures

This procedures computes the singular value decomposition (SVD) of a real `m` by `n` matrix `A`, optionally computing the left and/or right singular vectors. The SVD is written

```
A = U * SIGMA * transpose(V)
```

where SIGMA is an `m` by `n` matrix which is zero except for its `min(m,n)` diagonal elements, `U` is an`m` by `m` orthogonal matrix, and `V` is an`n` by `n` orthogonal matrix. The diagonal elements of SIGMA are the singular values of `A`, they are real and non-negative, and are returned in descending order. The first `min(m,n)` columns of `U` and `V` are the left and right singular vectors of `A`.

Note that the routine returns `V**T`, not `V`.

> **See Also:** LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms on page 173-11 for other subprograms in this group

Syntax

```
UTL_NLA.LAPACK_GESVD (
   jobu   IN       flag,
   jobvt  IN       flag,
   m      IN       POSITIVEN,
   n      IN       POSITIVEN,
   a      IN OUT   UTL_NLA_ARRAY_DBL,
   lda    IN       POSITIVEN,
   s      IN OUT   UTL_NLA_ARRAY_DBL,
   u      IN OUT   UTL_NLA_ARRAY_DBL,
   ldu    IN       POSITIVEN,
   vt     IN OUT   UTL_NLA_ARRAY_DBL,
   ldvt   IN       POSITIVEN,
   info   OUT      INTEGER,
   pack   IN       flag DEFAULT 'C');

UTL_NLA.LAPACK_GESVD (
   jobu   IN       flag,
   jobvt  IN       flag,
   m      IN       POSITIVEN,
   n      IN       POSITIVEN,
   a      IN OUT   UTL_NLA_ARRAY_FLT,
   lda    IN       POSITIVEN,
   s      IN OUT   UTL_NLA_ARRAY_FLT,
   u      IN OUT   UTL_NLA_ARRAY_FLT,
   ldu    IN       POSITIVEN,
   vt     IN OUT   UTL_NLA_ARRAY_FLT,
   ldvt   IN       POSITIVEN,
   info   OUT      INTEGER,
   pack   IN       flag DEFAULT 'C');
```

Parameters

Table 173–44 LAPACK_GESVD Procedure Parameters

Parameter	Description
jobu	Specifies options for computing all or part of the matrix U: ■ 'A': All m columns of U are returned in array U. ■ 'S': The first min(m,n) columns of U (the left singular vectors) are returned in the array U. ■ 'O': The first min(m,n) columns of U (the left singular vectors) are overwritten on the array a. jobu and jobvt cannot both be 'O ■ 'N': No columns of U (no left singular vectors) are computed.
jobvt	Specifies options for computing all or part of the matrix V**T: ■ 'A': All n rows of V**T are returned in the array vt. ■ 'S': The first min(m,n) rows of V**T (the right singular vectors) are returned in the array vt. ■ 'O': The first min(m,n) rows of V**T (the right singular vectors) are overwritten on the array a. jobvt and jobu cannot both be 'O'. ■ 'N': No rows of V**T (no right singular vectors) are computed.
m	The order of the matrix a. M >= 0.
n	The order of the matrix a. N >= 0.
a	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (lda, n). On entry, the n by n matrix A. On exit: ■ If jobu = 'O', A is overwritten with the first min(m,n) columns of U (the left singular vectors, stored columnwise); ■ If jobvt = 'O', A is overwritten with the first min(m,n) rows of V**T (the right singular vectors, stored rowwise); ■ If jobu.ne.'O' and jobvt.ne.'O', the contents of A are destroyed.
lda	The leading dimension of the array a. lda >= max(1,n).
s	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (min(m,n). The singular values of A, sorted so that S(i) >= S(i+1).
u	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldu,ucol).(ldu,m) if jobu = 'A' or (ldu,min(m,n)) if jobu = 'S'. ■ If jobu = 'A', U contains the m by m orthogonal matrix U. ■ If jobu = 'S', U contains the first min(m,n) columns of U (the left singular vectors, stored columnwise). ■ If jobu = 'N' or 'O', U is not referenced.
ldu	The leading dimension of the array U. ldu >= 1. If jobu = 'S' or 'a', ldu >= m.

Table 173–44 (Cont.) LAPACK_GESVD Procedure Parameters

Parameter	Description
vt	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldvt, n). ■ If jobvt = 'A', vt contains the n by n orthogonal matrix V**T. ■ If jobvt = 'S', vt contains the first min(m,n) rows of V**T (the right singular vectors, stored rowwise). ■ If jobvt = 'N' or 'O', vt is not referenced.
ldvt	The leading dimension of the array vt. ldvt >= 1. ■ If jobvt = 'A', ldvt >= n. ■ If jobvt = 'S', ldvt >= min(m,n).
info	■ = 0 : successful exit ■ < 0 : If info = -i, the i-th argument had an illegal value ■ > 0 : If SBDSQR did not converge, info specifies how many superdiagonals of an intermediate bidiagonal form B did not converge to zero.
pack	(Optional) Flags the packing of the matricies: ■ 'C': column-major (default) ■ 'R': row-major

LAPACK_GEEV Procedures

This procedures computes for an `n` by `n` real nonsymmetric matrix `A`, the eigenvalues and, optionally, the left and/or right eigenvectors.

- The right eigenvector `v(j)` of `A` satisfies `A * v(j) = lambda(j) * v(j)` where `lambda(j)` is its eigenvalue.
- The left eigenvector `u(j)` of `A` satisfies `u(j)**H * A = lambda(j) * u(j)**H` where `u(j)**H` denotes the conjugate transpose of `u(j)`.

The computed eigenvectors are normalized to have Euclidean norm equal to `1` and largest component real.

> **See Also:** LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms on page 173-11 for other subprograms in this group

Syntax

```
UTL_NLA.LAPACK_GEEV (
   jobvl   IN      flag,
   jobvr   IN      flag,
   n       IN      POSITIVEN,
   a       IN OUT  UTL_NLA_ARRAY_DBL,
   lda     IN      POSITIVEN,
   wr      IN OUT  UTL_NLA_ARRAY_DBL,
   wi      IN OUT  UTL_NLA_ARRAY_DBL,
   vl      IN OUT  UTL_NLA_ARRAY_DBL,
   ldvl    IN      POSITIVEN,
   vr      IN OUT  UTL_NLA_ARRAY_DBL,
   ldvr    IN      POSITIVEN,
   info    OUT     INTEGER,
   pack    IN      flag DEFAULT 'C');

UTL_NLA.LAPACK_GEEV (
   jobvl   IN      flag,
   jobvr   IN      flag,
   n       IN      POSITIVEN,
   a       IN OUT  UTL_NLA_ARRAY_FLT,
   lda     IN      POSITIVEN,
   wr      IN OUT  UTL_NLA_ARRAY_FLT,
   wi      IN OUT  UTL_NLA_ARRAY_FLT,
   vl      IN OUT  UTL_NLA_ARRAY_FLT,
   ldvl    IN      POSITIVEN,
   vr      IN OUT  UTL_NLA_ARRAY_FLT,
   ldvr    IN      POSITIVEN,
   info    OUT     INTEGER,
   pack    IN      flag DEFAULT 'C');
```

Parameters

Table 173–45 LAPACK_GEEV Procedure Parameters

Parameter	Description
`jobvl`	▪ `'N'`: Left eigenvectors of `A` are not computed. ▪ `'V'`: Left eigenvectors of `A` are computed.
`jobvr`	▪ `'N'`: Right eigenvectors of `A` are not computed. ▪ `'V'`: Right eigenvectors of `A` are computed.

Table 173–45 (Cont.) LAPACK_GEEV Procedure Parameters

Parameter	Description
n	The order of the matrix a. N >= 0.
a	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (lda, n). ■ On entry, the n by n matrix A. ■ On exit, A has been overwritten.
lda	The leading dimension of the array a. lda >= max(1,n).
wr	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n). wr and wi contain the real and imaginary parts respectively of the computed eigenvalues. Complex conjugate pairs of eigenvalues will appear consecutively with the eigenvalue having the positive imaginary part first.
wi	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldz, n). wr and wi contain the real and imaginary parts respectively of the computed eigenvalues. Complex conjugate pairs of eigenvalues will appear consecutively with the eigenvalue having the positive imaginary part first.
vl	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n). ■ If jobvl = 'V', the left eigenvectors u(j) are stored one after another in the columns of vl, in the same order as their eigenvalues. ■ If jobvs = 'N', vl is not referenced. ■ If the j-th eigenvalue is real, then u(j) = VL(:,j), the j-th column of vl. ■ If the j-th and (j+1)-st eigenvalues form a complex conjugate pair, then u(j) = VL(:,j) + i*VL(:,j+1) and u(j+1) = VL(:,j) - i*VL(:,j+1).
ldv1	The leading dimension of the array vl. ldvl >= 1. If jobvl = 'v', ldvl >= n.
vr	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldvr, n). ■ If jobvr = 'V', the right eigenvectors v(j) are stored one after another in the columns of vr, in the same order as their eigenvalues.. ■ If jobvr = 'N', vr is not referenced. ■ If the j-th eigenvalue is real, then v(j) = VR(:,j), the j-th column of vr. ■ If the j-th and (j+1)-st eigenvalues form a complex conjugate pair, then v(j) = VR(:,j) + i*VR(:,j+1) and v(j+1) = VR(:,j) - i*VR(:,j+1).
ldvr	The leading dimension of the array vr. vr.ldvr >= 1. If jobvr = 'V', ldvr >= N
info	■ = 0 : successful exit ■ < 0 : if info = -i, the i-th argument had an illegal value ■ > 0 : if info = i, and i is <= N: the QR algorithm failed to compute all the eigenvalues, and no eigenvectors have been computed. Elements i+1:N of wr and wi contain eigenvalues which have converged..

Table 173–45 (Cont.) LAPACK_GEEV Procedure Parameters

Parameter	Description
`pack`	(Optional) Flags the packing of the matricies: ■ `'C'`: column-major (default) ■ `'R'`: row-major

LAPACK_GTSV Procedures

This procedure solves the equation

```
a * x = b
```

where a is an n by n tridiagonal matrix, by Gaussian elimination with partial pivoting.

Note that the equation a'*x = b may be solved by interchanging the order of the arguments du and dl.

> **See Also:** LAPACK Driver Routines (Linear Equations) Subprograms on page 173-10 for other subprograms in this group

Syntax

```
UTL_NLA.LAPACK_GTSV (
   n      IN      POSITIVEN,
   nrhs   IN      POSITIVEN,
   dl     IN OUT  UTL_NLA_ARRAY_DBL,
   d      IN OUT  UTL_NLA_ARRAY_DBL,
   du     IN OUT  UTL_NLA_ARRAY_DBL,
   b      IN OUT  UTL_NLA_ARRAY_DBL,
   ldb    IN      POSITIVEN,
   info       OUT  INTEGER,
   pack   IN      flag DEFAULT 'C');

UTL_NLA.LAPACK_GTSV (
   n      IN      POSITIVEN,
   nrhs   IN      POSITIVEN,
   dl     IN OUT  UTL_NLA_ARRAY_FLT,
   d      IN OUT  UTL_NLA_ARRAY_FLT,
   du     IN OUT  UTL_NLA_ARRAY_FLT,
   b      IN OUT  UTL_NLA_ARRAY_FLT,
   ldb    IN      POSITIVEN,
   info       OUT  INTEGER,
   pack   IN      flag DEFAULT 'C');
```

Parameters

Table 173–46 LAPACK_GTSV Procedure Parameters

Parameter	Description
n	The order of the matrix a .n >= 0
nrhs	The number of right-hand sides, which is the number of columns of the matrix b. nrhs >= 0.
dl	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n-1). On entry, dl must contain the (n-1) sub-diagonal elements of a. On exit, dl is overwritten by the (n-2) elements of the second super-diagonal of the upper triangular matrix U from the LU factorization of a, in dl(1), ..., dl(n-2).
d	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n). On entry, d must contain the diagonal elements of a. On exit, d is overwritten by the n diagonal elements of U.

Table 173–46 (Cont.) LAPACK_GTSV Procedure Parameters

Parameter	Description
du	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n-1). On entry, du must contain the (n-1) super-diagonal elements of a. On exit, du is overwritten by the (n-1) elements of the first super-diagonal of U.
b	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (LDB, nrhs). On entry, the n by nrhs matrix of right hand side matrix b. On exit, if info = 0, the n by nrhs solution matrix X.
ldb	The leading dimension of the array b. ldb >= max (1, n)
info	■ = 0 : successful exit ■ < 0 : if info = -i , the i-th argument had an illegal value ■ > 0 : if info = i, U(i,i) is exactly zero, and the solution has not been computed. The factorization has not been completed unless i = n.
pack	(Optional) Flags the packing of the matrices: ■ 'C': column-major (default) ■ 'R': row-major

LAPACK_PBSV Procedures

This procedure computes the solution to a real system of linear equations

```
a * x = b
```

where `a` is an n by n symmetric positive definite band matrix and `x` and `b` are `n` by `nrhs` matrices.

The Cholesky decomposition is used to factor `A` as

`A = U**T*U` if `UPLO ='U'`

or

`A = L * L**T` if `UPLO = 'L'`

where `U` is an upper triangular matrix and `L` is a lower triangular matrix. The factored form of `A` is then used to solve the system of equations `A * X = B`.

> **See Also:** LAPACK Driver Routines (Linear Equations) Subprograms on page 173-10 for other subprograms in this group

Syntax

```
UTL_NLA.LAPACK_PBSV (
uplo       IN      flag,
n          IN      POSITIVEN,
kd         IN      NATURALN,
nrhs       IN      POSITIVEN,
ab         IN OUT  UTL_NLA_ARRAY_DBL,
ldab       IN      POSITIVEN,
b          IN OUT  UTL_NLA_ARRAY_DBL,
ldb        IN      POSITIVEN,
info       OUT     INTEGER,
pack       IN      flag DEFAULT 'C');

UTL_NLA.LAPACK_PBSV (
uplo      IN      flag,
n         IN      POSITIVEN,
kd        IN      NATURALN,
nrhs      IN      POSITIVEN,
ab        IN OUT  UTL_NLA_ARRAY_FLT,
ldab      IN      POSITIVEN,
b         IN OUT  UTL_NLA_ARRAY_FLT,
ldb       IN      POSITIVEN,
info      OUT     INTEGER,
pack      IN      flag DEFAULT 'C');
```

Parameters

Table 173–47 LAPACK_PBSV Procedure Parameters

Parameter	Description
`uplo`	▪ `uplo = 'U'`. Upper triangular of `A` is stored. ▪ `uplo = 'L'`. Lower triangular of `A` is stored.
`n`	The number of linear equations, that is, the order of the matrix `a .n >= 0`

Table 173–47 (Cont.) LAPACK_PBSV Procedure Parameters

Parameter	Description
kd	The number of superdiagonals of the matrix A if uplo = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.
nrhs	The number of right-hand sides, which is the number of columns of the matrix b. nrhs >= 0.
ab	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldab, n). On entry, the upper or lower triangle of the symmetric band matrix a, stored in the first kd+1 rows of the array. The j-th column of a is stored in the j-th column of the array ab is as follows: ■ if uplo = 'U',AB(KD+1+i-j,j) = A(i,j) for max(1,j-KD)<=i<=j; ■ if uplo = 'L',AB(1+i-j,j) = A(i,j) for j<=i<=min(N,j+KD) .See below for further details. On exit, if info = 0, the triangular factor U or L from the Cholesky factorization A = U**T*U or A = L*L**T of the band matrix A, in the same storage format as a.
ldab	The leading dimension of the array ab. ldb >= kd+1
b	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldb, nrhs). On entry, the n by nrhs matrix of right hand side matrix b. On exit, if info = 0 , the n by nrhs solution matrix X.
ldb	The leading dimension of the array b. ldb >= max(1,n)
info	■ = 0 : successful exit ■ < 0 : if info = -i , the i-th argument had an illegal value ■ > 0 : if info = i, the leading minor of order a of a is not positive definite, so the factorization could not be completed, and the solution has not been computed.
pack	(Optional) Flags the packing of the matrices: ■ 'C': column-major (default) ■ 'R': row-major

LAPACK_POSV Procedures

This procedure computes the solution to a real system of linear equations

```
a * x = b
```

where `a` is an n by n symmetric positive definite matrix and `x` and `b` are `n` by `nrhs` matrices.

The Cholesky decomposition is used to factor `A` as

`A = U**T* U` if `uplo = 'U'`

or

`A = L * L**T` if `UPLO = 'L'`

where `U` is an upper triangular matrix and `L` is a lower triangular matrix. The factored form of `A` is then used to solve the system of equations `A * X = B`.

> **See Also:** LAPACK Driver Routines (Linear Equations) Subprograms on page 173-10 for other subprograms in this group

Syntax

```
UTL_NLA.LAPACK_POSV (
   uplo      IN      flag,
   n         IN      POSITIVEN,
   nrhs      IN      POSITIVEN,
   a         IN OUT  UTL_NLA_ARRAY_DBL,
   lda       IN      POSITIVEN,
   b         IN OUT  UTL_NLA_ARRAY_DBL,
   ldb       IN      POSITIVEN,
   info      OUT     INTEGER,
   pack      IN      flag DEFAULT 'C');

UTL_NLA.LAPACK_POSV (
   uplo      IN      flag,
   n         IN      POSITIVEN,
   nrhs      IN      POSITIVEN,
   a         IN OUT  UTL_NLA_ARRAY_FLT,
   lda       IN      POSITIVEN,
   b         IN OUT  UTL_NLA_ARRAY_FLT,
   ldb       IN      POSITIVEN,
   info      OUT     INTEGER,
   pack      IN      flag DEFAULT 'C');
```

Parameters

Table 173–48 LAPACK_POSV Procedure Parameters

Parameter	Description
`uplo`	■ `uplo` = 'U'. Upper triangular of `A` is stored. ■ `uplo` = `'L'`. Lower triangular of `A` is stored.
`n`	The number of linear equations, that is, the order of the matrix `a` .`n >= 0`
`nrhs`	The number of right-hand sides, which is the number of columns of the matrix `b`. `nrhs >= 0`.

Table 173–48 (Cont.) LAPACK_POSV Procedure Parameters

Parameter	Description
a	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (lda, n). If uplo = 'U', the leading NRHS n by n upper triangular part of a contains the upper NRHS triangular part of the matrix A, and the strictly lower NRHS triangular part of A is not referenced. If uplo = ''L', then rhs leading n by n lower triangular part of a contains the lower nrhs triangular part of the matrix a, and the strictly upper nrhs triangular part of a is not referenced. On exit, if info = 0, the factor U or L from the Cholesky factorization A = U**T*U or A = L*L**T.
lda	The leading dimension of the array a. lda >= max (1, n)
b	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldb, nrhs). On entry, the n by nrhs matrix of right hand side matrix b. On exit, if info = 0 , the n by nrhs solution matrix X.
ldb	The leading dimension of the array b. ldb >= max(1,n)
info	■ = 0 : successful exit ■ < 0 : if info = -i , the i-th argument had an illegal value ■ > 0 : if info = i, the leading minor of order i of a is not positive definite, so the factorization could not be completed, and the solution has not been computed.
pack	(Optional) Flags the packing of the matrices: ■ 'C': column-major (default) ■ 'R': row-major

LAPACK_PPSV Procedures

This procedure computes the solution to a real system of linear equations

```
a * x = b
```

where `a` is an n by n symmetric positive definite matrix stored in packed format and `x` and `b` are `n` by `nrhs` matrices.

The Cholesky decomposition is used to factor `A` as

`A = U**T* U` if `UPLO = 'U'`

or

`A = L * L**T` if `UPLO = 'L'`

where **U** is an upper triangular matrix and `L` is a lower triangular matrix. The factored form of `A` is then used to solve the system of equations `A * X = B`.

> **See Also:** LAPACK Driver Routines (Linear Equations) Subprograms on page 173-10 for other subprograms in this group

Syntax

```
UTL_NLA.LAPACK_PPSV (
   uplo      IN      flag,
   n         IN      POSITIVEN,
   nrhs      IN      POSITIVEN,
   ap        IN OUT  UTL_NLA_ARRAY_DBL,
   b         IN OUT  UTL_NLA_ARRAY_DBL,
   ldb       IN      POSITIVEN,
   info      OUT     INTEGER,
   pack      IN      flag DEFAULT 'C');

UTL_NLA.LAPACK_PPSV (
   uplo      IN      flag,
   n         IN      POSITIVEN,
   nrhs      IN      POSITIVEN,
   ap        IN OUT  UTL_NLA_ARRAY_FLT,
   b         IN OUT  UTL_NLA_ARRAY_FLT,
   ldb       IN      POSITIVEN,
   info      OUT     INTEGER,
   pack      IN      flag DEFAULT 'C');
```

Parameters

Table 173–49 LAPACK_PPSV Procedure Parameters

Parameter	Description
`uplo`	▪ `uplo = 'U'` . Upper triangular of `A` is stored. ▪ `uplo = 'L'`. Lower triangular of `A` is stored.
`n`	The number of linear equations, that is, the order of the matrix `a` .`n >= 0`
`nrhs`	The number of right-hand sides, which is the number of columns of the matrix `b`. `nrhs >= 0`.

Table 173–49 (Cont.) LAPACK_PPSV Procedure Parameters

Parameter	Description
ap	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n*(n+1)/2). On entry, the upper or lower triangle of the symmetric matrix a, packed columnwise in a linear array. The j-th column of a is stored in the array ap as follows: If uplo = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; If uplo = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n; On exit, if info = 0, the factor U or 'L' from the Cholesky factorization A = U**T*U or A = L*L**T in the same storage format as A.
b	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldb, nrhs). On entry, the n by nrhs matrix of right hand side matrix b. On exit, if info = 0 , the n by nrhs solution matrix X.
ldb	The leading dimension of the array b. ldb >= max(1,n)
info	■ = 0 : successful exit ■ < 0 : if info = -i , the i-th argument had an illegal value ■ > 0 : if info = i, the leading minor of order i of a is not positive definite, so the factorization could not be completed, and the solution has not been computed.
pack	(Optional) Flags the packing of the matrices: ■ 'C': column-major (default) ■ 'R': row-major

LAPACK_PTSV Procedures

This procedure computes the solution to a real system of linear equations

```
a * x = b
```

where a is an n by n symmetric positive definite tridiagonal matrix, and x and b are n by nrhs matrices.

a is factored as A = L*D*L**T, and the factored form of a is then used to solve the system of equations.

See Also: LAPACK Driver Routines (Linear Equations) Subprograms on page 173-10 for other subprograms in this group

Syntax

```
UTL_NLA.LAPACK_PTSV (
   n      IN       POSITIVEN,
   nrhs   IN       POSITIVEN,
   d      IN OUT   UTL_NLA_ARRAY_DBL,
   e      IN OUT   UTL_NLA_ARRAY_DBL,
   b      IN OUT   UTL_NLA_ARRAY_DBL,
   ldb    IN       POSITIVEN,
   info   OUT      INTEGER,
   pack   IN       flag  DEFAULT 'C');

UTL_NLA.LAPACK_PTSV (
   n      IN       POSITIVEN,
   nrhs   IN       POSITIVEN,
   d      IN OUT   UTL_NLA_ARRAY_FLT,
   e      IN OUT   UTL_NLA_ARRAY_FLT,
   b      IN OUT   UTL_NLA_ARRAY_FLT,
   ldb    IN       POSITIVEN,
   info   OUT      INTEGER,
   pack   IN       flag DEFAULT 'C');
```

Parameters

Table 173–50 LAPACK_PTSV Procedure Parameters

Parameter	Description
n	The order of the matrix a. N >= 0.
nrhs	The number of right-hand sides, which is the number of columns of the matrix b. nrhs >= 0.
d	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n). On entry, the n diagonal elements of the tridiagonal matrix a. On exit, the n diagonal elements of the diagonal matrix d from the factorization A = L*D*L**T.
e	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n-1). On entry, the (n-1) subdiagonal elements of the tridiagonal matrix a. On exit, the (n-1) diagonal elements of the unit bidiagonal factor L from the factorization A = L*D*L**T of a.(e can also be regarded as the superdiagonal of the unit bidiagonal factor U from the U**T*D*U factorization of a)

Table 173–50 (Cont.) LAPACK_PTSV Procedure Parameters

Parameter	Description
b	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldb, nrhs). On entry, the n by nrhs matrix of right hand side matrix b. On exit, if info = 0 , the n by nrhs solution matrix X.
ldb	The leading dimension of the array b. ldb >= max(1,n)
info	■ = 0 : successful exit ■ < 0 : if info = -i , the i-th argument had an illegal value ■ > 0 : if info = i, the leading minor of order i of a is not positive definite, so the factorization could not be completed, and the solution has not been computed.
pack	(Optional) Flags the packing of the matrices: ■ 'C': column-major (default) ■ 'R': row-major

LAPACK_SBEV Procedures

This procedure computes all the eigenvalues and, optionally, eigenvectors of a real symmetric band matrix `A`.

See Also: LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms on page 173-11 for other subprograms in this group

Syntax

```
UTL_NLA.LAPACK_SBEV (
   jobz     IN      flag,
   uplo     IN      flag,
   n        IN      POSITIVEN,
   kd       IN      NATURALN,
   ab       IN OUT  UTL_NLA_ARRAY_DBL,
   ldab     IN      POSITIVEN,
   w        IN OUT  UTL_NLA_ARRAY_DBL,
   z        IN OUT  UTL_NLA_ARRAY_DBL,
   ldz       IN      POSITIVEN,
   info     OUT     INTEGER,
   pack     IN      flag DEFAULT 'C');

UTL_NLA.LAPACK_SBEV (
   jobz     IN      flag,
   uplo     IN      flag,
   n        IN      POSITIVEN,
   kd       IN      NATURALN,
   ab       IN OUT  UTL_NLA_ARRAY_FLT,
   ldab     IN      POSITIVEN,
   w        IN OUT  UTL_NLA_ARRAY_FLT,
   z        IN OUT  UTL_NLA_ARRAY_FLT,
   ldz      IN      POSITIVEN,
   info     OUT     INTEGER,
   pack     IN      flag DEFAULT 'C');
```

Parameters

Table 173–51 LAPACK_SBEV Procedure Parameters

Parameter	Description
`jobz`	■ `'N'`: Compute eigenvalues only. ■ `'V'`: Compute eigenvalues and eigenvectors.
`uplo`	■ `'U'`: Upper triangle of A is stored. ■ `'L'`: Lower triangle of A is stored.
`n`	The order of the matrix `a. N >= 0`.
`kd`	The number of superdiagonals of the matrix `A` if `uplo = 'U'`, or the number of subdiagonals if `uplo = 'L'. kd >= 0`.

Table 173–51 (Cont.) LAPACK_SBEV Procedure Parameters

Parameter	Description
ab	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldab, n). On entry, the upper or lower triangle of the symmetric band matrix A stored in the first kd+1 rows of the array. The j-th column of A is stored in the j-th column of the array ab: ■ If uplo = 'U', ab(kd+1+i-j,j) = a(i,j) for max(1,j-kd)<=i<=j. ■ If uplo = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd). On exit, ab is overwritten by values generated during the reduction to tridiagonal form: ■ If uplo = 'U', the diagonal and first superdiagonal of the tridiagonal matrix T are returned in rows kd and kd+1 of ab. ■ If uplo = 'L', the diagonal and first subdiagonal of T are returned in the first two rows of ab.
ldab	The leading dimension of the array ab. ldab >= kd + 1.
w	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n). If info = 0, the eigenvalues in ascending order.
z	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n). ■ If jobz = 'V', then if info = 0, z contains the orthonormal eigenvectors of the matrix A, with the i-th column of z holding the eigenvector associated with w(i). ■ If jobz = 'N', then z is not referenced.
ldz	The leading dimension of the array z. ldz >= 1, and if jobz = 'v', ldz >= max(1,n).
info	■ = 0 : successful exit ■ < 0 : if info = -i, the i-th argument had an illegal value ■ > 0 : if info = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero
pack	(Optional) Flags the packing of the matricies: ■ 'C': column-major (default) ■ 'R': row-major

LAPACK_SBEVD Procedures

This procedure computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix A. If eigenvectors are desired, it uses a divide and conquer algorithm that makes mild assumptions about floating point arithmetic.

See Also: LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms on page 173-11 for other subprograms in this group

Syntax

```
UTL_NLA.LAPACK_SBEVD (
   jobz     IN      flag,
   uplo     IN      flag,
   n        IN      POSITIVEN,
   kd       IN      NATURALN,
   ab       IN OUT  UTL_NLA_ARRAY_DBL,
   ldab     IN      POSITIVEN,
   w        IN OUT  UTL_NLA_ARRAY_DBL,
   z        IN OUT  UTL_NLA_ARRAY_DBL,
   ldz       IN      POSITIVEN,
   info     OUT     INTEGER,
   pack     IN      flag DEFAULT 'C');

UTL_NLA.LAPACK_SBEVD (
   jobz     IN      flag,
   uplo     IN      flag,
   n        IN      POSITIVEN,
   kd       IN      NATURALN,
   ab       IN OUT  UTL_NLA_ARRAY_FLT,
   ldab     IN      POSITIVEN,
   w        IN OUT  UTL_NLA_ARRAY_FLT,
   z        IN OUT  UTL_NLA_ARRAY_FLT,
   ldz      IN      POSITIVEN,
   info     OUT     INTEGER,
   pack     IN      flag DEFAULT 'C');
```

Parameters

Table 173–52 LAPACK_SBEVD Procedure Parameters

Parameter	Description
jobz	■ 'N': Compute eigenvalues only. ■ 'V': Compute eigenvalues and eigenvectors.
uplo	■ 'U': Upper triangle of A is stored. ■ 'L': Lower triangle of A is stored.
n	The order of the matrix a. N >= 0.
kd	The number of superdiagonals of the matrix A if uplo = 'U', or the number of subdiagonals if uplo = 'L'. kd >= 0.

Table 173–52 (Cont.) LAPACK_SBEVD Procedure Parameters

Parameter	Description
ab	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldab, n). On entry, the upper or lower triangle of the symmetric band matrix A stored in the first kd+1 rows of the array. The j-th column of A is stored in the j-th column of the array ab: ■ If uplo = 'U',ab(kd+1+i-j,j) = a(i,j) for max(1,j-kd)<=i<=j. ■ If uplo = 'L',AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd). On exit, ab is overwritten by values generated during the reduction to tridiagonal form: ■ If uplo = 'U', the diagonal and first superdiagonal of the tridiagonal matrix T are returned in rows kd and kd+1 of ab. ■ If uplo = 'L', the diagonal and first subdiagonal of T are returned in the first two rows of ab.
ldab	The leading dimension of the array ab. ldab >= kd + 1.
w	UTL_NLA_ARRAY_FLT/DBL,DIMENSION (ldz,n). If info = 0, the eigenvalues in ascending order.
z	UTL_NLA_ARRAY_FLT/DBL,DIMENSION (n). ■ If jobz = 'V', then if info = 0, z contains the orthonormal eigenvectors of the matrix A, with the i-th column of z holding the eigenvector associated with w(i). ■ If jobz = 'N', then z is not referenced.
ldz	The leading dimension of the array z. ldz >= 1, and if jobz = 'v',ldz >= max(1,n).
info	■ = 0 : successful exit ■ < 0 : if info = -i, the i-th argument had an illegal value ■ > 0 : if info = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero
pack	(Optional) Flags the packing of the matricies: ■ 'C': column-major (default) ■ 'R': row-major

LAPACK_SPEV Procedures

This procedure computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix `A` in packed storage.

> **See Also:** LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms on page 173-11 for other subprograms in this group

Syntax

```
UTL_NLA.LAPACK_SPEV (
   jobz     IN      flag,
   uplo     IN      flag,
   n        IN      POSITIVEN,
   ap       IN OUT  UTL_NLA_ARRAY_DBL,
   w        IN OUT  UTL_NLA_ARRAY_DBL,
   z        IN OUT  UTL_NLA_ARRAY_DBL,
   ldz      IN      POSITIVEN,
   info     OUT     INTEGER,
   pack     IN      flag DEFAULT 'C');

UTL_NLA.LAPACK_SPEV (
   jobz     IN      flag,
   uplo     IN      flag,
   n        IN      POSITIVEN,
   ap       IN OUT  UTL_NLA_ARRAY_FLT,
   w        IN OUT  UTL_NLA_ARRAY_FLT,
   z        IN OUT  UTL_NLA_ARRAY_FLT,
   ldz      IN      POSITIVEN,
   info     OUT     INTEGER,
   pack     IN      flag DEFAULT 'C');
```

Parameters

Table 173–53 LAPACK_SPEV Procedure Parameters

Parameter	Description
`jobz`	▪ `'N'`: Compute eigenvalues only. ▪ `'V'`: Compute eigenvalues and eigenvectors.
`uplo`	▪ `'U'`: Upper triangle of `A` is stored. ▪ `'L'`: Lower triangle of A is stored.
`n`	The order of the matrix `a. N >= 0`.

Table 173–53 (Cont.) LAPACK_SPEV Procedure Parameters

Parameter	Description
ap	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n*(n+1)/2). On entry, the upper or lower triangle of the symmetric matrix a packed columnwise in a linear array. The j-th column of a is stored in the array ap: ■ If uplo = 'U',ap(i + (j-1)*j/2) = a(i,j) for 1<=i<=j. ■ If uplo = 'L',ap(i + (j-1)*(2*n-j)/2) = a(i,j) for j<=i<=n. On exit, ap is overwritten by values generated during the reduction to tridiagonal form: ■ If uplo = 'U', the diagonal and first superdiagonal of the tridiagonal matrix T overwrite the corresponding elements of A. ■ If uplo = 'L', the diagonal and first subdiagonal of T overwrite the corresponding elements of A.
w	UTL_NLA_ARRAY_FLT/DBL,DIMENSION (n). If info = 0, the eigenvalues in ascending order.
z	UTL_NLA_ARRAY_FLT/DBL,DIMENSION (ldz,n). ■ If jobz = 'V', then if info = 0, z contains the orthonormal eigenvectors of the matrix A, with the i-th column of z holding the eigenvector associated with w(i). ■ If jobz = 'N', then z is not referenced.
ldz	The leading dimension of the array z. ldz >= 1, and if jobz = 'v',ldz >= max(1,n).
info	■ = 0 : successful exit ■ < 0 : if info = -i, the i-th argument had an illegal value ■ > 0 : if info = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero
pack	(Optional) Flags the packing of the matricies: ■ 'C': column-major (default) ■ 'R': row-major

LAPACK_SPEVD Procedures

This procedure computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix A in packed storage. If eigenvectors are desired, it uses a divide and conquer algorithm. The divide and conquer algorithm makes very mild assumptions about floating point arithmetic.

> **See Also:** LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms on page 173-11 for other subprograms in this group

Syntax

```
UTL_NLA.LAPACK_SPEVD (
   jobz     IN      flag,
   uplo     IN      flag,
   n        IN      POSITIVEN,
   ap       IN OUT  UTL_NLA_ARRAY_DBL,
   w        IN OUT  UTL_NLA_ARRAY_DBL,
   z        IN OUT  UTL_NLA_ARRAY_DBL,
   ldz       IN      POSITIVEN,
   info     OUT     INTEGER,
   pack     IN      flag DEFAULT 'C');

UTL_NLA.LAPACK_SPEVD (
   jobz     IN      flag,
   uplo     IN      flag,
   n        IN      POSITIVEN,
   ap       IN OUT  UTL_NLA_ARRAY_FLT,
   w        IN OUT  UTL_NLA_ARRAY_FLT,
   z        IN OUT  UTL_NLA_ARRAY_FLT,
   ldz      IN      POSITIVEN,
   info     OUT     INTEGER,
   pack     IN      flag DEFAULT 'C');
```

Parameters

Table 173–54 LAPACK_SPEVD Procedure Parameters

Parameter	Description
jobz	■ 'N': Compute eigenvalues only. ■ 'V': Compute eigenvalues and eigenvectors.
uplo	■ 'U': Upper triangle of A is stored. ■ 'L': Lower triangle of A is stored.
n	The order of the matrix a. N >= 0.

Table 173–54 (Cont.) LAPACK_SPEVD Procedure Parameters

Parameter	Description
ap	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n*(n+1)/2). On entry, the upper or lower triangle of the symmetric matrix a packed columnwise in a linear array. The j-th column of a is stored in the array ap: ■ If uplo = 'U',ap(i + (j-1)*j/2) = a(i,j) for 1<=i<=j. ■ If uplo = 'L',ap(i + (j-1)*(2*n-j)/2) = a(i,j) for j<=i<=n. On exit, ap is overwritten by values generated during the reduction to tridiagonal form: ■ If uplo = 'U', the diagonal and first superdiagonal of the tridiagonal matrix T overwrite the corresponding elements of A. ■ If uplo = 'L', the diagonal and first subdiagonal of T overwrite the corresponding elements of A.
w	UTL_NLA_ARRAY_FLT/DBL,DIMENSION (n). If info = 0, the eigenvalues in ascending order.
z	UTL_NLA_ARRAY_FLT/DBL,DIMENSION (ldz,n). ■ If jobz = 'V', then if info = 0, z contains the orthonormal eigenvectors of the matrix A, with the i-th column of z holding the eigenvector associated with w(i). ■ If jobz = 'N', then z is not referenced.
ldz	The leading dimension of the array z. ldz >= 1, and if jobz = 'v',ldz >= max(1,n).
info	■ = 0 : successful exit ■ < 0 : if info = -i, the i-th argument had an illegal value ■ > 0 : if info = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero
pack	(Optional) Flags the packing of the matricies: ■ 'C': column-major (default) ■ 'R': row-major

LAPACK_SPSV Procedures

This procedure computes the solution to a real system of linear equations

```
a * x = b
```

where `a` is an n by n symmetric matrix stored in packed format, and `x` and `b` are n by nrhs matrices.

The diagonal pivoting method is used to factor `A` as

`A = U * D * U**T`, if `UPLO = 'U'`

or

`A = L * D * L**T`, if `UPLO = 'L'`

where `U` (or `L`) is a product of permutation and unit upper (lower) triangular matrices, and `D` is symmetric and block diagonal with 1 by 1 and 2 by 2 diagonal blocks. The factored form of `A` is then used to solve the system of equations `A * X = B`.

See Also: LAPACK Driver Routines (Linear Equations) Subprograms on page 173-10 for other subprograms in this group

Syntax

```
UTL_NLA.LAPACK_SPSV (
   uplo    IN      flag,
   n       IN      POSITIVEN,
   nrhs    IN      POSITIVEN,
   ap      IN OUT  UTL_NLA_ARRAY_DBL,
   ipiv    IN OUT  UTL_NLA_ARRAY_INT,
   b       IN OUT  UTL_NLA_ARRAY_DBL,
   ldb     IN      POSITIVEN,
   info    OUT     INTEGER,
   pack    IN      flag DEFAULT 'C');

UTL_NLA.LAPACK_SPSV (
   uplo   IN       flag,
   n      IN       POSITIVEN,
   nrhs   IN       POSITIVEN,
   ap     IN OUT   UTL_NLA_ARRAY_FLT,
   ipiv   IN OUT   UTL_NLA_ARRAY_INT,
   b      IN OUT   UTL_NLA_ARRAY_FLT,
   ldb    IN       POSITIVEN,
   info   OUT      INTEGER,
   pack   IN       flag DEFAULT 'C');
```

Parameters

Table 173–55 LAPACK_SPSV Procedure Parameters

Parameter	Description
`uplo`	▪ `uplo = 'U'`. Upper triangular of `A` is stored. ▪ `uplo = 'L'` . Lower triangular of `A` is stored.
`n`	The number of linear equations, which is the order of the matrix `a`. `N >= 0`.
`nrhs`	The number of right-hand sides, which is the number of columns of the matrix `b`. `nrhs >= 0`.

Table 173–55 (Cont.) LAPACK_SPSV Procedure Parameters

Parameter	Description
ap	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n*(n+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array ap as follows: ■ uplo = 'U': AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j ■ uplo = 'L': AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n See below for further details. On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U**T or A = L*D*L**T as computed by SSPTRF, stored as a packed triangular matrix in the same storage format as A.
ipiv	INTEGER array, DIMENSION (n). Details of the interchanges and the block structure of d, as determined by SSPTRF. ■ If ipiv(k) > 0, then rows and columns k and ipiv(k) were interchanged, and d(k,k) is a 1 by1 diagonal block. ■ If uplo = 'U' and ipiv(k) = ipiv(k-1) < 0, then rows and columns k-1 and -ipiv(k) were interchanged and d(k-1:k,k-1:k) is a 2 by 2 diagonal block. ■ If uplo = 'L' and ipiv(k) = ipiv(k+1) < 0, then rows and columns k+1 and -ipiv(k) were interchanged and d(k:k+1,k:k+1) is a 2 by 2 diagonal block.
b	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldb, nrhs). On entry, the n by nrhs right hand side matrix b. On exit, if info = 0 , the n by nrhs solution matrix X.
ldb	The leading dimension of the array b. ldb >= max(1,n)
info	■ = 0 : successful exit ■ < 0 : if info = -i, the i-th argument had an illegal value ■ > 0 : if info = i, d(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix d is exactly singular, so the solution could not be computed.
pack	(Optional) Flags the packing of the matrices: ■ 'C': column-major (default) ■ 'R': row-major

LAPACK_STEV Procedures

This procedure computes all eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix A.

> **See Also:** LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms on page 173-11 for other subprograms in this group

Syntax

```
UTL_NLA.LAPACK_STEV (
   jobz     IN      flag,
   n        IN      POSITIVEN,
   d        IN OUT  UTL_NLA_ARRAY_DBL,
   e        IN OUT  UTL_NLA_ARRAY_DBL,
   z        IN OUT  UTL_NLA_ARRAY_DBL,
   ldz      IN      POSITIVEN,
   info     OUT     INTEGER,
   pack     IN      flag DEFAULT 'C');

UTL_NLA.LAPACK_STEV (
   jobz     IN      flag,
   n        IN      POSITIVEN,
   d        IN OUT  UTL_NLA_ARRAY_FLT,
   e        IN OUT  UTL_NLA_ARRAY_FLT,
   z        IN OUT  UTL_NLA_ARRAY_FLT,
   ldz      IN      POSITIVEN,
   info     OUT     INTEGER,
   pack     IN      flag DEFAULT 'C');
```

Parameters

Table 173–56 LAPACK_STEV Procedure Parameters

Parameter	Description
jobz	■ 'N': Compute eigenvalues only. ■ 'V': Compute eigenvalues and eigenvectors.
n	The order of the matrix a. N >= 0.
d	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n). ■ On entry, the n diagonal elements of the tridiagonal matrix A. ■ On exit, if info = 0, the eigenvalues in ascending order.
e	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n). ■ On entry, the (n-1) subdiagonal elements of the tridiagonal matrix A, stored in elements 1 to n-1 of e. e(n) need not be set, but is used by the subprogram. ■ On exit, the contents of e are destroyed.
z	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldz, n). ■ If jobz = 'V', then if info = 0, z contains the orthonormal eigenvectors of the matrix A, with the i-th column of z holding the eigenvector associated with d(i). ■ If jobz = 'N', then z is not referenced.

Table 173–56 (Cont.) LAPACK_STEV Procedure Parameters

Parameter	Description
ldz	The leading dimension of the array z. ldz >= 1, and if jobz = 'v',ldz >= max(1,n).
info	■ = 0 : successful exit ■ < 0 : if info = -i, the i-th argument had an illegal value ■ > 0 : if info = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero
pack	(Optional) Flags the packing of the matricies: ■ 'C': column-major (default) ■ 'R': row-major

LAPACK_STEVD Procedures

This procedure computes all eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix. If eigenvectors are desired, it uses a divide and conquer algorithm that makes mild assumptions about floating point arithmetic.

See Also: LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms on page 173-11 for other subprograms in this group

Syntax

```
UTL_NLA.LAPACK_STEVD (
   jobz     IN      flag,
   n        IN      POSITIVEN,
   d        IN OUT  UTL_NLA_ARRAY_DBL,
   e        IN OUT  UTL_NLA_ARRAY_DBL,
   z        IN OUT  UTL_NLA_ARRAY_DBL,
   ldz      IN      POSITIVEN,
   info     OUT     INTEGER,
   pack     IN      flag DEFAULT 'C');

UTL_NLA.LAPACK_STEVD(
   jobz     IN      flag,
   n        IN      POSITIVEN,
   d        IN OUT  UTL_NLA_ARRAY_FLT,
   e        IN OUT  UTL_NLA_ARRAY_FLT,
   z        IN OUT  UTL_NLA_ARRAY_FLT,
   ldz      IN      POSITIVEN,
   info     OUT     INTEGER,
   pack     IN      flag DEFAULT 'C');
```

Parameters

Table 173–57 LAPACK_STEVD Procedure Parameters

Parameter	Description
jobz	■ 'N': Compute eigenvalues only. ■ 'V': Compute eigenvalues and eigenvectors.
n	The order of the matrix a. N >= 0.
d	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n). ■ On entry, the n diagonal elements of the tridiagonal matrix A. ■ On exit, if info = 0, the eigenvalues in ascending order.
e	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n). ■ On entry, the (n-1) subdiagonal elements of the tridiagonal matrix A, stored in elements 1 to n-1 of e. e(n) need not be set, but is used by the subprogram. ■ On exit, the contents of e are destroyed.
z	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldz, n). ■ If jobz = 'V', then if info = 0, z contains the orthonormal eigenvectors of the matrix A, with the i-th column of z holding the eigenvector associated with d(i). ■ If jobz = 'N', then z is not referenced.

Table 173–57 (Cont.) LAPACK_STEVD Procedure Parameters

Parameter	Description
ldz	The leading dimension of the array z. ldz >= 1, and if jobz = 'v',ldz >= max(1,n).
info	■ = 0 : successful exit ■ < 0 : if info = -i, the i-th argument had an illegal value ■ > 0 : if info = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero
pack	(Optional) Flags the packing of the matricies: ■ 'C': column-major (default) ■ 'R': row-major

LAPACK_SYEV Procedures

This procedure computes all eigenvalues and, optionally, eigenvectors of a real symmetric matrix A.

> **See Also:** LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms on page 173-11 for other subprograms in this group

Syntax

```
UTL_NLA.LAPACK_SYEV (
   jobz     IN      flag,
   uplo     IN      flag,
   n        IN      POSITIVEN,
   a        IN OUT  UTL_NLA_ARRAY_DBL,
   lda      IN      POSITIVEN,
   w        IN OUT  UTL_NLA_ARRAY_DBL,
   info     OUT     INTEGER,
   pack     IN      flag DEFAULT 'C');

UTL_NLA.LAPACK_SYEV (
   jobz     IN      flag,
   uplo     IN      flag,
   n        IN      POSITIVEN,
   a        IN OUT  UTL_NLA_ARRAY_FLT,
   lda      IN      POSITIVEN,
   w        IN OUT  UTL_NLA_ARRAY_FLT,
   info     OUT     INTEGER,
   pack     IN      flag DEFAULT 'C');
```

Parameters

Table 173–58 LAPACK_SYEV Procedure Parameters

Paramete	Description
jobz	■ 'N': Compute eigenvalues only. ■ 'V': Compute eigenvalues and eigenvectors.
uplo	■ 'U': Upper triangle of A is stored. ■ 'L': Upper triangle of A is stored.
n	The order of the matrix a. N >= 0.
a	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (lda, n). On entry, the symmetric matrix a: ■ If uplo = 'U', the leading n by n upper triangular part of a contains the upper triangular part of the matrix a. ■ If uplo = 'L', the leading n byn lower triangular part of a contains the lower triangular part of the matrix a. On exit: ■ If jobz = 'V', then if info = 0, a contains the orthonormal eigenvectors of the matrix a. ■ If jobz = 'N', then on exit the lower triangle (if uplo = 'L') or the upper triangle (if uplo='U') of a, including the diagonal, is destroyed.
lda	The leading dimension of the array a. lda >= max(1,n).

Table 173–58 (Cont.) LAPACK_SYEV Procedure Parameters

Paramete	Description
w	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n). If info = 0, the eigenvalues in ascending order.
info	■ = 0 : successful exit ■ < 0 : if info = -i, the i-th argument had an illegal value ■ > 0 : if info = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero
pack	(Optional) Flags the packing of the matricies: ■ 'C': column-major (default) ■ 'R': row-major

LAPACK_SYEVD Procedures

This procedure computes all eigenvalues and, optionally, eigenvectors of a real symmetric matrix A. If eigenvectors are desired, it uses a divide and conquer algorithm that makes mild assumptions about floating point arithmetic.

See Also: LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms on page 173-11 for other subprograms in this group

Syntax

```
UTL_NLA.LAPACK_SYEVD (
   jobz     IN      flag,
   uplo     IN      flag,
   n        IN      POSITIVEN,
   a        IN OUT  UTL_NLA_ARRAY_DBL,
   lda      IN      POSITIVEN,
   w        IN OUT  UTL_NLA_ARRAY_DBL,
   info     OUT     INTEGER,
   pack     IN      flag DEFAULT 'C');

UTL_NLA.LAPACK_SYEVD (
   jobz     IN      flag,
   uplo     IN      flag,
   n        IN      POSITIVEN,
   a        IN OUT  UTL_NLA_ARRAY_FLT,
   lda      IN      POSITIVEN,
   w        IN OUT  UTL_NLA_ARRAY_FLT,
   info     OUT     INTEGER,
   pack     IN      flag DEFAULT 'C');
```

Parameters

Table 173–59 LAPACK_SYEVD Procedure Parameters

Parameter	Description
jobz	■ 'N': Compute eigenvalues only. ■ 'V': Compute eigenvalues and eigenvectors.
uplo	■ 'U' : Upper triangle of A is stored. ■ 'L': Upper triangle of A is stored.
n	The order of the matrix a. N >= 0.
a	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (lda, n). On entry, the symmetric matrix a: ■ If uplo = 'U', the leading n by n upper triangular part of a contains the upper triangular part of the matrix a. ■ If uplo = 'L', the leading n byn lower triangular part of a contains the lower triangular part of the matrix a. On exit: ■ If jobz = 'V', then if info = 0, a contains the orthonormal eigenvectors of the matrix a. ■ If jobz = 'N', then on exit the lower triangle (if uplo = 'L') or the upper triangle (if uplo = 'U') of a, including the diagonal, is destroyed.

Table 173–59 (Cont.) LAPACK_SYEVD Procedure Parameters

Parameter	Description
lda	The leading dimension of the array a. lda >= max(1,n).
w	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n). If info = 0, the eigenvalues in ascending order.
info	■ = 0 : successful exit ■ < 0 : if info = -i, the i-th argument had an illegal value ■ > 0 : if info = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero
pack	(Optional) Flags the packing of the matricies: ■ 'C': column-major (default) ■ 'R': row-major

LAPACK_SYSV Procedures

This procedure computes the solution to a real system of linear equations

```
a * x = b
```

where `a` is an n by n symmetric matrix, and `x` and `b` are n by nrhs matrices.

The diagonal pivoting method is used to factor `A` as

`A = U * D * U**T`, if `UPLO = 'U'`

or

`A = L * D * L**T`, if `UPLO = 'L'`

where `U` (or `L`) is a product of permutation and unit upper (lower) triangular matrices, and `D` is symmetric and block diagonal with 1 by 1 and 2 by 2 diagonal blocks. The factored form of `A` is then used to solve the system of equations `A * X = B`.

> **See Also:** LAPACK Driver Routines (Linear Equations) Subprograms on page 173-10 for other subprograms in this group

Syntax

```
UTL_NLA.LAPACK_SYSV (
   uplo    IN      flag,
   n       IN      POSITIVEN,
   nrhs    IN      POSITIVEN,
   a       IN OUT  UTL_NLA_ARRAY_DBL,
   lda     IN      POSITIVEN,
   ipiv    IN OUT  UTL_NLA_ARRAY_INT,
   b       IN OUT  UTL_NLA_ARRAY_DBL,
   ldb     IN      POSITIVEN,
   info    OUT     INTEGER,
   pack    IN      flag DEFAULT 'C');

UTL_NLA.LAPACK_SYSV (
   uplo   IN       flag,
   n      IN       POSITIVEN,
   nrhs   IN       POSITIVEN,
   a      IN OUT   UTL_NLA_ARRAY_FLT,
   lda    IN       POSITIVEN,
   ipiv   IN OUT   UTL_NLA_ARRAY_INT,
   b      IN OUT   UTL_NLA_ARRAY_FLT,
   ldb    IN       POSITIVEN,
   info   OUT      INTEGER,
   pack   IN       flag DEFAULT 'C');
```

Parameters

Table 173–60 LAPACK_SYSV Procedure Parameters

Parameter	Description
`uplo`	▪ `uplo = 'U'`. Upper triangular of `A` is stored. ▪ `uplo = 'L'` . Lower triangular of `A` is stored.
`n`	The number of linear equations, which is the order of the matrix `a`. `N >= 0`.

Table 173–60 (Cont.) LAPACK_SYSV Procedure Parameters

Parameter	Description
nrhs	The number of right-hand sides, which is the number of columns of the matrix b. nrhs >= 0.
a	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n-1). On entry, the symmetric matrix a. If UPLO = 'U', the leading n by n upper triangular part of a contains the upper triangular part of the matrix a, and the strictly lower triangular part of a is not referenced. If uplo = 'L', the leading n by n lower triangular part of a contains the lower triangular part of the matrix a, and the strictly upper triangular part of a is not referenced. On exit, if info = 0, the block diagonal matrix d and the multipliers used to obtain the factor U or L from the factorization A = U*D*U**T or A = L*D*L**T as computed by SSYTRF.
lda	The leading dimension of the array a. lda >= max(1,n)
ipiv	INTEGER array, DIMENSION (ldb, nrhs). Details of the interchanges and the block structure of d, as determined by SSYTRF. ■ If ipiv(k) > 0, then rows and columns k and ipiv(k) were interchanged, and d(k,k) is a 1 by1 diagonal block. ■ If uplo = 'U' and ipiv(k) = ipiv(k-1) < 0, then rows and columns k-1 and -ipiv(k) were interchanged and d(k-1:k,k-1:k) is a 2 by 2 diagonal block. ■ If uplo = 'L' and ipiv(k) = ipiv(k+1) < 0, then rows and columns k+1 and -ipiv(k) were interchanged and d(k:k+1,k:k+1) is a 2 by 2 diagonal block.
b	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldb, nrhs). On entry, the n by nrhs matrix of right hand side matrix b. On exit, if info = 0 , the n by nrhs solution matrix X.
ldb	The leading dimension of the array b. ldb >= max(1,n)
info	■ = 0 : successful exit ■ < 0 : if info = -i , the i-th argument had an illegal value ■ > 0 : if info = i, d(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix d is exactly singular, so the solution could not be computed.
pack	(Optional) Flags the packing of the matrices: ■ 'C': column-major (default) ■ 'R': row-major

174

UTL_RAW

The `UTL_RAW` package provides SQL functions for manipulating `RAW` datatypes.

This chapter contains the following topics:

Using UTL_RAW

Overview

This package is necessary because normal SQL functions do not operate on RAWs, and PL/SQL does not allow overloading between a RAW and a CHAR datatype. UTL_RAW also includes subprograms that convert various COBOL number formats to, and from, RAWs.

UTL_RAW is not specific to the database environment, and it may actually be used in other environments. For this reason, the prefix UTL has been given to the package, instead of DBMS.

Operational Notes

UTL_RAW allows a RAW "record" to be composed of many elements. By using the RAW datatype, character set conversion will not be performed, keeping the RAW in its original format when being transferred through remote procedure calls.

With the RAW functions, you can manipulate binary data that was previously limited to the hextoraw and rawtohex functions.

Summary of UTL_RAW Subprograms

Table 174–1 UTL_RAW Package Subprograms

Subprogram	Description
BIT_AND Function on page 174-7	Performs bitwise logical "and" of the values in RAW r1 with RAW r2 and returns the "anded" result RAW
BIT_COMPLEMENT Function on page 174-8	Performs bitwise logical "complement" of the values in RAW r and returns the "complement'ed" result RAW
BIT_OR Function on page 174-9	Performs bitwise logical "or" of the values in RAW r1 with RAW r2 and returns the "or'd" result RAW
BIT_XOR Function on page 174-10	Performs bitwise logical "exclusive or" of the values in RAW r1 with RAW r2 and returns the "xor'd" result RAW
CAST_FROM_BINARY_DOUBLE Function on page 174-11	Returns the binary representation of a BINARY_DOUBLE (in RAW)
CAST_FROM_BINARY_FLOAT Function on page 174-12	Returns the binary representation of a BINARY_FLOAT (in RAW)
CAST_FROM_BINARY_INTEGER Function on page 174-13	Returns the binary representation of a BINARY_INTEGER (in RAW)
CAST_FROM_NUMBER Function on page 174-14	Returns the binary representation of a NUMBER (in RAW)
CAST_TO_BINARY_DOUBLE Function on page 174-15	Casts the binary representation of a RAW into a BINARY_DOUBLE
CAST_TO_BINARY_FLOAT Function on page 174-17	Casts the binary representation of a RAW into a BINARY_FLOAT
CAST_TO_BINARY_INTEGER Function on page 174-19	Casts the binary representation of a BINARY_INTEGER (in RAW) into a BINARY_INTEGER
CAST_TO_NUMBER Function on page 174-20	Casts the binary representation of a NUMBER (in RAW) into a NUMBER
CAST_TO_RAW Function on page 174-21	Converts a VARCHAR2 represented using n data bytes into a RAW with n data bytes
CAST_TO_VARCHAR2 Function on page 174-22	Converts a RAW represented using n data bytes into VARCHAR2 with n data bytes
CAST_TO_NVARCHAR2 Function on page 174-23	Converts a RAW represented using n data bytes into NVARCHAR2 with n data bytes
COMPARE Function on page 174-24	Compares RAW r1 against RAW r2
CONCAT Function on page 174-25	Concatenates up to 12 RAWs into a single RAW
CONVERT Function on page 174-26	Converts RAW r from character set from_charset to character set to_charset and returns the resulting RAW
COPIES Function on page 174-27	Returns n copies of r concatenated together
LENGTH Function on page 174-28	Returns the length in bytes of a RAW r

Table 174–1 (Cont.) UTL_RAW Package Subprograms

Subprogram	Description
OVERLAY Function on page 174-29	Overlays the specified portion of target `RAW` with overlay `RAW`, starting from byte position `pos` of target and proceeding for `len` bytes
REVERSE Function on page 174-31	Reverses a byte sequence in `RAW r` from end to end
SUBSTR Function on page 174-32	Returns `len` bytes, starting at `pos` from `RAW r`
TRANSLATE Function on page 174-34	Translates the bytes in the input `RAW r` according to the bytes in the translation `RAWs from_set` and `to_set`
TRANSLITERATE Function on page 174-36	Converts the bytes in the input `RAW r` according to the bytes in the transliteration `RAWs from_set` and `to_set`
XRANGE Function on page 174-38	Returns a `RAW` containing all valid 1-byte encodings in succession, beginning with the value `start_byte` and ending with the value `end_byte`

BIT_AND Function

This function performs bitwise logical "and" of the values in RAW r1 with RAW r2 and returns the "anded" result RAW.

Syntax

```
UTL_RAW.BIT_AND (
   r1 IN RAW,
   r2 IN RAW)
RETURN RAW;
```

Pragmas

```
pragma restrict_references(bit_and, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 174–2 BIT_AND Function Parameters

Parameter	Description
r1	RAW to "and" with r2.
r2	RAW to "and" with r1.

Return Values

Table 174–3 BIT_AND Function Return Values

Return	Description
RAW	Containing the "and" of r1 and r2.
NULL	Either r1 or r2 input parameter was NULL.

Usage Notes

If r1 and r2 differ in length, the and operation is terminated after the last byte of the shorter of the two RAWs, and the unprocessed portion of the longer RAW is appended to the partial result. The result length equals the longer of the two input RAWs.

BIT_COMPLEMENT Function

This function performs bitwise logical "complement" of the values in RAW r and returns the complement'ed result RAW. The result length equals the input RAW r length.

Syntax

```
UTL_RAW.BIT_COMPLEMENT (
   r IN RAW)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(bit_complement, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 174–4 BIT_COMPLEMENT Function Parameters

Parameter	Description
r	RAW to perform "complement" operation.

Return Values

Table 174–5 BIT_COMPLEMENT Function Return Values

Return	Description
RAW	The "complement" of r1.
NULL	If r input parameter was NULL.

BIT_OR Function

This function performs bitwise logical "or" of the values in RAW r1 with RAW r2 and returns the or'd result RAW.

Syntax

```
UTL_RAW.BIT_OR (
   r1 IN RAW,
   r2 IN RAW)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(bit_or, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 174–6 BIT_OR Function Parameters

Parameters	Description
r1	RAW to "or" with r2.
r2	RAW to "or" with r1.

Return Values

Table 174–7 BIT_OR Function Return Values

Return	Description
RAW	Containing the "or" of r1 and r2.
NULL	Either r1 or r2 input parameter was NULL.

Usage Notes

If r1 and r2 differ in length, then the "or" operation is terminated after the last byte of the shorter of the two RAWs, and the unprocessed portion of the longer RAW is appended to the partial result. The result length equals the longer of the two input RAWs.

BIT_XOR Function

This function performs bitwise logical "exclusive or" of the values in RAW r1 with RAW r2 and returns the xor'd result RAW.

Syntax

```
UTL_RAW.BIT_XOR (
   r1 IN RAW,
   r2 IN RAW)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(bit_xor, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 174–8 BIT_XOR Function Parameters

Parameter	Description
r1	RAW to "xor" with r2.
r2	RAW to "xor" with r1.

Return Values

Table 174–9 BIT_XOR Function Return Values

Return	Description
RAW	Containing the "xor" of r1 and r2.
NULL	If either r1 or r2 input parameter was NULL.

Usage Notes

If r1 and r2 differ in length, then the "xor" operation is terminated after the last byte of the shorter of the two RAWs, and the unprocessed portion of the longer RAW is appended to the partial result. The result length equals the longer of the two input RAWs.

CAST_FROM_BINARY_DOUBLE Function

This function returns the binary representation of a BINARY_DOUBLE (in RAW).

Syntax

```
UTL_RAW.CAST_FROM_BINARY_DOUBLE(
   n          IN BINARY_DOUBLE,
   endianess IN PLS_INTEGER DEFAULT 1)
RETURN RAW;
```

Pragmas

```
pragma restrict_references(cast_from_binary_double, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 174–10 CAST_FROM_BINARY_DOUBLE Function Parameters

Parameter	Description
n	The BINARY_DOUBLE value.
endianess	A PLS_INTEGER representing big-endian or little-endian architecture. The default is big-endian.

Return Values

The binary representation of the BINARY_DOUBLE value, or NULL if the input is NULL.

Usage Notes

- An 8-byte binary_double value maps to the IEEE 754 double-precision format as follows:

```
byte 0: bit 63 ~ bit 56
byte 1: bit 55 ~ bit 48
byte 2: bit 47 ~ bit 40
byte 3: bit 39 ~ bit 32
byte 4: bit 31 ~ bit 24
byte 5: bit 23 ~ bit 16
byte 6: bit 15 ~ bit  8
byte 7: bit  7 ~ bit  0
```

- The parameter endianess describes how the bytes of BINARY_DOUBLE are mapped to the bytes of RAW. In the following matrix, rb0 ~ rb7 refer to the bytes in raw and db0 ~ db7 refer to the bytes in BINARY_DOUBLE.

	rb0	rb1	rb2	rb3	rb4	rb5	rb6	rb7
big_endian	db0	db1	db2	db3	db4	db5	db6	db7
little_endian	db7	db6	db5	db4	db3	db2	db1	db0

- In case of machine-endian, the 8 bytes of the BINARY_DOUBLE argument are copied straight across into the RAW return value. The effect is the same if the user has passed big_endian on a big-endian machine, or little_endian on a little-endian machine.

CAST_FROM_BINARY_FLOAT Function

This function returns the binary representation of a BINARY_FLOAT (in RAW).

Syntax

```
UTL_RAW.CAST_FROM_BINARY_FLOAT(
   n          IN BINARY_FLOAT,
   endianess IN PLS_INTEGER DEFAULT 1)
RETURN RAW;
```

Pragmas

```
pragma restrict_references(cast_from_binary_float, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 174–11 CAST_FROM_BINARY_FLOAT Function Parameters

Parameter	Description
n	The BINARY_FLOAT value.
endianess	A PLS_INTEGER representing big-endian or little-endian architecture. The default is big-endian.

Return Values

The binary representation (RAW) of the BINARY_FLOAT value, or NULL if the input is NULL.

Usage Notes

- A 4-byte binary_float value maps to the IEEE 754 single-precision format as follows:

```
byte 0: bit 31 ~ bit 24
byte 1: bit 23 ~ bit 16
byte 2: bit 15 ~ bit  8
byte 3: bit 7 ~  bit  0
```

- The parameter endianess describes how the bytes of BINARY_FLOAT are mapped to the bytes of RAW. In the following matrix, rb0 ~ rb3 refer to the bytes in RAW and fb0 ~ fb3 refer to the bytes in BINARY_FLOAT.

	rb0	rb1	rb2	rb3
big_endian	fbo	fb1	fb2	fb3
little_endian	fb3	fb2	fb1	fb0

- In case of machine-endian, the 4 bytes of the BINARY_FLOAT argument are copied straight across into the RAW return value. The effect is the same if the user has passed big_endian on a big-endian machine, or little_endian on a little-endian machine.

CAST_FROM_BINARY_INTEGER Function

This function returns the binary representation of a BINARY_INTEGER (in RAW).

Syntax

```
UTL_RAW.CAST_FROM_BINARY_INTEGER (
   n          IN BINARY_INTEGER
   endianess  IN PLS_INTEGER DEFAULT BIG_ENDIAN)
RETURN RAW;
```

Pragmas

```
pragma restrict_references(cast_from_binary_integer, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 174–12 CAST_FROM_BINARY_INTEGER Function Parameters

Parameter	Description
n	The BINARY_INTEGER value.
endianess	A PLS_INTEGER representing big-endian or little-endian architecture. The default is big-endian.

Return Values

The binary representation of the BINARY_INTEGER value.

CAST_FROM_NUMBER Function

This function returns the binary representation of a NUMBER (in RAW).

Syntax

```
UTL_RAW.CAST_FROM_NUMBER (
   n  IN NUMBER)
 RETURN RAW;
```

Pragmas

```
pragma restrict_references(cast_from_number, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 174–13 CAST_FROM_NUMBER Function Parameters

Parameter	Description
n	The NUMBER value.

Return Values

The binary representation of the NUMBER value.

CAST_TO_BINARY_DOUBLE Function

This function casts the binary representation of a BINARY_DOUBLE (in RAW) into a BINARY_DOUBLE.

Syntax

```
UTL_RAW.CAST_TO_BINARY_DOUBLE (
   r          IN RAW
   endianess  IN PLS_INTEGER DEFAULT 1)
RETURN BINARY_DOUBLE;
```

Pragmas

```
pragma restrict_references(cast_to_binary_double, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 174–14 CAST_TO_BINARY_DOUBLE Function Parameters

Parameter	Description
r	The binary representation of a BINARY_INTEGER.
endianess	A PLS_INTEGER representing big-endian or little-endian architecture. The default is big-endian.

Return Values

The BINARY_DOUBLE value.

Usage Notes

- If the RAW argument is more than 8 bytes, only the first 8 bytes are used and the rest of the bytes are ignored. If the result is -0, +0 is returned. If the result is NaN, the value BINARY_DOUBLE_NAN is returned.
- If the RAW argument is less than 8 bytes, a VALUE_ERROR exception is raised.
- An 8-byte binary_double value maps to the IEEE 754 double-precision format as follows:

```
byte 0: bit 63 ~ bit 56
byte 1: bit 55 ~ bit 48
byte 2: bit 47 ~ bit 40
byte 3: bit 39 ~ bit 32
byte 4: bit 31 ~ bit 24
byte 5: bit 23 ~ bit 16
byte 6: bit 15 ~ bit  8
byte 7: bit  7 ~ bit  0
```

- The parameter endianess describes how the bytes of BINARY_DOUBLE are mapped to the bytes of RAW. In the following matrix, rb0 ~ rb7 refer to the bytes in raw and db0 ~ db7 refer to the bytes in BINARY_DOUBLE.

	rb0	rb1	rb2	rb3	rb4	rb5	rb6	rb7
big_endian	db0	db1	db2	db3	db4	db5	db6	db7
little_endian	db7	db6	db5	db4	db3	db2	db1	db0

- In case of machine-endian, the 8 bytes of the `RAW` argument are copied straight across into the `BINARY_DOUBLE` return value. The effect is the same if the user has passed `big_endian` on a big-endian machine, or `little_endian` on a little-endian machine.

CAST_TO_BINARY_FLOAT Function

This function casts the binary representation of a BINARY_FLOAT (in RAW) into a BINARY_FLOAT.

Syntax

```
UTL_RAW.CAST_TO_BINARY_FLOAT (
   r          IN RAW
   endianess  IN PLS_INTEGER DEFAULT 1)
RETURN BINARY_FLOAT;
```

Pragmas

```
pragma restrict_references(cast_to_binary_float, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 174–15 CAST_TO_BINARY_FLOAT Function Parameters

Parameter	Description
r	The binary representation of a BINARY_FLOAT.
endianess	A PLS_INTEGER representing big-endian or little-endian architecture. The default is big-endian.

Return Values

The BINARY_FLOAT value.

Usage Notes

- If the RAW argument is more than 4 bytes, only the first 4 bytes are used and the rest of the bytes are ignored. If the result is -0, +0 is returned. If the result is NaN, the value BINARY_FLOAT_NAN is returned.
- If the RAW argument is less than 4 bytes, a VALUE_ERROR exception is raised.
- A 4-byte binary_float value maps to the IEEE 754 single-precision format as follows:

```
byte 0: bit 31 ~ bit 24
byte 1: bit 23 ~ bit 16
byte 2: bit 15 ~ bit  8
byte 3: bit 7 ~  bit  0
```

- The parameter endianess describes how the bytes of BINARY_FLOAT are mapped to the bytes of RAW. In the following matrix, rb0 ~ rb3 refer to the bytes in RAW and fb0 ~ fb3 refer to the bytes in BINARY_FLOAT.

	rb0	rb1	rb2	rb3
big_endian	fbo	fb1	fb2	fb3
little_endian	fb3	fb2	fb1	fb0

- In case of machine-endian, the 4 bytes of the RAW argument are copied straight across into the BINARY_FLOAT return value. The effect is the same if the user has

passed `big_endian` on a big-endian machine, or `little_endian` on a little-endian machine.

CAST_TO_BINARY_INTEGER Function

This function casts the binary representation of a BINARY_INTEGER (in RAW) into a BINARY_INTEGER.

Syntax

```
UTL_RAW.CAST_TO_BINARY_INTEGER (
   r          IN RAW
   endianess  IN PLS_INTEGER DEFAULT BIG_ENDIAN)
RETURN BINARY_INTEGER;
```

Pragmas

```
pragma restrict_references(cast_to_binary_integer, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 174–16 CAST_TO_BINARY_INTEGER Function Parameters

Parameter	Description
r	The binary representation of a BINARY_INTEGER.
endianess	A PLS_INTEGER representing big-endian or little-endian architecture. The default is big-endian.

Return Values

The BINARY_INTEGER value

CAST_TO_NUMBER Function

This function casts the binary representation of a NUMBER (in RAW) into a NUMBER.

Syntax

```
UTL_RAW.CAST_TO_NUMBER (
   r  IN RAW)
 RETURN NUMBER;
```

Pragmas

```
pragma restrict_references(cast_to_number, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 174–17 CAST_TO_NUMBER function Parameters

Parameter	Description
r	The binary representation of a NUMBER

Return Values

The NUMBER value.

CAST_TO_RAW Function

This function converts a VARCHAR2 represented using n data bytes into a RAW with n data bytes. The data is not modified in any way; only its datatype is recast to a RAW datatype.

Syntax

```
UTL_RAW.CAST_TO_RAW (
   c  IN VARCHAR2)
RETURN RAW;
```

Pragmas

```
pragma restrict_references(cast_to_raw, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 174–18 CAST_TO_RAW Function Parameters

Parameter	Description
c	VARCHAR2 to be changed to a RAW.

Return Values

Table 174–19 CAST_TO_RAW Function Return Values

Return	Description
RAW	Containing the same data as the input VARCHAR2 and equal byte length as the input VARCHAR2 and without a leading length field.
NULL	If c input parameter was NULL.

CAST_TO_VARCHAR2 Function

This function converts a RAW represented using n data bytes into VARCHAR2 with n data bytes.

> **Note:** When casting to a VARCHAR2, the current Globalization Support character set is used for the characters within that VARCHAR2.

Syntax

```
UTL_RAW.CAST_TO_VARCHAR2 (
   r IN RAW)
RETURN VARCHAR2;
```

Pragmas

```
pragma restrict_references(cast_to_VARCHAR2, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 174–20 CAST_TO_VARCHAR2 Function Parameters

Parameter	Description
r	RAW (without leading length field) to be changed to a VARCHAR2.

Return Values

Table 174–21 CAST_TO_VARCHAR2 Function Return Values

Return	Description
VARCHAR2	Containing having the same data as the input RAW.
NULL	If r input parameter was NULL.

CAST_TO_NVARCHAR2 Function

This function converts a RAW represented using n data bytes into NVARCHAR2 with n data bytes.

> **Note:** When casting to a NVARCHAR2, the current Globalization Support character set is used for the characters within that NVARCHAR2.

Syntax

```
UTL_RAW.CAST_TO_NVARCHAR2 (
   r IN RAW)
RETURN NVARCHAR2;
```

Pragmas

```
pragma restrict_references(cast_to_NVARCHAR2, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 174–22 CAST_TO_NVARCHAR2 Function Parameters

Parameter	Description
r	RAW (without leading length field) to be changed to a NVARCHAR2).

Return Values

Table 174–23 CAST_TO_NVARCHAR2 Function Return Values

Return	Description
NVARCHAR2	Containing having the same data as the input RAW.
NULL	If r input parameter was NULL.

COMPARE Function

This function compares RAW r1 against RAW r2. If r1 and r2 differ in length, then the shorter RAW is extended on the right with pad if necessary.

Syntax

```
UTL_RAW.COMPARE (
   r1  IN RAW,
   r2  IN RAW,
   pad IN RAW DEFAULT NULL)
  RETURN NUMBER;
```

Pragmas

```
pragma restrict_references(compare, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 174–24 COMPARE Function Parameters

Parameter	Description
r1	1st RAW to be compared, may be NULL or 0 length.
r2	2nd RAW to be compared, may be NULL or 0 length.
pad	This is an optional parameter. Byte to extend whichever of r1 or r2 is shorter. The default: x'00'

Return Values

Table 174–25 COMPARE Function Return Values

Return	Description
NUMBER	Equals 0 if RAW byte strings are both NULL or identical; or, Equals position (numbered from 1) of the first mismatched byte.

CONCAT Function

This function concatenates up to 12 RAWs into a single RAW. If the concatenated size exceeds 32K, then an error is returned

Syntax

```
UTL_RAW.CONCAT (
   r1  IN RAW DEFAULT NULL,
   r2  IN RAW DEFAULT NULL,
   r3  IN RAW DEFAULT NULL,
   r4  IN RAW DEFAULT NULL,
   r5  IN RAW DEFAULT NULL,
   r6  IN RAW DEFAULT NULL,
   r7  IN RAW DEFAULT NULL,
   r8  IN RAW DEFAULT NULL,
   r9  IN RAW DEFAULT NULL,
   r10 IN RAW DEFAULT NULL,
   r11 IN RAW DEFAULT NULL,
   r12 IN RAW DEFAULT NULL)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(concat, WNDS, RNDS, WNPS, RNPS);
```

Parameters

r1....r12 are the RAW items to concatenate.

Return Values

Table 174–26 CONCAT Function Return Values

Return	Description
RAW	Containing the items concatenated.

Exceptions

There is an error if the sum of the lengths of the inputs exceeds the maximum allowable length for a RAW, which is 32767 bytes.

CONVERT Function

This function converts RAW r from character set from_charset to character set to_charset and returns the resulting RAW.

Both from_charset and to_charset must be supported character sets defined to the Oracle server.

Syntax

```
UTL_RAW.CONVERT (
   r            IN RAW,
   to_charset   IN VARCHAR2,
   from_charset IN VARCHAR2)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(convert, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 174–27 CONVERT Function Parameters

Parameter	Description
r	RAW byte-string to be converted.
to_charset	Name of Globalization Support character set to which r is converted.
from_charset	Name of Globalization Support character set in which r is supplied.

Return Values

Table 174–28 CONVERT Function Return Values

Return	Description
RAW	Byte string r converted according to the specified character sets.

Exceptions

Table 174–29 CONVERT Function Exceptions

Error	Description
VALUE_ERROR	Either: - r missing, NULL, or 0 length - from_charset or to_charset missing, NULL, or 0 length - from_charset or to_charset names invalid or unsupported

COPIES Function

This function returns n copies of r concatenated together.

Syntax

```
UTL_RAW.COPIES (
   r IN RAW,
   n IN NUMBER)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(copies, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 174–30 COPIES Function Parameters

Parameters	Description
r	RAW to be copied
n	Number of times to copy the RAW (must be positive).

Return Values

This returns the RAW copied n times.

Exceptions

Table 174–31 COPIES Function Exceptions

Error	Description
VALUE_ERROR	Either: - r is missing, NULL or 0 length - n < 1 - Length of result exceeds maximum length of a RAW

LENGTH Function

This function returns the length in bytes of a `RAW` `r`.

Syntax

```
UTL_RAW.LENGTH (
   r  IN RAW)
RETURN NUMBER;
```

Pragmas

```
pragma restrict_references(length, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 174–32 LENGTH Function Parameters

Parameter	Description
`r`	The `RAW` byte stream to be measured.

Return Values

Table 174–33 LENGTH Function Return Values

Return	Description
`NUMBER`	The current length of the `RAW`.

OVERLAY Function

This function overlays the specified portion of target RAW with overlay_str RAW, starting from byte position pos of target and proceeding for len bytes.

Syntax

```
UTL_RAW.OVERLAY (
   overlay_str IN RAW,
   target      IN RAW,
   pos         IN BINARY_INTEGER DEFAULT 1,
   len         IN BINARY_INTEGER DEFAULT NULL,
   pad         IN RAW            DEFAULT NULL)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(overlay, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 174–34 OVERLAY Function Parameters

Parameters	Description
overlay_str	Byte-string used to overlay target.
target	Byte-string which is to be overlaid.
pos	Position in target (numbered from 1) to start overlay.
len	The number of target bytes to overlay.
pad	Pad byte used when overlay len exceeds overlay_str length or pos exceeds target length.

Defaults and Optional Parameters

Table 174–35 OVERLAY Function Optional Parameters

Optional Parameter	Description
pos	1
len	To the length of overlay_str
pad	x'00'

Return Values

Table 174–36 OVERLAY Function Return Values

Return	Description
RAW	The target byte_string overlaid as specified.

Usage Notes

If overlay_str has less than len bytes, then it is extended to len bytes using the pad byte. If overlay_str exceeds len bytes, then the extra bytes in overlay_str are ignored. If len bytes beginning at position pos of target exceeds the length of target, then target is extended to contain the entire length of overlay_str.

If len is specified, it must be greater than or equal to 0. If pos is specified, it must be greater than or equal to 1. If pos exceeds the length of target, then target is padded with pad bytes to position pos, and target is further extended with overlay_str bytes.

Exceptions

Table 174–37 OVERLAY Function Exceptions

Error	Description
VALUE_ERROR	Either: - Overlay_str is NULL or has 0 length - Target is missing or undefined - Length of target exceeds maximum length of a RAW - len < 0 - pos < 1

REVERSE Function

This function reverses a byte sequence in RAW r from end to end. For example, x'0102F3' would be reversed to x'F30201', and 'xyz' would be reversed to 'zyx'.The result length is the same as the input RAW length.

Syntax

```
UTL_RAW.REVERSE (
   r IN RAW)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(reverse, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 174–38 REVERSE Function Parameters

Parameter	Description
r	RAW to reverse.

Return Values

Table 174–39 REVERSE Function Return Values

Return	Description
RAW	Containing the "reverse" of r.

Exceptions

Table 174–40 REVERSE Function Exceptions

Error	Description
VALUE_ERROR	R is NULL or has 0 length.

SUBSTR Function

This function returns `len` bytes, starting at `pos` from `RAW r`.

Syntax

```
UTL_RAW.SUBSTR (
   r   IN RAW,
   pos IN BINARY_INTEGER,
   len IN BINARY_INTEGER DEFAULT NULL)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(substr, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 174–41 SUBSTR Function Parameters

Parameter	Description
`r`	The `RAW` byte-string from which a portion is extracted.
`pos`	The byte position in `r` at which to begin extraction.
`len`	The number of bytes from `pos` to extract from `r` (optional).

Defaults and Optional Parameters

Table 174–42 SUBSTR Function Exceptions

Optional Parameter	Description
`len`	Position `pos` through to the end of `r`.

Return Values

Table 174–43 SUBSTR Function Return Values

Return	Description
`portion of r`	Beginning at `pos` for `len` bytes long.
`NULL`	`r` input parameter was `NULL`.

Usage Notes

If `pos` is positive, then `SUBSTR` counts from the beginning of `r` to find the first byte. If `pos` is negative, then `SUBSTR` counts backward from the end of the `r`. The value `pos` cannot be 0.

If `len` is omitted, then `SUBSTR` returns all bytes to the end of `r`. The value `len` cannot be less than 1.

Exceptions

Table 174–44 SUBSTR Function Exceptions

Error	Description
VALUE_ERROR	VALUE_ERROR is returned if: ■ pos = 0 or > length of r ■ len < 1 or > length of r - (pos-1)

TRANSLATE Function

This function translates the bytes in the input RAW r according to the bytes in the translation RAWs from_set and to_set. If a byte in r has a matching byte in from_set, then it is replaced by the byte in the corresponding position in to_set, or deleted.

Bytes in r, but undefined in from_set, are copied to the result. Only the first (leftmost) occurrence of a byte in from_set is used. Subsequent duplicates are not scanned and are ignored.

Syntax

```
UTL_RAW.TRANSLATE (
   r        IN RAW,
   from_set IN RAW,
   to_set   IN RAW)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(translate, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 174–45 TRANSLATE Function Parameters

Parameter	Description
r	RAW source byte-string to be translated.
from_set	RAW byte-codes to be translated, if present in r.
to_set	RAW byte-codes to which corresponding from_str bytes are translated.

Return Values

Table 174–46 TRANSLATE Function Return Values

Return	Description
RAW	Translated byte-string.

Usage Notes

If to_set is shorter than from_set, then the extra from_set bytes have no translation correspondence and any bytes in r matching.

Note: Difference from TRANSLITERATE:

- Translation RAWs have no defaults.

- r bytes undefined in the to_set translation RAW are deleted.

- Result RAW may be shorter than input RAW r.

Exceptions

Table 174–47 TRANSLATE Function Exceptions

Error	Description
VALUE_ERROR	Either: - r is NULL or has 0 length - from_set is NULL or has 0 length - to_set is NULL or has 0 length

TRANSLITERATE Function

This function converts the bytes in the input RAW r according to the bytes in the transliteration RAWs from_set and to_set. Successive bytes in r are looked up in the from_set, and, if not found, copied unaltered to the result RAW. If found, then they are replaced in the result RAW by either corresponding bytes in the to_set, or the pad byte when no correspondence exists.

Bytes in r, but undefined in from_set, are copied to the result. Only the first (leftmost) occurrence of a byte in from_set is used. Subsequent duplicates are not scanned and are ignored. The result RAW is always the same length as r.

Syntax

```
UTL_RAW.TRANSLITERATE (
   r        IN RAW,
   to_set   IN RAW DEFAULT NULL,
   from_set IN RAW DEFAULT NULL,
   pad      IN RAW DEFAULT NULL)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(transliterate, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 174–48 TRANSLITERATE Function Parameters

Parameter	Description
r	RAW input byte-string to be converted.
from_set	RAW byte-codes to be converted, if present in r (any length).
to_set	RAW byte-codes to which corresponding from_set bytes are converted (any length).
pad	1 byte used when to-set is shorter than the from_set.

Defaults and Optional Parameters

Table 174–49 TRANSLITERATE Function Optional Parameters

Optional Parameter	Description
from_set	x f00' through x fff'
to_set	To the NULL string and effectively extended with pad to the length of from_set as necessary.
pad	x'00'.

Return Values

Table 174–50 TRANSLITERATE Function Return Values

Return	Description
RAW	Converted byte-string.

Usage Notes

If the to_set is shorter than the from_set, then the pad byte is placed in the result RAW when a selected from_set byte has no corresponding to_set byte (as if the to_set were extended to the same length as the from_set with pad bytes).

Note: Difference from TRANSLATE:

- r bytes undefined in to_set are padded.

- Result RAW is always same length as input RAW r.

Exceptions

Table 174–51 TRANSLITERATE Function Exceptions

Error	Description
VALUE_ERROR	R is NULL or has 0 length.

XRANGE Function

This function returns a RAW containing all valid 1-byte encodings in succession, beginning with the value start_byte and ending with the value end_byte. If start_byte is greater than end_byte, then the succession of resulting bytes begins with start_byte, wraps through x fFF f to x f00 f, and ends at end_byte. If specified, start_byte and end_byte must be single-byte RAWs.

Syntax

```
UTL_RAW.XRANGE (
   start_byte IN RAW DEFAULT NULL,
   end_byte   IN RAW DEFAULT NULL)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(xrange, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 174–52 XRANGE Function Parameters

Parameters	Description
start_byte	Beginning byte-code value of resulting sequence.
end_byte	Ending byte-code value of resulting sequence.

Defaults and Optional Parameters

```
start_byte - x f00 f
end_byte   - x fFF f
```

Return Values

Table 174–53 XRANGE Function Return Values

Return	Description
RAW	Containing succession of 1-byte hexadecimal encodings.

175

UTL_RECOMP

The UTL_RECOMP package recompiles invalid PL/SQL modules, Java classes, indextypes and operators in a database, either sequentially or in parallel.

This chapter contains the following topics:

Using UTL_RECOMP

Overview

This script is particularly useful after a major-version upgrade that typically invalidates all PL/SQL and Java objects. Although invalid objects are recompiled automatically on use, it is useful to run this script prior to operation because this will either eliminate or minimize subsequent latencies due to on-demand automatic recompilation at runtime.

Parallel recompilation can exploit multiple CPUs to reduce the time taken to recompile invalid objects. The degree of parallelism is specified by the first argument to RECOMP_PARALLEL Procedure.

In general, a parallelism setting of one thread for each available CPU provides a good initial setting. However, please note that the process of recompiling an invalid object writes a significant amount of data to system tables and is fairly I/O intensive. A slow disk system may be a significant bottleneck and limit speedups available from a higher degree of parallelism.

Operational Notes

- This package uses the job queue for parallel recompilation.
- This package must be run using `SQL*PLUS`.
- You must be connected `AS SYSDBA` to run this script.
- This package expects the following packages to have been created with VALID status:
  - `STANDARD` (`standard.sql`)
  - `DBMS_STANDARD` (`dbmsstdx.sql`)
  - `DBMS_JOB` (`dbmsjob.sql`)
  - `DBMS_RANDOM (dbmsrand.sql)`
- There should be no other DDL on the database while running entries in this package. Not following this recommendation may lead to deadlocks.

Examples

Recompile all objects sequentially:

```
EXECUTE UTL_RECOMP.RECOMP_SERIAL();
```

Recompile objects in schema SCOTT sequentially:

```
EXECUTE UTL_RECOMP.RECOMP_SERIAL('SCOTT');
```

Recompile all objects using 4 parallel threads:

```
EXECUTE UTL_RECOMP.RECOMP_PARALLEL(4);
```

Recompile objects in schema JOE using the number of threads specified in the parameter JOB_QUEUE_PROCESSES:

```
EXECUTE UTL_RECOMP.RECOMP_PARALLEL(NULL, 'JOE');
```

Summary of UTL_RECOMP Subprograms

Table 175–1 UTL_RECOMP Package Subprograms

Subprogram	Description
RECOMP_PARALLEL Procedure on page 175-7	Recompiles invalid objects in the database, or in a given schema, in parallel in dependency order
RECOMP_SERIAL Procedure on page 175-8	Recompiles invalid objects in a given schema or all invalid objects in the database

RECOMP_PARALLEL Procedure

This procedure is the main driver that recompiles invalid objects in the database, or in a given schema, in parallel in dependency order. It uses information in `dependency$` to order recompilation of dependents after parents.

Syntax

```
UTL_RECOMP.RECOMP_PARALLEL(
   threads  IN   PLS_INTEGER DEFAULT NULL,
   schema   IN   VARCHAR2    DEFAULT NULL,
   flags    IN   PLS_INTEGER DEFAULT 0);
```

Parameters

Table 175–2 RECOMP_PARALLEL Procedure Parameters

Parameter	Description
`threads`	The number of recompile threads to run in parallel. If `NULL`, use the value of '`job_queue_processes`'.
`schema`	The schema in which to recompile invalid objects. If `NULL`, all invalid objects in the database are recompiled.
`flags`	Flag values are intended for internal testing and diagnosability only.

Usage Notes

The parallel recompile exploits multiple CPUs to reduce the time taken to recompile invalid objects. However, please note that recompilation writes significant amounts of data to system tables, so the disk system may be a bottleneck and prevent significant speedups.

RECOMP_SERIAL Procedure

This procedure recompiles invalid objects in a given schema or all invalid objects in the database.

Syntax

```
UTL_RECOMP.RECOMP_SERIAL(
   schema    IN   VARCHAR2    DEFAULT NULL,
   flags     IN   PLS_INTEGER DEFAULT 0);
```

Parameters

Table 175–3 RECOMP_SERIAL Procedure Parameters

Parameter	Description
`schema`	The schema in which to recompile invalid objects. If `NULL`, all invalid objects in the database are recompiled.
`flags`	Flag values are intended for internal testing and diagnosability only.

176

UTL_REF

The UTL_REF package provides PL/SQL procedures to support reference-based operations. Unlike SQL, UTL_REF procedures enable you to write generic type methods without knowing the object table name.

This chapter contains the following topics:

Using UTL_REF

Overview

Oracle supports user-defined composite type or object type. Any instance of an object type is called an object. An object type can be used as the type of a column or as the type of a table.

In an object table, each row of the table stores an object. You can uniquely identify an object in an object table with an object identifier.

A reference is a persistent pointer to an object, and each reference can contain an object identifier. The reference can be an attribute of an object type, or it can be stored in a column of a table. Given a reference, an object can be retrieved.

Security Model

The procedural option is needed to use this package. This package must be created under SYS (connect/as sysdba). Operations provided by this package are performed under the current calling user, not under the package owner SYS.

You can use the UTL_REF package from stored PL/SQL procedures/packages on the server, as well as from client/side PL/SQL code.

When invoked from PL/SQL procedures/packages on the server, UTL_REF verifies that the invoker has the appropriate privileges to access the object pointed to by the REF.

> **Note:** This is in contrast to PL/SQL packages/procedures on the server which operate with definer's privileges, where the package owner must have the appropriate privileges to perform the desired operations.

Thus, if UTL_REF is defined under user SYS, and user A invokes UTL_REF.SELECT to select an object from a reference, then user A (the invoker) requires the privileges to check.

When invoked from client-side PL/SQL code, UTL_REF operates with the privileges of the client session under which the PL/SQL execution is being done.

Types

An object type is a composite datatype defined by the user or supplied as a library type. You can create the object type `employee_type` using the following syntax:

```
CREATE TYPE employee_type AS OBJECT (
   name    VARCHAR2(20),
   id      NUMBER,

member function GET_ID
    (name VARCHAR2)
   RETURN MEMBER);
```

The object type `employee_type` is a user-defined type that contains two attributes, `name` and `id`, and a member function, `GET_ID`().

You can create an object table using the following SQL syntax:

```
CREATE TABLE employee_table OF employee_type;
```

Exceptions

Exceptions can be returned during execution of UTL_REF functions for various reasons. For example, the following scenarios would result in exceptions:

- The object selected does not exist. This could be because either:
  1. The object has been deleted, or the given reference is dangling (invalid).
  2. The object table was dropped or does not exist.
- The object cannot be modified or locked in a serializable transaction. The object was modified by another transaction after the serializable transaction started.
- You do not have the privilege to select or modify the object. The caller of the UTL_REF subprogram must have the proper privilege on the object that is being selected or modified.

Table 176–1 UTL_REF Exceptions

Exceptions	Description
errnum == 942	Insufficient privileges.
errnum == 1031	Insufficient privileges.
errnum == 8177	Unable to serialize, if in a serializable transaction.
errnum == 60	Deadlock detected.
errnum == 1403	No data found (if the REF is NULL, and so on.).

The UTL_REF package does not define any named exceptions. You may define exception handling blocks to catch specific exceptions and to handle them appropriately.

Summary of UTL_REF Subprograms

Table 176–2 UTL_REF Package Subprograms

Subprogram	Description
DELETE_OBJECT Procedure on page 176-8	Deletes an object given a reference
LOCK_OBJECT Procedure on page 176-10	Locks an object given a reference
SELECT_OBJECT Procedure on page 176-11	Selects an object given a reference
UPDATE_OBJECT Procedure on page 176-12	Updates an object given a reference

DELETE_OBJECT Procedure

This procedure deletes an object given a reference. The semantic of this subprogram is similar to the following SQL statement:

```
DELETE FROM object_table
WHERE REF(t) = reference;
```

Unlike the preceding SQL statement, this subprogram does not require you to specify the object table name where the object resides.

Syntax

```
UTL_REF.DELETE_OBJECT (
   reference IN REF "<typename>");
```

Parameters

Table 176–3 DELETE_OBJECT Procedure Parameters

Parameter	Description
reference	Reference of the object to delete.

Exceptions

May be raised.

Examples

The following example illustrates usage of the UTL_REF package to implement this scenario: if an employee of a company changes their address, their manager should be notified.

... declarations of Address_t and others...

```
CREATE OR REPLACE TYPE Person_t (
   name    VARCHAR2(64),
   gender  CHAR(1),
   address Address_t,
   MEMBER PROCEDURE setAddress(addr IN Address_t)
);

CREATE OR REPLACE TYPE BODY Person_t (
   MEMBER PROCEDURE setAddress(addr IN Address_t) IS
   BEGIN
      address := addr;
   END;
);

CREATE OR REPLACE TYPE Employee_t (
```

Under Person_t: Simulate implementation of inheritance using a REF to Person_t and delegation of setAddress to it.

```
   thePerson  REF Person_t,
   empno      NUMBER(5),
   deptREF    Department_t,
   mgrREF     Employee_t,
   reminders  StringArray_t,
```

```
   MEMBER PROCEDURE setAddress(addr IN Address_t),
   MEMBER procedure addReminder(reminder VARCHAR2);
);

CREATE TYPE BODY Employee_t (
   MEMBER PROCEDURE setAddress(addr IN Address_t) IS
      myMgr Employee_t;
      meAsPerson Person_t;
   BEGIN
```

Update the address by delegating the responsibility to `thePerson`. Lock the Person object from the reference, and also select it:

```
      UTL_REF.LOCK_OBJECT(thePerson, meAsPerson);
      meAsPerson.setAddress(addr);
```

Delegate to `thePerson`:

```
        UTL_REF.UPDATE_OBJECT(thePerson, meAsPerson);
        if mgr is NOT NULL THEN
```

Give the manager a reminder:

```
         UTL_REF.LOCK_OBJECT(mgr);
         UTL_REF.SELECT_OBJECT(mgr, myMgr);
         myMgr.addReminder
         ('Update address in the employee directory for' ||
         thePerson.name || ', new address: ' || addr.asString);
         UTL_REF.UPDATE_OBJECT(mgr, myMgr);
      END IF;
   EXCEPTION
      WHEN OTHERS THEN
      errnum := SQLCODE;
      errmsg := SUBSTR(SQLERRM, 1, 200);
```

LOCK_OBJECT Procedure

This procedure locks an object given a reference. In addition, this procedure lets the program select the locked object. The semantic of this subprogram is similar to the following SQL statement:

```
SELECT VALUE(t)
  INTO object
  FROM object_table t
  WHERE REF(t) = reference
  FOR UPDATE;
```

Unlike the preceding SQL statement, this subprogram does not require you to specify the object table name where the object resides. It is not necessary to lock an object before updating/deleting it.

Syntax

```
UTL_REF.LOCK_OBJECT (
   reference IN REF "<typename>");

UTL_REF.LOCK_OBJECT (
   reference IN REF "<typename>",
   object    IN OUT "<typename>");
```

Parameters

Table 176–4 LOCK_OBJECT Procedure Parameters

Parameter	Description
`reference`	Reference of the object to lock.
`object`	The PL/SQL variable that stores the locked object. This variable should be of the same object type as the locked object.

Exceptions

May be raised.

SELECT_OBJECT Procedure

This procedure selects an object given its reference. The selected object is retrieved from the database and its value is put into the PL/SQL variable 'object'. The semantic of this subprogram is similar to the following SQL statement:

```
SELECT VALUE(t)
INTO object
FROM object_table t
WHERE REF(t) = reference;
```

Unlike the preceding SQL statement, this subprogram does not require you to specify the object table name where the object resides.

Syntax

```
UTL_REF.SELECT_OBJECT (
   reference IN REF "<typename>",
   object    IN OUT "<typename>");
```

Parameters

Table 176–5 SELECT_OBJECT Procedure Parameters

Parameter	Description
`reference`	Reference to the object to select or retrieve.
`object`	The PL/SQL variable that stores the selected object; this variable should be of the same object type as the referenced object.

Exceptions

May be raised.

UPDATE_OBJECT Procedure

This procedure updates an object given a reference. The referenced object is updated with the value contained in the PL/SQL variable 'object'. The semantic of this subprogram is similar to the following SQL statement:

```
UPDATE object_table t
SET VALUE(t) = object
WHERE REF(t) = reference;
```

Unlike the preceding SQL statement, this subprogram does not require you to specify the object table name where the object resides.

Syntax

```
UTL_REF.UPDATE_OBJECT (
   reference IN REF "<typename>",
   object    IN     "<typename>");
```

Parameters

Table 176–6 UPDATE_OBJECT Procedure Parameters

Parameter	Description
`reference`	Reference of the object to update.
`object`	The PL/SQL variable that contains the new value of the object. This variable should be of the same object type as the object to update.

Exceptions

May be raised.

177

UTL_SMTP

The `UTL_SMTP` package is designed for sending electronic mails (emails) over Simple Mail Transfer Protocol (SMTP) as specified by RFC821.

> **See Also:** How to use the SMTP package to send email in *Oracle Database Application Developer's Guide - Fundamentals*

This chapter contains the following topics:

Using UTL_SMTP

Overview

The protocol consists of a set of commands for an email client to dispatch emails to a SMTP server. The UTL_SMTP package provides interfaces to the SMTP commands. For many of the commands, the package provides both a procedural and a functional interface. The functional form returns the reply from the server for processing by the client. The procedural form checks the reply and will raise an exception if the reply indicates a transient (400-range reply code) or permanent error (500-range reply code). Otherwise, it discards the reply.

Note that the original SMTP protocol communicates using 7-bit ASCII. Using UTL_SMTP, all text data (in other words, those in VARCHAR2) will be converted to US7ASCII before it is sent over the wire to the server. Some implementations of SMTP servers that support SMTP extension 8BITMIME [RFC1652] support full 8-bit communication between client and server. The body of the DATA command may be transferred in full 8 bits, but the rest of the SMTP command and response should be in 7 bits. When the target SMTP server supports 8BITMIME extension, users of multibyte databases may convert their non-US7ASCII, multibyte VARCHAR2 data to RAW and use the WRITE_RAW_DATA subprogram to send multibyte data using 8-bit MIME encoding.

UTL_SMTP provides for SMTP communication as specified in RFC821, but does not provide an API to format the content of the message according to RFC 822 (for example, setting the subject of an electronic mail).You must format the message appropriately. In addition, UTL_SMTP does not have the functionality to implement an SMTP server for an email clients to send emails using SMTP.

Note : RFC documents are "Request for Comments" documents that describe proposed standards for public review on the Internet. For the actual RFC documents, please refer to:

`http://www.ietf.org/rfc/`

Types

CONNECTION Record Type

This is a PL/SQL record type used to represent an SMTP connection.

Syntax

```
TYPE connection IS RECORD (
    host              VARCHAR2(255),      -- remote host name
    port              PLS_INTEGER,        -- remote port number
    tx_timeout        PLS_INTEGER,        -- Transfer time out (in seconds)
    private_tcp_con   utl_tcp.connection, -- private, for implementation use
    private_state     PLS_INTEGER         -- private, for implementation use
);
```

Fields

Table 177–1 CONNECTION Record Type Fields

Field	Description
`host`	The name of the remote host when connection is established. `NULL` when no connection is established.
`port`	The port number of the remote SMTP server connected. `NULL` when no connection is established.
`tx_timeout`	The time in seconds that the `UTL_SMTP` package waits before giving up in a read or write operation in this connection. In read operations, this package gives up if no data is available for reading immediately. In write operations, this package gives up if the output buffer is full and no data is to be sent into the network without being blocked. `0` indicates not to wait at all. `NULL` indicates to wait forever.
`private_tcp_con`	Private, for implementation use only. You should not modify this field.
`private_state`	Private, for implementation use only. You should not modify this field.

Usage Notes

The read-only fields in a connection record are used to return information about the SMTP connection after the connection is successfully made with `open_connection()`. Changing the values of these fields has no effect on the connection. The fields `private_xxx` are for implementation use only. You should not modify these fields.

REPLY, REPLIES Record Types

These are PL/SQL record types used to represent an SMTP reply line. Each SMTP reply line consists of a reply code followed by a text message. While a single reply line is expected for most SMTP commands, some SMTP commands expect multiple reply lines. For those situations, a PL/SQL table of reply records is used to represent multiple reply lines.

Syntax

```
TYPE reply IS RECORD (
  code    PLS_INTEGER,       -- 3-digit reply code
  text    VARCHAR2(508)      -- text message
);
TYPE replies IS TABLE OF reply INDEX BY BINARY_INTEGER;   -- multiple reply lines
```

Fields

Table 177–2 REPLY, REPLIES Record Type Fields

Field	Description
`code`	The 3-digit reply code.
`text`	The text message of the reply.

Reply Codes

The following is a list of the SMTP reply codes.

Table 177–3 SMTP Reply Codes

Reply Code	Meaning
211	System status, or system help reply
214	Help message [Information on how to use the receiver or the meaning of a particular non-standard command; this reply is useful only to the human user]
220	<domain> Service ready
221	<domain> Service closing transmission channel
250	Requested mail action okay, completed
251	User not local; will forward to <forward-path>
252	OK, pending messages for node <node> started. Cannot VRFY user (for example, info is not local), but will take message for this user and attempt delivery.
253	OK, <messages> pending messages for node <node> started
354	Start mail input; end with <CRLF>.<CRLF>
355	Octet-offset is the transaction offset
421	<domain> Service not available, closing transmission channel (This may be a reply to any command if the service knows it must shut down.)
450	Requested mail action not taken: mailbox unavailable [for example, mailbox busy]
451	Requested action terminated: local error in processing
452	Requested action not taken: insufficient system storage
453	You have no mail.
454	TLS not available due to temporary reason. Encryption required for requested authentication mechanism.
458	Unable to queue messages for node <node>
459	Node <node> not allowed: reason
500	Syntax error, command unrecognized (This may include errors such as command line too long.)
501	Syntax error in parameters or arguments
502	Command not implemented
503	Bad sequence of commands
504	Command parameter not implemented
521	<Machine> does not accept mail.
530	Must issue a STARTTLS command first. Encryption required for requested authentication mechanism.
534	Authentication mechanism is too weak.
538	Encryption required for requested authentication mechanism.

Table 177–3 (Cont.) SMTP Reply Codes

Reply Code	Meaning
550	Requested action not taken: mailbox unavailable [for , mailbox not found, no access]
551	User not local; please try <forward-path>
552	Requested mail action terminated: exceeded storage allocation
553	Requested action not taken: mailbox name not allowed [for example, mailbox syntax incorrect]
554	Transaction failed

Exceptions

The table lists the exceptions that can be raised by the interface of the `UTL_SMTP` package. The network error is transferred to a reply code of 421- service not available.

Table 177–4 UTL_SMTP Exceptions

`INVALID_OPERATION`	Raised when an invalid operation is made. In other words, calling API other than `write_data(), write_raw_data()` or `close_data()` after open_data() is called, or calling `write_data(), write_raw_data()` or `close_data()` without first calling `open_data()`.
`TRANSIENT_ERROR`	Raised when receiving a reply code in 400 range.
`PERMANENT_ERROR`	Raised when receiving a reply code in 500 range.

Rules and Limits

No limitation or range-checking is imposed by the API. However, you should be aware of the following size limitations on various elements of SMTP. Sending data that exceed these limits may result in errors returned by the server.

Table 177–5 SMTP Size Limitation

Element	Size Limitation
`user`	The maximum total length of a user name is 64 characters.
`domain`	The maximum total length of a domain name or number is 64 characters.
`path`	The maximum total length of a reverse-path or forward-path is 256 characters (including the punctuation and element separators).
`command line`	The maximum total length of a command line including the command word and the `<CRLF>` is 512 characters.
`reply line`	The maximum total length of a reply line including the reply code and the `<CRLF>` is 512 characters.
`text line`	The maximum total length of a text line including the `<CRLF>` is 1000 characters (but not counting the leading dot duplicated for transparency).
`recipients buffer`	The maximum total number of recipients that must be buffered is 100 recipients.

Examples

The following example illustrates how UTL_SMTP is used by an application to send e-mail. The application connects to an SMTP server at port 25 and sends a simple text message.

```
DECLARE
  c UTL_SMTP.CONNECTION;

  PROCEDURE send_header(name IN VARCHAR2, header IN VARCHAR2) AS
  BEGIN
    UTL_SMTP.WRITE_DATA(c, name || ': ' || header || UTL_TCP.CRLF);
  END;

BEGIN
  c := UTL_SMTP.OPEN_CONNECTION('smtp-server.acme.com');
  UTL_SMTP.HELO(c, 'foo.com');
  UTL_SMTP.MAIL(c, 'sender@foo.com');
  UTL_SMTP.RCPT(c, 'recipient@foo.com');
  UTL_SMTP.OPEN_DATA(c);
  send_header('From',    '"Sender" <sender@foo.com>');
  send_header('To',      '"Recipient" <recipient@foo.com>');
  send_header('Subject', 'Hello');
  UTL_SMTP.WRITE_DATA(c, UTL_TCP.CRLF || 'Hello, world!');
  UTL_SMTP.CLOSE_DATA(c);
  UTL_SMTP.QUIT(c);
EXCEPTION
  WHEN utl_smtp.transient_error OR utl_smtp.permanent_error THEN
    BEGIN
      UTL_SMTP.QUIT(c);
    EXCEPTION
      WHEN UTL_SMTP.TRANSIENT_ERROR OR UTL_SMTP.PERMANENT_ERROR THEN
        NULL; -- When the SMTP server is down or unavailable, we don't have
              -- a connection to the server. The QUIT call will raise an
              -- exception that we can ignore.
    END;
    raise_application_error(-20000,
      'Failed to send mail due to the following error: ' || sqlerrm);
END;
```

Summary of UTL_SMTP Subprograms

Table 177–6 UTL_SMTP Package Subprograms

Subprogram	Description
CLOSE_DATA Function and Procedure on page 177-12	Closes the data session
COMMAND Function and Procedure on page 177-13	Performs a generic SMTP command
COMMAND_REPLIES Function on page 177-14	Performs initial handshaking with SMTP server after connecting
DATA Function and Procedure on page 177-15	Performs initial handshaking with SMTP server after connecting, with extended information returned
EHLO Function and Procedure on page 177-16	Performs initial handshaking with SMTP server after connecting, with extended information returned
HELO Function and Procedure on page 177-17	Performs initial handshaking with SMTP server after connecting
HELP Function on page 177-18	Sends `HELP` command
MAIL Function and Procedure on page 177-19	Initiates a mail transaction with the server, the destination is a mailbox
NOOP Function and Procedure on page 177-20	The null command
OPEN_CONNECTION Functions on page 177-21	Opens a connection to an SMTP server
OPEN_DATA Function and Procedure on page 177-23	Sends the DATA command
QUIT Function and Procedure on page 177-24	Terminates an SMTP session and disconnects from the server
RCPT Function on page 177-25	Specifies the recipient of an e-mail message
RSET Function and Procedure on page 177-26	Terminates the current mail transaction
VRFY Function on page 177-27	Verifies the validity of a destination e-mail address
WRITE_DATA Procedure on page 177-28	Writes a portion of the e-mail message
WRITE_RAW_DATA Procedure on page 177-30	Writes a portion of the e-mail message with `RAW` data

CLOSE_DATA Function and Procedure

The CLOSE_DATA call ends the e-mail message by sending the sequence <CR><LF>.<CR><LF> (a single period at the beginning of a line).

Syntax

```
UTL_SMTP.CLOSE_DATA (
   c     IN OUT NOCOPY connection)
RETURN reply;

UTL_SMTP.CLOSE_DATA (
   c     IN OUT NOCOPY connection);
```

Parameters

Table 177–7 CLOSE_DATA Function and Procedure Parameters

Parameter	Description
c	The SMTP connection.

Return Values

Table 177–8 CLOSE_DATA Function and Procedure Return Values

Return Value	Description
reply	Reply of the command (see REPLY, REPLIES Record Types). In cases where there are multiple replies, the last reply will be returned.

Usage Notes

The calls to OPEN_DATA, WRITE_DATA, WRITE_RAW_DATA and CLOSE_DATA must be made in the right order. A program calls OPEN_DATA to send the DATA command to the SMTP server. After that, it can call WRITE_DATA or WRITE_RAW_DATA repeatedly to send the actual data. The data is terminated by calling CLOSE_DATA. After OPEN_DATA is called, the only subprograms that can be called are WRITE_DATA, WRITE_RAW_DATA, or CLOSE_DATA. A call to other APIs will result in an INVALID_OPERATION exception being raised.

CLOSE_DATA should be called only after OPEN_CONNECTION, HELO or EHLO, MAIL, and RCPT have been called. The connection to the SMTP server must be open and a mail transaction must be active when this routine is called.

Note that there is no function form of WRITE_DATA because the SMTP server does not respond until the data-terminator is sent during the call to CLOSE_DATA.

COMMAND Function and Procedure

This function/procedure performs a generic SMTP command.

Syntax

```
UTL_SMTP.COMMAND (
   c     IN OUT NOCOPY    connection,
   cmd   IN               VARCHAR2,
   arg   IN               VARCHAR2 DEFAULT NULL)
RETURN reply;

UTL_SMTP.COMMAND (
   c     IN OUT NOCOPY    connection,
   cmd   IN               VARCHAR2,
   arg   IN               VARCHAR2 DEFAULT NULL);
```

Parameters

Table 177–9 COMMAND Function and Procedure Parameters

Parameter	Description
c	The SMTP connection.
cmd	The SMTP command to send to the server.
arg	The optional argument to the SMTP argument. A space will be inserted between cmd and arg.

Return Values

Table 177–10 COMMAND Function and Procedure Return Values

Return Value	Description
reply	Reply of the command (see REPLY, REPLIES Record Types). In cases where there are multiple replies, the last reply will be returned.

Usage Notes

This function is used to invoke generic SMTP commands. Use COMMAND if only a single reply line is expected. Use COMMAND_REPLIES if multiple reply lines are expected.

For COMMAND, if multiple reply lines are returned from the SMTP server, it returns the last reply line only.

COMMAND_REPLIES Function

This functions performs a generic SMTP command.

Syntax

```
UTL_SMTP.COMMAND_REPLIES (
   c     IN OUT NOCOPY    connection,
   cmd   IN               VARCHAR2,
   arg   IN               VARCHAR2 DEFAULT NULL)
RETURN replies;
```

Parameters

Table 177–11 COMMAND_REPLIES Function Parameters

Parameter	Description
`c`	The SMTP connection.
`cmd`	The SMTP command to send to the server.
`arg`	The optional argument to the SMTP argument. A space will be inserted between `cmd` and `arg`.

Return Values

Table 177–12 COMMAND_REPLIES Function Return Values

Return Value	Description
`replies`	Reply of the command (see REPLY, REPLIES Record Types).

Usage Notes

This function is used to invoke generic SMTP commands. Use `COMMAND` if only a single reply line is expected. Use `COMMAND_REPLIES` if multiple reply lines are expected.

For `COMMAND`, if multiple reply lines are returned from the SMTP server, it returns the last reply line only.

DATA Function and Procedure

This function/procedure specifies the body of an e-mail message.

Syntax

```
UTL_SMTP.DATA (
   c     IN OUT NOCOPY connection
   body  IN  VARCHAR2 CHARACTER SET ANY_CS)
RETURN reply;

UTL_SMTP.DATA (
   c     IN OUT NOCOPY connection
   body  IN VARCHAR2 CHARACTER SET ANY_CS);
```

Parameters

Table 177–13 DATA Function and Procedure Parameters

Parameter	Description
c	The SMTP Connection.
body	The text of the message to be sent, including headers, in [RFC822] format.

Return Values

Table 177–14 DATA Function and Procedure Return Values

Return Value	Description
reply	Reply of the command (see REPLY, REPLIES Record Types). In cases where there are multiple replies, the last reply will be returned.

Usage Notes

The application must ensure that the contents of the body parameter conform to the MIME(RFC822) specification. The DATA routine will terminate the message with a <CR><LF>.<CR><LF> sequence (a single period at the beginning of a line), as required by RFC821. It will also translate any sequence of <CR><LF>.<CR><LF> (single period) in body to <CR><LF>..<CR><LF> (double period). This conversion provides the transparency as described in Section 4.5.2 of RFC821.

The DATA call should be called only after OPEN_CONNECTION, HELO or EHLO, MAIL and RCPT have been called. The connection to the SMTP server must be open, and a mail transaction must be active when this routine is called.

The expected response from the server is a message beginning with status code 250. The 354 response received from the initial DATA command will not be returned to the caller.

EHLO Function and Procedure

This function/procedure performs initial handshaking with SMTP server after connecting, with extended information returned.

Syntax

```
UTL_SMTP.EHLO (
   c       IN OUT NOCOPY connection,
   domain  IN)
RETURN replies;

UTL_SMTP.EHLO (
   c       IN OUT NOCOPY connection,
   domain  IN);
```

Parameters

Table 177–15 EHLO Function and Procedure Parameters

Parameter	Description
c	The SMTP connection.
domain	The domain name of the local (sending) host. Used for identification purposes.

Return Values

Table 177–16 EHLO Function and Procedure Return Values

Return Value	Description
replies	Reply of the command (see REPLY, REPLIES Record Types).

Usage Notes

The EHLO interface is identical to HELO except that it allows the server to return more descriptive information about its configuration. [RFC1869] specifies the format of the information returned, which the PL/SQL application can retrieve using the functional form of this call. For compatibility with HELO, each line of text returned by the server begins with status code 250.

Related Functions

HELO

HELO Function and Procedure

This function/procedure performs initial handshaking with SMTP server after connecting.

Syntax

```
UTL_SMTP.HELO (
   c       IN OUT NOCOPY   connection,
   domain  IN              VARCHAR2)
RETURN reply;

UTL_SMTP.HELO (
   c       IN OUT NOCOPY   connection,
   domain  IN              VARCHAR2);
```

Parameters

Table 177–17 HELO Function and Procedure Parameters

Parameter	Description
c	The SMTP connection.
domain	The domain name of the local (sending) host. Used for identification purposes.

Return Values

Table 177–18 HELO Function and Procedure Return Values

Return Value	Description
reply	Reply of the command (see REPLY, REPLIES Record Types). In cases where there are multiple replies, the last reply will be returned.

Usage Notes

RFC 821 specifies that the client must identify itself to the server after connecting. This routine performs that identification. The connection must have been opened through a call to OPEN_CONNECTION Functions before calling this routine.

The expected response from the server is a message beginning with status code 250.

Related Functions

EHLO

HELP Function

This function sends the `HELP` command.

Syntax

```
UTL_SMTP.HELP (
   c         IN OUT NOCOPY   connection,
   command   IN              VARCHAR2 DEFAULT NULL)
RETURN replies;
```

Parameters

Table 177–19 HELP Function Parameters

Parameter	Description
`c`	The SMTP connection.
`command`	The command to get the help message.

Return Values

Table 177–20 HELP Function Return Values

Return Value	Description
`replies`	Reply of the command (see REPLY, REPLIES Record Types).

MAIL Function and Procedure

This function/procedure initiates a mail transaction with the server. The destination is a mailbox.

Syntax

```
UTL_SMTP.MAIL (
   c           IN OUT NOCOPY   connection,
   sender      IN              VARCHAR2,
   parameters  IN              VARCHAR2 DEFAULT NULL)
RETURN reply;

UTL_SMTP.MAIL (
   c           IN OUT NOCOPY   connection,
   sender      IN              VARCHAR2,
   parameters  IN              VARCHAR2 DEFAULT NULL);
```

Parameters

Table 177–21 MAIL Function and Procedure Parameters

Parameter	Description
c	The SMTP connection.
sender	The e-mail address of the user sending the message.
parameters	The additional parameters to MAIL command as defined in Section 6 of [RFC1869]. It should follow the format of "XXX=XXX (XXX=XXX ....)".

Return Values

Table 177–22 MAIL Function and Procedure Return Values

Return Value	Description
reply	Reply of the command (see REPLY, REPLIES Record Types). In cases where there are multiple replies, the last reply will be returned.

Usage Notes

This command does not send the message; it simply begins its preparation. It must be followed by calls to RCPT and DATA to complete the transaction. The connection to the SMTP server must be open and a HELO or EHLO command must have already been sent.

The expected response from the server is a message beginning with status code 250.

NOOP Function and Procedure

The null command.

Syntax

```
UTL_SMTP.NOOP (
   c  IN OUT NOCOPY connection)
RETURN reply;

UTL_SMTP.NOOP (
   c  IN OUT NOCOPY connection);
```

Parameter

Table 177–23 NOOP Function and Procedure Parameters

Parameter	Description
c	The SMTP connection.

Return Values

Table 177–24 NOOP Function and Procedure Return Values

Return Value	Description
reply	Reply of the command (see REPLY, REPLIES Record Types). In cases where there are multiple replies, the last reply will be returned.

Usage Notes

This command has no effect except to elicit a successful reply from the server. It can be issued at any time after the connection to the server has been established with OPEN_CONNECTION. The NOOP command can be used to verify that the server is still connected and is listening properly.

This command will always reply with a single line beginning with status code 250.

OPEN_CONNECTION Functions

These functions open a connection to an SMTP server.

Syntax

```
UTL_SMTP.OPEN_CONNECTION (
   host        IN  VARCHAR2,
   port        IN  PLS_INTEGER DEFAULT 25,
   c           OUT connection,
   tx_timeout  IN  PLS_INTEGER DEFAULT NULL)
RETURN reply;

UTL_SMTP.OPEN_CONNECTION (
   host        IN  VARCHAR2,
   port        IN  PLS_INTEGER DEFAULT 25,
   tx_timeout  IN  PLS_INTEGER DEFAULT NULL)
RETURN connection;
```

Parameters

Table 177–25 OPEN_CONNECTION Functions Parameters

Parameter	Description
host	The name of the SMTP server host
port	The port number on which SMTP server is listening (usually 25).
c	The SMTP connection.
tx_timeout	The time in seconds that the UTL_SMTP package waits before giving up in a read or write operation in this connection. In read operations, this package gives up if no data is available for reading immediately. In write operations, this package gives up if the output buffer is full and no data is to be sent into the network without being blocked. 0 indicates not to wait at all. NULL indicates to wait forever.

Return Values

Table 177–26 OPEN_CONNECTION Functions Return Values

Return Value	Description
reply	Reply of the command (see REPLY, REPLIES Record Types). In cases where there are multiple replies, the last reply will be returned.

Usage Notes

- The expected response from the server is a message beginning with status code 220.
- The version of OPEN_CONNECTION that returns UTL_SMTP.CONNECTION record checks the reply code returned by an SMTP server when the connection is first established. It raises an exception when the reply indicates an error. Otherwise, it discards the reply. If a user is interested in examining the reply, he or she can invoke the version of OPEN_CONNECTION that returns REPLY.

- A timeout on the `WRITE` operations feature is not supported in the current release of this package.

OPEN_DATA Function and Procedure

OPEN_DATA sends the DATA command after which you can use WRITE_DATA and WRITE_RAW_DATA to write a portion of the e-mail message.

Syntax

```
UTL_SMTP.OPEN_DATA (
   c     IN OUT NOCOPY connection)
RETURN reply;

UTL_SMTP.OPEN_DATA (
   c     IN OUT NOCOPY connection);
```

Parameters

Table 177–27 OPEN_DATA Function and Procedure Parameters

Parameter	Description
c	The SMTP connection.
data	The portion of the text of the message to be sent, including headers, in [RFC822] format.

Return Values

Table 177–28 OPEN_DATA Function and Procedure Function Return Values

Return Value	Description
reply	Reply of the command (see REPLY, REPLIES Record Types). In cases where there are multiple replies, the last reply will be returned.

Usage Notes

The calls to OPEN_DATA, WRITE_DATA, WRITE_RAW_DATA and CLOSE_DATA must be made in the right order. A program calls OPEN_DATA to send the DATA command to the SMTP server. After that, it can call WRITE_DATA or WRITE_RAW_DATA repeatedly to send the actual data. The data is terminated by calling CLOSE_DATA. After OPEN_DATA is called, the only subprograms that can be called are WRITE_DATA, WRITE_RAW_DATA, or CLOSE_DATA. A call to other APIs will result in an INVALID_OPERATION exception being raised.

OPEN_DATA should be called only after OPEN_CONNECTION, HELO or EHLO, MAIL, and RCPT have been called. The connection to the SMTP server must be open and a mail transaction must be active when this routine is called.

QUIT Function and Procedure

This function terminates an SMTP session and disconnects from the server.

Syntax

```
UTL_SMTP.QUIT (
   c  IN OUT NOCOPY connection)
RETURN reply;

UTL_SMTP.QUIT (
   c  IN OUT NOCOPY connection);
```

Parameter

Table 177–29 QUIT Function and Procedure Parameters

Parameter	Description
c	The SMTP connection.

Return Values

Table 177–30 QUIT Function and Procedure Function Return Values

Return Value	Description
reply	Reply of the command (see REPLY, REPLIES Record Types). In cases where there are multiple replies, the last reply will be returned.

Usage Notes

The QUIT command informs the SMTP server of the client's intent to terminate the session. It then closes the connection established by OPEN_CONNECTION which must have been called before executing this command. If a mail transaction is in progress when QUIT is issued, it is abandoned in the same manner as RSET.

The function form of this command returns a single line beginning with the status code 221 on successful termination. In all cases, the connection to the SMTP server is closed. The fields REMOTE_HOST and REMOTE_PORT of c are reset.

Related Functions

RSET

RCPT Function

This function/procedure specifies the recipient of an e-mail message.

Syntax

```
UTL_SMTP.RCPT (
   c           IN OUT NOCOPY     connection,
   recipient   IN                VARCHAR2,
   parameters  IN                VARCHAR2 DEFAULT NULL)
RETURN reply;

UTL_SMTP.RCPT (
   c           IN OUT NOCOPY     connection,
   recipient   IN                VARCHAR2,
   parameters  IN                VARCHAR2 DEFAULT NULL);
```

Table 177–31 RCPT Function and Procedure Parameters

Parameter	Description
`c`	The SMTP connection.
`recipient`	The e-mail address of the user to which the message is being sent.
`parameters`	The additional parameters to RCPT command as defined in Section 6 of [RFC1869]. It should follow the format of "XXX=XXX (XXX=XXX ....)".

Return Values

Table 177–32 RCPT Function and Procedure Function Return Values

Return Value	Description
`reply`	Reply of the command (see REPLY, REPLIES Record Types). In cases where there are multiple replies, the last reply will be returned.

Usage Notes

To send a message to multiple recipients, call this routine multiple times. Each invocation schedules delivery to a single e-mail address. The message transaction must have been begun by a prior call to `MAIL`, and the connection to the mail server must have been opened and initialized by prior calls to `OPEN_CONNECTION` and `HELO` or `EHLO` respectively.

The expected response from the server is a message beginning with status code 250 or 251.

RSET Function and Procedure

This function terminates the current mail transaction.

Syntax

```
UTL_SMTP.RSET (
   c  IN OUT NOCOPY connection)
RETURN reply;

UTL_SMTP.RSET (
   c  IN OUT NOCOPY connection);
```

Parameters

Table 177–33 RSET Function and Procedure Parameters

Parameter	Description
c	The SMTP connection.

Return Values

Table 177–34 RSET Function and Procedure Return Values

Return Value	Description
reply	Reply of the command (see REPLY, REPLIES Record Types). In cases where there are multiple replies, the last reply will be returned.

Usage Notes

This command allows the client to abandon a mail message it was in the process of composing. No mail will be sent. The client can call RSET at any time after the connection to the SMTP server has been opened by means of OPEN_CONNECTION until DATA or OPEN_DATA is called. Once the email data has been sent, it will be too late to prevent the email from being sent.

The server will always respond to RSET with a message beginning with status code 250.

Related Functions

QUIT

VRFY Function

This function verifies the validity of a destination e-mail address.

Syntax

```
UTL_SMTP.VRFY (
   c          IN OUT NOCOPY connection
   recipient  IN VARCHAR2)
RETURN reply;
```

Parameters

Table 177–35 VRFY Function Parameters

Parameter	Description
`c`	The SMTP connection.
`recipient`	The e-mail address to be verified.

Return Values

Table 177–36 VRFY Function Return Values

Return Value	Description
`reply`	Reply of the command (see REPLY, REPLIES Record Types). In cases where there are multiple replies, the last reply will be returned.

Usage Notes

The server attempts to resolve the destination address `recipient`. If successful, it returns the recipient's full name and fully qualified mailbox path. The connection to the server must have already been established by means of `OPEN_CONNECTION` and `HELO` or `EHLO` before making this request.

Successful verification returns one or more lines beginning with status code 250 or 251.

WRITE_DATA Procedure

Use WRITE_DATA to write a portion of the e-mail message. A repeat call to WRITE_DATA appends data to the e-mail message.

Syntax

```
UTL_SMTP.WRITE_DATA (
   c     IN OUT NOCOPY connection,
   data  IN VARCHAR2 CHARACTER SET ANY_CS);
```

Parameters

Table 177–37 WRITE_DATA Procedure Parameters

Parameter	Description
c	The SMTP connection.
data	The portion of the text of the message to be sent, including headers, in [RFC822] format.

Usage Notes

The calls to OPEN_DATA, WRITE_DATA, WRITE_RAW_DATA and CLOSE_DATA must be made in the right order. A program calls OPEN_DATA to send the DATA command to the SMTP server. After that, it can call WRITE_DATA or WRITE_RAW_DATA repeatedly to send the actual data. The data is terminated by calling CLOSE_DATA. After OPEN_DATA is called, the only subprograms that can be called are WRITE_DATA, WRITE_RAW_DATA, or CLOSE_DATA. A call to other APIs will result in an INVALID_OPERATION exception being raised.

The application must ensure that the contents of the body parameter conform to the MIME(RFC822) specification. The DATA routine will terminate the message with a <CR><LF>.<CR><LF> sequence (a single period at the beginning of a line), as required by RFC821. It will also translate any sequence of <CR><LF>.<CR><LF> (single period) in the body to <CR><LF>..<CR><LF> (double period). This conversion provides the transparency as described in Section 4.5.2 of RFC821.

Notice that this conversion is not bullet-proof. Consider this code fragment:

```
UTL_SMTP.WRITE_DATA('some message.' || chr(13) || chr(10));
UTL_SMTP.WRITE_DATA('.' || chr(13) || chr(10));
```

Since the sequence <CR><LF>.<CR><LF> is split between two calls to WRITE_DATA, the implementation of WRITE_DATA will not detect the presence of the data-terminator sequence, and therefore, will not perform the translation. It will be the responsibility of the user to handle such a situation, or it may result in premature termination of the message data.

WRITE_DATA should be called only after OPEN_CONNECTION, HELO or EHLO, MAIL, and RCPT have been called. The connection to the SMTP server must be open and a mail transaction must be active when this routine is called.

Note that there is no function form of WRITE_DATA because the SMTP server does not respond until the data-terminator is sent during the call to CLOSE_DATA.

Text (VARCHAR2) data sent using WRITE_DATA is converted to US7ASCII before it is sent. If the text contains multibyte characters, each multibyte character in the text that cannot be converted to US7ASCII is replaced by a '?' character. If 8BITMIME extension

is negotiated with the SMTP server using the EHLO subprogram, multibyte VARCHAR2 data can be sent by first converting the text to RAW using the UTL_RAW package, and then sending the RAW data using WRITE_RAW_DATA.

WRITE_RAW_DATA Procedure

Use WRITE_RAW_DATA to write a portion of the e-mail message. A repeat call to WRITE_RAW_DATA appends data to the e-mail message.

Syntax

```
UTL_SMTP.WRITE_RAW_DATA (
   c     IN OUT NOCOPY connection
   data  IN RAW);
```

Parameters

Table 177–38 WRITE_RAW_DATA Procedure Parameters

Parameter	Description
c	The SMTP connection.
data	The portion of the text of the message to be sent, including headers, in [RFC822] format.

Usage Notes

The calls to OPEN_DATA, WRITE_DATA, WRITE_RAW_DATA and CLOSE_DATA must be made in the right order. A program calls OPEN_DATA to send the DATA command to the SMTP server. After that, it can call WRITE_DATA or WRITE_RAW_DATA repeatedly to send the actual data. The data is terminated by calling CLOSE_DATA. After OPEN_DATA is called, the only subprograms that can be called are WRITE_DATA, WRITE_RAW_DATA, or CLOSE_DATA. A call to other APIs will result in an INVALID_OPERATION exception being raised.

The application must ensure that the contents of the body parameter conform to the MIME(RFC822) specification. The DATA routine will terminate the message with a <CR><LF>.<CR><LF> sequence (a single period at the beginning of a line), as required by RFC821. It will also translate any sequence of <CR><LF>.<CR><LF> (single period) in the body to <CR><LF>..<CR><LF> (double period). This conversion provides the transparency as described in Section 4.5.2 of RFC821.

Notice that this conversion is not bullet-proof. Consider this code fragment:

```
UTL_SMTP.WRITE_DATA('some message.' || chr(13) || chr(10));
UTL_SMTP.WRITE_DATA('.' || chr(13) || chr(10));
```

Since the sequence <CR><LF>.<CR><LF> is split between two calls to WRITE_DATA, the implementation of WRITE_DATA will not detect the presence of the data-terminator sequence, and therefore, will not perform the translation. It will be the responsibility of the user to handle such a situation, or it may result in premature termination of the message data.

XXX_DATA should be called only after OPEN_CONNECTION, HELO or EHLO, MAIL, and RCPT have been called. The connection to the SMTP server must be open and a mail transaction must be active when this routine is called.

Note that there is no function form of WRITE_DATA because the SMTP server does not respond until the data-terminator is sent during the call to CLOSE_DATA.

Text (VARCHAR2) data sent using WRITE_DATA is converted to US7ASCII before it is sent. If the text contains multibyte characters, each multibyte character in the text that cannot be converted to US7ASCII is replaced by a '?' character. If 8BITMIME extension

is negotiated with the SMTP server using the EHLO subprogram, multibyte VARCHAR2 data can be sent by first converting the text to RAW using the UTL_RAW package, and then sending the RAW data using WRITE_RAW_DATA.

178

UTL_TCP

With the `UTL_TCP` package and its procedures and functions, PL/SQL applications can communicate with external TCP/IP-based servers using TCP/IP. Because many Internet application protocols are based on TCP/IP, this package is useful to PL/SQL applications that use Internet protocols and e-mail.

This chapter contains the following topics:

Using UTL_TCP

Overview

The UTL_TCP package provides TCP/IP client-side access functionality in PL/SQL.

Types

CONNECTION Type

This is a PL/SQL record type used to represent a TCP/IP connection.

Syntax

```
TYPE connection IS RECORD (
    remote_host    VARCHAR2(255),
    remote_port    PLS_INTEGER,
    local_host     VARCHAR2(255),
    local_port     PLS_INTEGER,
    charset        VARCHAR2(30),
    newline        VARCHAR2(2),
    tx_timeout     PLS_INTEGER,
    private_sd     PLS_INTEGER);
```

Fields

Table 178–1 Connection Record Type Fields

Field	Description
`remote_host`	The name of the remote host when connection is established. `NULL` when no connection is established.
`remote_port`	The port number of the remote host connected. NULL when no connection is established.
`local_host`	The name of the local host used to establish the connection. `NULL` when no connection is established.
`local_port`	The port number of the local host used to establish the connection. `NULL` when no connection is established.
`charset`	The on-the-wire character set. Since text messages in the database may be encoded in a character set that is different from the one expected on the wire (that is, the character set specified by the communication protocol, or the one stipulated by the other end of the communication), text messages in the database will be converted to and from the on-the-wire character set as they are sent and received on the network.
`newline`	The newline character sequence. This newline character sequence is appended to the text line sent by `WRITE_LINE` API.
`tx_timeout`	A time in seconds that the UTL_TCP package waits before giving up in a read or write operation in this connection. In read operations, this package gives up if no data is available for reading immediately. In write operations, this package gives up if the output buffer is full and no data is to be sent in the network without being blocked. Zero (0) indicates not to wait at all. NULL indicates to wait forever.

Usage Notes

The fields in a connection record are used to return information about the connection, which is often made using `OPEN_CONNECTION`. Changing the values of those fields

has no effect on the connection. The fields `private_XXXX` are for implementation use only. You should not modify the values.

In the current release of the `UTL_TCP` package, the parameters `local_host` and `local_port` are ignored when `open_connection` makes a TCP/IP connection. It does not attempt to use the specified local host and port number when the connection is made. The `local_host` and `local_port` fields will not be set in the connection record returned by the function.

Time out on write operations is not supported in the current release of the `UTL_TCP` package.

CRLF

The character sequence carriage-return line-feed. It is the newline sequence commonly used many communication standards.

Syntax

```
CRLF varchar2(10);
```

Usage Notes

This package variable defines the newline character sequence commonly used in many Internet protocols. This is the default value of the newline character sequence for `WRITE_LINE`, specified when a connection is opened. While such protocols use `<CR><LF>` to denote a new line, some implementations may choose to use just line-feed to denote a new line. In such cases, users can specify a different newline character sequence when a connection is opened.

This CRLF package variable is intended to be a constant that denotes the carriage-return line-feed character sequence. Do not modify its value. Modification may result in errors in other PL/SQL applications.

Exceptions

The exceptions raised by the TCP/IP package are listed in Table 178–2.

Table 178–2 TCP/IP Exceptions

Exception	Description
BUFFER_TOO_SMALL	Buffer is too small for input that requires look-ahead.
END_OF_INPUT	Raised when no more data is available to read from the connection.
NETWORK_ERROR	Generic network error.
BAD_ARGUMENT	Bad argument passed in an API call (for example, a negative buffer size).
TRANSFER_TIMEOUT	No data is read and a read time out occurred.
PARTIAL_MULTIBYTE_CHAR	No complete character is read and a partial multibyte character is found at the end of the input.

Rules and Limits

The interface provided in the package only allows connections to be initiated by the PL/SQL program. It does not allow the PL/SQL program to accept connections initiated outside the program.

Examples

The following code example illustrates how the TCP/IP package can be used to retrieve a Web page over HTTP. It connects to a Web server listening at port 80 (standard port for HTTP) and requests the root document.

```
DECLARE
  c  utl_tcp.connection;  -- TCP/IP connection to the Web server
  ret_val pls_integer;
BEGIN
  c := utl_tcp.open_connection(remote_host => 'www.acme.com',
                               remote_port =>  80,
                               charset     => 'US7ASCII');  -- open connection
  ret_val := utl_tcp.write_line(c, 'GET / HTTP/1.0');    -- send HTTP request
  ret_val := utl_tcp.write_line(c);
  BEGIN
    LOOP
      dbms_output.put_line(utl_tcp.get_line(c, TRUE));  -- read result
    END LOOP;
  EXCEPTION
    WHEN utl_tcp.end_of_input THEN
      NULL; -- end of input
  END;
  utl_tcp.close_connection(c);
END;
```

The following code example illustrates how the TCP/IP package can be used by an application to send e-mail (also known as email from PL/SQL). The application connects to an SMTP server at port 25 and sends a simple text message.

```
PROCEDURE send_mail (sender    IN VARCHAR2,
                     recipient IN VARCHAR2,
                   message   IN VARCHAR2)
IS
    mailhost   VARCHAR2(30) := 'mailhost.mydomain.com';
    smtp_error  EXCEPTION;
    mail_conn   utl_tcp.connection;
    PROCEDURE smtp_command(command IN VARCHAR2,
                           ok      IN VARCHAR2 DEFAULT '250')
    IS
        response varchar2(3);
        len pls_integer;
    BEGIN
        len := utl_tcp.write_line(mail_conn, command);
        response := substr(utl_tcp.get_line(mail_conn), 1, 3);
        IF (response <> ok) THEN
            RAISE smtp_error;
        END IF;
    END;

BEGIN
    mail_conn := utl_tcp.open_connection(remote_host => mailhost,
                                         remote_port => 25,
                                         charset     => 'US7ASCII');
    smtp_command('HELO ' || mailhost);
    smtp_command('MAIL FROM: ' || sender);
    smtp_command('RCPT TO: ' || recipient);
    smtp_command('DATA', '354');
    smtp_command(message);
```

```
    smtp_command('QUIT', '221');
    utl_tcp.close_connection(mail_conn);
EXCEPTION
    WHEN OTHERS THEN
        -- Handle the error
END;
```

Summary of UTL_TCP Subprograms

Table 178–3 UTL_TCP Package Subprograms

Subprogram	Description
AVAILABLE Function on page 178-11	Determines the number of bytes available for reading from a TCP/IP connection
CLOSE_ALL_ CONNECTIONS Procedure on page 178-13	Closes all open TCP/IP connections
CLOSE_CONNECTION Procedure on page 178-14	Closes an open TCP/IP connection
FLUSH Procedure on page 178-15	Transmits all data in the output buffer, if a buffer is used, to the server immediately
GET_LINE Function on page 178-16	Convenient forms of the read functions, which return the data read instead of the amount of data read
GET_RAW Function on page 178-17	Convenient forms of the read functions, which return the data read instead of the amount of data read
GET_TEXT Function on page 178-18	Convenient forms of the read functions, which return the data read instead of the amount of data read
OPEN_CONNECTION Function on page 178-19	Opens a TCP/IP connection to a specified service
READ_LINE Function on page 178-21	Receives a text line from a service on an open connection
READ_RAW Function on page 178-23	Receives binary data from a service on an open connection
READ_TEXT Function on page 178-24	Receives text data from a service on an open connection
WRITE_LINE Function on page 178-26	Transmits a text line to a service on an open connection
WRITE_RAW Function on page 178-27	Transmits a binary message to a service on an open connection
WRITE_TEXT Function on page 178-28	Transmits a text message to a service on an open connection

AVAILABLE Function

This function determines the number of bytes available for reading from a TCP/IP connection. It is the number of bytes that can be read immediately without blocking. Determines if data is ready to be read from the connection.

Syntax

```
UTL_TCP.AVAILABLE (
   c        IN OUT NOCOPY connection,
   timeout  IN PLS_INTEGER DEFAULT 0)
RETURN num_bytes PLS_INTEGER;
```

Parameters

Table 178–4 AVAILABLE Function Parameters

Parameter	Description
c	The TCP connection to determine the amount of data that is available to be read from.
timeout	A time in seconds to wait before giving up and reporting that no data is available. Zero (0) indicates not to wait at all. NULL indicates to wait forever.

Return Values

Table 178–5 AVAILABLE Function Return Values

Parameter	Description
num_bytes	The number of bytes available for reading without blocking.

Usage Notes

The connection must have already been opened through a call to OPEN_CONNECTION. Users may use this API to determine if data is available to be read before calling the read API so that the program will not be blocked because data is not ready to be read from the input.

The number of bytes available for reading returned by this function may less than than what is actually available. On some platforms, this function may only return 1, to indicate that some data is available. If you are concerned about the portability of your application, assume that this function returns a positive value when data is available for reading, and 0 when no data is available. This function returns a positive value when all the data at a particular connection has been read and the next read will result in the END_OF_INPUT exception.

The following example illustrates using this function in a portable manner:

```
DECLARE
   c   utl_tcp.connection
   data VARCHAR2(256);
   len  PLS_INTEGER;
BEGIN
   c := utl_tcp.open_connection(...);
   LOOP
      IF (utl_tcp.available(c) > 0) THEN
         len := utl_tcp.read_text(c, data, 256);
```

```
      ELSE
         ---do some other things
        . . . .
      END IF
   END LOOP;
END;
```

Related Functions

READ_RAW, READ_TEXT, READ_LINE

CLOSE_ALL_CONNECTIONS Procedure

This procedure closes all open TCP/IP connections.

Syntax

```
UTL_TCP.CLOSE_ALL_CONNECTIONS;
```

Usage Notes

This call is provided to close all connections before a PL/SQL program avoid dangling connections.

Related Functions

OPEN_CONNECTION, CLOSE_CONNECTION.

CLOSE_CONNECTION Procedure

This procedure closes an open TCP/IP connection.

Syntax

```
UTL_TCP.CLOSE_CONNECTION (
   c IN OUT NOCOPY connection);
```

Parameters

Table 178–6 CLOSE_CONNECTION Procedure Parameters

Parameter	Description
c	The TCP connection to close.

Usage Notes

Connection must have been opened by a previous call to `OPEN_CONNECTION`. The fields `remote_host, remote_port, local_host, local_port` and `charset` of c will be reset after the connection is closed.

An open connection must be closed explicitly. An open connection will remain open when the PL/SQL record variable that stores the connection goes out-of-scope in the PL/SQL program. Failing to close unwanted connections may result in unnecessary tying up of local and remote system resources.

FLUSH Procedure

This procedure transmits all data in the output buffer, if a buffer is used, to the server immediately.

Syntax

```
UTL_TCP.FLUSH (
   c IN OUT NOCOPY connection);
```

Parameters

Table 178–7 FLUSH Procedure Parameters

Parameter	Description
c	The TCP connection to send data to.

Usage Notes

The connection must have already been opened through a call to OPEN_CONNECTION.

Related Functions

WRITE_RAW, WRITE_TEXT, WRITE_LINE

GET_LINE Function

This function returns the data read instead of the amount of data read.

Syntax

```
UTL_TCP.GET_LINE (
   c           IN OUT NOCOPY connection,
   remove_crlf IN            BOOLEAN DEFAULT FALSE,
   peek        IN            BOOLEAN DEFAULT FALSE)
 RETURN VARCHAR2;
```

Parameters

Table 178–8 GET_LINE Function Parameters

Parameter	Description
c	The TCP connection to receive data from.
remove_crlf	If TRUE, the trailing CR/LF character(s) are removed from the received message.
peek	Normally, you want to read the data and remove it from the input queue, that is, consume it. In some situations, you may just want to look ahead at the data, that is, peek at it, without removing it from the input queue, so that it is still available for reading (or even peeking) in the next call. To keep the data in the input queue, set this flag to TRUE and set up an input buffer before the connection is opened. The amount of data you can peeked at (that is, read but keep in the input queue) must be less than the size of input buffer.

Usage Notes

- The connection must have already been opened through a call to OPEN_CONNECTION.
- See READ_LINE for the read time out, character set conversion, buffer size, and multibyte character issues.

Related Functions

GET_RAW, GET_TEXT, READ_LINE

GET_RAW Function

This function returns the data read instead of the amount of data read.

Syntax

```
UTL_TCP.GET_RAW (
   c     IN OUT NOCOPY connection,
   len   IN            PLS_INTEGER DEFAULT 1,
   peek  IN            BOOLEAN     DEFAULT FALSE)
 RETURN RAW;
```

Parameters

Table 178–9 GET_RAW Function Parameters

Parameter	Description
c	The TCP connection to receive data from.
len	The number of bytes (or characters for VARCHAR2) of data to receive. Default is 1.
peek	Normally, you want to read the data and remove it from the input queue, that is, consume it. In some situations, you may just want to look ahead at the data, that is, peek at it, without removing it from the input queue, so that it is still available for reading (or even peeking) in the next call. To keep the data in the input queue, set this flag to TRUE and set up an input buffer before the connection is opened. The amount of data you can peeked at (that is, read but keep in the input queue) must be less than the size of input buffer.
remove_crlf	If TRUE, the trailing CR/LF character(s) are removed from the received message.

Usage Notes

The connection must have already been opened through a call to OPEN_CONNECTION.

For all the get_* APIs described in this section, see the corresponding READ_* API for the read time out issue. For GET_TEXT and GET_LINE, see the corresponding READ_* API for character set conversion, buffer size, and multibyte character issues.

Related Functions

GET_RAW, GET_TEXT, READ_RAW, READ_TEXT, READ_LINE

GET_TEXT Function

This function returns the data read instead of the amount of data read.

Syntax

```
UTL_TCP.GET_TEXT (
   c    IN OUT NOCOPY connection,
   len  IN            PLS_INTEGER DEFAULT 1,
   peek IN            BOOLEAN     DEFAULT FALSE)
 RETURN VARCHAR2;
```

Parameters

Table 178–10 GET_TEXT Function Parameters

Parameter	Description
c	The TCP connection to receive data from.
len	The number of bytes (or characters for VARCHAR2) of data to receive. Default is 1.
peek	Normally, you want to read the data and remove it from the input queue, that is, consume it. In some situations, you may just want to look ahead at the data, that is, peek at it, without removing it from the input queue, so that it is still available for reading (or even peeking) in the next call. To keep the data in the input queue, set this flag to TRUE and set up an input buffer before the connection is opened. The amount of data you can peeked at (that is, read but keep in the input queue) must be less than the size of input buffer.
remove_crlf	If TRUE, the trailing CR/LF character(s) are removed from the received message.

Usage Notes

The connection must have already been opened through a call to OPEN_CONNECTION.

For all the get_* APIs described in this section, see the corresponding read_* API for the read time out issue. For GET_TEXT and GET_LINE, see the corresponding READ_* API for character set conversion, buffer size, and multibyte character issues.

Related Functions

READ_RAW, READ_TEXT, READ_LINE

OPEN_CONNECTION Function

This function opens a TCP/IP connection to a specified service.

Syntax

```
UTL_TCP.OPEN_CONNECTION  (
   remote_host      IN VARCHAR2,
   remote_port      IN PLS_INTEGER,
   local_host       IN VARCHAR2 DEFAULT NULL,
   local_port       IN PLS_INTEGER DEFAULT NULL,
   in_buffer_size   IN PLS_INTEGER DEFAULT NULL,
   out_buffer_size  IN PLS_INTEGER DEFAULT NULL,
   charset          IN VARCHAR2 DEFAULT NULL,
   newline          IN VARCHAR2 DEFAULT CRLF,
   tx_timeout       IN PLS_INTEGER DEFAULT NULL)
  RETURN connection;
```

Parameters

Table 178–11 OPEN_CONNECTION Function Parameters

Parameter	Description
remote_host	The name of the host providing the service. When remote_host is NULL, it connects to the local host.
remote_port	The port number on which the service is listening for connections.
local_host	The name of the host providing the service. NULL means don't care.
local_port	The port number on which the service is listening for connections. NULL means don't care.
in_buffer_size	The size of input buffer. The use of an input buffer can speed up execution performance in receiving data from the server. The appropriate size of the buffer depends on the flow of data between the client and the server, and the network condition. A 0 value means no buffer should be used. A NULL value means the caller does not care if a buffer is used or not. The maximum size of the input buffer is 32767 bytes.
out_buffer_size	The size of output buffer. The use of an output buffer can speed up execution performance in sending data to the server. The appropriate size of buffer depends on the flow of data between the client and the server, and the network condition. A 0 value means no buffer should be used. A NULL value means the caller does not care if a buffer is used or not. The maximum size of the output buffer is 32767 bytes.
charset	The on-the-wire character set. Since text messages in the database may be encoded in a character set that is different from the one expected on the wire (that is, the character set specified by the communication protocol, or the one stipulated by the other end of the communication), text messages in the database will be converted to and from the on-the-wire character set as they are sent and received on the network using READ_TEXT, READ_LINE, WRITE_TEXT and WRITE_LINE. Set this parameter to NULL when no conversion is needed.

Table 178–11 (Cont.) OPEN_CONNECTION Function Parameters

Parameter	Description
newline	The newline character sequence. This newline character sequence is appended to the text line sent by WRITE_LINE API.
tx_timeout	A time in seconds that the UTL_TCP package should wait before giving up in a read or write operations in this connection. In read operations, this package gives up if no data is available for reading immediately. In write operations, this package gives up if the output buffer is full and no data is to be sent in the network without being blocked. Zero (0) indicates not to wait at all. NULL indicates to wait forever.

Return Values

Table 178–12 OPEN_CONNECTION Function Return Values

Parameter	Description
connection	A connection to the targeted TCP/IP service.

Usage Notes

Note that connections opened by this UTL_TCP package can remain open and be passed from one database call to another in a shared server configuration. However, the connection must be closed explicitly. The connection will remain open when the PL/SQL record variable that stores the connection goes out-of-scope in the PL/SQL program. Failing to close unwanted connections may result in unnecessary tying up of local and remote system resources.

In the current release of the UTL_TCP package, the parameters local_host and local_port are ignored when open_connection makes a TCP/IP connection. It does not attempt to use the specified local host and port number when the connection is made. The local_host and local_port fields will not be set in the connection record returned by the function.

Time out on write operations is not supported in the current release of the UTL_TCP package.

Related Functions

CLOSE_CONNECTION, CLOSE_ALL_CONNECTIONS

READ_LINE Function

This function receives a text line from a service on an open connection. A line is terminated by a line-feed, a carriage-return or a carriage-return followed by a line-feed.

Syntax

```
UTL_TCP.READ_LINE (
   c           IN OUT NOCOPY connection,
   data        IN OUT NOCOPY VARCHAR2 CHARACTER SET ANY_CS,
   peek        IN            BOOLEAN DEFAULT FALSE)
 RETURN num_chars PLS_INTEGER;
```

Parameters

Table 178–13 READ_LINE Function Parameters

Parameter	Description
c	The TCP connection to receive data from.
data	The data received.
remove_crlf	If TRUE, the trailing CR/LF character(s) are removed from the received message.
peek	Normally, you want to read the data and remove it from the input queue, that is, consume it. In some situations, you may just want to look ahead at the data, that is, peek at it, without removing it from the input queue, so that it is still available for reading (or even peeking) in the next call. To keep the data in the input queue, set this flag to TRUE and set up an input buffer before the connection is opened. The amount of data you can peeked at (that is, read but keep in the input queue) must be less than the size of input buffer.

Return Values

Table 178–14 READ_LINE Function Return Values

Parameter	Description
num_chars	The actual number of characters of data received.

Usage Notes

The connection must have already been opened through a call to OPEN_CONNECTION. This function does not return until the end-of-line have been reached, or the end of input has been reached. Text messages will be converted from the on-the-wire character set, specified when the connection was opened, to the database character set before they are returned to the caller.

If transfer time out is set when the connection is opened, this function waits for each data packet to be ready to read until time out occurs. If it occurs, this function stops reading and returns all the data read successfully. If no data is read successfully, the transfer_timeout exception is raised. The exception can be handled and the read operation can be retried later.

If a partial multibyte character is found at the end of input, this function stops reading and returns all the complete multibyte characters read successfully. If no complete

character is read successfully, the `partial_multibyte_char` exception is raised. The exception can be handled and the bytes of that partial multibyte character can be read as binary by the `READ_RAW` function. If a partial multibyte character is seen in the middle of the input because the remaining bytes of the character have not arrived and read time out occurs, the `transfer_timeout` exception is raised instead. The exception can be handled and the read operation can be retried later.

Related Functions

`READ_RAW, READ_TEXT, AVAILABLE`

READ_RAW Function

This function receives binary data from a service on an open connection.

Syntax

```
UTL_TCP.READ_RAW (
   c     IN OUT NOCOPY connection,
   data  IN OUT NOCOPY RAW,
   len   IN            PLS_INTEGER DEFAULT 1,
   peek  IN            BOOLEAN     DEFAULT FALSE)
 RETURN num_bytes PLS_INTEGER;
```

Parameters

Table 178–15 READ_RAW Function Parameters

Parameter	Description
c	The TCP connection to receive data from.
data (IN OUT COPY)	The data received.
len	The number of bytes of data to receive.
peek	Normally, you want to read the data and remove it from the input queue, that is, consume it. In some situations, you may just want to look ahead at the data, that is, peek at it, without removing it from the input queue, so that it is still available for reading (or even peeking) in the next call. To keep the data in the input queue, set this flag to TRUE and set up an input buffer before the connection is opened. The amount of data you can peeked at (that is, read but keep in the input queue) must be less than the size of input buffer.

Return Values

Table 178–16 READ_RAW Function Return Values

Parameter	Description
num_bytes	The actual number of bytes of data received.

Usage Notes

The connection must have already been opened through a call to OPEN_CONNECTION. This function does not return until the specified number of bytes have been read, or the end of input has been reached.

If transfer time out is set when the connection is opened, this function waits for each data packet to be ready to read until time out occurs. If it occurs, this function stops reading and returns all the data read successfully. If no data is read successfully, the transfer_timeout exception is raised. The exception can be handled and the read operation can be retried later.

Related Functions

READ_TEXT, READ_LINE, AVAILABLE

READ_TEXT Function

This function receives text data from a service on an open connection.

Syntax

```
UTL_TCP.READ_TEXT (
   c    IN OUT NOCOPY connection,
   data IN OUT NOCOPY VARCHAR2 CHARACTER SET ANY_CS,
   len  IN            PLS_INTEGER DEFAULT 1,
   peek IN            BOOLEAN     DEFAULT FALSE)
RETURN num_chars PLS_INTEGER;
```

Parameters

Table 178–17 READ_TEXT Function Parameters

Parameter	Description
c	The TCP connection to receive data from.
data	The data received.
len	The number of characters of data to receive.
peek	Normally, users want to read the data and remove it from the input queue, that is, consume it. In some situations, users may just want to look ahead at the data without removing it from the input queue so that it is still available for reading (or even peeking) in the next call. To keep the data in the input queue, set this flag to TRUE and an input buffer must be set up when the connection is opened. The amount of data that you can peek at (that is, read but keep in the input queue) must be less than the size of input buffer.

Return Values

Table 178–18 READ_TEXT Function Return Values

Parameter	Description
num_chars	The actual number of characters of data received.

Usage Notes

The connection must have already been opened through a call to OPEN_CONNECTION. This function does not return until the specified number of characters has been read, or the end of input has been reached. Text messages will be converted from the on-the-wire character set, specified when the connection was opened, to the database character set before they are returned to the caller.

Unless explicitly overridden, the size of a VARCHAR2 buffer is specified in terms of bytes, while the parameter len refers to the maximum number of characters to be read. When the database character set is multibyte, where a single character may consist of more than 1 byte, you should ensure that the buffer can hold the maximum of characters. In general, the size of the VARCHAR2 buffer should equal the number of characters to be read, multiplied by the maximum number of bytes of a character of the database character set.

If transfer time out is set when the connection is opened, this function waits for each data packet to be ready to read until time out occurs. If it occurs, this function stops

reading and returns all the data read successfully. If no data is read successfully, the `transfer_timeout` exception is raised. The exception can be handled and the read operation can be retried later.

If a partial multibyte character is found at the end of input, this function stops reading and returns all the complete multibyte characters read successfully. If no complete character is read successfully, the `partial_multibyte_char` exception is raised. The exception can be handled and the bytes of that partial multibyte character can be read as binary by the `READ_RAW` function. If a partial multibyte character is seen in the middle of the input because the remaining bytes of the character have not arrived and read time out occurs, the `transfer_timeout` exception is raised instead. The exception can be handled and the read operation can be retried later.

Related Functions

`READ_RAW, READ_LINE, AVAILABLE`

WRITE_LINE Function

This function transmits a text line to a service on an open connection. The newline character sequence will be appended to the message before it is transmitted.

Syntax

```
UTL_TCP.WRITE_LINE (
   c    IN OUT NOCOPY connection,
   data IN            VARCHAR2 DEFAULT NULL CHARACTER SET ANY_CS)
  RETURN PLS_INTEGER;
```

Parameters

Table 178–19 WRITE_LINE Function Parameters

Parameter	Description
c	The TCP connection to send data to.
data	The buffer containing the data to be sent.

Return Values

Table 178–20 WRITE_LINE Function Return Values

Parameter	Description
num_chars	The actual number of characters of data transmitted.

Usage Notes

The connection must have already been opened through a call to OPEN_CONNECTION. Text messages will be converted to the on-the-wire character set, specified when the connection was opened, before they are transmitted on the wire.

Related Functions

WRITE_RAW, WRITE_TEXT, FLUSH

WRITE_RAW Function

This function transmits a binary message to a service on an open connection. The function does not return until the specified number of bytes have been written.

Syntax

```
UTL_TCP.WRITE_RAW (
   c    IN OUT NOCOPY connection,
   data IN            RAW,
   len  IN            PLS_INTEGER DEFAULT NULL)
 RETURN num_bytes PLS_INTEGER;
```

Parameters

Table 178–21 WRITE_RAW Function Parameters

Parameter	Description
c	The TCP connection to send data to.
data	The buffer containing the data to be sent.
len	The number of bytes of data to transmit. When len is NULL, the whole length of data is written.

Return Values

Table 178–22 WRITE_RAW Function Return Values

Parameter	Description
num_bytes	The actual number of bytes of data transmitted.

Usage Notes

The connection must have already been opened through a call to OPEN_CONNECTION.

Related Functions

WRITE_TEXT, WRITE_LINE, FLUSH

WRITE_TEXT Function

This function transmits a text message to a service on an open connection.

Syntax

```
UTL_TCP.WRITE_TEXT (
   c    IN OUT NOCOPY connection,
   data IN            VARCHAR2 CHARACTER SET ANY_CS,
   len  IN            PLS_INTEGER DEFAULT NULL)
 RETURN num_chars PLS_INTEGER;
```

Parameters

Table 178–23 WRITE_TEXT Function Parameters

Parameter	Description
c	The TCP connection to send data to.
data	The buffer containing the data to be sent.
len	The number of characters of data to transmit. When len is NULL, the whole length of data is written. The actual amount of data written may be less because of network condition.

Return Values

Table 178–24 WRITE_TEXT Function Return Values

Parameter	Description
num_chars	The actual number of characters of data transmitted.

Usage Notes

The connection must have already been opened through a call to OPEN_CONNECTION. Text messages will be converted to the on-the-wire character set, specified when the connection was opened, before they are transmitted on the wire.

Related Functions

WRITE_RAW, WRITE_LINE, FLUSH

179

UTL_URL

The UTL_URL package has two functions: ESCAPE and UNESCAPE.

See Also: Chapter 168, "UTL_HTTP"

This chapter contains the following topics:

- Using UTL_URL
  - Overview
  - Exceptions
  - Examples
- Summary of UTL_URL Subprograms

Using UTL_URL

Overview

A Uniform Resource Locator (URL) is a string that identifies a Web resource, such as a page or a picture. Use a URL to access such resources by way of the HyperText Transfer Protocol (HTTP). For example, the URL for Oracle's Web site is:

```
http://www.oracle.com
```

Normally, a URL contains English alphabetic characters, digits, and punctuation symbols. These characters are known as the *unreserved characters*. Any other characters in URLs, including multibyte characters or binary octet codes, must be escaped to be accurately processed by Web browsers or Web servers. Some punctuation characters, such as dollar sign ($), question mark (?), colon (:), and equals sign (=), are reserved as delimiters in a URL. They are known as the *reserved characters*. To literally process these characters, instead of treating them as delimiters, they must be escaped.

The unreserved characters are:

- A through Z, a through z, and 0 through 9
- Hyphen (-), underscore (_), period (.), exclamation point (!), tilde (~), asterisk (*), accent ('), left parenthesis (( ), right parenthesis ( ))

The reserved characters are:

- Semi-colon (;) slash (/), question mark (?), colon (:), at sign (@), ampersand (&), equals sign (=), plus sign (+), dollar sign ($), and comma (,)

The UTL_URL package has two functions that provide escape and unescape mechanisms for URL characters. Use the escape function to escape a URL before the URL is used fetch a Web page by way of the UTL_HTTP package. Use the unescape function to unescape an escaped URL before information is extracted from the URL.

For more information, refer to the Request For Comments (RFC) document RFC2396. Note that this URL escape and unescape mechanism is different from the x-www-form-urlencoded encoding mechanism described in the HTML specification:

```
http://www.w3.org/TR/html
```

Exceptions

Table 179–1 lists the exceptions that can be raised when the UTL_URL package API is invoked.

Table 179–1 UTL_URL Exceptions

Exception	Error Code	Reason
BAD_URL	29262	The URL contains badly formed escape code sequences
BAD_FIXED_WIDTH_CHARSET	29274	Fixed-width multibyte character set is not allowed as a URL character set.

Examples

You can implement the `x-www-form-urlencoded` encoding using the `UTL_URL.ESCAPE` function as follows:

```
CREATE OR REPLACE FUNCTION form_url_encode (
   data    IN VARCHAR2,
   charset IN VARCHAR2) RETURN VARCHAR2 AS
BEGIN
  RETURN utl_url.escape(data, TRUE, charset); -- note use of TRUE
END;
```

For decoding data encoded with the `form-URL-encode scheme`, the following function implements the decording scheme:

```
CREATE OR REPLACE FUNCTION form_url_decode(
   data    IN VARCHAR2,
   charset IN VARCHAR2) RETURN VARCHAR2 AS
BEGIN
  RETURN utl_url.unescape(
     replace(data, '+', ' '),
     charset);
END;
```

Summary of UTL_URL Subprograms

Table 179–2 UTL_URL Package Subprograms

Subprogram	Description
ESCAPE Function on page 179-7	Returns a URL with illegal characters (and optionally reserved characters) escaped using the `%2-digit-hex-code` format
UNESCAPE Function on page 179-9	Unescapes the escape character sequences to their original forms in a URL. Convert the `%XX` escape character sequences to the original characters

ESCAPE Function

This function returns a URL with illegal characters (and optionally reserved characters) escaped using the `%2-digit-hex-code` format.

Syntax

```
UTL_URL.ESCAPE (
   url                  IN VARCHAR2 CHARACTER SET ANY_CS,
   escape_reserved_chars IN BOOLEAN  DEFAULT  FALSE,
   url_charset          IN VARCHAR2 DEFAULT  utl_http.body_charset)
 RETURN VARCHAR2;
```

Parameters

Table 179–3 ESCAPE Function Parameters

Parameter	Description
`url`	The original URL
`escape_reserved_chars`	Indicates whether the URL reserved characters should be escaped. If set to `TRUE`, both the reserved and illegal URL characters are escaped. Otherwise, only the illegal URL characters are escaped. The default value is `FALSE`.
`url_charset`	When escaping a character (single-byte or multibyte), determine the target character set that character should be converted to before the character is escaped in %hex-code format. If `url_charset` is `NULL`, the database charset is assumed and no character set conversion will occur. The default value is the current default body character set of the `UTL_HTTP` package, whose default value is `ISO-8859-1`. The character set can be named in Internet Assigned Numbers Authority (IANA) or in the Oracle naming convention.

Usage Notes

Use this function to escape URLs that contain illegal characters as defined in the URL specification RFC 2396. The legal characters in URLs are:

- `A` through `Z`, `a` through `z`, and `0` through `9`
- Hyphen (`-`), underscore (_), period (`.`), exclamation point (`!`), tilde (`~`), asterisk (`*`), accent (`'`), left parenthesis (`(`), right parenthesis (`)`)

The reserved characters consist of:

- Semi-colon (`;`) slash (`/`), question mark (`?`), colon (`:`), at sign (`@`), ampersand (`&`), equals sign (`=`), plus sign (`+`), dollar sign (`$`), and comma (`,`)

Many of the reserved characters are used as delimiters in the URL. You should escape characters beyond those listed here by using escape_url. Also, to use the reserved characters in the name-value pairs of the query string of a URL, those characters must be escaped separately. An escape_url cannot recognize the need to escape those characters because once inside a URL, those characters become indistinguishable from the actual delimiters. For example, to pass a name-value pair `$logon=scott/tiger` into the query string of a URL, escape the `$` and `/` separately as `%24logon=scott%2Ftiger` and use it in the URL.

Normally, you will escape the entire URL, which contains the reserved characters (delimiters) that should not be escaped. For example:

```
utl_url.escape('http://www.acme.com/a url with space.html')
```

Returns:

```
http://foo.com/a%20url%20with%20space.html
```

In other situations, you may want to send a query string with a value that contains reserved characters. In that case, escape only the value fully (with `escape_reserved_chars` set to `TRUE`) and then concatenate it with the rest of the URL. For example:

```
url := 'http://www.acme.com/search?check=' || utl_url.escape
('Is the use of the "$" sign okay?', TRUE);
```

This expression escapes the question mark (`?`), dollar sign (`$`), and space characters in `'Is the use of the "$" sign okay?'` but not the `?` after `search` in the URL that denotes the use of a query string.

The Web server that you intend to fetch Web pages from may use a character set that is different from that of your database. In that case, specify the url_charset as the Web server character set so that the characters that need to be escaped are escaped in the target character set. For example, a user of an EBCDIC database who wants to access an ASCII Web server should escape the URL using `US7ASCII` so that a space is escaped as `%20` (hex code of a space in ASCII) instead of `%40` (hex code of a space in EBCDIC).

This function does not validate a URL for the proper URL format.

UNESCAPE Function

This function unescapes the escape character sequences to its original form in a URL, to convert the `%XX` escape character sequences to the original characters.

Syntax

```
UTL_URL.UNESCAPE (
   url            IN VARCHAR2 CHARACTER SET ANY_CS,
   url_charset    IN VARCHAR2 DEFAULT utl_http.body_charset)
                  RETURN VARCHAR2;
```

Parameters

Table 179–4 UNESCAPE Function Parameters

Parameter	Description
`url`	The URL to unescape
`url_charset`	After a character is unescaped, the character is assumed to be in the `source_charset` character set and it will be converted from the `source_charset` to the database character set before the URL is returned. If `source_charset` is `NULL`, the database charset is assumed and no character set conversion occurred. The default value is the current default body character set of the `UTL_HTTP` package, whose default value is "ISO-8859-1". The character set can be named in Internet Assigned Numbers Authority (IANA) or Oracle naming convention.

Usage Notes

The Web server that you receive the URL from may use a character set that is different from that of your database. In that case, specify the url_charset as the Web server character set so that the characters that need to be unescaped are unescaped in the source character set. For example, a user of an EBCDIC database who receives a URL from an ASCII Web server should unescape the URL using `US7ASCII` so that `%20` is unescaped as a space (0x20 is the hex code of a space in ASCII) instead of a `?` (because 0x20 is not a valid character in EBCDIC).

This function does not validate a URL for the proper URL format.

180

WPG_DOCLOAD

The WPG_DOCLOAD package provides an interface to download files, BLOBs and BFILEs.

> **See Also:** For more information about implementation of this package:
>
> - *Oracle HTTP Server Administrator's Guide*
> - *Oracle Application Server mod_plsql User's Guide*

The chapter contains the following topics:

Using WPG_DOCLOAD

- Constants

Constants

NAME_COL_LEN

The `NAME` column in your document table must be the same as the value of `name_col_len`.

```
name_col_len CONSTANT pls_integer := 64;
```

MIMET_COL_LEN

The `MIME_TYPE` column in your document table must be the same as the value of `mimet_col_len`.

```
mimet_col_len CONSTANT pls_integer := 48;
```

MAX_DOCTABLE_NAME_LEN

The name length of your document table must be less than `max_doctable_name_len`.

```
max_doctable_name_len CONSTANT pls_integer := 256;
```

Summary of WPG_DOCLOAD Subprograms

Table 180–1 WPG_DOCLOAD Package Subprograms

Subprogram	Description
DOWNLOAD_FILE Procedures on page 180-5	Downloads files, `BLOBS` and `BFILES`

DOWNLOAD_FILE Procedures

There are three versions of this procedure:

- The first version downloads files and is invoked from within a document download procedure to signal the PL/SQL Gateway that `p_filename` is to be downloaded from the document table to the client's browser.
- The second version can be called from within any procedure to signal the PL/SQL Gateway that `p_blob` is to be downloaded to the client's browser.
- The third version can be called from within any procedure to signal the PL/SQL Gateway that `p_bfile` is to be downloaded to the client's browser.

Syntax

```
WPG_DOCLOAD.DOWNLOAD_FILE(
   p_filename      IN             VARCHAR2,
   p_bcaching      IN             BOOLEAN DEFAULT TRUE);


WPG_DOCLOAD.DOWNLOAD_FILE(
   p_blob          IN OUT NOCOPY  BLOB);

WPG_DOCLOAD.DOWNLOAD_FILE(
   p_bfile         IN OUT         BFILE);
```

Parameters

Table 180–2 DOWNLOAD_FILE Procedure Parameters

Parameter	Description
`p_filename`	The file to download from the document table.
`p_blob`	The `BLOB` to download.
`p_bfile`	The `BFILE` to download (see Usage Notes).
`p_bcaching`	Whether browser caching is enabled (see Usage Notes).

Usage Notes

- Normally, a document will be downloaded to the browser unless the browser sends an 'If-Modified-Since' header to the gateway indicating that it has the requested document in its cache. In that case, the gateway will determine if the browser's cached copy is up to date, and if it is, it will send an HTTP 304 status message to the browser indicating that the browser should display the cached copy. However, because a document URL and a document do not necessarily have a one-to-one relationship in the PL/SQL Web Gateway, in some cases it may be undesirable to have the cached copy of a document displayed. In those cases, the `p_bcaching` parameter should be set to FALSE to indicate to the gateway to ignore the 'If-Modified-Since' header, and download the document.
- `p_bfile` and `p_blob` are declared as `IN OUT` because the locator is initially opened to check for file accessibility and existence. The open operation can only be performed if the locator is writable and readable.

181

ANYDATA TYPE

An ANYDATA TYPE contains an instance of a given type, plus a description of the type. In this sense, an ANYDATA is self-describing. An ANYDATA can be persistently stored in the database.

This chapter contains the following topics:

Using ANYDATA TYPE

Restrictions

Persistent storage of `ANYDATA` instances whose type contains embedded LOBs other than `BFILE`s is not currently supported.

Operational Notes

Construction

There are 2 ways to construct an ANYDATA. The CONVERT* calls enable construction of the ANYDATA in its entirety with a single call. They serve as explicit CAST functions from any type in the Oracle ORDBMS to ANYDATA.

```
STATIC FUNCTION ConvertBDouble(dbl IN BINARY_DOUBLE) return ANYDATA,
STATIC FUNCTION ConvertBfile(b IN BFILE) RETURN ANYDATA,
STATIC FUNCTION ConvertBFloat(fl IN BINARY_FLOAT) return ANYDATA,
STATIC FUNCTION ConvertBlob(b IN BLOB) RETURN ANYDATA,
STATIC FUNCTION ConvertChar(c IN CHAR) RETURN ANYDATA,
STATIC FUNCTION ConvertClob(c IN CLOB) RETURN ANYDATA,
STATIC FUNCTION ConvertCollection(col IN "collection_type") RETURN ANYDATA,
STATIC FUNCTION ConvertDate(dat IN DATE) RETURN ANYDATA,
STATIC FUNCTION ConvertIntervalDS(inv IN INTERVAL DAY TO SECOND) return ANYDATA,
STATIC FUNCTION ConvertIntervalYM(invIN INTERVAL YEAR TO MONTH) return ANYDATA,
STATIC FUNCTION ConvertNchar(nc IN NCHAR) return ANYDATA,
STATIC FUNCTION ConvertNClob(nc IN NCLOB) return ANYDATA,
STATIC FUNCTION ConvertNumber(num IN NUMBER) RETURN ANYDATA,
STATIC FUNCTION ConvertNVarchar2(nc IN NVARCHAR2) return ANYDATA,
STATIC FUNCTION ConvertObject(obj IN "<object_type>") RETURN ANYDATA,
STATIC FUNCTION ConvertRaw(r IN RAW) RETURN ANYDATA,
STATIC FUNCTION ConvertRef(rf IN REF "<object_type>") RETURN ANYDATA,
STATIC FUNCTION ConvertTimestamp(ts IN TIMESTAMP) return ANYDATA,
STATIC FUNCTION ConvertTimestampTZ(ts IN TIMESTAMP WITH TIMEZONE) return ANYDATA,
STATIC FUNCTION ConvertTimestampLTZ(ts IN TIMESTAMP WITH LOCAL TIMEZONE) return
ANYDATA,
STATIC FUNCTION ConvertURowid(rid IN UROWID) return ANYDATA,
STATIC FUNCTION ConvertVarchar(c IN VARCHAR) RETURN ANYDATA,
STATIC FUNCTION ConvertVarchar2(c IN VARCHAR2) RETURN ANYDATA,
```

The second way to construct an ANYDATA is a piece by piece approach. The BEGINCREATE Static Procedure call begins the construction process and ENDCREATE Member Procedure call finishes the construction process. In between these two calls, the individual attributes of an object type or the elements of a collection can be set using SET* calls. For piece by piece access of the attributes of objects and elements of collections, the PIECEWISE Member Procedure should be invoked prior to GET* calls.

Note: The ANYDATA has to be constructed or accessed sequentially starting from its first attribute (or collection element). The BEGINCREATE call automatically begins the construction in a piece-wise mode. There is no need to call PIECEWISE immediately after BEGINCREATE. ENDCREATE should be called to finish the construction process (before which any access calls can be made).

Access

Access functions are available based on SQL. These functions do not throw exceptions on type-mismatch. Instead, they return NULL if the type of the ANYDATA does not correspond to the type of access. If you wish to use only ANYDATA functions of the appropriate types returned in a query, you should use a WHERE clause which uses GETTYPENAME and choose the type you are interested in (say "SYS.NUMBER"). Each of

these functions returns the value of a specified datatype inside a SYS.ANYDATA wrapper.

```
MEMBER FUNCTION AccessBDouble(self IN ANYDATA) return BINARY_DOUBLE
   DETERMINISTIC,
MEMBER FUNCTION AccessBfile(self IN ANYDATA) return BFILE,
MEMBER FUNCTION AccessBFloat(self IN ANYDATA) return BINARY_FLOAT
   DETERMINISTIC,
MEMBER FUNCTION AccessBlob(self IN ANYDATA) return BLOB,
MEMBER FUNCTION AccessChar(self IN ANYDATA) return CHAR,
MEMBER FUNCTION AccessClob(self IN ANYDATA) return CLOB,
MEMBER FUNCTION AccessDate(self IN ANYDATA) return DATE,
MEMBER FUNCTION AccessIntervalYM(self IN ANYDATA) return INTERVAL YEAR TO MONTH,
MEMBER FUNCTION AccessIntervalDS(self IN ANYDATA) return INTERVAL DAY TO SECOND,
MEMBER FUNCTION AccessNchar(self IN ANYDATA) return NCHAR,
MEMBER FUNCTION AccessNClob(self IN ANYDATA) return NCLOB
MEMBER FUNCTION AccessNumber(self IN ANYDATA) return NUMBER,
MEMBER FUNCTION AccessNVarchar2(self IN ANYDATA) return NVARCHAR2,
MEMBER FUNCTION AccessRaw(self IN ANYDATA) return RAW,
MEMBER FUNCTION AccessTimestamp(self IN ANYDATA) return TIMESTAMP,
MEMBER FUNCTION AccessTimestampLTZ(self IN ANYDATA) return TIMESTAMP WITH LOCAL
   TIMEZONE,
MEMBER FUNCTION AccessTimestampTZ(self IN ANYDATA) return TIMESTAMP WITH
   TIMEZONE,
MEMBER FUNCTION AccessURowid(self IN ANYDATA) return UROWID DETERMINISTIC
MEMBER FUNCTION AccessVarchar(self IN ANYDATA) return VARCHAR,
MEMBER FUNCTION AccessVarchar2(self IN ANYDATA) return VARCHAR2,
```

Summary of ANYDATA Subprograms

Table 181–1 ANYDATA Type Subprograms

Subprogram	Description
BEGINCREATE Static Procedure on page 181-7	Begins creation process on a new `ANYDATA`
ENDCREATE Member Procedure on page 181-8	Ends creation of an `ANYDATA`
GET* Member Functions on page 181-9	Gets the current data value (which should be of appropriate type)
GETTYPE Member Function on page 181-12	Gets the Type of the `ANYDATA`
GETTYPENAME Member Function on page 181-13	Get the fully qualified type name for the `ANYDATA`
PIECEWISE Member Procedure on page 181-14	Sets the `MODE` of access of the current data value to be an attribute at a time (if the data value is of `TYPECODE_OBJECT`)
SET* Member Procedures on page 181-15	Sets the current data value.

BEGINCREATE Static Procedure

This procedure begins the creation process on a new ANYDATA.

Syntax

```
STATIC PROCEDURE BeginCreate(
   dtype          IN OUT NOCOPY AnyType,
   adata          OUT NOCOPY ANYDATA);
```

Parameters

Table 181–2 BEGINCREATE Procedure Parameters

Parameter	Description
dtype	The type of the ANYDATA. (Should correspond to OCI_TYPECODE_OBJECT or a Collection typecode.)
adata	ANYDATA being constructed.

Exception

DBMS_TYPES.INVALID_PARAMETERS: dtype is invalid (not fully constructed, and similar deficits.)

Usage Notes

There is no need to call PIECEWISE immediately after this call. The construction process begins in a piece-wise manner automatically.

ENDCREATE Member Procedure

This procedure ends creation of an ANYDATA. Other creation functions cannot be called after this call.

Syntax

```
MEMBER PROCEDURE EndCreate(
   self         IN OUT NOCOPY ANYDATA);
```

Parameters

Table 181–3 ENDCREATE Procedure Parameter

Parameter	Description
self	An ANYDATA.

GET* Member Functions

These functions get the current data value (which should be of appropriate type).

The type of the current data value depends on the MODE with which we are accessing (depending on whether we have invoked the PIECEWISE call).

If PIECEWISE has NOT been called, we are accessing the ANYDATA in its entirety and the type of the data value should match the type of the ANYDATA.

If PIECEWISE has been called, we are accessing the ANYDATA piece-wise. The type of the data value should match the type of the attribute (or collection element) at the current position.

Syntax

```
MEMBER FUNCTION GetBDouble(
   self         IN ANYDATA,
   dbl          OUT NOCOPY BINARY_DOUBLE)
RETURN PLS_INTEGER;

MEMBER FUNCTION GetBfile(
   self         IN ANYDATA,
   b            OUT NOCOPY BFILE)
   RETURN       PLS_INTEGER;

MEMBER FUNCTION GetBFloat(
   self         IN ANYDATA,
   fl           OUT NOCOPY BINARY_FLOAT)
RETURN PLS_INTEGER;

MEMBER FUNCTION GetBlob(
   self         IN ANYDATA,
   b            OUT NOCOPY BLOB)
   RETURN       PLS_INTEGER;

MEMBER FUNCTION GetChar(
   self         IN ANYDATA,
   c            OUT NOCOPY CHAR)
   RETURN       PLS_INTEGER;

MEMBER FUNCTION GetClob(
   self         IN ANYDATA,
   c            OUT NOCOPY CLOB)
   RETURN       PLS_INTEGER;

MEMBER FUNCTION GetCollection(
   self         IN ANYDATA,
   col          OUT NOCOPY "<collection_type>")
   RETURN       PLS_INTEGER;

MEMBER FUNCTION GetDate(
   self         IN ANYDATA,
   dat          OUT NOCOPY DATE)
   RETURN       PLS_INTEGER;

MEMBER FUNCTION GetIntervalDS(
   self         IN ANYDATA,
   inv          OUT NOCOPY INTERVAL DAY TO SECOND)
 RETURN         PLS_INTEGER;
```

```
MEMBER FUNCTION GetIntervalYM(
   self         IN ANYDATA,
   inv          OUT NOCOPY INTERVAL YEAR TO MONTH)
 RETURN         PLS_INTEGER;

MEMBER FUNCTION GetNchar(
   self         IN ANYDATA,
   nc           OUT NOCOPY NCHAR)
 RETURN         PLS_INTEGER;

MEMBER FUNCTION GetNClob(
   self         IN ANYDATA,
   nc           OUT NOCOPY NCLOB)
 RETURN         PLS_INTEGER;

MEMBER FUNCTION GetNumber(
   self         IN ANYDATA,
   num          OUT NOCOPY NUMBER)
   RETURN       PLS_INTEGER;

MEMBER FUNCTION GetNVarchar2(
   self         IN ANYDATA,
   nc           OUT NOCOPY NVARCHAR2)
 RETURN         PLS_INTEGER;

MEMBER FUNCTION GetObject(
   self         IN ANYDATA,
   obj          OUT NOCOPY "<object_type>")
   RETURN       PLS_INTEGER;

MEMBER FUNCTION GetRaw(
   self         IN ANYDATA,
   r            OUT NOCOPY RAW)
   RETURN       PLS_INTEGER;

MMEMBER FUNCTION GetRef(
   self         IN ANYDATA,
   rf           OUT NOCOPY REF "<object_type>")
   RETURN       PLS_INTEGER;

MEMBER FUNCTION GetTimestamp(
   self         IN ANYDATA,
   ts           OUT NOCOPY TIMESTAMP)
 RETURN PLS_INTEGER;

MEMBER FUNCTION GetTimestampTZ(
   self         IN ANYDATA,
   ts           OUT NOCOPY TIMESTAMP WITH TIME ZONE)
 RETURN PLS_INTEGER;

MEMBER FUNCTION GetTimestampLTZ(
   self         IN ANYDATA,
   ts           OUT NOCOPY TIMESTAMP WITH LOCAL TIME ZONE)
 RETURN PLS_INTEGER;

MEMBER FUNCTION GetVarchar(
   self         IN ANYDATA,
   c            OUT NOCOPY VARCHAR)
   RETURN       PLS_INTEGER;
```

```
MEMBER FUNCTION GetVarchar2(
   self         IN ANYDATA,
   c            OUT NOCOPY VARCHAR2)
   RETURN       PLS_INTEGER;
```

Parameters

Table 181–4 GET* Function Parameter

Parameter	Description
self	An ANYDATA.
num	The number to be obtained.

Return Values

DBMS_TYPES.SUCCESS or DBMS_TYPES.NO_DATA

The return value is relevant only if PIECEWISE has been already called (for a collection). In such a case, DBMS_TYPES.NO_DATA signifies the end of the collection when all elements have been accessed.

Exceptions

DBMS_TYPES.TYPE_MISMATCH: When the expected type is different from the passed in type.

DBMS_TYPES.INVALID_PARAMETERS: Invalid Parameters (if it is not appropriate to add a number at this point in the creation process).

DBMS_TYPES.INCORRECT_USAGE: Incorrect usage.

GETTYPE Member Function

This function gets the typecode of the ANYDATA.

Syntax

```
MEMBER FUNCTION GETTYPE(
   self          IN ANYDATA,
   typ           OUT NOCOPY AnyType)
   RETURN        PLS_INTEGER;
```

Parameters

Table 181–5 GETTYPE Function Parameter

Parameter	Description
self	An ANYDATA.
typ	The AnyType corresponding to the ANYDATA. May be NULL if it does not represent a user-defined type.

Return Values

The typecode corresponding to the type of the ANYDATA.

GETTYPENAME Member Function

This function gets the fully qualified type name for the ANYDATA.

If the ANYDATA is based on a built-in type, this function will return NUMBER and other relevant information.

If it is based on a user defined type, this function will return *schema_name.type_name*, for example, SCOTT.FOO.

If it is based on a transient anonymous type, this function will return NULL.

Syntax

```
MEMBER FUNCTION GETTYPENAME(
   self         IN ANYDATA)
   RETURN       VARCHAR2;
```

Parameters

Table 181–6 GETTYPENAME Function Parameter

Parameter	Description
self	An ANYDATA.

Return Values

Type name of the ANYDATA.

PIECEWISE Member Procedure

This procedure sets the MODE of access of the current data value to be an attribute at a time (if the data value is of TYPECODE_OBJECT).

It sets the MODE of access of the data value to be a collection element at a time (if the data value is of collection type). Once this call has been made, subsequent calls to SET* and GET* will sequentially obtain individual attributes or collection elements.

Syntax

```
MEMBER PROCEDURE PIECEWISE(
   self         IN OUT NOCOPY ANYDATA);
```

Parameters

Table 181–7 PIECEWISE Procedure Parameters

Parameter	Description
self	The current data value.

Exceptions

- DBMS_TYPES.INVALID_PARAMETERS
- DBMS_TYPES.INCORRECT_USAGE: On incorrect usage.

Usage Notes

The current data value must be of an OBJECT or COLLECTION type before this call can be made.

Piece-wise construction and access of nested attributes that are of object or collection types is not supported.

SET* Member Procedures

Sets the current data value.

This is a list of procedures that should be called depending on the type of the current data value. The type of the data value should be the type of the attribute at the current position during the piece-wise construction process.

Syntax

```
MEMBER PROCEDURE SETBDOUBLE(
   self        IN OUT NOCOPY ANYDATA,
   dbl         IN BINARY_DOUBLE,
   last_elem   IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETBFILE(
   self        IN OUT NOCOPY ANYDATA,
   b           IN BFILE,
   last_elem   IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETBFLOAT(
   self        IN OUT NOCOPY ANYDATA,
   fl          IN            BINARY_FLOAT,
   last_elem   IN            boolean DEFAULT FALSE);

MEMBER PROCEDURE SETBLOB(
   self        IN OUT NOCOPY ANYDATA,
   b           IN BLOB,
   last_elem   IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETCHAR(
   self        IN OUT NOCOPY ANYDATA,
   c           IN CHAR,
   last_elem   IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETCLOB(
   self        IN OUT NOCOPY ANYDATA,
   c           IN CLOB,
   last_elem   IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETCOLLECTION(
   self        IN OUT NOCOPY ANYDATA,
   col         IN "<collectyion_type>",
   last_elem   IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETDATE(
   self        IN OUT NOCOPY ANYDATA,
   dat         IN DATE,
   last_elem   IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETINTERVALDS(
   self        IN OUT NOCOPY ANYDATA,
   inv         IN INTERVAL DAY TO SECOND,
   last_elem IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETINTERVALYM(
   self        IN OUT NOCOPY ANYDATA,
   inv         IN INTERVAL YEAR TO MONTH,
   last_elem   IN boolean DEFAULT FALSE);
```

```
MEMBER PROCEDURE SETNCHAR(
   self        IN OUT NOCOPY ANYDATA,
   nc          IN NCHAR,
   last_elem   IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETNCLOB(
   self        IN OUT NOCOPY ANYDATA,
   nc          IN NClob,
   last_elem   IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETNUMBER(
   self        IN OUT NOCOPY ANYDATA,
   num         IN NUMBER,
   last_elem   IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETNVARCHAR2(
   self        IN OUT NOCOPY ANYDATA,
   nc          IN NVarchar2,
   last_elem   IN boolean DEFAULT FALSE),

MEMBER PROCEDURE SETOBJECT(
   self        IN OUT NOCOPY ANYDATA,
   obj         IN "<object_type>",
   last_elem   IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETRAW(
   self        IN OUT NOCOPY ANYDATA,
   r           IN RAW,
   last_elem   IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETREF(
   self        IN OUT NOCOPY ANYDATA,
   rf          IN REF "<object_type>",
   last_elem   IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETTIMESTAMP(
   self       IN OUT NOCOPY ANYDATA,
   ts         IN TIMESTAMP,
   last_elem  IN BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETTIMESTAMPTZ(self IN OUT NOCOPY ANYDATA,
   ts          IN TIMESTAMP WITH TIME ZONE,
   last_elem   IN BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETTIMESTAMPLTZ(
   self IN OUT NOCOPY ANYDATA,
   ts IN TIMESTAMP WITH LOCAL TIME ZONE,
   last_elem IN boolean DEFAULT FALSE),

MEMBER PROCEDURE SETVARCHAR(
   self        IN OUT NOCOPY ANYDATA,
   c           IN VARCHAR,
   last_elem   IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETVARCHAR2(
   self        IN OUT NOCOPY ANYDATA,
   c           IN VARCHAR2,
   last_elem   IN boolean DEFAULT FALSE);
```

Parameters

Table 181–8 SET Procedure Parameters*

Parameter	Description
self	An ANYDATA.
num	The number, and associated information, that is to be set.
last_elem	Relevant only if ANYDATA represents a collection. Set to TRUE if it is the last element of the collection, FALSE otherwise.

Exceptions

- DBMS_TYPES.INVALID_PARAMETERS: Invalid Parameters (if it is not appropriate to add a number at this point in the creation process).
- DBMS_TYPES.INCORRECT_USAGE: Incorrect usage.
- DBMS_TYPES.TYPE_MISMATCH: When the expected type is different from the passed in type.

Usage Notes

When BEGINCREATE is called, construction has already begun in a piece-wise fashion. Subsequent calls to SET* will set the successive attribute values.

If the ANYDATA is a standalone collection, the SET* call will set the successive collection elements.

182

ANYDATASET TYPE

An ANYDATASET TYPE contains a description of a given type plus a set of data instances of that type. An ANYDATASET can be persistently stored in the database if desired, or it can be used as interface parameters to communicate self-descriptive sets of data, all of which belong to a certain type.

This chapter contains the following topics:

Construction

The ANYDATASET needs to be constructed value by value, sequentially.

For each data instance (of the type of the ANYDATASET), the ADDINSTANCE function must be invoked. This adds a new data instance to the ANYDATASET. Subsequently, SET* can be called to set each value in its entirety.

The MODE of construction/access can be changed to attribute/collection element wise by making calls to PIECEWISE.

- If the type of the ANYDATASET is TYPECODE_OBJECT, individual attributes will be set with subsequent SET* calls. Likewise on access.
- If the type of the current data value is a collection type individual collection elements will be set with subsequent SET* calls. Likewise on access. This call is very similar to ANYDATA.PIECEWISE call defined for the type ANYDATA.

Note that there is no support for piece-wise construction and access of nested (not top level) attributes that are of object types or collection types.

ENDCREATE should be called to finish the construction process (before which no access calls can be made).

Summary of ANYDATASET TYPE Subprograms

Table 182–1 ANYDATASET Type Subprograms

Subprogram	Description
ADDINSTANCE Member Procedure on page 182-4	Adds a new data instance to an ANYDATASET.
BEGINCREATE Static Procedure on page 182-5	Creates a new ANYDATASET which can be used to create a set of data values of the given ANYTYPE.
ENDCREATE Member Procedure on page 182-6	Ends Creation of a ANYDATASET. Other creation functions cannot be called after this call.
GET* Member Functions on page 182-7	Gets the current data value (which should be of appropriate type).
GETCOUNT Member Function on page 182-10	Gets the number of data instances in an ANYDATASET.
GETINSTANCE Member Function on page 182-11	Gets the next instance in an ANYDATASET.
GETTYPE Member Function on page 182-12	Gets the ANYTYPE describing the type of the data instances in an ANYDATASET. current data value (which should be of appropriate type).
GETTYPENAME Member Function on page 182-13	Gets the AnyType describing the type of the data instances in an ANYDATASET.
PIECEWISE Member Procedure on page 182-14	Sets the MODE of construction, access of the data value to be an attribute at a time (if the data value is of TYPECODE_OBJECT).
SET* Member Procedures on page 182-15	Sets the current data value.

ADDINSTANCE Member Procedure

This procedure adds a new data instance to an ANYDATASET.

Syntax

```
MEMBER PROCEDURE AddInstance(
   self          IN OUT NOCOPY ANYDATASET);
```

Parameters

Table 182–2 ADDINSTANCE Procedure Parameter

Parameter	Description
self	The ANYDATASET being constructed.

Exceptions

```
DBMS_TYPES.invalid_parameters: Invalid parameters.
DBMS_TYPES.incorrect_usage: On incorrect usage.
```

Usage Notes

The data instances have to be added sequentially. The previous data instance must be fully constructed (or set to NULL) before a new one can be added.

This call DOES NOT automatically set the mode of construction to be piece-wise. The user has to explicitly call PIECEWISE if a piece-wise construction of the instance is intended.

BEGINCREATE Static Procedure

This procedure creates a new ANYDATASET which can be used to create a set of data values of the given ANYTYPE.

Syntax

```
STATIC PROCEDURE BeginCreate(
   typecode     IN PLS_INTEGER,
   rtype        IN OUT NOCOPY AnyType,
   aset         OUT NOCOPY ANYDATASET);
```

Parameters

Table 182–3 BEGINCREATE Procedure Parameter

Parameter	Description
typecode	The typecode for the type of the ANYDATASET.
dtype	The type of the data values. This parameter is a must for user-defined types like TYPECODE_OBJECT, Collection typecodes, and similar others.
aset	The ANYDATASET being constructed.

Exceptions

DBMS_TYPES.invalid_parameters: dtype is invalid (not fully constructed, and like errors.)

ENDCREATE Member Procedure

This procedure ends Creation of a ANYDATASET. Other creation functions cannot be called after this call.

Syntax

```
MEMBER PROCEDURE ENDCREATE(
   self              IN OUT NOCOPY ANYDATASET);
```

Parameters

Table 182–4 ENDCREATE Procedure Parameter

Parameter	Description
self	The ANYDATASET being constructed.

GET* Member Functions

These functions get the current data value (which should be of appropriate type).

The type of the current data value depends on the MODE with which you are accessing it (depending on how we have invoked the PIECEWISE call). If PIECEWISE has not been called, we are accessing the instance in its entirety and the type of the data value should match the type of the ANYDATASET.

If PIECEWISE has been called, we are accessing the instance piece-wise. The type of the data value should match the type of the attribute (or collection element) at the current position.

Syntax

```
MEMBER FUNCTION GETBDOUBLE(
   self        IN ANYDATASET,
   dbl         OUT NOCOPY BINARY_DOUBLE)
   RETURN PLS_INTEGER;

MEMBER FUNCTION GETBFLOAT(
   self        IN ANYDATASET,
   fl          OUT NOCOPY BINARY_FLOAT)
   RETURN PLS_INTEGER;

MEMBER FUNCTION GETBFILE(
   self        IN ANYDATASET,
   b           OUT NOCOPY BFILE)
   RETURN      PLS_INTEGER;

MEMBER FUNCTION GETBLOB(
   self        IN ANYDATASET,
   b           OUT NOCOPY BLOB)
   RETURN      PLS_INTEGER;

MEMBER FUNCTION GETCHAR(
   self        IN ANYDATASET,
   c           OUT NOCOPY CHAR)
   RETURN      PLS_INTEGER;

MEMBER FUNCTION GETCLOB(
   self        IN ANYDATASET,
   c           OUT NOCOPY CLOB)
   RETURN      PLS_INTEGER;

MEMBER FUNCTION GETCOLLECTION(
   self        IN ANYDATASET,
   col         OUT NOCOPY "<collection_type>")
   RETURN      PLS_INTEGER;

MEMBER FUNCTION GETDATE(
   self        IN ANYDATASET,
   dat         OUT NOCOPY DATE)
   RETURN      PLS_INTEGER;

MEMBER FUNCTION GETINTERVALDS(
   self        IN ANYDATASET,
   inv         IN OUT NOCOPY INTERVAL DAY TO SECOND)
 RETURN PLS_INTEGER;
```

```
MEMBER FUNCTION GETINTERVALYM(
   self        IN ANYDATASET,
   inv IN OUT NOCOPY INTERVAL YEAR TO MONTH)
 RETURN PLS_INTEGER;

MEMBER FUNCTION GETNCHAR(
   self        IN ANYDATASET,
   nc          OUT NOCOPY NCHAR)
 RETURN PLS_INTEGER;

MEMBER FUNCTION GETNCLOB(
   self        IN ANYDATASET,
   nc          OUT NOCOPY NCLOB)
 RETURN PLS_INTEGER;

MEMBER FUNCTION GETNUMBER(
   self        IN ANYDATASET,
   num         OUT NOCOPY NUMBER)
   RETURN      PLS_INTEGER;

MEMBER FUNCTION GETNVARCHAR2(
   self        IN ANYDATASET,
   nc          OUT NOCOPY NVARCHAR2)
 RETURN PLS_INTEGER;

MEMBER FUNCTION GETOBJECT(
   self        IN ANYDATASET,
   obj         OUT NOCOPY "<object_type>")
   RETURN      PLS_INTEGER;

MEMBER FUNCTION GETRAW(
   self        IN ANYDATASET,
   r           OUT NOCOPY RAW)
   RETURN      PLS_INTEGER;

MEMBER FUNCTION GETREF(
   self        IN ANYDATASET,
   rf          OUT NOCOPY REF "<object_type>")
   RETURN      PLS_INTEGER;

MEMBER FUNCTION GETTIMESTAMP(
   self        IN ANYDATASET,
 RETURN PLS_INTEGER;

MEMBER FUNCTION GETTIMESTAMPLTZ(
   self        IN ANYDATASET,
   ts          OUT NOCOPY TIMESTAMP WITH LOCAL TIME ZONE)
 RETURN PLS_INTEGER;

MEMBER FUNCTION GETTIMESTAMPTZ(
   self        IN ANYDATASET,
   ts          OUT NOCOPY TIMESTAMP WITH TIME ZONE)
 RETURN PLS_INTEGER,

MEMBER FUNCTION GETUROWID(
   self        IN ANYDATASET,
   rid         OUT NOCOPY UROWID)
 RETURN PLS_INTEGER
```

```
MEMBER FUNCTION GETVARCHAR(
   self        IN ANYDATASET,
   c           OUT NOCOPY VARCHAR)
   RETURN      PLS_INTEGER;

MEMBER FUNCTION GETVARCHAR2(
   self        IN ANYDATASET,
   c           OUT NOCOPY VARCHAR2)
   RETURN      PLS_INTEGER;
```

Parameters

Table 182–5 GET* Procedure Parameters

Parameter	Description
self	The ANYDATASET being accessed.
num	The number, and associated information., that is to be obtained.

Return Values

DBMS_TYPES.SUCCESS or DBMS_TYPES.NO_DATA

The return value is relevant only if PIECEWISE has been already called (for a collection). In such a case, DBMS_TYPES.NO_DATA signifies the end of the collection when all elements have been accessed.

Exceptions

DBMS_TYPES.INVALID_PARAMETERs: Invalid Parameters (if it is not appropriate to add a number at this point in the creation process).

DBMS_TYPES.INCORRECT_USAGE: Incorrect usage

DBMS_TYPES.TYPE_MISMATCH: When the expected type is different from the passed in type.

GETCOUNT Member Function

This function gets the number of data instances in an ANYDATASET.

Syntax

```
MEMBER FUNCTION GetCount(
   self        IN ANYDATASET)
   RETURN      PLS_INTEGER;
```

Parameter

Table 182–6 GETCOUNT Function Parameter

Parameter	Description
self	The ANYDATASET being accessed.

Return Values

The number of data instances.

GETINSTANCE Member Function

This function gets the next instance in an ANYDATASET. Only sequential access to the instances in an ANYDATASET is allowed. After this function has been called, the GET* functions can be invoked on the ANYDATASET to access the current instance. If PIECEWISE is called before doing the GET* calls, the individual attributes (or collection elements) can be accessed.

It is an error to invoke this function before the ANYDATASET is fully created.

Syntax

```
MEMBER FUNCTION GETINSTANCE(
   self           IN OUT NOCOPY ANYDATASET)
   RETURN         PLS_INTEGER;
```

Parameters

Table 182–7 GETINSTANCE Function Parameter

Parameter	Description
self	The ANYDATASET being accessed.

Return Values

DBMS_TYPES.SUCCESS or DBMS_TYPES.NO_DATA

DBMS_TYPES.NO_DATA signifies the end of the ANYDATASET (all instances have been accessed).

Usage Notes

This function should be called even before accessing the first instance.

GETTYPE Member Function

Gets the AnyType describing the type of the data instances in an ANYDATASET.

Syntax

```
MEMBER FUNCTION GETTYPE(
   self           IN ANYDATASET,
   typ            OUT NOCOPY AnyType)
   RETURN         PLS_INTEGER;
```

Parameters

Table 182–8 GETTYPE Function Parameter

Parameter	Description
self	The ANYDATASET.
typ	The ANYTYPE corresponding to the AnyData. May be NULL if it does not represent a user-defined function.

Return Values

The typecode corresponding to the type of the ANYDATA.

GETTYPENAME Member Function

This procedure gets the fully qualified type name for the ANYDATASET.

If the ANYDATASET is based on a built-in, this function will return NUMBER and associated information.

If it is based on a user defined type, this function will return *schema_name.type_name*. for example, SCOTT.FOO.

If it is based on a transient anonymous type, this function will return NULL.

Syntax

```
MEMBER FUNCTION GETTYPENAME(
   self           IN ANYDATASET)
   RETURN         VARCHAR2;
```

Parameter

Table 182–9 GETTYPENAME Function Parameter

Parameter	Description
self	The ANYDATASET being constructed.

Return Values

Type name of the ANYDATASET.

PIECEWISE Member Procedure

This procedure sets the MODE of construction, access of the data value to be an attribute at a time (if the data value is of TYPECODE_OBJECT).

It sets the MODE of construction, access of the data value to be a collection element at a time (if the data value is of a collection TYPE). Once this call has been made, subsequent SET* and GET* calls will sequentially obtain individual attributes or collection elements.

Syntax

```
MEMBER PROCEDURE PIECEWISE(
   self         IN OUT NOCOPY ANYDATASET);
```

Parameters

Table 182–10 PIECEWISE Procedure Parameter

Parameter	Description
self	The ANYDATASET being constructed.

Exceptions

DBMS_TYPES.INVALID_PARAMETERS: Invalid parameters.

DBMS_TYPES.INCORRECT_USAGE: On incorrect usage.

Usage Notes

The current data value must be of an object or collectyon type before this call can be made. There is no support for piece-wise construction or access of embedded object type attributes or nested collections.

SET* Member Procedures

This procedure sets the current data value.

The type of the current data value depends on the MODE with which we are constructing (depending on how we have invoked the PIECEWISE call). The type of the current data should be the type of the ANYDATASET if PIECEWISE has NOT been called. The type should be the type of the attribute at the current position if PIECEWISE has been called.

Syntax

```
MEMBER PROCEDURE SETBDOUBLE(
   self              IN OUT NOCOPY ANYDATASET,
   dbl               IN BINARY_DOUBLE,
   last_elem         IN BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETBFLOAT(
   self              IN OUT NOCOPY ANYDATASET,
   fl                IN BINARY_FLOAT,
   last_elem         IN BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETBFILE(
   self              IN OUT NOCOPY ANYDATASET,
   b                 IN BFILE,
   last_elem BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETBLOB(
   self              IN OUT NOCOPY ANYDATASET,
   b                 IN BLOB,
   last_elem BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETCHAR(
   self              IN OUT NOCOPY ANYDATASET,
   c                 IN CHAR,
   last_elem BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETCLOB(
   self              IN OUT NOCOPY ANYDATASET,
   c                 IN CLOB,
   last_elem BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETCOLLECTION(
   self              IN OUT NOCOPY ANYDATASET,
   col               IN "<collection_type>",
   last_elem BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETDATE(
   self              IN OUT NOCOPY ANYDATASET,
   dat               IN DATE,
   last_elem BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETINTERVALDS(
   self              IN OUT NOCOPY ANYDATASET,
   inv               IN INTERVAL DAY TO SECOND,
   last_elem         IN BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETINTERVALYM(
   self              IN OUT NOCOPY ANYDATASET,
```

```
   inv               IN INTERVAL YEAR TO MONTH,
   last_elem         IN BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETNCHAR(
   self              IN OUT NOCOPY ANYDATASET,
   nc                IN NCHAR,
   last_elem IN BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETNCLOB(

s
elf              IN OUT NOCOPY ANYDATASET,
  nc                IN NClob,
  last_elem         IN BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETNUMBER(
   self              IN OUT NOCOPY ANYDATASET,
   num               IN NUMBER,
   last_elem BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETNVARCHAR2(
   self             IN OUT NOCOPY ANYDATASET,
   nc               IN NVarchar2,
   last_elem        IN BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETOBJECT(
   self              IN OUT NOCOPY ANYDATASET,
   obj               IN "<object_type>",
   last_elem BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETRAW(
   self              IN OUT NOCOPY ANYDATASET,
   r                 IN RAW,
   last_elem BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETREF(
   self              IN OUT NOCOPY ANYDATASET,
   rf                IN REF "<object_type>",
   last_elem BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETTIMESTAMP(
   self              IN OUT NOCOPY ANYDATASET,
   ts                IN TIMESTAMP,
   last_elem IN BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETTIMESTAMPLTZ(
   self             IN OUT NOCOPY ANYDATASET,
   ts               IN TIMESTAMP WITH LOCAL TIME ZONE,
   last_elem        IN BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETTIMESTAMPTZ(
   self             IN OUT NOCOPY ANYDATASET,
   ts               IN TIMESTAMP WITH TIME ZONE,
   last_elem        IN BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETUROWID(
   self              IN OUT NOCOPY ANYDATASET,
   rid               IN UROWID,
   last_elem IN BOOLEAN DEFAULT FALSE);
```

```
MEMBER PROCEDURE SETVARCHAR(
   self              IN OUT NOCOPY ANYDATASET,
   c                 IN VARCHAR,
   last_elem BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETVARCHAR2(
   self              IN OUT NOCOPY ANYDATASET,
   c                 IN VARCHAR2,
   last_elem BOOLEAN DEFAULT FALSE);
```

Parameters

Table 182–11 SET* Procedure Parameters

Parameter	Description
`self`	The `ANYDATASET` being accessed.
`num`	The number, and associated information, that is to be set.
`last_elem`	Relevant only if `PIECEWISE` has been already called (for a collection). Set to TRUE if it is the last element of the collection, FALSE otherwise.

Exceptions

- `DBMS_TYPES.INVALID_PARAMETERS`: Invalid parameters (if it is not appropriate to add a number at this point in the creation process).
- `DBMS_TYPES.INCORRECT_USAGE`: Incorrect usage.
- `DBMS_TYPES.TYPE_MISMATCH`: When the expected type is different from the passed in type.

183

ANYTYPE TYPE

An `ANYTYPE TYPE` can contain a type description of any persistent SQL type, named or unnamed, including object types and collection types. It can also be used to construct new transient type descriptions.

New persistent types can only be created using the `CREATE TYPE` statement. Only new transient types can be constructed using the `ANYTYPE` interfaces.

This chapter discusses the following:

Summary of ANYTYPE Subprograms

Table 183–1 ANYTYPE Type Subprograms

Subprogram	Description
BEGINCREATE Static Procedure on page 183-3	Creates a new instance of `ANYTYPE` which can be used to create a transient type description.
SETINFO Member Procedure on page 183-4	Sets any additional information required for constructing a `COLLECTION` or builtin type.
ADDATTR Member Procedure on page 183-6	Adds an attribute to an `ANYTYPE` (of typecode `DBMS_TYPES.TYPECODE_OBJECT`).
ENDCREATE Member Procedure on page 183-7	Ends creation of a transient `ANYTYPE`. Other creation functions cannot be called after this call.
GETPERSISTENT Static Function on page 183-8	Returns an `ANYTYPE` corresponding to a persistent type created earlier using the `CREATE TYPE` SQL statement.
GETINFO Member Function on page 183-9	Gets the type information for the `ANYTYPE`.
GETATTRELEMINFO Member Function on page 183-10	Gets the type information for an attribute of the type (if it is of `TYPECODE_OBJECT`). Gets the type information for a collection's element type if the *self* parameter is of a collection type.

BEGINCREATE Static Procedure

This procxedure creates a new instance of ANYTYPE which can be used to create a transient type description.

Syntax

```
STATIC PROCEDURE BEGINCREATE(
   typecode       IN          PLS_INTEGER,
   atype          OUT NOCOPY  ANYTYPE);
```

Parameters

Table 183–2 BEGINCREATE Procedure Parameters

Parameter	Description
typecode	Use a constant from DBMS_TYPES package. Typecodes for user-defined type: ■ DBMS_TYPES.TYPECODE_OBJECT ■ DBMS_TYPES.TYPECODE_VARRAY or ■ DBMS_TYPES.TYPECODE_TABLE Typecodes for builtin types: ■ DBMS_TYPES.TYPECODE_NUMBER, and similar types.
atype	ANYTYPE for a transient type

SETINFO Member Procedure

This procedure sets any additional information required for constructing a COLLECTION or builtin type.

Syntax

```
MEMBER PROCEDURE SETINFO(
   self          IN OUT NOCOPY ANYTYPE,
   prec          IN PLS_INTEGER,
   scale         IN PLS_INTEGER,
   len           IN PLS_INTEGER,
   csid          IN PLS_INTEGER,
   csfrm         IN PLS_INTEGER,
   atype         IN ANYTYPE DEFAULT NULL,
   elem_tc       IN PLS_INTEGER DEFAULT NULL,
   elem_count    IN PLS_INTEGER DEFAULT 0);
```

Parameters

Table 183–3 SETINFO Procedure Parameters

Parameter	Description
self	The transient ANYTYPE that is being constructed.
prec	Optional.Required if typecode represents a NUMBER. Give precision and scale. Ignored otherwise.
scale	Optional.Required if typecode represents a NUMBER. Give precision and scale. Ignored otherwise.
len	Optional. Required if typecode represents a RAW, CHAR, VARCHAR, or VARCHAR2 type. Gives length.
csid	Required if typecode represents types requiring character information such as CHAR, VARCHAR, VARCHAR2, or CFILE.
csfrm	Required if typecode represents types requiring character information such as CHAR, VARCHAR, VARCHAR2, or CFILE.
atype	Optional. Required if collection element typecode is a user-defined type such as TYPECODE_OBJECT, and similar others.. It is also required for a built-in type that needs user-defined type information such as TYPECODE_REF. This parameter is not needed otherwise.

The Following Parameters Are Required For Collection Types

Table 183–4 SETINFO Procedure Parameters - Collection Types

Parameter	Description
elem_tc	Must be of the collection element's typecode (from DBMS_TYPES package).
elem_count	Pass 0 for elem_count if the self represents a nested table (TYPECODE_TABLE). Otherwise pass the collection count if *self* represents a VARRAY.

Exceptions

- DBMS_TYPES.INVALID_PARAMETER: Invalid Parameters (typecode, typeinfo)

- DBMS_TYPES.INCORRECT_USAGE: Incorrect usage (cannot call after calling ENDCREATE, and similar actions.)

Usage Notes

It is an error to call this function on an ANYTYPE that represents a persistent user defined type.

ADDATTR Member Procedure

This procedure adds an attribute to an ANYTYPE (of typecode DBMS_TYPES.TYPECODE_OBJECT).

Syntax

```
MEMBER PROCEDURE ADDATTR(
   self          IN OUT NOCOPY ANYTYPE,
   aname         IN VARCHAR2,
   typecode      IN PLS_INTEGER,
   prec          IN PLS_INTEGER,
   scale         IN PLS_INTEGER,
   len           IN PLS_INTEGER,
   csid          IN PLS_INTEGER,
   csfrm         IN PLS_INTEGER,
   attr_type     IN ANYTYPE DEFAULT NULL);
```

Parameters

Table 183–5 ADDATTR Procedure Parameters

Parameter	Description
self	The transient ANYTYPE that is being constructed. Must be of type DBMS_TYPES.TYPECODE_OBJECT.
aname	Optional. Attribute's name. Could be NULL.
typecode	Attribute's typecode. Can be built-in or user-defined typecode (from DBMS_TYPES package).
prec	Optional. Required if typecode represents a NUMBER. Give precision and scale. Ignored otherwise.
scale	Optional. Required if typecode represents a NUMBER. Give precision and scale. Ignored otherwise.
len	Optional. Required if typecode represents a RAW, CHAR, VARCHAR, or VARCHAR2 type. Give length.
csid	Optional. Required if typecode represents a type requiring character information, such as CHAR, VARCHAR, VARCHAR2, CFILE.
csfrm	Optional. Required if typecode represents a type requiring character information, such as CHAR, VARCHAR, VARCHAR2, CFILE.
attr_type	Optional. ANYTYPE corresponding to a user-defined type. This parameter is required if the attribute is a user defined type.

Exceptions

- DBMS_TYPES.INVALID_PARAMETERS: Invalid Parameters (typecode, typeinfo)
- DBMS_TYPES.INCORRECT_USAGE: Incorrect usage (cannot call after calling EndCreate, and similar actions.)

ENDCREATE Member Procedure

This procedure ends creation of a transient ANYTYPE. Other creation functions cannot be called after this call.

Syntax

```
MEMBER PROCEDURE ENDCREATE(
   self           IN OUT NOCOPY ANYTYPE);
```

Parameter

Table 183–6 ENDCREATE Procedure Parameter

Parameter	Description
self	The transient ANYTYPE that is being constructed.

GETPERSISTENT Static Function

This procedure returns an ANYTYPE corresponding to a persistent type created earlier using the CREATE TYPE SQL statement.

Syntax

```
STATIC FUNCTION GETPERSISTENT(
   schema_name      IN VARCHAR2,
   type_name        IN VARCHAR2,
   version          IN VARCHAR2 DEFAULT NULL)
   RETURN           ANYTYPE;
```

Parameters

Table 183–7 GETPERSISTENT Function Parameters

Parameter	Description
schema_name	Schema name of the type.
type_name	Type name.
version	Type version.

Return Values

An ANYTYPE corresponding to a persistent type created earlier using the CREATE TYPE SQL statement.

GETINFO Member Function

This function gets the type information for the ANYTYPE.

Syntax

```
MEMBER FUNCTION GETINFO (
   self        IN ANYTYPE,
   prec        OUT PLS_INTEGER,
   scale       OUT PLS_INTEGER,
   len         OUT PLS_INTEGER,
   csid        OUT PLS_INTEGER,
   csfrm       OUT PLS_INTEGER,
   schema_name OUT VARCHAR2,
   type_name   OUT VARCHAR2,
   version     OUT varchar2,
   count       OUT PLS_INTEGER)
   RETURN      PLS_INTEGER;
```

Parameters

Table 183–8 GETINFO Function Parameters

Parameter	Description
self	The ANYTYPE.
prec	If typecode represents a number. Gives precision and scale. Ignored otherwise.
scale	If typecode represents a number. Gives precision and scale. Ignored otherwise.
len	If typecode represents a RAW, CHAR, VARCHAR, or VARCHAR2 type. Gives length.
csid	If typecode represents a type requiring character information such as: CHAR, VARCHAR, VARCHAR2, CFILE.
csid	If typecode represents a type requiring character information such as: CHAR, VARCHAR, VARCHAR2, CFILE.
schema_name	Type's schema (if persistent).
type_name	Type's typename.
version	Type's version.
count	If *self* is a VARRAY, this gives the VARRAY count. If self is of TYPECODE_OBJECT, this gives the number of attributes.

Return Values

The typecode of *self*.

Exceptions

- DBMS_TYPES.INVALID_PARAMETERS: Invalid Parameters (position is beyond bounds or the ANYTYPE is not properly Constructed).

GETATTRELEMINFO Member Function

This function gets the type information for an attribute of the type (if it is of TYPECODE_OBJECT). Gets the type information for a collection's element type if the *self* parameter is of a collection type.

Syntax

```
MEMBER FUNCTION GETATTRELEMINFO (
   self           IN ANYTYPE,
   pos            IN PLS_INTEGER,
   prec           OUT PLS_INTEGER,
   scale          OUT PLS_INTEGER,
   len            OUT PLS_INTEGER,
   csid           OUT PLS_INTEGER,
   csfrm          OUT PLS_INTEGER,
   attr_elt_type  OUT ANYTYPE
   aname          OUT VARRCHAR2)
   RETURN         PLS_INTEGER;
```

Parameters

Table 183–9 GETATTRELEMINFO Function Parameters

Parameter	Description
self	The ANYTYPE.
pos	If self is of TYPECODE_OBJECT, this gives the attribute position (starting at 1). It is ignored otherwise.
prec	If attribute/collection element typecode represents a NUMBER. Gives precision and scale. Ignored otherwise.
scale	If attribute/collection element typecode represents a NUMBER. Gives precision and scale. Ignored otherwise.
len	If typecode represents a RAW, CHAR, VARCHAR, or VARCHAR2 type. Gives length.
csid, csfrm	If typecode represents a type requiring character information such as: CHAR, VARCHAR, VARCHAR2, CFILE. Gives character set ID, character set form.
attr_elt_type	If attribute/collection element typecode represents a user-defined type, this returns the ANYTYPE corresponding to it. User can subsequently describe the *attr_elt_type*.
aname	Attribute name (if it is an attribute of an object type, NULL otherwise).

Return Values

The typecode of the attribute or collection element.

Exceptions

DBMS_TYPES.INVALID_PARAMETERS: Invalid Parameters (position is beyond bounds or the ANYTYPE is not properly constructed).

184

Oracle Streams AQ TYPEs

This chapter describes the types used with Oracle Streams Advanced Queuing (AQ) packages for PL/SQL, `DBMS_AQ`, and `DBMS_AQADM`.

> **See Also:** *Oracle Streams Advanced Queuing User's Guide and Reference* for information about using Oracle Streams AQ.

This chapter contains the following topics:

Summary of Types

AQ$_AGENT Type

This type identifies a producer or a consumer of a message.

Syntax

```
TYPE SYS.AQ$_AGENT IS OBJECT (
   name       VARCHAR2(30),
   address    VARCHAR2(1024),
   protocol   NUMBER  DEFAULT 0);
```

Attributes

Table 184–1 AQ$_AGENT Attributes

Attribute	Description
`name`	Name of a producer or consumer of a message. The name must follow object name guidelines in the *Oracle Database SQL Reference* with regard to reserved characters.
`address`	Protocol-specific address of the recipient. If the protocol is `0`, then the address is of the form [*schema.*]*queue*[@*dblink*].
	For example, a queue named `emp_messages` in the `HR` queue at the site `dbs1.net` has the address: hr.emp_messages@dbs1.net
`protocol`	Protocol to interpret the address and propagate the message. Protocols 1-127 are reserved for internal use. If the protocol number is in the range 128 - 255, the address of the recipient is not interpreted by Oracle Streams AQ.

AQ$_AGENT_LIST_T Type

This type identifies the list of agents for which `DBMS_AQ.LISTEN` listens.

See Also: "AQ$_AGENT Type" on page 184-3

Syntax

```
TYPE SYS.AQ$_AGENT_LIST_T IS TABLE OF SYS.AQ$_AGENT
   INDEX BY BINARY INTEGER;
```

AQ$_DESCRIPTOR Type

This type specifies the Oracle Streams AQ descriptor received by the AQ PL/SQL callbacks upon notification.

See Also: "MESSAGE_PROPERTIES_T Type" on page 184-22

Syntax

```
TYPE SYS.AQ$_DESCRIPTOR IS OBJECT (
   queue_name      VARCHAR2(61),
   consumer_name   VARCHAR2(30),
   msg_id          RAW(16),
   msg_prop        MSG_PROP_T,
   gen_desc        AQ$_NTFN_DESCRIPTOR);
```

Attributes

Table 184–2 AQ$_DESCRIPTOR Attributes

Attribute	Description
queue_name	Name of the queue in which the message was enqueued which resulted in the notification
consumer_name	Name of the consumer for the multiconsumer queue
msg_id	Identification number of the message
msg_prop	Message properties specified by the MSG_PROP_T type
gen_desc	Indicates the timeout specifications

AQ$_NTFN_DESCRIPTOR Type

This type is for timeout specifications.

Syntax

```
TYPE SYS.AQ$_NTFN_DESCRIPTOR IS OBJECT(
   ntfn_flags   NUMBER)
```

Attributes

Table 184–3 AQ$_DESCRIPTOR Attributes

Attribute	Description
`ntfn_flags`	Set to 1 if the notifications are already removed after a stipulated timeout; otherwise the value is 0.

AQ$_POST_INFO Type

Specifies anonymous subscriptions to which you want to post messages.

Syntax

```
TYPE SYS.AQ$_POST_INFO IS OBJECT (
   name        VARCHAR2(128),
   namespace   NUMBER,
   payload     RAW(2000)  DEFAULT NULL);
```

Attributes

Table 184–4 AQ$_POST_INFO Attributes

Attribute	Description
`name`	Name of the anonymous subscription to which you want to post
`namespace`	To receive notifications from other applications through `DBMS_AQ.POST` or `OCISubscriptionPost()`, the namespace must be `DBMS_AQ.NAMESPACE_ANONYMOUS`
`payload`	The payload to be posted to the anonymous subscription

AQ$_POST_INFO_LIST Type

Identifies the list of anonymous subscriptions to which you want to post messages.

See Also: AQ$_POST_INFO Type on page 184-7

Syntax

```
TYPE SYS.AQ$_POST_INFO_LIST AS VARRAY(1024) OF SYS.AQ$_POST_INFO;
```

AQ$_PURGE_OPTIONS_T Type

This type specifies the options available for purging a queue table.

See Also: PURGE_QUEUE_TABLE Procedure on page 17-42.

Syntax

```
TYPE AQ$_PURGE_OPTIONS_T is RECORD (
   block           BOOLEAN       DEFAULT FALSE
   delivery_mode   PLS_INTEGER   DEFAULT PERSISTENT);
```

Table 184–5 AQ$_PURGE_OPTIONS_T Type Attributes

Attribute	Description
block	TRUE/FALSE. ■ If block is TRUE, then an exclusive lock on all the queues in the queue table is held while purging the queue table. This will cause concurrent enqueuers and dequeuers to block while the queue table is purged. The purge call always succeeds if block is TRUE. ■ The default for block is FALSE. This will not block enqueuers and dequeuers, but it can cause the purge to fail with an error during high concurrency times.
delivery_mode	Kind of messages to purge, either DBMS_AQ.BUFFERED or DBMS_AQ.PERSISTENT

AQ$_RECIPIENT_LIST_T Type

Identifies the list of agents that receive the message. This type can be used only when the queue is enabled for multiple dequeues.

See Also: "AQ$_AGENT Type" on page 184-3

Syntax

```
TYPE SYS.AQ$_RECIPIENT_LIST_T IS TABLE OF SYS.AQ$_AGENT
   INDEX BY BINARY_INTEGER;
```

AQ$_REG_INFO Type

This type identifies a producer or a consumer of a message.

Syntax

```
TYPE SYS.AQ$_REG_INFO IS OBJECT (
   name        VARCHAR2(128),
   namespace   NUMBER,
   callback    VARCHAR2(4000),
   context     RAW(2000)  DEFAULT NULL,
   qosflags    NUMBER,
   timeout     NUMBER);
```

Attributes

Table 184–6 AQ$_REG_INFO Type Attributes

Attribute	Description
`name`	Specifies the name of the subscription. The subscription name is of the form *`schema.queue`* if the registration is for a single consumer queue or *`schema.queue:consumer_name`* if the registration is for a multiconsumer queues.
`namespace`	Specifies the namespace of the subscription. To receive notification from Oracle Streams AQ queues, the namespace must be `DBMS_AQ.NAMESPACE_AQ`. To receive notifications from other applications through `DBMS_AQ.POST` or `OCISubscriptionPost()`, the namespace must be `DBMS_AQ.NAMESPACE_ANONYMOUS`.
`callback`	Specifies the action to be performed on message notification. For HTTP notifications, use `http://www.company.com:8080`. For e-mail notifications, use `mailto://xyz@company.com`. For raw message payload for the `PLSQLCALLBACK` procedure, use `plsql://`*`schema.procedure`*`?PR=0`. For user-defined type message payload converted to XML for the `PLSQLCALLBACK` procedure, use `plsql://`*`schema.procedure`*`?PR=1`
`context`	Specifies the context that is to be passed to the callback function
`quosflags`	Can be set to one or more of the following values to specify the notification quality of service: ▪ `NTFN_QOS_RELIABLE`- This value specifies that reliable notification is required. Reliable notifications persist across instance and database restarts. ▪ `NTFN_QOS_PAYLOAD` - This value specifies that payload delivery is required. It is supported only for client notification and only for `RAW` queues. ▪ `NTFN_QOS_PURGE_ON_NTFN` - This value specifies that the registration is to be purged automatically when the first notification is delivered to this registration location.
`timeout`	Specifies an automatic expiration period for the registration. If you want no timeout, then set this attribute to 0.

Usage Notes

You can use the following notification mechanisms:

- OCI callback
- e-mail callback

- PL/SQL callback

Table 184–7 shows the actions performed for nonpersistent queues for different notification mechanisms when RAW presentation is specified. Table 184–8 shows the actions performed when XML presentation is specified.

Table 184–7 Actions Performed for Nonpersistent Queues When RAW Presentation Specified

Queue Payload Type	OCI Callback	E-mail	PL/SQL Callback
`RAW`	OCI callback receives the RAW data in the payload.	Not supported	PL/SQL callback receives the RAW data in the payload.
`Oracle object type`	Not supported	Not supported	Not supported

Table 184–8 Actions Performed for Nonpersistent Queues When XML Presentation Specified

Queue Payload Type	OCI Callback	E-mail	PL/SQL Callback
`RAW`	OCI callback receives the XML data in the payload.	XML data is formatted as a SOAP message and e-mailed to the registered e-mail address.	PL/SQL callback receives the XML data in the payload.
`Oracle object type`	OCI callback receives the XML data in the payload.	XML data is formatted as a SOAP message and e-mailed to the registered e-mail address.	PL/SQL callback receives the XML data in the payload.

AQ$_REG_INFO_LIST Type

Identifies the list of registrations to a queue.

See Also: "AQ$_REG_INFO Type" on page 184-11

Syntax

```
TYPE SYS.AQ$_REG_INFO_LIST AS VARRAY(1024) OF SYS.AQ$_REG_INFO;
```

AQ$_SUBSCRIBER_LIST_T Type

Identifies the list of subscribers that subscribe to a queue.

See Also: "AQ$_AGENT Type" on page 184-3

Syntax

```
TYPE SYS.AQ$_SUBSCRIBER_LIST_T IS TABLE OF SYS.AQ$_AGENT
   INDEX BY BINARY_INTEGER;
```

DEQUEUE_OPTIONS_T Type

Specifies the options available for the dequeue operation.

Syntax

```
TYPE DEQUEUE_OPTIONS_T IS RECORD (
   consumer_name     VARCHAR2(30)    DEFAULT NULL,
   dequeue_mode      BINARY_INTEGER  DEFAULT REMOVE,
   navigation        BINARY_INTEGER  DEFAULT NEXT_MESSAGE,
   visibility        BINARY_INTEGER  DEFAULT ON_COMMIT,
   wait              BINARY_INTEGER  DEFAULT FOREVER,
   msgid             RAW(16)         DEFAULT NULL,
   correlation       VARCHAR2(128)   DEFAULT NULL,
   deq_condition     VARCHAR2(4000)  DEFAULT NULL,
   signature         aq$_sig_prop    DEFAULT NULL,
   transformation    VARCHAR2(61)    DEFAULT NULL,
   delivery_mode     PLS_INTEGER     DEFAULT PERSISTENT);
```

Attributes

Table 184–9 DEQUEUE_OPTIONS_T Attributes

Attribute	Description
consumer_name	Name of the consumer. Only those messages matching the consumer name are accessed. If a queue is not set up for multiple consumers, then this field should be set to NULL. For secure queues, consumer_name must be a valid AQ agent name, mapped to the database user performing the dequeue operation, through dbms_aqadm.enable_db_access procedure call.
dequeue_mode	Specifies the locking behavior associated with the dequeue. Possible settings are: BROWSE: Read the message without acquiring any lock on the message. This specification is equivalent to a select statement. LOCKED: Read and obtain a write lock on the message. The lock lasts for the duration of the transaction. This setting is equivalent to a select for update statement. REMOVE: Read the message and delete it. This setting is the default. The message can be retained in the queue table based on the retention properties. REMOVE_NODATA: Mark the message as updated or deleted. The message can be retained in the queue table based on the retention properties.

Table 184–9 (Cont.) DEQUEUE_OPTIONS_T Attributes

Attribute	Description
navigation	Specifies the position of the message that will be retrieved. First, the position is determined. Second, the search criterion is applied. Finally, the message is retrieved. Possible settings are: NEXT_MESSAGE: Retrieve the next message that is available and matches the search criteria. If the previous message belongs to a message group, then AQ retrieves the next available message that matches the search criteria and belongs to the message group. This setting is the default. NEXT_TRANSACTION: Skip the remainder of the current transaction group (if any) and retrieve the first message of the next transaction group. This setting can only be used if message grouping is enabled for the current queue. FIRST_MESSAGE: Retrieves the first message which is available and matches the search criteria. This setting resets the position to the beginning of the queue. FIRST_MESSAGE_MULTI_GROUP: indicates that a call to DBMS_AQ.DEQUEUE_ARRAY will reset the position to the beginning of the queue and dequeue messages (possibly across different transaction groups) that are available and match the search criteria, until reaching the ARRAY_SIZE limit. Refer to the TRANSACTION_GROUP attribute for the message to distinguish between transaction groups. NEXT_MESSAGE_MULTI_GROUP: indicates that a call to DBMS_AQ.DEQUEUE_ARRAY will dequeue the next set of messages (possibly across different transaction groups) that are available and match the search criteria, until reaching the ARRAY_SIZE limit. Refer to the TRANSACTION_GROUP attribute for the message to distinguish between transaction groups.
visibility	Specifies whether the new message is dequeued as part of the current transaction.The visibility parameter is ignored when using the BROWSE dequeue mode. Possible settings are: ON_COMMIT: The dequeue will be part of the current transaction. This setting is the default. IMMEDIATE: The dequeue operation is not part of the current transaction, but an autonomous transaction which commits at the end of the operation.
wait	Specifies the wait time if there is currently no message available which matches the search criteria. Possible settings are: FOREVER: Wait forever. This setting is the default. NO_WAIT: Do not wait. number: Wait time in seconds.
msgid	Specifies the message identifier of the message to be dequeued.
correlation	Specifies the correlation identifier of the message to be dequeued. Special pattern matching characters, such as the percent sign (%) and the underscore (_) can be used. If more than one message satisfies the pattern, then the order of dequeuing is undetermined.

Table 184–9 (Cont.) DEQUEUE_OPTIONS_T Attributes

Attribute	Description
deq_condition	A conditional expression based on the message properties, the message data properties, and PL/SQL functions. A deq_condition is specified as a Boolean expression using syntax similar to the WHERE clause of a SQL query. This Boolean expression can include conditions on message properties, user data properties (object payloads only), and PL/SQL or SQL functions (as specified in the WHERE clause of a SQL query). Message properties include priority, corrid and other columns in the queue table To specify dequeue conditions on a message payload (object payload), use attributes of the object type in clauses. You must prefix each attribute with tab.user_data as a qualifier to indicate the specific column of the queue table that stores the payload. The deq_condition parameter cannot exceed 4000 characters. If more than one message satisfies the dequeue condition, then the order of dequeuing is undetermined.
signature	Currently not implemented
transformation	Specifies a transformation that will be applied after dequeuing the message. The source type of the transformation must match the type of the queue.
delivery_mode	The dequeuer specifies the delivery mode of the messages it wishes to dequeue in the dequeue options. It can be BUFFERED or PERSISTENT or PERSISTENT_OR_BUFFERED. The message properties of the dequeued message indicate the delivery mode of the dequeued message. Array dequeue is only supported for buffered messages with an array size of '1'.

ENQUEUE_OPTIONS_T Type

Specifies the options available for the enqueue operation.

Syntax

```
TYPE SYS.ENQUEUE_OPTIONS_T IS RECORD (
   visibility            BINARY_INTEGER  DEFAULT ON_COMMIT,
   relative_msgid        RAW(16)         DEFAULT NULL,
   sequence_deviation    BINARY_INTEGER  DEFAULT NULL,
   transformation        VARCHAR2(61)    DEFAULT NULL,
   delivery_mode         PLS_INTEGER     NOT NULL DEFAULT PERSISTENT);
```

Attributes

Table 184–10 ENQUEUE_OPTIONS_T Attributes

Attribute	Description
`visibility`	Specifies the transactional behavior of the enqueue request. Possible settings are: `ON_COMMIT`: The enqueue is part of the current transaction. The operation is complete when the transaction commits. This setting is the default. `IMMEDIATE`: The enqueue operation is not part of the current transaction, but an autonomous transaction which commits at the end of the operation. This is the only value allowed when enqueuing to a non-persistent queue.
`relative_msgid`	Specifies the message identifier of the message which is referenced in the sequence deviation operation. This field is valid only if `BEFORE` is specified in `sequence_deviation`. This parameter is ignored if sequence deviation is not specified.
`sequence_deviation`	Specifies whether the message being enqueued should be dequeued before other messages already in the queue. Possible settings are: `BEFORE`: The message is enqueued ahead of the message specified by `relative_msgid`. `TOP`: The message is enqueued ahead of any other messages.
`transformation`	Specifies a transformation that will be applied before enqueuing the message. The return type of the transformation function must match the type of the queue.
`delivery_mode`	Specifies a transformation that will be applied before enqueuing the message. The return type of the transformation function must match the type of the queue.

SYS.MSG_PROP_T Type

This type is used in PL/SQL notification, as one field in aq$_descriptor, to pass message properties of an AQ message to the PL/SQL notification client callback.

Syntax

```
CREATE or replace TYPE sys.msg_prop_t AS OBJECT (
   priority         NUMBER,
   delay            NUMBER,
   expiration       NUMBER,
   correlation      VARCHAR2(128),
   attempts         NUMBER,
   exception_queue  VARCHAR2(51),
   enqueue_time     DATE,
   state            NUMBER,
   sender_id        aq$_agent,
   original_msgid   RAW(16),
   delivery_mode    NUMBER);
```

Parameters

Table 184–11 SYS.MSG_PROP_T Type Attributes

Parameter	Description
priority	Specifies the priority of the message. A smaller number indicates higher priority. The priority can be any number, including negative numbers.
delay	Specifies the delay of the enqueued message. The delay represents the number of seconds after which a message is available for dequeuing. Dequeuing by msgid overrides the delay specification. A message enqueued with delay set is in the WAITING state, and when the delay expires, the message goes to the READY state. DELAY processing requires the queue monitor to be started. However the queue monitor is started automatically by the system if needed. Delay is set by the producer who enqueues the message. The possible settings follow: NO_DELAY: The message is available for immediate dequeuing number: The number of seconds to delay the message
expiration	Specifies the expiration of the message. It determines, in seconds, the duration the message is available for dequeuing. This parameter is an offset from the time the message is ready for dequeue. Expiration processing requires the queue monitor to be running. However the queue monitor is started automatically by the system if needed. The possible settings follow: NEVER: The message does not expire number: The number of seconds message remains in READY state. If the message is not dequeued before it expires, then it is moved to the exception queue in the EXPIRED state.
correlation	Returns the identifier supplied by the producer of the message at enqueue time.
attempts	Returns the number of attempts that have been made to dequeue the message. This parameter cannot be set at enqueue time.

Table 184–11 (Cont.) SYS.MSG_PROP_T Type Attributes

Parameter	Description
exception_queue	Specifies the name of the queue into which the message is moved if it cannot be processed successfully. Messages are moved automatically into the exception queue. Messages are moved into the exception queue in the following cases: ■ RETRY_COUNT, the number of unsuccessful dequeue attempts, has exceeded the specification for the MAX_RETRIES parameter in the DBMS_AQADM.CREATE_QUEUE procedure during queue creation. For multiconsumer queues, the message becomes eligible to be moved to the exception queue even if failed dequeue attempts exceeds the MAX_RETRIES parameter for only one of the consumers. But the message will not be moved until either all other consumers have successfully consumed the message or failed more than MAX_RETRIES. You can view MAX_RETRIES for a queue in the ALL_QUEUES data dictionary view. If a dequeue transaction fails because the server process dies (including ALTER SYSTEM KILL SESSION) or SHUTDOWN ABORT on the instance, then RETRY_COUNT is not incremented. ■ A message was not dequeued before the expiration time elapsed. ■ Message propagation to the specified destination queue failed with one of the following errors: * There were no recipients for the multiconsumer destination queue. * Recipients were specified for a single-consumer destination queue. * Destination queue was an exception queue * There was an error when applying transformation. The default is the exception queue associated with the queue table. If the exception queue specified does not exist at the time of the move, then the message is moved to the default exception queue associated with the queue table, and a warning is logged in the alert log. If the default exception queue is specified, then the parameter returns a NULL value at dequeue time.
enqueue_time	Specifies the time the message was enqueued. This value is determined by the system and cannot be set by the user at enqueue time.
state	Specifies the state of the message at the time of the dequeue. This parameter cannot be set at enqueue time. The possible states follow: ■ DBMS_AQ.READY: The message is ready to be processed. ■ DBMS_AQ.WAITING: The message delay has not yet been reached. ■ DBMS_AQ.PROCESSED: The message has been processed and is retained. ■ DBMSAQ.EXPIRED: The message has been moved to the exception queue.

Table 184–11 (Cont.) SYS.MSG_PROP_T Type Attributes

Parameter	Description
sender_id	The application-sender identification specified at enqueue time by the message producer. Sender id is of type aq$_agent. Sender name is required for secure queues at enqueue time. This must be a valid AQ agent name, mapped to the database user performing the enqueue operation, through dbms_aqadm.enable_db_access procedure call. Sender address and protocol should not be specified. The Sender id in the message properties returned at dequeue time may have a sender address if the message was propagated from another queue. The value of the address is the source_queue, source database name if it was a remote database [format source_queue@source_database_name]
original_msgid	This parameter is used by Oracle Streams AQ for propagating messages.
delivery_mode	DBMS_AQ.BUFFERED or DBMS_AQ.PERSISTENT.

MESSAGE_PROPERTIES_T Type

This type is defined inside the DBMS_AQ package, and describes the information that AQ uses to convey the state of individual messages. These are set at enqueue time, and their values are returned at dequeue time.

See Also: AQ$_RECIPIENT_LIST_T Type on page 184-10

Syntax

```
TYPE message_properties_t IS RECORD (
   priority               BINARY_INTEGER  NOT NULL DEFAULT 1,
   delay                  BINARY_INTEGER  NOT NULL DEFAULT NO_DELAY,
   expiration             BINARY_INTEGER  NOT NULL DEFAULT NEVER,
   correlation            VARCHAR2(128)   DEFAULT NULL,
   attempts               BINARY_INTEGER,
   recipient_list         AQ$_RECIPIENT_LIST_T,
   exception_queue        VARCHAR2(61)    DEFAULT NULL,
   enqueue_time           DATE,
   state                  BINARY_INTEGER,
   sender_id              SYS.AQ$_AGENT   DEFAULT NULL,
   original_msgid         RAW(16)         DEFAULT NULL,
   signature              aq$_sig_prop    DEFAULT NULL,
   transaction_group      VARCHAR2(30)    DEFAULT NULL,
   user_property          SYS.ANYDATA     DEFAULT NULL
   delivery_mode          PLS_INTEGER     NOT NULL DEFAULT DBMS_AQ.PERSISTENT);
```

Attributes

Table 184–12 MESSAGE_PROPERTIES_T Attributes

Attribute	Description
`priority`	Specifies the priority of the message. A smaller number indicates higher priority. The priority can be any number, including negative numbers.
`delay`	Specifies the delay of the enqueued message. The delay represents the number of seconds after which a message is available for dequeuing. Dequeuing by `msgid` overrides the delay specification. A message enqueued with delay set is in the `WAITING` state, and when the delay expires, the message goes to the `READY` state. `DELAY` processing requires the queue monitor to be started. However the queue monitor is started automatically by the system if needed. Delay is set by the producer who enqueues the message. The possible settings follow: `NO_DELAY`: The message is available for immediate dequeuing number: The number of seconds to delay the message

Table 184–12 (Cont.) MESSAGE_PROPERTIES_T Attributes

Attribute	Description
`expiration`	Specifies the expiration of the message. It determines, in seconds, the duration the message is available for dequeuing. This parameter is an offset from the time the message is ready for dequeue. Expiration processing requires the queue monitor to be running. However the queue monitor is started automatically by the system if needed. The possible settings follow: `NEVER`: The message does not expire number: The number of seconds message remains in `READY` state. If the message is not dequeued before it expires, then it is moved to the exception queue in the `EXPIRED` state.
`correlation`	Returns the identifier supplied by the producer of the message at enqueue time.
`attempts`	Returns the number of attempts that have been made to dequeue the message. This parameter cannot be set at enqueue time.
`recipient_list`	This parameter is only valid for queues that allow multiple consumers. The default recipients are the queue subscribers. This parameter is not returned to a consumer at dequeue time. For type definition, see the "AQ$_AGENT Type" on page 184-3.

Table 184–12 (Cont.) MESSAGE_PROPERTIES_T Attributes

Attribute	Description
exception_queue	Specifies the name of the queue into which the message is moved if it cannot be processed successfully. Messages are moved automatically into the exception queue. Messages are moved into the exception queue in the following cases: ▪ RETRY_COUNT, the number of unsuccessful dequeue attempts, has exceeded the specification for the MAX_RETRIES parameter in the DBMS_AQADM.CREATE_QUEUE procedure during queue creation. For multiconsumer queues, the message becomes eligible to be moved to the exception queue even if failed dequeue attempts exceeds the MAX_RETRIES parameter for only one of the consumers. But the message will not be moved until either all other consumers have successfully consumed the message or failed more than MAX_RETRIES. You can view MAX_RETRIES for a queue in the ALL_QUEUES data dictionary view. If a dequeue transaction fails because the server process dies (including ALTER SYSTEM KILL SESSION) or SHUTDOWN ABORT on the instance, then RETRY_COUNT is not incremented. ▪ A message was not dequeued before the expiration time elapsed. ▪ Message propagation to the specified destination queue failed with one of the following errors: * There were no recipients for the multiconsumer destination queue. * Recipients were specified for a single-consumer destination queue. * Destination queue was an exception queue * There was an error when applying transformation. The default is the exception queue associated with the queue table. If the exception queue specified does not exist at the time of the move, then the message is moved to the default exception queue associated with the queue table, and a warning is logged in the alert log. If the default exception queue is specified, then the parameter returns a NULL value at dequeue time.
enqueue_time	Specifies the time the message was enqueued. This value is determined by the system and cannot be set by the user at enqueue time.
state	Specifies the state of the message at the time of the dequeue. This parameter cannot be set at enqueue time. The possible states follow: ▪ DBMS_AQ.READY: The message is ready to be processed. ▪ DBMS_AQ.WAITING: The message delay has not yet been reached. ▪ DBMS_AQ.PROCESSED: The message has been processed and is retained. ▪ DBMSAQ.EXPIRED: The message has been moved to the exception queue.

Table 184–12 (Cont.) MESSAGE_PROPERTIES_T Attributes

Attribute	Description
sender_id	The application-sender identification specified at enqueue time by the message producer. Sender id is of type aq$_agent. Sender name is required for secure queues at enqueue time. This must be a valid AQ agent name, mapped to the database user performing the enqueue operation, through dbms_aqadm.enable_db_access procedure call. Sender address and protocol should not be specified. The Sender id in the message properties returned at dequeue time may have a sender address if the message was propagated from another queue. The value of the address is the source_queue, source database name if it was a remote database [format source_queue@source_database_name]
original_msgid	This parameter is used by Oracle Streams AQ for propagating messages.
signature	Currently not implemented
transaction_group	Specifies the transaction_group for the dequeued message. Messages belonging to the same transaction group will have the same value for this attribute. This attribute is only set by the DBMS_AQ.DEQUEUE_ARRAY. This attribute cannot be used to set the transaction group of a message through DBMS_AQ.ENQUEUE or DBMS_AQ.ENQUEUE_ARRAY calls.
user_property	This optional attribute is used to store additional information about the payload.
delivery_mode	The message publisher specifies the delivery mode in the message_properties. This can be DBMS_AQ.BUFFERED or DBMS_AQ.PERSISTENT. Array enqueue is only supported for buffered messages with an array size of '1'.

MESSAGE_PROPERTIES_ARRAY_T Type

This type is used by `dbms_aq.enqueue_array` and `dbms_aq.dequeue_array` calls to hold the set of message properties. Each element in the `payload_array` should have a corresponding element in the `MESSAGE_PROPERTIES_ARRAY_T VARRAY`.

See Also: "MESSAGE_PROPERTIES_T Type" on page 184-22

Syntax

```
TYPE MESSAGE_PROPERTIES_ARRAY_T IS VARRAY (2147483647)
   OF MESSAGE_PROPERTIES_T;
```

MSGID_ARRAY_T Type

The `msgid_array_t` type is used in `dbms_aq.enqueue_array` and `dbms_aq.dequeue_array` calls to hold the set of message IDs that correspond to the enqueued or dequeued messages.

Syntax

```
TYPE MSGID_ARRAY_T IS TABLE OF RAW(16) INDEX BY BINARY_INTEGER
```

185

Database URI TYPEs

Oracle supports the `UriType` family of types that can be used to store and query Uri-refs inside the database. The `UriType` itself is an abstract object type and the `HTTPURITYPE`, `XDBURITYPE` and `DBURITYPE` are subtypes of it.

You can create a `UriType` column and store instances of the `DBURITYPE`, `XDBURITYPE` or the `HTTPURITYPE` inside of it. You can also define your own subtypes of the `UriType` to handle different URL protocols.

Oracle also provides a `UriFactory` package that can be used as a factory method to automatically generate various instances of these `UriTypes` by scanning the prefix, such as `http://` or `/oradb`. You can also register your subtype and provide the prefix that you support. For instance, if you have written a subtype to handle the gopher protocol, you can register the prefix `gopher://` to be handled by your subtype. The `UriFactory` will then generate your subtype instance for any URL starting with that prefix.

This chapter contains the following topics:

See Also:

- *Oracle XML DB Developer's Guide*

Summary of URITYPE Supertype Subprograms

The `UriType` is the abstract super type. It provides a standard set of functions to get the value pointed to by the URI. The actual implementation of the protocol must be defined by the subtypes of this type.

Instances of this type cannot be created directly. However, you can create columns of this type and store subtype instances in it, and also select from columns without knowing the instance of the URL stored.

Table 185–1 URITYPE Type Subprograms

Method	Description
GETBLOB on page 185-3	Returns the `BLOB` located at the address specified by the URL.
GETCLOB on page 185-4	Returns the `CLOB` located at the address specified by the URL.
GETCONTENTTYPE on page 185-5	Returns the URL, in escaped format, stored inside the `UriType` instance.
GETEXTERNALURL on page 185-6	Returns the URL, in escaped format, stored inside the `UriType` instance.
GETURL on page 185-7	Returns the URL, in non-escaped format, stored inside the `UriType` instance.
GETXML on page 185-8	Returns the `XMLType` located at the address specified by the URL.

GETBLOB

This function returns the BLOB located at the address specified by the URL. This function can be overridden in the subtype instances. The options are described in the following table.

Syntax	Description
MEMBER FUNCTION getBlob() RETURN BLOB;	This function returns the BLOB located at the address specified by the URL.
MEMBER FUNCTION getBlob(content OUT VARCHAR2) RETURN BLOB;	This function returns the BLOB located at the address specified by the URL and the content type.
FUNCTION getBlob(csid IN NUMBER) RETURN BLOB;	This function returns the BLOB located at the address specified by the URL in the specified character set.

Parameter	IN / OUT	Description
content	(OUT)	Content type of the document to which URI is pointing.
csid	(IN)	Character set id of the document. Must be a valid Oracle id and greater than 0; otherwise returns an error

GETCLOB

This function returns the CLOB located at the address specified by the URL. This function can be overridden in the subtype instances. This function returns either a permanent CLOB or a temporary CLOB. If a temporary CLOB is returned, it must be freed. The options are described in the following table.

Syntax	Description
MEMBER FUNCTION getClob() RETURN clob;	This function returns the CLOB located at the address specified by the URL.
MEMBER FUNCTION getClob(content OUT VARCHAR2) RETURN clob;	This function returns the CLOB located at the address specified by the URL and the content type.

Parameter	IN / OUT	Description
content	(OUT)	Content type of the document to which URI is pointing.

GETCONTENTTYPE

This function returns the content type of the document pointed to by the URI. This function can be overridden in the subtype instances. This function returns the content type as VARCHAR2.

Syntax

```
MEMBER FUNCTION getContentType()
RETURN VARCHAR2;
```

GETEXTERNALURL

This function returns the URL, in escaped format, stored inside the `UriType` instance. The subtype instances override this member function to provide additional semantics. For instance, the `HTTPURITYPE` function does not store the prefix `http://` in the URL itself. When generating the external URL, it appends the prefix and generates it. For this reason, use the `getExternalUrl` function or the `getUrl` function to get to the URL value instead of using the attribute present in the `UriType` instance.

Syntax

```
MEMBER FUNCTION getExternalUrl()
RETURN varchar2;
```

GETURL

This function returns the URL, in non-escaped format, stored inside the `UriType` instance. The subtype instances override this member function to provide additional semantics. For instance, the `HTTPURITYPE` function does not store the prefix `http://` in the URL itself. When generating the external URL, it appends the prefix and generates it. For this reason, use the `getExternalUrl` function or the `getUrl` function to get to the URL value instead of using the attribute present in the `UriType` instance.

Syntax

```
MEMBER FUNCTION getUrl()
RETURN varchar2;
```

GETXML

This function returns the XMLType located at the address specified by the URL. This function can be overridden in the subtype instances. The options are described in the following table.

Syntax	Description
`MEMBER FUNCTION getXML()` `RETURN XMLType;`	This function returns the XMLType located at the address specified by the URL.

`MEMBER FUNCTION getXML(`
`content OUT VARCHAR2)`
`RETURN XMLType;`
	This function returns the XMLType located at the address specified by the URL and the content type.

Parameter	IN / OUT	Description
content	(OUT)	Content type of the document to which URI is pointing.

Summary of HTTPURITYPE Subtype Subprograms

The HTTPURITYPE is a subtype of the UriType that provides support for the HTTP protocol. This uses the UTL_HTTP package underneath to access the HTTP URLs. Proxy and secure wallets are not supported in this release.

Table 185–2 HTTPURITYPE Type Subprorgams

Method	Description
CREATEURI on page 185-10	Creates an instance of HTTPURITYPE from the given URI.
GETBLOB on page 185-11	Returns the BLOB located at the address specified by the URL.
GETCLOB on page 185-12	Returns the CLOB located at the address specified by the URL.
GETCONTENTTYPE on page 185-13	Returns the content type of the document pointed to by the URI.
GETEXTERNALURL on page 185-14	Returns the URL, in escaped format, stored inside the UriType instance.
GETURL on page 185-7	Returns the URL, in non-escaped format, stored inside the UriType instance.
GETXML on page 185-16	Returns the XMLType located at the address specified by the URL
HTTPURITYPE on page 185-17	Creates an instance of HTTPURITYPE from the given URI.

CREATEURI

This static function constructs a HTTPURITYPE instance. The HTTPURITYPE instance does not contain the prefix http:// in the stored URL.

Syntax

```
STATIC FUNCTION createUri(
   url IN varchar2)
RETURN HTTPURITYPE;
```

Parameter	IN / OUT	Description
url	(IN)	The URL string containing a valid HTTP URL; escaped format.

GETBLOB

This function returns the BLOB located at the address specified by the HTTP URL.

Syntax	Description
MEMBER FUNCTION getBlob() RETURN blob;	This function returns the BLOB located at the address specified by the HTTP URL.
MEMBER FUNCTION getBlob(content OUT VARCHAR2) RETURN blob;	This function returns the BLOB located at the address specified by the HTTP URL and the content type.
FUNCTION getBlob(csid IN NUMBER) RETURN BLOB;	This function returns the BLOB located at the address specified by the URL in the specified character set.

Parameter	IN / OUT	Description
content	(OUT)	Content type of the document to which URI is pointing.
csid	(IN)	Character set id of the document. Must be a valid Oracle id and greater than 0; otherwise returns an error.

GETCLOB

This function returns the `CLOB` located by the HTTP URL address. If a temporary `CLOB` is returned, it must be freed.

Syntax	Description
`MEMBER FUNCTION getClob()` `RETURN clob;`	Returns the `CLOB` located at the address specified by the HTTP URL.

`MEMBER FUNCTION getClob(`
`content OUT VARCHAR2)`
`RETURN clob;`
	Returns the `CLOB` located at the address specified by the HTTP URL and the content type.

Parameter	IN / OUT	Description
`content`	`(OUT)`	Content type of the document to which URI is pointing.

GETCONTENTTYPE

Returns the content type of the document pointed to by the URI.

Syntax

```
MEMBER FUNCTION getContentType()
RETURN VARCHAR2;
```

GETEXTERNALURL

This function returns the URL, in escaped format, stored inside the `HTTPURITYPE` instance. The subtype instances override this member function. The `HTTPURITYPE` function does not store the prefix `http://`, but generates it for the external URL.

Syntax

```
MEMBER FUNCTION getExternalUrl()
RETURN varchar2;
```

GETURL

This function returns the URL, in non-escaped format, stored inside the `HTTPURITYPE` instance.

Syntax

```
MEMBER FUNCTION getUrl()
RETURN varchar2;
```

GETXML

This function returns the `XMLType` located at the address specified by the URL. An error is thrown if the address does not point to a valid XML document.

Syntax	Description
`MEMBER FUNCTION getXML()` `RETURN XMLType;`	This function returns the `XMLType` located at the address specified by the URL.

`MEMBER FUNCTION getXML(`
`content OUT VARCHAR2)`
`RETURN XMLType;`
	This function returns the `XMLType` located at the address specified by the URL and the content type.

Parameter	IN / OUT	Description
`content`	`(OUT)`	Content type of the document to which URI is pointing.

HTTPURITYPE

This constructs a `HTTPURITYPE` instance. The `HTTPURITYPE` instance does not contain the prefix `http://` in the stored URL.

Syntax

```
CONSTRUCTOR FUNCTION HTTPURITYPE(
   url IN varchar2);
```

Parameter	IN / OUT	Description
`url`	`(IN)`	The URL string containing a valid HTTP URL. The URL string is expected in escaped format. For example, non-url characters are represented as the hexadecimal value for the UTF-8 encoding of those characters.

Summary of DBURITYPE Subtype Subprogams

The DBURITYPE is a subtype of the UriType that provides support for DBUri-refs. A DBUri-ref is an intra-database URL that can be used to reference any row or row-column data in the database. The URL is specified as an XPath expression over a XML visualization of the database. The schemas become elements which contain tables and views. These tables and view further contain the rows and columns inside them.

Table 185–3 DBURITYPE Type Subprograms

Method	Description
CREATEURI on page 185-19	Constructs a DBURITYPE instance.
DBURITYPE on page 185-20	Creates an instance of DBURITYPE from the given URI.
GETBLOB on page 185-21	Returns the BLOB located at the address specified by the DBURITYPE instance.
GETCLOB on page 185-22	Returns the CLOB located at the address specified by the DBURITYPE instance.
GETCONTENTTYPE on page 185-23	Returns the content type of the document pointed to by the URI.
GETEXTERNALURL on page 185-24	Returns the URL, in escaped format, stored inside the DBURITYPE instance.
GETURL on page 185-25	Returns the URL, in non-escaped format, stored inside the DBURITYPE instance.
GETXML on page 185-26	Returns the XMLType located at the address specified by the URL

CREATEURI

This static function constructs a DBURITYPE instance. Parses the URL given and creates a DBURITYPE instance.

Syntax

```
STATIC FUNCTION createUri(
   url IN varchar2)
RETURN DBURITYPE;
```

Parameter	IN / OUT	Description
url	(IN)	The URL string, in escaped format, containing a valid DBURITYPE.

DBURITYPE

This constructs a DBURITYPE instance.

Syntax

```
CONSTRUCTOR FUNCTION DBURITYPE(
   url IN varchar2);
```

Parameter	IN / OUT	Description
url	(IN)	The URL string containing a valid DBURITYPE. The URL string is expected in escaped format. For example, non-URL characters are represented as the hexadecimal value for the UTF-8 encoding of those characters.

GETBLOB

This function returns the BLOB located at the address specified by the URL. The options are described in the following table.

Syntax	Description
MEMBER FUNCTION getBlob() RETURN blob;	This function returns the BLOB located at the address specified by the URL.
MEMBER FUNCTION getBlob(content OUT VARCHAR2) RETURN blob;	This function returns the BLOB located at the address specified by the URL and the content type.
FUNCTION getBlob(csid IN NUMBER) RETURN BLOB;	This function returns the BLOB located at the address specified by the URL in the specified character set.

Parameter	IN / OUT	Description
content	(OUT)	Content type of the document to which URI is pointing.
csid	(IN)	Character set id of the document. Must be a valid Oracle id and greater than 0; otherwise returns an error.

GETCLOB

This function returns the CLOB located at the address specified by the DBURITYPE instance. If a temporary CLOB is returned, it must be freed. The document returned may be an XML document or a text document. When the DBUri-ref identifies an element in the XPath, the result is a well-formed XML document. On the other hand, if it identifies a text node, then what is returned is only the text content of the column or attribute. The options are described in the following table.

Syntax	Description
MEMBER FUNCTION getClob() RETURN clob;	Returns the CLOB located at the address specified by the DBURITYPE instance.
MEMBER FUNCTION getClob(content OUT VARCHAR2) RETURN clob;	Returns the CLOB located at the address specified by the DBURITYPE instance and the content type.

Parameter	IN / OUT	Description
content	(OUT)	Content type of the document to which URI is pointing.

GETCONTENTTYPE

This function returns the content type of the document pointed to by the URI.

Syntax

```
MEMBER FUNCTION getContentType()
RETURN VARCHAR2;
```

GETEXTERNALURL

This function returns the URL, in escaped format, stored inside the `DBURITYPE` instance. The `DBUri` servlet URL that processes the `DBURITYPE` has to be appended before using the escaped URL in web pages.

Syntax

```
MEMBER FUNCTION getExternalUrl()
RETURN varchar2;
```

GETURL

This function returns the URL, in non-escaped format, stored inside the `DBURITYPE` instance.

Syntax

```
MEMBER FUNCTION getUrl()
RETURN varchar2;
```

GETXML

This function returns the XMLType located at the address specified by the URL. The options are described in the following table.

Syntax	Description
MEMBER FUNCTION getXML() RETURN XMLType;	This function returns the XMLType located at the address specified by the URL.
MEMBER FUNCTION getXML(content OUT VARCHAR2) RETURN XMLType;	This function returns the XMLType located at the address specified by the URL and the content type.

Parameter	IN / OUT	Description
content	(OUT)	Content type of the document to which URI is pointing.

Summary of XDBURITYPE Subtype Subprograms

XDBURITYPE is a new subtype of URIType. It provides a way to expose documents in the Oracle XML DB hierarchy as URIs that can be embedded in any URIType column in a table. The URL part of the URI is the hierarchical name of the XML document it refers to. The optional fragment part uses the XPath syntax, and is separated from the URL part by '#'. The more general XPointer syntax for specifying a fragment is not currently supported.

Table 185–4 XDBURITYPE Type Subprograms

Method	Description
CREATEURI on page 185-28	Returns the UriType corresponding to the specified URL.
GETBLOB on page 185-29	Returns the BLOB corresponding to the contents of the document specified by the XDBURITYPE instance.
GETCLOB on page 185-22	Returns the CLOB corresponding to the contents of the document specified by the XDBURITYPE instance.
GETCONTENTTYPE on page 185-31	Returns the content type of the document pointed to by the URI.
GETEXTERNALURL on page 185-24	Returns the URL, in escaped format, stored inside the XDBURITYPE instance.
GETURL on page 185-25	Returns the URL, in non-escaped format, stored inside the XDBURITYPE instance.
GETXML on page 185-34	Returns the XMLType corresponding to the contents of the document specified by the URL.
XDBURITYPE on page 185-35	Creates an instance of XDBURITYPE from the given URI.

CREATEURI

This static function constructs a XDBURITYPE instance. Parses the URL given and creates a XDBURITYPE instance.

Syntax

```
STATIC FUNCTION createUri(
   url IN varchar2)
RETURN XDBURITYPE
```

Parameter	IN / OUT	Description
url	(IN)	The URL string, in escaped format, containing a valid XDBURITYPE.

GETBLOB

This function returns the BLOB located at the address specified by the XDBURITYPE instance. The options are described in the following table.

Syntax	Description
MEMBER FUNCTION getBlob() RETURN blob;	This function returns the BLOB located at the address specified by the URL.
MEMBER FUNCTION getBlob(content OUT VARCHAR2) RETURN blob;	This function returns the BLOB located at the address specified by the URL and the content type.
FUNCTION getBlob(csid IN NUMBER) RETURN BLOB;	This function returns the BLOB located at the address specified by the URL in the specified character set.

Parameter	IN / OUT	Description
content	(OUT)	Content type of the document to which URI is pointing.
csid	(IN)	Character set id of the document. Must be a valid Oracle id and greater than 0; otherwise returns an error.

GETCLOB

This function returns the CLOB located at the address specified by the XDBURITYPE instance. If a temporary CLOB is returned, it must be freed. The options are described in the following table.

Syntax	Description
MEMBER FUNCTION getClob() RETURN clob;	Returns the CLOB located at the address specified by the XDBUirType instance.
MEMBER FUNCTION getClob(content OUT VARCHAR2) RETURN clob;	Returns the CLOB located at the address specified by the XDBUirType instance and the content type.

Parameter	IN / OUT	Description
content	(OUT)	Content type of the document to which URI is pointing.

GETCONTENTTYPE

This function returns the content type of the document pointed to by the URI. This function returns the content type as `VARCHAR2`.

Syntax

```
MEMBER FUNCTION getContentType()
RETURN VARCHAR2;
```

GETEXTERNALURL

This function returns the URL, in escaped format, stored inside the `XDBURITYPE` instance.

Syntax

```
MEMBER FUNCTION getExternalUrl()
RETURN varchar2;
```

GETURL

This function returns the URL, in non-escaped format, stored inside the `XDBURITYPE` instance.

Syntax

```
MEMBER FUNCTION getUrl()
RETURN varchar2;
```

GETXML

This function returns the XMLType located at the address specified by the URL. The options are described in the following table.

Syntax	Description
MEMBER FUNCTION getXML() RETURN XMLType;	This function returns the XMLType located at the address specified by the URL.
MEMBER FUNCTION getXML(content OUT VARCHAR2) RETURN XMLType;	This function returns the XMLType located at the address specified by the URL and the content type.

Parameter	IN / OUT	Description
content	(OUT)	Content type of the document to which URI is pointing.

XDBURITYPE

This constructs a `XDBURITYPE` instance.

Syntax

```
CONSTRUCTOR FUNCTION XDBURITYPE(
   url IN varchar2);
```

Parameter	IN / OUT	Description
`url`	`(IN)`	The URL string containing a valid `XDBUirType`. The URL string is expected in escaped format. For example, non-URL characters are represented as the hexadecimal value for the UTF-8 encoding of those characters.

Summary of URIFACTORY Package Subprograms

The `UriFactory` package contains factory methods that can be used to generate the appropriate instance of the URI types without having to hard code the implementation in the program.

The `UriFactory` package also provides the ability to register new subtypes of the `UriType` to handle various other protocols. For example, you can invent a new protocol `ecom://` and define a subtype of the `UriType` to handle that protocol and register it with `UriFactory`. After that any factory method would generate the new subtype instance if it sees the `ecom://` prefix.

Table 185–5 URIFACTORY Type Subprograms

Method	Description
GETURI on page 185-37	Returns the correct URL handler for the given URL string.
ESCAPEURI on page 185-38	Returns a URL in escaped format.
UNESCAPEURI on page 185-39	Returns a URL in unescaped format.
REGISTERURLHANDLER on page 185-40	Registers a particular type name for handling a particular URL.
UNREGISTERURLHANDLER on page 185-41	Unregisters a URL handler.

GETURI

This factory method returns the correct URI handler for the given URI string. It returns a subtype instance of the `UriType` that can handle the protocol. By default, it always creates an `XDBURITYPE` instance, if it cannot resolve the URL. A URL handler can be registered for a particular prefix using the REGISTERURLHANDLER function. If the prefix matches, GETURI would then use that subtype.

Syntax

```
FUNCTION getUri(
   url IN Varchar2)
RETURN UriType;
```

Parameter	IN / OUT	Description
`uri`	`(IN)`	The URL string, in escaped format, containing a valid HTTP URL.

ESCAPEURI

This function returns a URL in escaped format. The subtype instances override this member function to provide additional semantics. For instance, the `HTTPURITYPE` does not store the prefix `http://` in the URL itself. When generating the external URL, it appends the prefix and generates it. For this reason, use the GETEXTERNALURL function or the GETURI function to get to the URL value instead of using the attribute present in the `UriType`.

Syntax

```
MEMBER FUNCTION escapeUri()
RETURN varchar2;
```

Parameter	IN / OUT	Description
`url`	`(IN)`	The URL string to be returned in escaped format.

UNESCAPEURI

This function returns a URL in unescaped format. This function is the reverse of the ESCAPEURI function. This function scans the string and converts any non-URL hexadecimal characters into the equivalent UTF-8 characters. Since the return type is a VARCHAR2, the characters would be converted into the equivalent characters as defined by the database character set.

Syntax

```
FUNCTION unescapeUri()
RETURN varchar2;
```

Parameter	IN / OUT	Description
url	(IN)	The URL string to be returned in unescaped format.

REGISTERURLHANDLER

Registers a particular type name for handling a particular URL. The type specified must be valid and must be a subtype of the `UriType` or one of its subtypes. It must also implement the `createUri` static member function. This function is called by the GETURI function to generate an instance of the type. The `stripprefix` parameter indicates that the prefix must be stripped off before calling this function.

Syntax

```
PROCEDURE registerUrlHandler(
   prefix IN varchar2,
   schemaName IN varchar2,
   typename IN varchar2,
   ignoreCase IN boolean := true,
   stripprefix IN boolean := true);
```

Parameter	IN / OUT	Description
`prefix`	`(IN)`	The prefix to handle; for example, `http://`.
`schemaName`	`(IN)`	Name of the schema where the type resides; case sensitive.
`typename`	`(IN)`	The name of the type to handle the URL; case sensitive.
`ignoreCase`	`(IN)`	Ignore case when matching prefixes.
`stripprefix`	`(IN)`	Strip prefix before generating the instance of the type.

UNREGISTERURLHANDLER

This procedure unregisters a URL handler. This only unregisters user registered handler prefixes and not predefined system prefixes such as `http://`.

Syntax

```
PROCEDURE unregisterUrlHandler(
   prefix IN varchar2);
```

Parameter	IN / OUT	Description
`prefix`	`(IN)`	The prefix to be unregistered.

186

Expression Filter Types

The Expression Filter feature is supplied with a set of predefined types and public synonyms for these types. Most of these types are used for configuring index parameters with the Expression Filter procedural APIs. The `EXF$TABLE_ALIAS` type is used to support expressions defined on one or more database tables.

> **See Also:** *Oracle Database Application Developer's Guide - Rules Manager and Expression Filter* for more information.

This chapter contains the following topics:

Summary of Expression FilterTypes

Table 186–1 describes the Expression Filter object types.

All the values and names passed to the types defined in this chapter are not case sensitive. To preserve the case, you use double quotation marks around the values.

Table 186–1 Expression Filter Object Types

Object Type Name	Description
EXF$ATTRIBUTE	Specifies the stored and indexed attributes for the Expression Filter indexes
EXF$ATTRIBUTE_LIST	Specifies a list of stored and indexed attributes when configuring index parameters
EXF$INDEXOPER	Specifies a list of common operators in predicates with a stored or an indexed attribute
EXF$TABLE_ALIAS	Indicates a special form of elementary attribute used to manage expressions defined on one or more database tables
EXF$XPATH_TAG	Configures an XML element or an XML attribute for indexing a set of XPath predicates
EXF$XPATH_TAGS	Specifies a list of XML tags when configuring the Expression Filter index parameters

EXF$ATTRIBUTE

The EXF$ATTRIBUTE type is used to handle stored and indexed attributes for the Expression Filter indexes.

Syntax

```
CREATE or REPLACE TYPE EXF$ATTRIBUTE AS OBJECT attr_name    VARCHAR2(350),
                                               attr_oper    EXF$INDEXOPER,
                                               attr_indexed VARCHAR2(5);
```

Attributes

Table 186–2 EXF$ATTRIBUTE Object Type Attributes

Attribute	Description
attr_name	The arithmetic expression that constitutes the stored or indexed attribute
attr_oper	The list of common operators in the predicates with the attribute. Default value: EXF$INDEXOPER('all')
attr_indexed	TRUE if the attribute is indexed, else FALSE. Default value: FALSE.

Usage Notes

The EXF$ATTRIBUTE type is used to specify the stored and indexed attributes for an Expression Filter index using the DBMS_EXPFIL.DEFAULT_INDEX_PARAMETERS procedure. When values for attr_oper and attr_indexed fields are omitted during EXF$ATTRIBUTE instantiation, it is considered a stored attribute with a default value for common operators (EXF$INDEXOPER('all')).

Examples

A stored attribute with no preference on the list of common operators is represented as follows:

```
exf$attribute (attr_name => 'HorsePower(Model, Year)')
```

An indexed attribute is represented as follows:

```
exf$attribute (attr_name => 'HorsePower(Model, Year)',
               attr_indexed => 'TRUE')
```

An indexed attribute with a list of common operators is represented as follows:

```
exf$attribute (attr_name => 'HorsePower(Model, Year)',
               attr_oper => exf$indexoper('=','<','>','>=','<='),
               attr_indexed => 'TRUE')
```

EXF$ATTRIBUTE_LIST

The `EXF$ATTRIBUTE_LIST` type is used to specify a list of stored and indexed attributes while configuring the index parameters.

Syntax

```
CREATE or REPLACE TYPE EXF$ATTRIBUTE_LIST as VARRAY(490) of exf$attribute;
```

Attributes

None.

Usage Notes

Also see the "DEFAULT_INDEX_PARAMETERS Procedure" for more information

Examples

A list of stored and indexed attributes can be represented as follows:

```
exf$attribute_list (
       exf$attribute (attr_name => 'Model',
                      attr_oper => exf$indexoper('='),
                      attr_indexed => 'TRUE'),
       exf$attribute (attr_name => 'Price',
                      attr_oper => exf$indexoper('all'),
                      attr_indexed => 'TRUE'),
       exf$attribute (attr_name => 'HorsePower(Model, Year)',
                      attr_oper => exf$indexoper('=','<','>','>=','<='),
                      attr_indexed => 'FALSE')
    )
```

EXF$INDEXOPER

The EXF$INDEXOPER type is used to specify the list of common operators in predicates with a stored or an indexed attribute.

Syntax

```
CREATE or REPLACE TYPE EXFSYS.EXF$INDEXOPER as VARRAY(20) of VARCHAR2(15);
```

The values for the EXF$INDEXOPER array are expected to be from the list in the following table:

Value	Predicate Description
=	Equality predicates
>	Greater than predicates
<	Less than predicates
>=	Greater than or equal to predicates
<=	Less than or equal to predicates
!= or <> or ^=	Not equal to predicates
IS NULL	IS NULL predicates
IS NOT NULL	IS NOT NULL predicates
ALL	All the operators listed in this table starting with the equality predicate through the IS NOT NULL predicate
NVL	Predicates with NVL (equality) operator
LIKE	Predicates with LIKE operator
BETWEEN	BETWEEN predicates

Attributes

None.

Usage Notes

- A value of ALL for one of the EXF$INDEXOPER items implies that all the simple operators (=,>,<,>=,<=,!=, IS NULL, IS NOT NULL) are common in the predicates with an attribute. This value can be used along with one or more complex operators (NVL, LIKE and BETWEEN).
- A predicate with a BETWEEN operator is treated as two predicates with binary operators, one with '>=' operator and another with '<=' operator. By default, only one of these operators is indexed, and the other operator is evaluated by value substitution. However, if predicates with the BETWEEN operator are common for an attribute (stored or indexed), both the binary operators resulting from the BETWEEN operator can be indexed by specifying BETWEEN in the EXF$INDEXOPER VARRAY. However, because this uses additional space in the predicate table, this operator should be used only when majority of predicates with an attribute use the BETWEEN operator.
- When the LIKE operator is chosen as one of the common operators for an attribute, LIKE predicates on that attributes are indexed. Indexing a LIKE operator is beneficial only if the VARCHAR2 constant on the right-hand side of the

predicate does not lead with a wild-card character. For example, indexing a `LIKE` operator will filter the following predicates efficiently:

```
company LIKE 'General%'
company LIKE 'Proctor%'
```

But, the following predicates are evaluated as sparse predicates in the last stage:

```
company LIKE '%Electric'
company LIKE "%Gamble'
```

Examples

An attribute with a list of common operators is represented as follows:

```
exf$attribute (attr_name => 'HorsePower(Model, Year)',
               attr_oper => exf$indexoper('=','<','>','>=','<=', 'between'),
               attr_indexed => 'TRUE')
```

EXF$TABLE_ALIAS

A EXF$TABLE_ALIAS type is a special form of elementary attribute that can be included in the attribute set. These attributes are used to manage expressions defined on one or more database tables.

Syntax

```
CREATE or REPLACE TYPE EXF$TABLE_ALIAS AS OBJECT table_name VARCHAR2(70);
```

Attributes

Table 186–3 EXF$TABLE_ALIAS Attribute

Attribute	Description
table_name	Name of the table with a possible schema extension

Usage Notes

The concept of a table alias attribute is captured in the Expression Filter dictionary and the corresponding attribute in the attribute set's object type is created with a VARCHAR2 datatype. (Also see Appendix A in Oracle Database Application Developer's Guide - Rules Manager and Expression Filter and "ADD_ELEMENTARY_ATTRIBUTE Procedures".)

Examples

For a set of expressions defined on database tables, the corresponding table alias attributes are configured as follows:

```
BEGIN
  DBMS_EXPFIL.ADD_ELEMENTARY_ATTRIBUTE (
                                attr_set  => 'HRAttrSet',
                                attr_name => 'EMP',
                                tab_alias => exf$table_alias('SCOTT.EMP'));
  DBMS_EXPFIL.ADD_ELEMENTARY_ATTRIBUTE (
                                attr_set  => 'HRAttrSet',
                                attr_name => 'DEPT',
                                tab_alias => exf$table_alias('DEPT'));
END;
```

The Expression column using the previous attribute set can store expressions of form EMP.JOB = 'Clerk' and EMP.NAME = 'Joe', where JOB and NAME are the names of the columns in the SCOTT.EMP table.

EXF$XPATH_TAG

The EXF$XPATH_TAG type is used to configure an XML element or an XML attribute for indexing a set of XPath predicates.

Syntax

```
CREATE or REPLACE TYPE EXF$XPATH_TAG AS OBJECT tag_name    VARCHAR2(350),
                                               tag_indexed VARCHAR2(5),
                                               tag_type    VARCHAR2(30);
```

Attributes

Table 186–4 EXF$XPATH_TAG Attributes

Attribute	Description
tag_name	Name of the XML element or attribute. The name for an XML attribute is formatted as: <ElementName>@<AttributeName>. Optionally, the element name can be prefixed with its namespace URL as in <Namespace URL>:<ElementName>@<AttributeName>.
tag_indexed	TRUE if XML tag is indexed; otherwise FALSE. Default: TRUE if the tag is a positional filter. FALSE if the tag is a value filter.
tag_type	Datatype for the value in the case of value filter. NULL for positional filters.

Usage Notes

EXF$XPATH_TAG type is used to configure an XML element or an attribute as a positional or a value filter for an Expression Filter index (see "Index Tuning for XPath Predicates" in Oracle Database Application Developer's Guide - Rules Manager and Expression Filter). An instance of the EXF$XPATH_TAG type with NULL value for tag_type configures the XML tag as a positional filter. In the current release, the only other possible values for the tag_type attribute are strings (CHAR or VARCHAR) and such tags are configured as value filters. By default, all positional filters are indexed and the value filters are not indexed. This behavior can be overridden by setting a TRUE or FALSE value for the tag_indexed attribute accordingly.

Examples

An XML element can be configured as a positional filter and be indexed using the following instance of the EXF$XPATH_TAG type.

```
exf$xpath_tag(tag_name    => 'stereo',       --- XML element
              tag_indexed => 'TRUE',         --- indexed predicate group
              tag_type    => null)           --- positional filter
```

An XML attribute can be configured as a value filter and be indexed using the following type instance.

```
exf$xpath_tag(tag_name    => 'stereo@make',  --- XML attribute
              tag_indexed => 'TRUE',         --- indexed predicate group
              tag_type    => 'VARCHAR(15)')  --- value filter
```

The following commands configure the two filters shown previously using the namespace URL for the corresponding elements.

```
exf$xpath_tag(tag_name => 'http://www.auto.com/car.xsd:stereo',
```

```
                tag_indexed => 'TRUE', --- indexed predicate group
                tag_type => null)      --- positional filter

exf$xpath_tag(tag_name => 'http://www.auto.com/car.xsd:stereo@make'
                tag_indexed => 'TRUE',     --- indexed predicate group
                tag_type => 'VARCHAR(15)') --- value filter
```

EXF$XPATH_TAGS

The EXF$XPATH_TAGS type is used to specify a list of XML tags while configuring the Expression Filter index parameters.

Syntax

```
CREATE or REPLACE TYPE EXF$XPATH_TAGS as VARRAY(490) of EXF$XPATH_TAG;
```

Attributes

None.

Usage Notes

EXF$XPATH_TAGS type is used to specify a list of XML tags while configuring the Expression Filter index parameters. (See "DEFAULT_INDEX_PARAMETERS Procedure".)

Examples

A list of XML tags configured as positional and value filters can be represented as follows:

```
exf$xpath_tags(
           exf$xpath_tag(tag_name    => 'stereo@make',  --- XML attribute
                         tag_indexed => 'TRUE',
                         tag_type    => 'VARCHAR(15)'), --- value filter
           exf$xpath_tag(tag_name    => 'stereo',       --- XML element
                         tag_indexed => 'FALSE',
                         tag_type    => null),          --- positional filter
           exf$xpath_tag(tag_name    => 'memory',       --- XML element
                         tag_indexed => 'TRUE',
                         tag_type    => 'VARCHAR(10)')  --- value filter
          )
```

187

JMS Types

PL/SQL users can use the `DBMS_AQ` package to enqueue and dequeue messages from JMS queues. The JMS types member and static functions and procedures in this chapter are needed to populate JMS messages for enqueuing or to interpret a dequeued JMS message.

This chapter contains these topics:

Using JMS Types

Overview

Java Message Service (JMS) is a well known public standard interface for accessing messaging systems. Oracle JMS (OJMS) implements JMS based on Oracle Streams Advanced Queuing (AQ) and a relational database system (RDBMS). Messages are stored in queues as OJMS specific ADTs. Java clients use OJMS packages to enqueue, dequeue, and manipulate these messages.

PL/SQL users, on the other hand, use the `DBMS_AQ` package to enqueue and dequeue JMS messages and the member functions in this chapter to populate and interpret them. Oracle Streams AQ offers such member functions for the following JMS ADTs:

- `aq$_jms_header`
- `aq$_jms_message`
- `aq$_jms_text_message`
- `aq$_jms_bytes_message`
- `aq$_jms_map_message`
- `aq$_jms_stream_message`

In addition to these populating and interpreting member functions, Oracle Streams AQ offers:

- Casting between `aq$_jms_message` and other message ADTs.
- PL/SQL stored procedures for converting JMS selectors to equivalent Oracle Streams AQ rules

Java Versus PL/SQL Data Types

Data types do not map one-to-one between PL/SQL and Java.

Some Java types, such as `BYTE` and `SHORT`, are not present in PL/SQL. PL/SQL type `INT` was chosen to represent these types. If a PL/SQL `INT` value intended to hold a Java `BYTE` or `SHORT` value exceeds the corresponding range Java enforces, an out-of-range error is thrown.

Other Java types have more than one counterpart in PL/SQL with different capabilities. A Java String can be represented by both `VARCHAR2` and `CLOB`, but `VARCHAR2` has a maximum limit of 4000 bytes. When retrieving `TEXT` data from map, stream, and bytes message types, a `CLOB` is always returned. When updating the map, stream and bytes message types, users can submit either a `VARCHAR2` or `CLOB`.

Similarly, a Java `BYTE ARRAY` can be represented by both `RAW` and `BLOB`, with `RAW` having a maximum size of 32767. When retrieving `BYTE ARRAY` data from map, stream, and bytes message types, a `BLOB` is always returned. When updating the map, stream and bytes message types, users can submit either a `RAW` or `BLOB`.

> **See Also:** JMS specification 3.11.3, Conversion Provided by StreamMessage and MapMessage

New JMS Support in Oracle Database 10*g*

In Oracle Database 10*g*, a new `AQ$_JMS_VALUE` ADT has been added in the `SYS` schema for OJMS PL/SQL users. It is specifically used to implement the `read_object` procedure of `aq$_jms_stream_message` and `get_object` procedure of `aq$_jms_map_message`, to mimic the Java general object class `Object`. `AQ$_JMS_VALUE` ADT can represent any data type that JMS `StreamMessage` and `MapMessage` can hold.

The collection ADT `AQ$_JMS_NAMEARRAY` was added for the `getNames` method of `MapMessage`. It holds an array of names.

In this release the ADT `AQ$_JMS_EXCEPTION` was added to represent a Java exception thrown in an OJMS JAVA stored procedure on the PL/SQL side. Now you can retrieve a Java exception thrown by an OJMS stored procedure and analyze it on the PL/SQL side.

More on Bytes, Stream and Map Messages

Oracle uses Java stored procedure to implement some of the procedures of `AQ$_MAP_MESSAGE`, `AQ$_JMS_STREAM_MESSAGE`, and `AQ$_JMS_BYTES_MESSAGE` types. These types have some common functionalities that are different from `AQ$_JMS_TEXT_MESSAGE` type. This section discusses these common functionalities.

This section contains these topics:

- Using Java Stored Procedures to Encode and Decode Oracle Streams AQ Messages
- Read-Only and Write-Only Modes Enforced for Stream and Bytes Messages
- Differences Between Bytes and Stream Messages
- Getting and Setting Bytes, Map, and Stream Messages as RAW Bytes

Using Java Stored Procedures to Encode and Decode Oracle Streams AQ Messages

The major difference between map, stream, bytes, and other messages is that the message payload is encoded as a byte stream by JAVA. Retrieving and updating these payloads in PL/SQL therefore requires Oracle JAVA stored procedures.

A message payload is stored in two places during processing. On the PL/SQL side it is stored as the data members of a JMS message ADT, and on the Jserv side it is stored as a static variable. (Jserv is the JVM inside Oracle Database.) When the payload is processed, the payload data is first transformed to a static variable on the Jserv side. Once the static variable is initialized, all later updates on the message payload are performed on this static variable. At the end of processing, payload data is flushed back to the PL/SQL side.

Oracle provides member procedures that maintain the status of the Jserv static variable and enforce rules when calling these member procedures. These procedures are in the following ADTs:

- `aq$_jms_bytes_message`
- `aq$_jms_map_message`
- `aq$_jms_stream_message`

Initialize the Jserv Static Variable

Before you make any other calls to manipulate the payload data, the Jserv static variable must be properly initialized. This is done by calling the `prepare` or `clear_body` procedure. The `prepare` procedure uses the payload data in PL/SQL ADTs to initialize the static variable, while `clear_body` initializes the static variable to an empty payload (empty hashtable or stream).

Note: It is important to call the `prepare` or `clear_body` procedure before any other calls to properly initialize the Jserv static variables. Usually these two methods are called once at the beginning. But they can be called multiple times for one message. Any call of these two methods without first calling the `flush` procedure wipes out all updates made to the messages.

Get the Payload Data Back to PL/SQL

Calling the `flush` procedure synchronizes changes made to the Jserv static variable back to the PL/SQL ADTs. The `flush` call is required when you want the changes made to be reflected in the ADT payload. It is important to synchronize the changes back to the ADT, because it is the ADT payload that matters.

Garbage Collect the Static Variable

The `clean` procedure forces garbage collection of the static variable. It is there to do cleanup and free JVM memory. You can avoid memory leaks by doing it immediately after finishing processing the message.

Use a Message Store: A Static Variable Collection

Instead of a single static variable, Oracle uses a collection of static variables to process the message payload on the Jserv side. This collection is called the message store. Each map, bytes, or stream message type has its own message store within one session.

Oracle uses the operation ID parameter to locate the correct static variable to work on within the message store. Initialization calls such as `prepare` and `clear_body` give users an operation ID, which is used in later message access.

After users complete message processing, they must call the `clean` procedure with the operation ID to clean up the message store. This avoids possible memory leaks. The `clean_all` static procedures of message ADTs `aq$_jms_bytes_message`, `aq$_jms_map_message`, and `aq$_jms_stream_message` clean up all static variables of their corresponding message stores.

Typical Calling Sequences

This section describes typical procedures for retrieving and populating messages.

Here is a typical procedure for retrieving messages

1. Call `prepare` for a message.

   This call also gives you an operation ID if you do not specify one.
2. Call multiple retrieving procedures with the provided operation ID.
3. Call the `clean` procedure with the provided operation ID.

Here is a typical procedure for populating messages:

1. Call `clear_body` for a message.

   For `aq$_jms_map_message`, you can also call `prepare` to update the message based on the existing payload. This call also gives you an operation ID if you do not specify one.
2. Call multiple updating procedures with the provided operation ID.
3. Call the `flush` method with the provided operation ID.
4. Call the `clean` procedure with the provided operation ID.

Read-Only and Write-Only Modes Enforced for Stream and Bytes Messages

According to the JMS specification, when a message is received, its body is read-only. Users can call the `clear_body` method to make the body writable. This method erases the current message body and sets the message body to be empty.

The OJMS JAVA API follows the rule set by JMS specification. In updating the JMS message ADTs in PL/SQL, however, Oracle enforces the rule selectively:

- Map messages

  The restriction is relaxed, because adding more entries on top of a existing map payload is a convenient way for users to update the payload. Therefore there are no read-only or write-only modes for map messages.

- Stream and bytes messages

  The restriction is not relaxed, because these payloads use a stream when reading and writing data. It is difficult to update the payload while in the middle of a stream. Oracle enforces read-only and write-only modes in processing stream and bytes message payloads. Calling the `prepare` procedure initializes the message payload in read-only mode. Calling the `clear_body` procedure initializes the message payload in write-only mode.

  Calling the `reset` procedure resets the pointer to the beginning of the stream and switches the mode from write-only to read-only. The `reset` procedure keeps the updates made to the message payload in the Jserv static variable.

  The `prepare` procedure, on the other hand, overwrites the message payload in the Jserv static variable with the payload in the PL/SQL ADT.

  Oracle provides member function `get_mode` for users to query the mode.

Differences Between Bytes and Stream Messages

Member functions of bytes messages are not exactly the same as those of stream messages. Stream messages are encoded using Java `ObjectOutputStream` and bytes messages are encoded using Java `DataOutputStream`. In stream messages each primitive type is written and read as a Java Object, but in a bytes message they are written and read as raw bytes according to the encoding mechanism of `DataOutputStream`.

For stream messages, the `read_bytes` method works on a stream of bytes to the end of the byte array field written by the corresponding `write_bytes` method. The `read_bytes` method of bytes message works on a stream of bytes to the end of the whole byte stream. This is why the `read_bytes` member procedure of `aq$_bytes_message` also requires a `length` parameter to tell how long it is to read.

You will not see a type conversion error raised by bytes message, because bytes messages do not support type conversion.

Methods `get_unsigned_byte` and `get_unsigned_short` are available for bytes messages, but not for stream messages. This is because stream messages read Java objects, and there are no Java objects as unsigned bytes or unsigned shorts.

Methods `read_string` and `write_string` methods are not available for bytes messages. The bytes message ADT must enforce some character encoding. It has methods `read_utf` and `write_utf` which support `utf-8` encoding.

Note: All data written by bytes messages use `DataOutputStream` as the basis. See JDK API documentation JavaSoft.com for details on how the data is encoded into bytes.

Getting and Setting Bytes, Map, and Stream Messages as RAW Bytes

The payloads of bytes, map, and stream message types are stored as either `RAW` or `BLOB` in the database. In this release Oracle Streams AQ provides the following member functions to set and get these payloads as raw bytes without interpreting them:

```
set_bytes(payload IN BLOB)
set_bytes(payload IN RAW)
get_bytes(payload OUT BLOB)
get_bytes(payload OUT RAW)
```

These functions were provided for bytes messages in Oracle9*i* Release 2 (9.2).

Upcasting and Downcasting Between General and Specific Messages

OJMS ADT `aq$_jms_message` is used to represent a general message, so that different types of messages can reside on the same Oracle Streams AQ queue. Oracle Streams AQ supports retrieving and populating of `aq$_jms_message` by supporting upcasting and downcasting between this ADT and ADTs of specific message types.

To read an `aq$_jms_message`, you must first downcast it to a specific message type according to its `message_type` field

To populate an `aq$_jms_message`, you must first populate a specific message and upcast it to `aq$_jms_message`. This avoids copying all member functions of other specific message ADTs to this ADT. It also guarantees that the manipulation of this ADT is consistent with other specific message ADTs.

JMS Types Error Reporting

Table 187–1 lists Oracle JMS types related errors.

Table 187–1 Oracle JMS Types Errors

ORA error number	dbms_jms_plsql package constants	Explanation
ORA-24190	ERROR_DATA_OVERFLOW	The payload data exceeds the size that an out parameter can hold. For example, the get_text procedure with a VARCHAR2 parameter of aq$_jms_text_message or get_bytes procedure with a RAW parameter of aq$_jms_bytes_message.
ORA-24191	ERROR_PROP_NAME_EXIST	Setting a property that is previous set
ORA-24192	ERROR_PROP_NAME_NULL	Occurs when setting a property with null property name.
ORA-24193	ERROR_EXCEED_RANGE	PL/SQL number type exceeds the valid range of the respective Java type. For example set_byte_property, set_short_property of aq$_jms_head ADT; set_byte and set_short of aq$_jms_map_message ADT; write_byte and write_short of aq$_jms_stream_message and aq$_jms_bytes_message ADT.
ORA-24194	ERROR_TYPE_MISMATCH	The type conversion between the Java type of the retrieving method and the Java type of a field of the payload is not valid.
ORA-24195	ERROR_MAP_TOO_LARGE	The size of the map exceeds the aq$_jms_namearray ADT capacity. The current size limit is 1024. You can use the get_names function with offset and length parameters to retrieve the name array in multiple small chunks.
ORA-24196	ERROR_WRONG_MODE	The message payload is being accessed with a wrong access mode. For example, trying to read a message payload with write-only mode or trying to write a message payload with the read-only mode.
ORA-24197	ERROR_JAVA_EXCEPTION	ORA-24197 error is raised when a Java exception is raised that does not fit in any of the other error categories. You can use the get_exception static procedure of aq$_jms_map_message, aq$_jms_bytes_message, and aq$_jms_stream_message to retrieve the exception information last thrown by the Java stored procedure. A single static variable is used to store the last exception and is overwritten if another exception is thrown before you retrieve it. A new ADT aq$_jms_exception is created to represent the exception information on the PL/SQL side.
ORA-24198	ERROR_INVALID_ID	An invalid operation ID is being provided to access a message.
ORA-24199	ERROR_STORE_OVERFLOW	The number of messages (with the same type) that users are trying to manipulate exceeds the size of the message store on the Java stored procedure side. The current size of the store is 20. It unusual to need to manipulate more than 20 messages at the same time. A common mistake is to forget to call the clean procedure after using one message. The clean procedure frees the message slot for use by other messages attempting access.

Oracle JMS Type Constants

This section lists some useful constants when dealing with message type functions.

DBMS_AQ Package Constants

DBMS_AQ package constants specify different types of JMS messages. They are useful when dealing with general message types during upcasting and downcasting or constructing a general message with a specific message type:

```
JMS_TEXT_MESSAGE   CONSTANT BINARY_INTEGER;
JMS_BYTES_MESSAGE  CONSTANT BINARY_INTEGER;
JMS_STREAM_MESSAGE CONSTANT BINARY_INTEGER;
JMS_MAP_MESSAGE    CONSTANT BINARY_INTEGER;
JMS_OBJECT_MESSAGE CONSTANT BINARY_INTEGER;
```

SYS.DBMS_JMS_PLSQL Package Constants

SYS.DBMS_JMS_PLSQL package constants are new in Oracle Database 10*g*.

These constants specify the mode of message payload. They are useful when interpreting the mode of the message payload returned from the get_mode function:

```
MESSAGE_ACCESS_READONLY  CONSTANT PLS_INTEGER;
MESSAGE_ACCESS_WRITEONLY CONSTANT PLS_INTEGER;
```

These constants specify the ADT type of an Oracle Streams AQ queue. They are useful during the conversion of JMS selectors to Oracle Streams AQ rules:

```
DESTPLOAD_JMSTYPE CONSTANT PLS_INTEGER;
DESTPLOAD_USERADT CONSTANT PLS_INTEGER;
DESTPLOAD_ANYDATA CONSTANT PLS_INTEGER;
```

These constants specify the type of data that can be held by a aq$_jms_value type. They are useful when interpreting the aq$_jms_value returned by the get_object method of AQ$_JMS_MAP_MESSAGE or read_object method of AQ$_JMS_STREAM_MESSAGE:

```
DATA_TYPE_BYTE           CONSTANT PLS_INTEGER;
DATA_TYPE_SHORT          CONSTANT PLS_INTEGER;
DATA_TYPE_INTEGER        CONSTANT PLS_INTEGER;
DATA_TYPE_LONG           CONSTANT PLS_INTEGER;
DATA_TYPE_FLOAT          CONSTANT PLS_INTEGER;
DATA_TYPE_DOUBLE         CONSTANT PLS_INTEGER;
DATA_TYPE_BOOLEAN        CONSTANT PLS_INTEGER;
DATA_TYPE_CHARACTER      CONSTANT PLS_INTEGER;
DATA_TYPE_STRING         CONSTANT PLS_INTEGER;
DATA_TYPE_BYTES          CONSTANT PLS_INTEGER;
DATA_TYPE_UNSIGNED_BYTE  CONSTANT PLS_INTEGER;
DATA_TYPE_UNSIGNED_SHORT CONSTANT PLS_INTEGER;
```

These constants specify the error number of the ORA errors that can be raised by the functions of message type ADTs. They are useful in user error handlers:

```
ERROR_DATA_OVERFLOW   CONSTANT PLS_INTEGER := -24190;
ERROR_PROP_NAME_EXIST CONSTANT PLS_INTEGER := -24191;
ERROR_PROP_NAME_NULL  CONSTANT PLS_INTEGER := -24192;
ERROR_EXCEED_RANGE    CONSTANT PLS_INTEGER := -24193;
ERROR_TYPE_MISMATCH   CONSTANT PLS_INTEGER := -24194;
ERROR_MAP_TOO_LARGE   CONSTANT PLS_INTEGER := -24195;
ERROR_WRONG_MODE      CONSTANT PLS_INTEGER := -24196;
```

```
ERROR_JAVA_EXCEPTION  CONSTANT PLS_INTEGER := -24197;
ERROR_INVALID_ID      CONSTANT PLS_INTEGER := -24198;
ERROR_STORE_OVERFLOW  CONSTANT PLS_INTEGER := -24199;
```

CONVERT_JMS_SELECTOR

Oracle Database includes three stored procedures to help users convert JMS selectors into Oracle Streams AQ rules. These rules can be used in `ADD_SUBSCRIBER` operations as subscriber rules or in `DEQUEUE` operations as dequeue conditions. These procedures are in the `SYS.dbms_jms_plsql` package.

Convert with Minimal Specification

The first procedure assumes the destination payload type is one of the JMS ADTs whose corresponding constant is `dbms_jms_plsql.DESTPLOAD_JMSTYPE` and also assumes that the J2EE compliant mode is true.

Syntax

```
Function convert_jms_selector(selector IN VARCHAR2) RETURN VARCHAR2
```

Returns

The converted Oracle Streams AQ rule or null if there is any conversion error.

Exceptions

ORA-24197 if the Java stored procedure throws an exception during execution.

Convert with Destination Payload Type Specified

The second procedure takes one more parameter: `dest_pload_type`. The conversion of a JMS selector to an Oracle Streams AQ rule happens only if this parameter is `SYS.dbms_jms_plsql.DESTPLOAD_JMSTYPE` or `SYS.dbms_jms_plsql.DESTPLOAD_ANYDATA`. The function returns exactly the same `VARCHAR2` value as the selector parameter if the `dest_pload_type` parameter is `SYS.dbms_jms_plsql.DESTPLOAD_USERADT`. The function returns null if `dest_pload_type` parameter is none of these three constants.

This function assumes that the J2EE compliant mode is true.

Syntax

```
Function convert_jms_selector(
   selector IN VARCHAR2,
   dest_pload_type IN PLS_INTEGER)
RETURN VARCHAR2
```

Returns

The converted Oracle Streams AQ rule or null if there is any conversion error.

Exceptions

ORA-24197 if the Java stored procedure throws an exception during execution.

Convert with Destination Payload Type and Compliant Mode Specified

The third procedure takes a `dest_pload_type` parameter and a `compliant` parameter. The conversion of a JMS selector to an Oracle Streams AQ rule happens only if the `dest_pload_type` parameter is `SYS.dbms_jms_plsql.DESTPLOAD_JMSTYPE` or `SYS.dbms_jms_plsql.DESTPLOAD_ANYDATA`. The function returns exactly the same `VARCHAR2` value as the selector parameter if the `dest_pload_type`

parameter is `SYS.dbms_jms_plsql.DESTPLOAD_USERADT`. The function returns null if the `dest_pload_type` parameter is none of these three constants.

The `compliant` parameter controls if the conversion is in J2EE compliant mode or not. The noncompliant conversion of a JMS selector is for backward compatibility.

Syntax

```
Function convert_jms_selector(
   selector         IN  VARCHAR2,
   dest_pload_type  IN  PLS_INTEGER,
   compliant        IN  BOOLEAN )
```

Returns

The converted Oracle Streams AQ rule or null if there is any conversion error.

Exceptions

ORA-24197 if the Java stored procedure throws an exception during execution.

Summary of JMS Types

- SYS.AQ$_JMS_MESSAGE Type
- SYS.AQ$_JMS_TEXT_MESSAGE Type
- SYS.AQ$_JMS_BYTES_MESSAGE Type
- SYS.AQ$_JMS_MAP_MESSAGE Type
- SYS.AQ$_JMS_STREAM_MESSAGE Type
- SYS.AQ$_JMS_OBJECT_MESSAGE Type
- SYS.AQ$_JMS_NAMESARRAY Type
- SYS.AQ$_JMS_VALUE Type
- SYS.AQ$_JMS_EXCEPTION Type

SYS.AQ$_JMS_MESSAGE Type

This ADT type can represent any of five different JMS message types: text message, bytes message, stream message, map message, or object message. Queues created using this ADT can therefore store all five types of JMS messages.

This section contains these topics:

Syntax

```
TYPE AQ$_JMS_MESSAGE AS OBJECT(
 header        aq$_jms_header,
 senderid      varchar2(100),
 message_type  INT,
 text_len      INT,
 bytes_len     INT,
 text_vc       varchar2(4000),
 bytes_raw     raw(2000),
 text_lob      clob,
 bytes_lob     blob,
 STATIC FUNCTION  construct (mtype      IN  INT)
   RETURN aq$_jms_message,
 STATIC FUNCTION  construct (text_msg   IN  aq$_jms_text_message)
   RETURN aq$_jms_message,
 STATIC FUNCTION  construct (bytes_msg  IN  aq$_jms_bytes_message)
   RETURN aq$_jms_message,
 STATIC FUNCTION  construct (stream_msg IN  aq$_jms_stream_message)
   RETURN aq$_jms_message,
 STATIC FUNCTION  construct (map_msg    IN  aq$_jms_map_message)
   RETURN aq$_jms_message,
 STATIC FUNCTION  construct (object_msg IN  aq$_jms_object_message)
   RETURN aq$_jms_message,
 MEMBER FUNCTION  cast_to_bytes_msg  RETURN aq$_jms_bytes_message,
 MEMBER FUNCTION  cast_to_map_msg    RETURN aq$_jms_map_message,
 MEMBER FUNCTION  cast_to_object_msg RETURN aq$_jms_object_message,
 MEMBER FUNCTION  cast_to_stream_msg RETURN aq$_jms_stream_message,
 MEMBER FUNCTION  cast_to_text_msg   RETURN aq$_jms_text_message,
 MEMBER PROCEDURE set_replyto  (replyto IN sys.aq$_agent),
 MEMBER PROCEDURE set_type     (type     IN  VARCHAR),
 MEMBER PROCEDURE set_userid   (userid   IN  VARCHAR),
 MEMBER PROCEDURE set_appid    (appid    IN  VARCHAR),
 MEMBER PROCEDURE set_groupid  (groupid  IN  VARCHAR),
 MEMBER PROCEDURE set_groupseq (groupseq IN  INT),
 MEMBER FUNCTION  get_replyto  RETURN sys.aq$_agent,
 MEMBER FUNCTION  get_type     RETURN VARCHAR,
 MEMBER FUNCTION  get_userid   RETURN VARCHAR,
 MEMBER FUNCTION  get_appid    RETURN VARCHAR,
 MEMBER FUNCTION  get_groupid  RETURN VARCHAR,
 MEMBER FUNCTION  get_groupseq RETURN INT,
 MEMBER PROCEDURE clear_properties,
```

```
MEMBER PROCEDURE set_boolean_property (property_name IN VARCHAR,
  property_value IN BOOLEAN),
MEMBER PROCEDURE set_byte_property    (property_name IN VARCHAR,
  property_value IN INT),
MEMBER PROCEDURE set_double_property  (property_name IN VARCHAR,
  property_value IN DOUBLE PRECISION),
MEMBER PROCEDURE set_float_property   (property_name IN VARCHAR,
  property_value IN FLOAT),
MEMBER PROCEDURE set_int_property     (property_name IN VARCHAR,
  property_value IN INT),
MEMBER PROCEDURE set_long_property    (property_name IN VARCHAR,
  property_value IN NUMBER),
MEMBER PROCEDURE set_short_property   (property_name IN VARCHAR,
  property_value IN INT),
MEMBER PROCEDURE set_string_property  (property_name IN VARCHAR,
  property_value IN VARCHAR),
MEMBER FUNCTION get_boolean_property (property_name IN VARCHAR) RETURN BOOLEAN,
MEMBER FUNCTION get_byte_property    (property_name IN VARCHAR) RETURN INT,
MEMBER FUNCTION get_double_property  (property_name IN VARCHAR)
  RETURN DOUBLE PRECISION,
MEMBER FUNCTION get_float_property   (property_name IN VARCHAR) RETURN FLOAT,
MEMBER FUNCTION get_int_property     (property_name IN VARCHAR) RETURN INT,
MEMBER FUNCTION get_long_property    (property_name IN VARCHAR) RETURN NUMBER,
MEMBER FUNCTION get_short_property   (property_name IN VARCHAR) RETURN INT,
MEMBER FUNCTION get_string_property  (property_name IN VARCHAR) RETURN VARCHAR,
MEMBER PROCEDURE set_text     (payload  IN  VARCHAR2),
MEMBER PROCEDURE set_text     (payload  IN  CLOB),
MEMBER PROCEDURE set_bytes    (payload  IN  RAW),
MEMBER PROCEDURE set_bytes    (payload  IN  BLOB),
MEMBER PROCEDURE get_text     (payload  OUT VARCHAR2),
MEMBER PROCEDURE get_text     (payload  OUT CLOB),
MEMBER PROCEDURE get_bytes    (payload  OUT RAW),
MEMBER PROCEDURE get_bytes    (payload  OUT BLOB));
```

CONSTRUCT Static Functions

There are six CONSTRUCT static functions in this type.

STATIC FUNCTION construct (mtype IN INT) RETURN aq$_jms_message
Creates an instance of aq$_jms_message, which can hold a specific type of JMS message (TextMessage, BytesMessage, MapMessage, StreamMessage or ObjectMessage). The message type of the created aq$_jms_message instance depends on the mtype parameter passed to the construct method. Once a message has been constructed, it can be used to store JMS messages of the type it has been constructed to hold.

The mtype parameter must be one of the following constants described in "Oracle JMS Type Constants" on page 187-11:

```
DBMS_AQ.JMS_TEXT_MESSAGE
DBMS_AQ.JMS_BYTES_MESSAGE
DBMS_AQ.JMS_STREAM_MESSAGE
DBMS_AQ.JMS_MAP_MESSAGE
DBMS_AQ.JMS_OBJECT_MESSAGE
```

STATIC FUNCTION construct (text_msg IN aq$_jms_text_message) RETURN aq$_jms_message
Creates an aq$_jms_message from an aq$_jms_text_message.

STATIC FUNCTION construct (bytes_msg IN aq$_jms_bytes_message) RETURN aq$_

jms_message;
Creates an aq$_jms_message from an aq$_jms_bytes_message.

STATIC FUNCTION construct (stream_msg IN aq$_jms_stream_message) RETURN aq$_jms_message;
Creates an aq$_jms_message from an aq$_jms_stream_message.

STATIC FUNCTION construct (map_msg IN aq$_jms_map_message) RETURN aq$_jms_message;
Creates an aq$_jms_message from an aq$_jms_map_message.

STATIC FUNCTION construct (object_msg IN aq$_jms_object_message) RETURN aq$_jms_message;
Creates an aq$_jms_message from an aq$_jms_object_message.

Cast Methods

cast_to_bytes_msg RETURN aq$_jms_bytes_message
Casts an aq$_jms_message to an aq$_jms_bytes_message. Returns an aq$_jms_bytes_message or null if the message_type attribute of the aq$_jms_message is not DBMS_AQ.JMS_BYTES_MESSAGE. This function raises ORA-24198 if the message_type field of the aq$_jms_message is not DBMS_AQJMS.JMS_BYTES_MESSAGE.

cast_to_map_msg RETURN aq$_jms_map_message
Casts an aq$_jms_message to an aq$_jms_map_message. Returns an aq$_jms_map_message or null if the message_type attribute of the aq$_jms_message is not DBMS_AQ.JMS_MAP_MESSAGE. This function raises ORA-24198 if the message_type field of the aq$_jms_message is not DBMS_AQJMS.JMS_MAP_MESSAGE.

cast_to_object_msg RETURN aq$_jms_object_message
Casts an aq$_jms_message to an aq$_jms_object_message. Returns an aq$_jms_object_message or null if the message_type attribute of the aq$_jms_message is not DBMS_AQ.JMS_OBJECT_MESSAGE. This function raises ORA-24198 if the message_type field of the aq$_jms_message is not DBMS_AQJMS.JMS_OBJECT_MESSAGE.

cast_to_stream_msg RETURN aq$_jms_stream_message
Casts an aq$_jms_message to an aq$_jms_stream_message. Returns an aq$_jms_stream_message or null if the message_type attribute of the aq$_jms_message is not DBMS_AQ.JMS_STREAM_MESSAGE. This function raises ORA-24198 if the message_type field of the aq$_jms_message is not DBMS_AQJMS.JMS_STREAM_MESSAGE.

cast_to_text_msg RETURN aq$_jms_text_message
Casts an aq$_jms_message to an aq$_jms_text_message. Returns an aq$_jms_text_message or null if the message_type attribute of the aq$_jms_message is not DBMS_AQ.JMS_TEXT_MESSAGE. This function raises ORA-24198 if the message_type field of the aq$_jms_message is not DBMS_AQJMS.JMS_TEXT_MESSAGE.

JMS Header Methods

set_replyto (replyto IN sys.aq$_agent)
Sets the replyto parameter, which corresponds to JMSReplyTo.

get_replyto RETURN sys.aq$_agent
Returns `replyto`, which corresponds to `JMSReplyTo`.

set_type (type IN VARCHAR)
Sets the JMS type, which can be any text and corresponds to `JMSType`.

get_type RETURN VARCHAR
Returns `type`, which corresponds to `JMSType`.

System Properties Methods

set_userid (userid IN VARCHAR)
Sets `userid`, which corresponds to `JMSXUserID`.

set_appid (appid IN VARCHAR)
Sets `appid`, which corresponds to `JMSXAppID`.

set_groupid (groupid IN VARCHAR)
Sets `groupid`, which corresponds to `JMSXGroupID`.

set_groupseq (groupseq IN INT)
Sets `groupseq`, which corresponds to `JMSXGroupSeq`.

get_userid RETURN VARCHAR
Returns `userid`, which corresponds to `JMSXUserID`.

get_appid RETURN VARCHAR
Returns `appid`, which corresponds to `JMSXAppID`.

get_groupid RETURN VARCHAR
Returns `groupid`, which corresponds to `JMSXGroupID`.

get_groupseq RETURN VARCHAR
Returns `groupseq`, which corresponds to `JMSXGroupSeq`.

User Properties Methods

clear_properties
Clears all user properties. This procedure does not affect system properties.

set_boolean_property (property_name IN VARCHAR, property_value IN BOOLEAN)
Checks whether `property_name` is null or exists. If it is not null, the procedure stores `property_value` in an internal representation (a `NUMBER` type). Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_byte_property (property_name IN VARCHAR, property_value IN INT)
Checks whether property_name is null or exists. If it is not null, the procedure checks whether `property_value` is within -128 to 127 (8-bits). This check is necessary because neither PL/SQL nor RDBMS defines the `byte` datatype. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_double_property (property_name IN VARCHAR, property_value IN DOUBLE PRECISION)
Checks whether `property_name` is null or exists. If it is not null, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_float_property (property_name IN VARCHAR, property_value IN FLOAT)
Checks whether `property_name` is null or exists. If it is not null, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_int_property (property_name IN VARCHAR, property_value IN INT)
Checks whether `property_name` is null or exists. If it is not null, the procedure checks whether `property_value` is within -2147483648 to 2147483647 (32-bits). This check is necessary because the `INT` datatype is 38 bits in PL/SQL and Oracle Database. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_long_property (property_name IN VARCHAR, property_value IN NUMBER)
Checks whether `property_name` is null or exists. If it is not null, the procedure stores `property_value`. In PL/SQL and Oracle Database, the `NUMBER` datatype is 38 bits. In Java, the long datatype is 64 bits. Therefore, no range check is needed. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_short_property (property_name IN VARCHAR, property_value IN INT)
Checks whether `property_name` is null or exists. If it is not null, the procedure checks whether `property_value` is within -32768 to 32767 (16-bits). This check is necessary because neither PL/SQL nor RDBMS defines the `short` datatype. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_string_property (property_name IN VARCHAR, property_value IN VARCHAR)
Checks whether `property_name` is null or exists. If it is not null, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

get_boolean_property (property_name IN VARCHAR) RETURN BOOLEAN
If the property with the corresponding property name passed in exists, and if it is a `BOOLEAN` property, then this function returns the value of the property. Otherwise it returns a null.

get_byte_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a `BYTE` property, then this function returns the value of the property. Otherwise it returns a null.

get_double_property (property_name IN VARCHAR) RETURN DOUBLE PRECISION
If the property with the corresponding property name passed in exists, and if it is a `DOUBLE` property, then this function returns the value of the property. Otherwise it returns a null.

get_float_property (property_name IN VARCHAR) RETURN FLOAT
If the property with the corresponding property name passed in exists, and if it is a `FLOAT` property, then this function returns the value of the property. Otherwise it returns a null.

get_int_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a `Integer` property, then this function returns the value of the property. Otherwise it returns a null.

get_long_property (property_name IN VARCHAR) RETURN NUMBER
If the property with the corresponding property name passed in exists, and if it is a `long` property, then this function returns the value of the property. Otherwise it returns a null.

get_short_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a `short` property, then this function returns the value of the property. Otherwise it returns a null.

get_string_property (property_name IN VARCHAR) RETURN VARCHAR
If the property with the corresponding property name passed in exists, and if it is a `STRING` property, then this function returns the value of the property. Otherwise it returns a null.

Payload Methods

set_text (payload IN VARCHAR2)
Sets the payload, a `VARCHAR2` value, to an internal representation.

set_text (payload IN CLOB),
Sets the payload, a `CLOB` value, to an internal representation.

set_bytes (payload IN RAW)
Sets the payload, a `RAW` value, to an internal representation.

set_bytes (payload IN BLOB)
Sets the payload, a `BLOB` value, to an internal representation.

get_text (payload OUT VARCHAR2)
Puts the internal representation of the payload into a `VARCHAR2` variable payload.

get_text (payload OUT CLOB)
Puts the internal representation of the payload into a `CLOB` variable payload.

get_bytes (payload OUT RAW)
Puts the internal representation of the payload into a `RAW` variable payload.

get_bytes (payload OUT BLOB)
Puts the internal representation of the payload into a `BLOB` variable payload.

SYS.AQ$_JMS_TEXT_MESSAGE Type

This type is the ADT used to store a `TextMessage` in an Oracle Streams AQ queue.

This section contains these topics:

Syntax

```
TYPE AQ$_JMS_TEXT_MESSAGE AS OBJECT(
 header    aq$_jms_header,
 text_len  INT,
 text_vc   varchar2(4000),
 text_lob  clob,
 STATIC FUNCTION construct    RETURN aq$_jms_text_message,
 MEMBER PROCEDURE set_replyto  (replyto  IN  sys.aq$_agent),
 MEMBER PROCEDURE set_type     (type     IN  VARCHAR),
 MEMBER FUNCTION  get_replyto RETURN sys.aq$_agent,
 MEMBER FUNCTION  get_type    RETURN VARCHAR,
 MEMBER PROCEDURE set_userid   (userid   IN  VARCHAR),
 MEMBER PROCEDURE set_appid    (appid    IN  VARCHAR),
 MEMBER PROCEDURE set_groupid  (groupid  IN  VARCHAR),
 MEMBER PROCEDURE set_groupseq (groupseq IN  INT),
 MEMBER FUNCTION get_userid   RETURN VARCHAR,
 MEMBER FUNCTION get_appid    RETURN VARCHAR,
 MEMBER FUNCTION get_groupid  RETURN VARCHAR,
 MEMBER FUNCTION get_groupseq RETURN INT,
 MEMBER PROCEDURE clear_properties,
 MEMBER PROCEDURE set_boolean_property(property_name IN VARCHAR,
   property_value IN BOOLEAN),
 MEMBER PROCEDURE set_byte_property   (property_name IN VARCHAR,
   property_value IN INT),
 MEMBER PROCEDURE set_double_property (property_name IN VARCHAR,
   property_value IN DOUBLE PRECISION),
 MEMBER PROCEDURE set_float_property  (property_name IN VARCHAR,
   property_value IN FLOAT),
 MEMBER PROCEDURE set_int_property    (property_name IN VARCHAR,
   property_value IN INT),
 MEMBER PROCEDURE set_long_property   (property_name IN VARCHAR,
   property_value IN NUMBER),
 MEMBER PROCEDURE set_short_property  (property_name IN VARCHAR,
   property_value IN INT),
 MEMBER PROCEDURE set_string_property (property_name IN VARCHAR,
   property_value IN VARCHAR),
 MEMBER FUNCTION get_boolean_property (property_name IN VARCHAR)
   RETURN BOOLEAN,
 MEMBER FUNCTION get_byte_property    (property_name IN VARCHAR) RETURN INT,
 MEMBER FUNCTION get_double_property  (property_name IN VARCHAR)
   RETURN DOUBLE PRECISION,
 MEMBER FUNCTION get_float_property   (property_name IN VARCHAR) RETURN FLOAT,
 MEMBER FUNCTION get_int_property     (property_name IN VARCHAR) RETURN INT,
 MEMBER FUNCTION get_long_property    (property_name IN VARCHAR) RETURN NUMBER,
 MEMBER FUNCTION get_short_property   (property_name IN VARCHAR) RETURN INT,
```

```
MEMBER FUNCTION get_string_property  (property_name IN VARCHAR)
  RETURN VARCHAR,
MEMBER PROCEDURE set_text            (payload  IN  VARCHAR2),
MEMBER PROCEDURE set_text            (payload  IN  CLOB),
MEMBER PROCEDURE get_text            (payload  OUT VARCHAR2),
MEMBER PROCEDURE get_text            (payload  OUT CLOB));
```

CONSTRUCT Function

STATIC FUNCTION construct RETURN aq$_jms_text_message
Creates an empty `aq$_jms_text_message`.

JMS Header Methods

set_replyto (replyto IN sys.aq$_agent)
Sets the `replyto` parameter, which corresponds to `JMSReplyTo` in JMS.

set_type (type IN VARCHAR)
Sets the JMS type, which can be any text, and which corresponds to `JMSType` in JMS.

get_replyto RETURN sys.aq$_agent
Returns `replyto`, which corresponds to `JMSReplyTo`.

get_type RETURN VARCHAR
Returns `type`, which corresponds to `JMSType`.

System Properties Methods

set_userid (userid IN VARCHAR)
Sets `userid`, which corresponds to `JMSXUserID` in JMS.

set_appid (appid IN VARCHAR)
Sets `appid`, which corresponds to `JMSXAppID` in JMS.

set_groupid (groupid IN VARCHAR)
Sets `groupid`, which corresponds to `JMSXGroupID` in JMS.

set_groupseq (groupseq IN INT)
Sets `groupseq`, which corresponds to `JMSXGroupSeq` in JMS.

get_userid RETURN VARCHAR
Returns `userid`, which corresponds to `JMSXUserID`.

get_appid RETURN VARCHAR
Returns `appid`, which corresponds to `JMSXAppID`.

get_groupid RETURN VARCHAR
Returns `groupid`, which corresponds to `JMSXGroupID`.

get_groupseq RETURN INT
Returns `groupseq`, which corresponds to `JMSXGroupSeq`.

User Properties Methods

clear_properties
Clears all user properties. This procedure does not affect system properties.

set_boolean_property (property_name IN VARCHAR, property_value IN BOOLEAN)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value` in an internal representation. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_byte_property (property_name IN VARCHAR, property_value IN INT)
Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -128 to 127 (8-bits). This check is necessary because neither PL/SQL nor RDBMS defines the `BYTE` datatype. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_double_property (property_name IN VARCHAR, property_value IN DOUBLE PRECISION)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_float_property (property_name IN VARCHAR, property_value IN FLOAT)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_int_property (property_name IN VARCHAR, property_value IN INT)
Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -2147483648 to 2147483647 (32-bits). This check is necessary because in PL/SQL and Oracle Database, the `INT` datatype is 38 bits. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_long_property (property_name IN VARCHAR, property_value IN NUMBER)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. In PL/SQL and Oracle Database, the `NUMBER` datatype is 38 bits. In Java, the `long` datatype is 64 bits. Therefore, no range check is needed.Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_short_property property_name IN VARCHAR, property_value IN INT)
Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -32768 to 32767 (16-bits). This check is necessary because neither PL/SQL nor RDBMS defines the `short` datatype. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_string_property (property_name IN VARCHAR, property_value IN VARCHAR)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

get_boolean_property (property_name IN VARCHAR) RETURN BOOLEAN
If the property with the corresponding property name passed in exists, and if it is a `BOOLEAN` property, then this function returns the value of the property. Otherwise it returns a null.

get_byte_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a `BYTE` property, then this function returns the value of the property. Otherwise it returns a null.

get_double_property (property_name IN VARCHAR) RETURN DOUBLE PRECISION
If the property with the corresponding property name passed in exists, and if it is a `DOUBLE` property, then this function returns the value of the property. Otherwise it returns a null.

get_float_property (property_name IN VARCHAR) RETURN FLOAT
If the property with the corresponding property name passed in exists, and if it is a `FLOAT` property, then this function returns the value of the property. Otherwise it returns a null.

get_int_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a `Integer` property, then this function returns the value of the property. Otherwise it returns a null.

get_long_property (property_name IN VARCHAR) RETURN NUMBER
If the property with the corresponding property name passed in exists, and if it is a `long` property, then this function returns the value of the property. Otherwise it returns a null.

get_short_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a `short` property, then this function returns the value of the property. Otherwise it returns a null.

get_string_property (property_name IN VARCHAR) RETURN VARCHAR)
If the property with the corresponding property name passed in exists, and if it is a `STRING` property, then this function returns the value of the property. Otherwise it returns a null.

Payload Methods

set_text (payload IN VARCHAR2)
Sets the payload, a `VARCHAR2` value, to an internal representation.

set_text (payload IN CLOB)
Sets the payload, a `CLOB` value, to an internal representation.

get_text (payload OUT VARCHAR2)
Puts the internal representation of the payload into a `VARCHAR2` variable payload.

get_text (payload OUT CLOB)
Puts the internal representation of the payload into a `CLOB` variable payload.

SYS.AQ$_JMS_BYTES_MESSAGE Type

This type is the ADT used to store a `BytesMessage` in an Oracle Streams AQ queue.

This section contains these topics:

Syntax

```
TYPE AQ$_JMS_BYTES_MESSAGE AS OBJECT(
 header     aq$_jms_header,
 bytes_len  INT,
 bytes_raw  raw(2000),
 bytes_lob  blob,
 STATIC FUNCTION construct RETURN aq$_jms_bytes_message,
 MEMBER PROCEDURE set_replyto  (replyto IN sys.aq$_agent),
 MEMBER PROCEDURE set_type     (type    IN VARCHAR),
 MEMBER FUNCTION get_replyto RETURN sys.aq$_agent,
 MEMBER FUNCTION get_type    RETURN VARCHAR,
 MEMBER PROCEDURE set_userid   (userid   IN VARCHAR),
 MEMBER PROCEDURE set_appid    (appid    IN VARCHAR),
 MEMBER PROCEDURE set_groupid  (groupid  IN VARCHAR),
 MEMBER PROCEDURE set_groupseq (groupseq IN INT),
 MEMBER FUNCTION get_userid   RETURN VARCHAR,
 MEMBER FUNCTION get_appid    RETURN VARCHAR,
 MEMBER FUNCTION get_groupid  RETURN VARCHAR,
 MEMBER FUNCTION get_groupseq RETURN INT,
 MEMBER PROCEDURE clear_properties,
 MEMBER PROCEDURE set_boolean_property(property_name IN VARCHAR,
   property_value IN BOOLEAN),
 MEMBER PROCEDURE set_byte_property   (property_name IN VARCHAR,
   property_value IN INT),
 MEMBER PROCEDURE set_double_property (property_name IN VARCHAR,
   property_value IN DOUBLE PRECISION),
 MEMBER PROCEDURE set_float_property  (property_name IN VARCHAR,
   property_value IN FLOAT),
 MEMBER PROCEDURE set_int_property    (property_name IN VARCHAR,
   property_value IN INT),
 MEMBER PROCEDURE set_long_property   (property_name IN VARCHAR,
   property_value IN NUMBER),
 MEMBER PROCEDURE set_short_property  (property_name IN VARCHAR,
   property_valuE IN INT),
 MEMBER PROCEDURE set_string_property (property_name IN VARCHAR,
   property_value IN VARCHAR),
 MEMBER FUNCTION get_boolean_property (property_name IN VARCHAR) RETURN BOOLEAN,
 MEMBER FUNCTION get_byte_property    (property_name IN VARCHAR) RETURN INT,
 MEMBER FUNCTION get_double_property  (property_name IN VARCHAR)
   RETURN DOUBLE PRECISION,
 MEMBER FUNCTION get_float_property   (property_name IN VARCHAR) RETURN FLOAT,
 MEMBER FUNCTION get_int_property     (property_name IN VARCHAR) RETURN INT,
 MEMBER FUNCTION get_long_property    (property_name IN VARCHAR) RETURN NUMBER,
 MEMBER FUNCTION get_short_property   (property_name IN VARCHAR) RETURN INT,
 MEMBER FUNCTION get_string_property  (property_name IN VARCHAR) RETURN VARCHAR,
```

```
MEMBER PROCEDURE set_bytes           (payload  IN RAW),
MEMBER PROCEDURE set_bytes           (payload  IN BLOB),
MEMBER PROCEDURE get_bytes           (payload  OUT RAW),
MEMBER PROCEDURE get_bytes           (payload  OUT BLOB),
MEMBER FUNCTION  prepare             (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER PROCEDURE reset               (id IN PLS_INTEGER),
MEMBER PROCEDURE flush               (id IN PLS_INTEGER),
MEMBER PROCEDURE clear_body          (id IN PLS_INTEGER),
MEMBER PROCEDURE clean               (id IN PLS_INTEGER),
STATIC PROCEDURE clean_all,
MEMBER FUNCTION get_mode           (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION read_boolean       (id IN PLS_INTEGER) RETURN BOOLEAN,
MEMBER FUNCTION read_byte          (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION read_bytes         (id IN PLS_INTEGER,
  value OUT NOCOPY BLOB, length IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION read_char          (id IN PLS_INTEGER) RETURN CHAR,
MEMBER FUNCTION read_double        (id IN PLS_INTEGER) RETURN DOUBLE PRECISION,
MEMBER FUNCTION read_float         (id IN PLS_INTEGER) RETURN FLOAT,
MEMBER FUNCTION read_int           (id IN PLS_INTEGER) RETURN INT,
MEMBER FUNCTION read_long          (id IN PLS_INTEGER) RETURN NUMBER,
MEMBER FUNCTION read_short         (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION read_unsigned_byte  (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION read_unsigned_short (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER PROCEDURE read_utf          (id IN PLS_INTEGER, value OUT NOCOPY CLOB),
MEMBER PROCEDURE write_boolean     (id IN PLS_INTEGER, value IN BOOLEAN),
MEMBER PROCEDURE write_byte        (id IN PLS_INTEGER, value IN PLS_INTEGER),
MEMBER PROCEDURE write_bytes       (id IN PLS_INTEGER, value IN RAW),
MEMBER PROCEDURE write_bytes       (id IN PLS_INTEGER, value IN BLOB),
MEMBER PROCEDURE write_bytes       (id IN PLS_INTEGER, value IN RAW,
  offset IN PLS_INTEGER, length IN PLS_INTEGER),
MEMBER PROCEDURE write_bytes       (id IN PLS_INTEGER, value IN BLOB,
  offset IN INT, length IN INT),
MEMBER PROCEDURE write_char        (id IN PLS_INTEGER, value IN CHAR),
MEMBER PROCEDURE write_double      (id IN PLS_INTEGER,
  value IN DOUBLE PRECISION),
MEMBER PROCEDURE write_float       (id IN PLS_INTEGER, value IN FLOAT),
MEMBER PROCEDURE write_int         (id IN PLS_INTEGER, value IN PLS_INTEGER),
MEMBER PROCEDURE write_long        (id IN PLS_INTEGER, value IN NUMBER),
MEMBER PROCEDURE write_short       (id IN PLS_INTEGER, value IN PLS_INTEGER),
MEMBER PROCEDURE write_utf         (id IN PLS_INTEGER, value IN VARCHAR2),
MEMBER PROCEDURE write_utf         (id IN PLS_INTEGER, value IN CLOB));
```

CONSTRUCT Function

STATIC FUNCTION construct RETURN aq$_jms_bytes_message
Creates an empty `aq$_jms_bytes_message`.

JMS Header Methods

set_replyto (replyto IN sys.aq$_agent)
Sets the `replyto` parameter, which corresponds to `JMSReplyTo` in JMS.

set_type (type IN VARCHAR)
Sets the JMS type, which can be any text, and which corresponds to `JMSType` in JMS.

get_replyto RETURN sys.aq$_agent
Returns `replyto`, which corresponds to `JMSReplyTo`.

get_type RETURN VARCHAR
Returns `type`, which corresponds to `JMSType`.

System Properties Methods

set_userid (userid IN VARCHAR)
Sets `userid`, which corresponds to `JMSXUserID` in JMS.

set_appid (appid IN VARCHAR)
Sets `appid`, which corresponds to `JMSXAppID` in JMS.

set_groupid (groupid IN VARCHAR)
Sets `groupid`, which corresponds to `JMSXGroupID` in JMS.

set_groupseq (groupseq IN INT)
Sets `groupseq`, which corresponds to `JMSXGroupSeq` in JMS.

get_userid RETURN VARCHAR
Returns `userid`, which corresponds to `JMSXUserID`.

get_appid RETURN VARCHAR
Returns `appid`, which corresponds to `JMSXAppID`.

get_groupid RETURN VARCHAR
Returns `groupid`, which corresponds to `JMSXGroupID`.

get_groupseq RETURN NUMBER
Returns `groupseq`, which corresponds to `JMSXGroupSeq`.

User Properties Methods

clear_properties
Clears all user properties. This procedure does not affect system properties.

set_boolean_property (property_name IN VARCHAR, property_value IN BOOLEAN)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value` in an internal representation. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_byte_property (property_name IN VARCHAR, property_value IN INT)
Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -128 to 127 (8-bits). This check is necessary because neither PL/SQL nor RDBMS defines the `BYTE` datatype. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_double_property (property_name IN VARCHAR, property_value IN DOUBLE PRECISION)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_float_property (property_name IN VARCHAR, property_value IN FLOAT)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_int_property (property_name IN VARCHAR, property_value IN INT)
Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -2147483648 to 2147483647 (32-bits). This check is necessary because in PL/SQL and Oracle Database, the `INT` datatype is 38 bits. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_long_property (property_name IN VARCHAR, property_value IN NUMBER)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. In PL/SQL and Oracle Database, the `NUMBER` datatype is 38 bits. In Java, the `long` datatype is 64 bits. Therefore, no range check is needed.Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_short_property (property_name IN VARCHAR, property_value IN INT)
Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -32768 to 32767 (16-bits). This check is necessary because neither PL/SQL nor RDBMS defines the `short` datatype. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_string_property (property_name IN VARCHAR, property_value IN VARCHAR)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

get_boolean_property (property_name IN VARCHAR) RETURN BOOLEAN
If the property with the corresponding property name passed in exists, and if it is a `BOOLEAN` property, then this function returns the value of the property. Otherwise it returns a null.

get_byte_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a `BYTE` property, then this function returns the value of the property. Otherwise it returns a null.

get_double_property (property_name IN VARCHAR) RETURN DOUBLE PRECISION
If the property with the corresponding property name passed in exists, and if it is a `DOUBLE` property, then this function returns the value of the property. Otherwise it returns a null.

get_float_property (property_name IN VARCHAR) RETURN FLOAT
If the property with the corresponding property name passed in exists, and if it is a `FLOAT` property, then this function returns the value of the property. Otherwise it returns a null.

get_int_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a `Integer` property, then this function returns the value of the property. Otherwise it returns a null.

get_long_property (property_name IN VARCHAR) RETURN NUMBER
If the property with the corresponding property name passed in exists, and if it is a `long` property, then this function returns the value of the property. Otherwise it returns a null.

get_short_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a `short` property, then this function returns the value of the property. Otherwise it returns a null.

get_string_property (property_name IN VARCHAR) RETURN VARCHAR
If the property with the corresponding property name passed in exists, and if it is a `STRING` property, then this function returns the value of the property. Otherwise it returns a null.

Payload Methods

set_bytes (payload in RAW)
Sets the payload, a `RAW` value, to an internal representation.

set_bytes (payload in BLOB)
Sets the payload, a `BLOB` value, to an internal representation.

get_bytes (payload out RAW)
Puts the internal representation of the payload into a `RAW` variable payload. Raises exception ORA-24190 if the length of the internal payload is more than 32767 (the maximum length of `RAW` in PL/SQL).

get_bytes (payload out BLOB)
Puts the internal representation of the payload into a `BLOB` variable payload.

prepare (id IN PLS_INTEGER) RETURN PLS_INTEGER
Takes the byte array stored in aq$_jms_bytes_message and decodes it as a Java object in the Java stored procedure. The result of the decoding is stored as a static variable in Jserv session memory. Parameter `id` is used to identify the slot where the Java object is stored in the Oracle Database JVM session memory. If `id` is null, then a new slot is created for this PL/SQL object. Subsequent JMS operations on the payload need to provide this operation ID.

This function also sets the message access mode to `MESSAGE_ACCESS_READONLY`. Subsequent calls of `write_XXX` procedure raise an ORA-24196 error. Users can call the `clear_body` procedure to set the message access mode to `MESSAGE_ACCESS_READONLY`.

This function raises ORA-24197 if the Java stored procedure throws an exception during execution, ORA-24198 if the operation ID is invalid, or ORA-24199 if the Java stored procedure message store overflows.

reset (id IN PLS_INTEGER)
Resets the starting position of the stream to the beginning and puts the bytes message in read-only mode. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

flush (id IN PLS_INTEGER)
Takes the static variable in Jserv and synchronizes the content back to the `aq$_jms_bytes_message`. This procedure will not affect the underlying access mode. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

clear_body (id IN PLS_INTEGER)
Sets the Java stored procedure static variable to empty payload. Parameter `id` is used to identify the slot where the Java object is stored in the Oracle Database JVM session

memory. If id is null, a new slot is created for this PL/SQL object. Subsequent JMS operations on the payload need to provide this operation ID.

It also sets the message access mode to MESSAGE_ACCESS_WRITEONLY. Later calls of read_XXX procedure raise ORA-24196 error. Users can call the reset or prepare procedures to set the message access mode to MESSAGE_ACCESS_READONLY. Write-only and read-only modes affect only the payload functions of AQ$_JMS_BYTES_MESSAGE. They do not affect the header functions.

This function raises ORA-24197 if the Java stored procedure throws an exception during execution, ORA-24198 if the operation ID is invalid, or ORA-24199 if the Java stored procedure message store overflows.

clean (id IN PLS_INTEGER)

Closes and cleans up the DataInputStream or DataOutputStream at the Java stored procedure side corresponding to the operation ID. It is very important to call this procedure to avoid memory leaks. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

clean_all

Closes and cleans up all the messages in the corresponding type of message store at the Java stored procedure side. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution.

get_mode (id IN PLS_INTEGER) RETURN PLS_INTEGER

Returns the current mode of this message. The return value is either SYS.dbms_jms.plsql.MESSAGE_ACCESS_READONLY or SYS.dbms_jms.plsql.MESSAGE_ACCESS_WRITEONLY. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

read_boolean (id IN PLS_INTEGER) RETURN BOOLEAN

Reads a Boolean value from the bytes message and returns the Boolean value read. Null is returned if the end of the message stream has been reached. Parameter id is the operation ID. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_byte (id IN PLS_INTEGER) RETURN PLS_INTEGER

Reads a BYTE value from the bytes message and returns the BYTE value read. Null is returned if the end of the stream has been reached. Because there is no BYTE type in PL/SQL, Oracle Database uses PLS_INTEGER to represent a BYTE. Although PL/SQL users get a PLS_INTEGER, they are guaranteed that the value is in the Java BYTE value range. If this value is issued with a write_byte function, then there will not be an out of range error. Parameter id is the operation ID. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_bytes (id IN PLS_INTEGER, value OUT NO COPY BLOB, length IN PLS_INTEGER) RETURN PLS_INTEGER

Reads length of the bytes from bytes message stream into value and returns the total number of bytes read. If there is no more data (because the end of the stream has been reached), then it returns -1. Raises exceptions ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_char (id IN PLS_INTEGER) RETURN CHAR
Reads a character value from the bytes message and returns the character value read. Null is returned if the end of the stream has been reached. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_double (id IN PLS_INTEGER) RETURN DOUBLE PRECISION
Reads a double from the bytes message and returns the character value read. Null is returned if the end of the stream has been reached. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_float (id IN PLS_INTEGER) RETURN FLOAT
Reads a float from the bytes message and returns the float read. Null is returned if the end of the stream has been reached. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_int (id IN PLS_INTEGER) RETURN INT
Reads an `INT` from the bytes message and returns the `INT` read. Null is returned if the end of the stream has been reached. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_long (id IN PLS_INTEGER) RETURN NUMBER
Reads a long from the bytes message and returns the long read. Null is returned if the end of the stream has been reached. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_short (id IN PLS_INTEGER) RETURN PLS_INTEGER
Reads a short value from the bytes message and returns the short value read. Null is returned if the end of the stream has been reached. Because there is no short type in PL/SQL, `PLS_INTEGER` is used to represent a `BYTE`. Although PL/SQL users get an `PLS_INTEGER`, they are guaranteed that the value is in the Java short value range. If this value is issued with a `write_short` function, then there will not be an out of range error. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_unsigned_byte (id IN PLS_INTEGER) RETURN PLS_INTEGER
Reads an unsigned 8-bit number from the bytes message stream and returns the next byte from the bytes message stream, interpreted as an unsigned 8-bit number. Null is returned if the end of the stream has been reached. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_unsigned_short (id IN PLS_INTEGER) RETURN PLS_INTEGER
Reads an unsigned 16-bit number from the bytes message stream and returns the next two bytes from the bytes message stream, interpreted as an unsigned 16-bit integer. Null is returned if the end of the stream has been reached. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_utf (id IN PLS_INTEGER, value OUT NOCOPY CLOB)
Reads a string that has been encoded using a UTF-8 format from the bytes message. Null is returned if the end of the stream has been reached. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_boolean (id IN PLS_INTEGER, value IN BOOLEAN)
Writes a Boolean to the bytes message stream as a 1-byte value. The value `true` is written as the value (byte)1. The value `false` is written as the value (byte)0. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_byte (id IN PLS_INTEGER, value IN PLS_INTEGER)
Writes a byte to the bytes message. Because there is no `BYTE` type in PL/SQL, `PLS_INTEGER` is used to represent a `BYTE`. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_bytes (id IN PLS_INTEGER, value IN RAW)
Writes an array of bytes to the bytes message. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_bytes (id IN PLS_INTEGER, value IN BLOB)
Writes an array of bytes to the bytes message. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_bytes (id IN PLS_INTEGER, value IN RAW, offset IN PLS_INTEGER, length IN PLS_INTEGER)
Writes a portion of a byte array to the bytes message stream. Parameter `offset` is the initial offset within the byte array. If the range [offset, offset+length] exceeds the boundary of the byte array value, then a Java IndexOutOfBounds exception is thrown in the Java stored procedure and this procedure raises error ORA-24197. The index starts from 0. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_bytes (id IN PLS_INTEGER, value IN BLOB, offset IN INT, length IN INT)
Writes a portion of a byte array to the bytes message stream. Parameter `offset` is the initial offset within the byte array. If the range [offset, offset+length] exceeds the boundary of the byte array value, then a Java IndexOutOfBounds exception is thrown in the Java stored procedure and this procedure raises error ORA-24197. The index starts from 0. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_char (id IN PLS_INTEGER, value IN CHAR)
Writes a character value to the bytes message. If this value has multiple characters, it is the first character that is written. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_double (id IN PLS_INTEGER, value IN DOUBLE PRECISION)
Writes a double to the bytes message. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_float (id IN PLS_INTEGER, value IN FLOAT)
Writes a float to the bytes message. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_int (id IN PLS_INTEGER, value IN PLS_INTEGER)
Writes an `INT` to the bytes message. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_long (id IN PLS_INTEGER, value IN NUMBER)
Writes a long to the bytes message. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_short (id IN PLS_INTEGER, value IN PLS_INTEGER)
Writes a short to the bytes message as two bytes, high byte first. Because there is no short type in PL/SQL, `INT` is used to represent a short. Raises exception ORA-24193 if the parameter value exceeds the valid range, ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_utf (id IN PLS_INTEGER, value IN VARCHAR2)
Writes a string to the bytes message stream using UTF-8 encoding in a machine-independent manner. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_utf (id IN PLS_INTEGER, value IN CLOB)
Writes a string to the bytes message stream using UTF-8 encoding in a machine-independent manner. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

SYS.AQ$_JMS_MAP_MESSAGE Type

This type is the ADT used to store a `MapMessage` in an Oracle Streams AQ queue.

This section contains these topics:

Syntax

```
TYPE aq$_jms_map_message AS object(
 header     aq$_jms_header,
 bytes_len  int,
 bytes_raw  raw(2000),
 bytes_lob  blob,
 STATIC FUNCTION  construct   RETURN aq$_jms_map_message,
 MEMBER PROCEDURE set_replyto  (replyto IN sys.aq$_agent),
 MEMBER PROCEDURE set_type     (type    IN VARCHAR),
 MEMBER FUNCTION get_replyto  RETURN sys.aq$_agent,
 MEMBER FUNCTION get_type     RETURN VARCHAR,
 MEMBER PROCEDURE set_userid   (userid   IN VARCHAR),
 MEMBER PROCEDURE set_appid    (appid    IN VARCHAR),
 MEMBER PROCEDURE set_groupid  (groupid  IN VARCHAR),
 MEMBER PROCEDURE set_groupseq (groupseq IN INT),
 MEMBER FUNCTION get_userid   RETURN VARCHAR,
 MEMBER FUNCTION get_appid    RETURN VARCHAR,
 MEMBER FUNCTION get_groupid  RETURN VARCHAR,
 MEMBER FUNCTION get_groupseq RETURN INT,
 MEMBER PROCEDURE clear_properties,
 MEMBER PROCEDURE set_boolean_property(property_name IN VARCHAR,
   property_value IN BOOLEAN),
 MEMBER PROCEDURE set_byte_property   (property_name IN VARCHAR,
   property_value IN INT),
 MEMBER PROCEDURE set_double_property (property_name IN VARCHAR,
   property_value IN DOUBLE PRECISION),
 MEMBER PROCEDURE set_float_property  (property_name IN VARCHAR,
   property_value IN FLOAT),
 MEMBER PROCEDURE set_int_property    (property_name IN VARCHAR,
   property_value IN INT),
 MEMBER PROCEDURE set_long_property   (property_name IN VARCHAR,
   property_value IN NUMBER),
 MEMBER PROCEDURE set_short_property  (property_name IN VARCHAR,
   property_valuE IN INT),
 MEMBER PROCEDURE set_string_property (property_name IN VARCHAR,
   property_value IN VARCHAR),
 MEMBER FUNCTION get_boolean_property (property_name IN VARCHAR) RETURN BOOLEAN,
 MEMBER FUNCTION get_byte_property    (property_name IN VARCHAR) RETURN INT,
 MEMBER FUNCTION get_double_property  (property_name IN VARCHAR)
   RETURN DOUBLE PRECISION,
 MEMBER FUNCTION get_float_property   (property_name IN VARCHAR) RETURN FLOAT,
 MEMBER FUNCTION get_int_property     (property_name IN VARCHAR) RETURN INT,
 MEMBER FUNCTION get_long_property    (property_name IN VARCHAR) RETURN NUMBER,
 MEMBER FUNCTION get_short_property   (property_name IN VARCHAR) RETURN INT,
 MEMBER FUNCTION get_string_property  (property_name IN VARCHAR) RETURN VARCHAR,
```

```
MEMBER PROCEDURE set_bytes   (payload IN RAW),
MEMBER PROCEDURE set_bytes   (payload IN BLOB),
MEMBER PROCEDURE get_bytes   (payload OUT RAW),
MEMBER PROCEDURE get_bytes   (payload OUT BLOB),
MEMBER FUNCTION  prepare     (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER PROCEDURE flush       (id  IN  PLS_INTEGER),
MEMBER PROCEDURE clear_body  (id IN PLS_INTEGER),
MEMBER PROCEDURE clean       (id  IN  PLS_INTEGER),
STATIC PROCEDURE clean_all,
MEMBER PROCEDURE set_boolean (id IN PLS_INTEGER, name IN VARCHAR2,
  value IN BOOLEAN),
MEMBER PROCEDURE set_byte    (id IN PLS_INTEGER, name IN VARCHAR2,
  value IN  PLS_INTEGER),
MEMBER PROCEDURE set_bytes   (id IN PLS_INTEGER, name IN VARCHAR2,
  value IN  RAW),
MEMBER PROCEDURE set_bytes   (id IN PLS_INTEGER, name IN VARCHAR2,
  value IN RAW, offset IN INT, length IN INT),
MEMBER PROCEDURE set_bytes   (id IN PLS_INTEGER, name IN VARCHAR2,
  value IN BLOB),
MEMBER PROCEDURE set_bytes   (id IN PLS_INTEGER, name IN VARCHAR2,
  value IN BLOB, offset IN INT, length IN INT),
MEMBER PROCEDURE set_char    (id IN PLS_INTEGER, name IN VARCHAR2,
  value IN  CHAR),
MEMBER PROCEDURE set_double  (id IN PLS_INTEGER, name IN VARCHAR2,
  value IN  DOUBLE PRECISION),
MEMBER PROCEDURE set_float   (id IN PLS_INTEGER, name IN VARCHAR2,
  value IN  FLOAT),
MEMBER PROCEDURE set_int     (id IN PLS_INTEGER, name IN VARCHAR2,
  value IN  PLS_INTEGER),
MEMBER PROCEDURE set_long    (id IN PLS_INTEGER, name IN VARCHAR2,
  value IN  NUMBER),
MEMBER PROCEDURE set_short   (id IN PLS_INTEGER, name IN VARCHAR2,
  value IN  PLS_INTEGER),
MEMBER PROCEDURE set_string  (id IN PLS_INTEGER, name IN VARCHAR2,
  value IN  VARCHAR2),
MEMBER PROCEDURE set_string  (id IN  PLS_INTEGER, name IN VARCHAR2,
  value IN  CLOB),
MEMBER FUNCTION get_boolean  (id IN PLS_INTEGER, name IN VARCHAR2)
  RETURN BOOLEAN,
MEMBER FUNCTION  get_byte    (id IN PLS_INTEGER, name IN VARCHAR2)
  RETURN PLS_INTEGER,
MEMBER PROCEDURE get_bytes   (id IN PLS_INTEGER, name IN VARCHAR2,
  value OUT NOCOPY BLOB),
MEMBER FUNCTION get_char     (id IN PLS_INTEGER, name IN VARCHAR2) RETURN CHAR,
MEMBER FUNCTION get_double   (id IN PLS_INTEGER, name IN VARCHAR2)
  RETURN DOUBLE PRECISION,
MEMBER FUNCTION get_float   (id IN PLS_INTEGER, name IN VARCHAR2) RETURN FLOAT,
MEMBER FUNCTION get_int     (id IN PLS_INTEGER, name IN VARCHAR2)
  RETURN PLS_INTEGER,
MEMBER FUNCTION get_long    (id IN PLS_INTEGER, name IN VARCHAR2)
  RETURN NUMBER,
MEMBER FUNCTION get_short   (id IN PLS_INTEGER, name IN VARCHAR2)
  RETURN PLS_INTEGER,
MEMBER PROCEDURE get_string (id IN PLS_INTEGER, name IN VARCHAR2,
  value OUT NOCOPY CLOB),
MEMBER FUNCTION get_names   (id IN PLS_INTEGER) RETURN aq$_jms_namearray,
MEMBER FUNCTION get_names   (id IN PLS_INTEGER, names OUT aq$_jms_namearray,
  offset IN  PLS_INTEGER, length IN  PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER PROCEDURE get_object (id IN PLS_INTEGER, name IN  VARCHAR2,
  value  OUT NOCOPY AQ$_JMS_VALUE),
```

```
MEMBER FUNCTION get_size    (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION item_exists (id IN PLS_INTEGER, name IN VARCHAR2)
  RETURN BOOLEAN);
```

CONSTRUCT Function

STATIC FUNCTION construct RETURN aq$_jms_map_message
Creates an empty `aq$_jms_map_message` object.

JMS Header Methods

set_replyto (replyto IN sys.aq$_agent)
Sets the `replyto` parameter, which corresponds to `JMSReplyTo` in JMS.

set_type (type IN VARCHAR)
Sets the JMS type, which can be any text, and which corresponds to `JMSType` in JMS.

get_replyto RETURN sys.aq$_agent
Returns `replyto`, which corresponds to `JMSReplyTo`.

get_type RETURN VARCHAR
Returns `type`, which corresponds to `JMSType`.

System Properties Methods

set_userid (userid IN VARCHAR)
Sets `userid`, which corresponds to `JMSXUserID` in JMS.

set_appid (appid IN VARCHAR)
Sets `appid`, which corresponds to `JMSXAppID` in JMS.

set_groupid (groupid IN VARCHAR)
Sets `groupid`, which corresponds to `JMSXGroupID` in JMS.

set_groupseq (groupseq IN INT)
Sets `groupseq`, which corresponds to `JMSXGroupSeq` in JMS.

get_userid RETURN VARCHAR
Returns `userid`, which corresponds to `JMSXUserID`.

get_appid RETURN VARCHAR
Returns `appid`, which corresponds to `JMSXAppID`.

get_groupid RETURN VARCHAR
Returns `groupid`, which corresponds to `JMSXGroupID`.

get_groupseq RETURN NUMBER
Returns `groupseq`, which corresponds to `JMSXGroupSeq`.

User Properties Methods

clear_properties
Clears all user properties. This procedure does not affect system properties.

set_boolean_property (property_name IN VARCHAR, property_value IN BOOLEAN)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value` in an internal representation. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_byte_property (property_name IN VARCHAR, property_value IN INT)
Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -128 to 127 (8-bits). This check is necessary because neither PL/SQL nor RDBMS defines the `BYTE` datatype. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_double_property (property_name IN VARCHAR, property_value IN DOUBLE PRECISION)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_float_property (property_name IN VARCHAR, property_value IN FLOAT)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_int_property (property_name IN VARCHAR, property_value IN INT)
Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -2147483648 to 2147483647 (32-bits). This check is necessary because in PL/SQL and Oracle Database, the `INT` datatype is 38 bits. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_long_property (property_name IN VARCHAR, property_value IN NUMBER)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. In PL/SQL and Oracle Database, the `NUMBER` datatype is 38 bits. In Java, the `long` datatype is 64 bits. Therefore, no range check is needed.Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_short_property (property_name IN VARCHAR, property_value IN INT)
Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -32768 to 32767 (16-bits). This check is necessary because neither PL/SQL nor RDBMS defines the `short` datatype. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_string_property (property_name IN VARCHAR, property_value IN VARCHAR)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

get_boolean_property (property_name IN VARCHAR) RETURN BOOLEAN
If the property with the corresponding property name passed in exists, and if it is a `BOOLEAN` property, then this function returns the value of the property. Otherwise it returns a null.

get_byte_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a `BYTE` property, then this function returns the value of the property. Otherwise it returns a null.

get_double_property (property_name IN VARCHAR) RETURN DOUBLE PRECISION
If the property with the corresponding property name passed in exists, and if it is a `DOUBLE` property, then this function returns the value of the property. Otherwise it returns a null.

get_float_property (property_name IN VARCHAR) RETURN FLOAT
If the property with the corresponding property name passed in exists, and if it is a `FLOAT` property, then this function returns the value of the property. Otherwise it returns a null.

get_int_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a `Integer` property, then this function returns the value of the property. Otherwise it returns a null.

get_long_property (property_name IN VARCHAR) RETURN NUMBER
If the property with the corresponding property name passed in exists, and if it is a `long` property, then this function returns the value of the property. Otherwise it returns a null.

get_short_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a `short` property, then this function returns the value of the property. Otherwise it returns a null.

get_string_property (property_name IN VARCHAR) RETURN VARCHAR
If the property with the corresponding property name passed in exists, and if it is a `STRING` property, then this function returns the value of the property. Otherwise it returns a null.

Payload Methods

set_bytes (payload IN RAW)
Sets the internal payload as a RAW variable without any interpretation. The payload of `aq$_jms_map_message` is stored as either `RAW` or `BLOB` in the database. This member function sets a payload as a RAW variable without interpreting it.

set_bytes (payload IN BLOB)
Sets the internal payload as a BLOB variable without any interpretation. The payload of `aq$_jms_map_message` is stored as either `RAW` or `BLOB` in the database. This member function sets a payload as a BLOB variable without interpreting it.

get_bytes (payload OUT RAW)
Puts the internal payload into a `RAW` variable without any interpretation. The payload of `aq$_jms_map_message` is stored as either `RAW` or `BLOB` in the database. This member function gets a payload as raw bytes without interpreting it. Raises exceptions ORA-24190 if the length of internal payload is more than 32767.

get_bytes (payload OUT BLOB)
Puts the internal payload into a `BLOB` variable without any interpretation. The payload of `aq$_jms_map_message` is stored as either `RAW` or `BLOB` in the database. This member function gets a payload as a `BLOB` without interpreting it.

prepare (id IN PLS_INTEGER) RETURN PLS_INTEGER
Takes the byte array stored in `aq$_jms_map_message` and decodes it as a Java object in the Java stored procedure. The result of the decoding is stored as a static variable in Jserv session memory. Parameter `id` is used to identify the slot where the Java object is stored in the Oracle Database JVM session memory. If `id` is null, then a new slot is created for this PL/SQL object. Subsequent JMS operations on the payload need to provide this operation ID.

This function raises ORA-24197 if the Java stored procedure throws an exception during execution, ORA-24198 if the operation ID is invalid, or ORA-24199 if the Java stored procedure message store overflows.

flush (id IN PLS_INTEGER)
Takes the static variable in Jserv and synchronizes the content back to `aq$_jms_map_message`. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

clear_body (id IN PLS_INTEGER)
Sets the Java stored procedure static variable to empty payload. Parameter `id` is used to identify the slot where the Java object is stored in the Oracle Database JVM session memory. If `id` is null, a new slot is created for this PL/SQL object. Subsequent JMS operations on the payload need to provide this operation ID.

This function raises ORA-24197 if the Java stored procedure throws an exception during execution, ORA-24198 if the operation ID is invalid, or ORA-24199 if the Java stored procedure message store overflows.

clean (id IN PLS_INTEGER)
Closes and cleans up the `DataInputStream` or `DataOutputStream` at the Java stored procedure side corresponding to the operation ID. It is very important to call this procedure to avoid memory leaks. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

clean_all
Closes and cleans up all the messages in the corresponding type of message store at the Java stored procedure side. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution.

set_boolean (id IN PLS_INTEGER, name IN VARCHAR2, value IN BOOLEAN)
Sets the Boolean value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_byte (id IN PLS_INTEGER, name IN VARCHAR2, value IN PLS_INTEGER)
Sets the `BYTE` value with the specified name in the map. Because there is no `BYTE` type in PL/SQL, `PLS_INTEGER` is used to represent a byte. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_bytes (id IN PLS_INTEGER, name IN VARCHAR2, value IN RAW))
Sets the byte array value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_bytes (id IN PLS_INTEGER, name IN VARCHAR2, value IN RAW, offset IN INT,

length IN INT)
Sets a portion of the byte array value with the specified name in the map. Parameter `offset` is the initial offset within the byte array, and parameter `length` is the number of bytes to use. If the range [offset … offset+length] exceeds the boundary of the byte array value, then a Java `IndexOutOfBounds` exception is thrown in the Java stored procedure and this procedure raises an ORA-24197 error. The index starts from 0. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_bytes (id IN PLS_INTEGER, name IN VARCHAR2, value IN BLOB)
Sets the byte array value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_bytes (id IN PLS_INTEGER, name IN VARCHAR2, value IN BLOB, offset IN INT, length IN INT)
Sets a portion of the byte array value with the specified name in the map. Parameter `offset` is the initial offset within the byte array, and parameter `length` is the number of bytes to use. If the range [offset … offset+length] exceeds the boundary of the byte array value, then a Java `IndexOutOfBounds` exception is thrown in the Java stored procedure, and this procedure raises an ORA-24197 error. The index starts from 0. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_char (id IN PLS_INTEGER, name IN VARCHAR2, value IN CHAR)
Sets the character value with the specified name in the map. If this value has multiple characters, then it is the first character that is used. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_double (id IN PLS_INTEGER, name IN VARCHAR2, value IN DOUBLE PRECISION)
Sets the double value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_float (id IN PLS_INTEGER, name IN VARCHAR2, value IN FLOAT)
This procedure is to set the float value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_int (id IN PLS_INTEGER, name IN VARCHAR2, value IN PLS_INTEGER)
Sets the int value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_long (id IN PLS_INTEGER, name IN VARCHAR2, value IN NUMBER)
Sets the long value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_short (id IN PLS_INTEGER, name IN VARCHAR2, value IN PLS_INTEGER)
Sets the short value with the specified name in the map. Because there is no short type in PL/SQL, `PLS_INTEGER` is used to represent a short. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_string (id IN PLS_INTEGER, name IN VARCHAR2, value IN VARCHAR2)
Sets the string value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_string (id IN PLS_INTEGER, name IN VARCHAR2, value IN CLOB))
Sets the string value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

get_boolean (id IN PLS_INTEGER, name IN VARCHAR2) RETURN BOOLEAN
Retrieves the Boolean value with the specified name. If there is no item by this name, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

get_byte (id IN PLS_INTEGER, name IN VARCHAR2) RETURN PLS_INTEGER
Retrieves the `BYTE` value with the specified name. If there is no item by this name, then null is returned. Because there is no `BYTE` type in PL/SQL, `PLS_INTEGER` is used to represent a byte. Although the PL/SQL users get an `PLS_INTEGER`, they are guaranteed that the value is in the Java `BYTE` value range. If this value is issued with a `set_byte` function, then there will not be an out of range error. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

get_bytes (id IN PLS_INTEGER, name IN VARCHAR2, value OUT NOCOPY BLOB)
Retrieves the byte array value with the specified name. If there is no item by this name, then null is returned. Because the size of the array might be larger than the limit of PL/SQL `RAW` type, a `BLOB` is always returned here. The `BLOB` returned is a copy, which means it can be modified without affecting the message payload. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

get_char (id IN PLS_INTEGER, name IN VARCHAR2) RETURN CHAR
Retrieves and returns the character value with the specified name. If there is no item by this name, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid.

get_double (id IN PLS_INTEGER, name IN VARCHAR2) RETURN DOUBLE PRECISION
Retrieves and returns the double value with the specified name. If there is no item by this name, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid.

get_float (id IN PLS_INTEGER, name IN VARCHAR2) RETURN FLOAT
Retrieves the float value with the specified name. If there is no item by this name, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

get_int (id IN PLS_INTEGER, name IN VARCHAR2) RETURN PLS_INTEGER
Retrieves the `INT` value with the specified name. If there is no item by this name, then null is returned. Raises exception ORA-24194 if the type conversion between the type

of real value and the expected type is invalid, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

get_long (id IN PLS_INTEGER, name IN VARCHAR2) RETURN NUMBER
Retrieves the long value with the specified name. If there is no item by this name, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

get_short (id IN PLS_INTEGER, name IN VARCHAR2) RETURN PLS_INTEGER
Retrieves the short value with the specified name. If there is no item by this name, then null is returned. Because there is no `short` type in PL/SQL, `INT` is used to represent a `short`. Although the PL/SQL users get an `PLS_INTEGER`, they are guaranteed that the value is in the Java short value range. If this value is issued with a `set_short` function, then there will not be an out of range error. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

get_string (id IN PLS_INTEGER, name IN VARCHAR2, value OUT NOCOPY CLOB)
Retrieves the string value with the specified name. If there is no item by this name, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

get_names (id IN PLS_INTEGER) RETURN aq$_jms_namearray
Retrieves all the names within the map message and returns them in a varray. Because `aq$_jms_namearray` has a size as 1024 and each element is a `VARCHAR(200)`, this function will return an error if the size of the name array of the payload exceeds the limit. Raises exception ORA-24195 if the size of the name array or the size of a name exceeds the limit.

get_names (id IN PLS_INTEGER, names OUT aq$_jms_namearray, offset IN PLS_INTEGER, length IN PLS_INTEGER) RETURN PLS_INTEGER
Retrieves a portion of the names within the map message. Because `aq$_jms_namearray` has a size as 1024 and each element is a `VARCHAR(200)`, this function will return an error if either limits are exceeded during the retrieval. (This means there is no sense to put a `length` parameter greater than 1024.) The index of the names of a map messages begins from 0. Parameter `offset` is the offset from which to start retrieving.

The function returns the number of names that have been retrieved. The names retrieved is the intersection of the interval [offset, offset+length-1] and interval [0, size-1] where size is the size of this map message. If the intersection is an empty set, then names will be returned as null and the function returns 0 as the number of names retrieved. If users iterate the names by retrieving in small steps, then this can be used to test that there are no more names to read from map message.

Raises exception ORA-24195 if the size of the name array or the size of a name exceed the limit, ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

get_object (id IN PLS_INTEGER, name IN VARCHAR2, value OUT NOCOPY AQ$_JMS_VALUE)
Returns a general value ADT `AQ$_JMS_VALUE`. If there is no item by this name, then null is returned.Users can use the `type` attribute of this ADT to interpret the data. See

the map in the AQ$_JMS_VALUE ADT for the correspondence among dbms_jms_plsql package constants, Java data type and AQ$_JMS_VALUE attribute. Note this member procedure might bring additional overhead compared to other get member procedures or functions. It is used only if the user does not know the data type of the fields within a message before hand. Otherwise it is a good idea to use a specific get member procedure or function. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

get_size (id IN PLS_INTEGER) RETURN PLS_INTEGER
Retrieves the size of the map message. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

item_exists (id IN PLS_INTEGER, name IN VARCHAR2) RETURN BOOLEAN
Indicates that an item exists in this map message by returning TRUE. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

SYS.AQ$_JMS_STREAM_MESSAGE Type

This type is the ADT used to store a `StreamMessage` in an Oracle Streams AQ queue.

This section contains these topics:

Syntax

```
TYPE aq$_jms_stream_message AS object(
 header     aq$_jms_header,
 bytes_len  int,
 bytes_raw  raw(2000),
 bytes_lob  blob,
 STATIC FUNCTION  construct RETURN aq$_jms_stream_message,
 MEMBER PROCEDURE set_replyto  (replyto IN sys.aq$_agent),
 MEMBER PROCEDURE set_type     (type    IN VARCHAR),
 MEMBER FUNCTION get_replyto  RETURN sys.aq$_agent,
 MEMBER FUNCTION get_type     RETURN VARCHAR,
 MEMBER PROCEDURE set_userid   (userid   IN VARCHAR),
 MEMBER PROCEDURE set_appid    (appid    IN VARCHAR),
 MEMBER PROCEDURE set_groupid  (groupid  IN VARCHAR),
 MEMBER PROCEDURE set_groupseq (groupseq IN INT),
 MEMBER FUNCTION get_userid   RETURN VARCHAR,
 MEMBER FUNCTION get_appid    RETURN VARCHAR,
 MEMBER FUNCTION get_groupid  RETURN VARCHAR,
 MEMBER FUNCTION get_groupseq RETURN INT,
 MEMBER PROCEDURE clear_properties,
 MEMBER PROCEDURE set_boolean_property(property_name IN VARCHAR,
   property_value IN BOOLEAN),
 MEMBER PROCEDURE set_byte_property   (property_name IN VARCHAR,
   property_value IN INT),
 MEMBER PROCEDURE set_double_property (property_name IN VARCHAR,
   property_value IN DOUBLE PRECISION),
 MEMBER PROCEDURE set_float_property  (property_name IN VARCHAR,
   property_value IN FLOAT),
 MEMBER PROCEDURE set_int_property    (property_name IN VARCHAR,
   property_value IN INT),
 MEMBER PROCEDURE set_long_property   (property_name IN VARCHAR,
   property_value IN NUMBER),
 MEMBER PROCEDURE set_short_property  (property_name IN VARCHAR,
   property_valuE IN INT),
 MEMBER PROCEDURE set_string_property (property_name IN VARCHAR,
   property_value IN VARCHAR),
 MEMBER FUNCTION get_boolean_property (property_name IN VARCHAR) RETURN BOOLEAN,
 MEMBER FUNCTION get_byte_property    (property_name IN VARCHAR) RETURN INT,
 MEMBER FUNCTION get_double_property  (property_name IN VARCHAR)
   RETURN DOUBLE PRECISION,
 MEMBER FUNCTION get_float_property   (property_name IN VARCHAR) RETURN FLOAT,
 MEMBER FUNCTION get_int_property     (property_name IN VARCHAR) RETURN INT,
 MEMBER FUNCTION get_long_property    (property_name IN VARCHAR) RETURN NUMBER,
 MEMBER FUNCTION get_short_property   (property_name IN VARCHAR) RETURN INT,
 MEMBER FUNCTION get_string_property  (property_name IN VARCHAR) RETURN VARCHAR,
```

```
MEMBER PROCEDURE set_bytes           (payload IN RAW),
MEMBER PROCEDURE set_bytes           (payload IN BLOB),
MEMBER PROCEDURE get_bytes           (payload OUT RAW),
MEMBER PROCEDURE get_bytes           (payload OUT BLOB),
MEMBER FUNCTION  prepare             (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER PROCEDURE reset               (id IN PLS_INTEGER),
MEMBER PROCEDURE flush               (id IN PLS_INTEGER),
MEMBER PROCEDURE clear_body          (id IN PLS_INTEGER),
MEMBER PROCEDURE clean               (id IN PLS_INTEGER),
STATIC PROCEDURE clean_all,
MEMBER FUNCTION  get_mode       (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION  read_boolean   (id IN PLS_INTEGER) RETURN BOOLEAN,
MEMBER FUNCTION  read_byte      (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION  read_bytes     (id IN PLS_INTEGER) RETURN BLOB,
MEMBER PROCEDURE read_bytes     (id IN PLS_INTEGER, value OUT NOCOPY BLOB),
MEMBER FUNCTION  read_char      (id IN PLS_INTEGER) RETURN CHAR,
MEMBER FUNCTION  read_double    (id IN PLS_INTEGER) RETURN DOUBLE PRECISION,
MEMBER FUNCTION  read_float     (id IN PLS_INTEGER) RETURN FLOAT,
MEMBER FUNCTION  read_int       (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION  read_long      (id IN PLS_INTEGER) RETURN NUMBER,
MEMBER FUNCTION  read_short     (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION  read_string RETURN CLOB,
MEMBER PROCEDURE read_string    (id IN PLS_INTEGER, value OUT NOCOPY CLOB),
MEMBER PROCEDURE read_object    (id IN PLS_INTEGER,
  value OUT NOCOPY AQ$_JMS_VALUE),
MEMBER PROCEDURE write_boolean  (id IN PLS_INTEGER, value IN BOOLEAN),
MEMBER PROCEDURE write_byte     (id IN PLS_INTEGER, value IN INT),
MEMBER PROCEDURE write_bytes    (id IN PLS_INTEGER, value IN RAW),
MEMBER PROCEDURE write_bytes    (id IN PLS_INTEGER, value IN RAW,
  offset IN INT, length IN INT),
MEMBER PROCEDURE write_bytes    (id IN PLS_INTEGER, value IN BLOB),
MEMBER PROCEDURE write_bytes    (id IN PLS_INTEGER, value IN BLOB,
  offset IN INT, length IN INT),
MEMBER PROCEDURE write_char     (id IN PLS_INTEGER, value IN CHAR),
MEMBER PROCEDURE write_double   (id IN PLS_INTEGER, value IN DOUBLE PRECISION),
MEMBER PROCEDURE write_float    (id IN PLS_INTEGER, value IN FLOAT),
MEMBER PROCEDURE write_int      (id IN PLS_INTEGER, value IN PLS_INTEGER),
MEMBER PROCEDURE write_long     (id IN PLS_INTEGER, value IN NUMBER),
MEMBER PROCEDURE write_short    (id IN PLS_INTEGER, value IN PLS_INTEGER),
MEMBER PROCEDURE write_string   (id IN PLS_INTEGER, value IN VARCHAR2),
MEMBER PROCEDURE write_string   (id IN PLS_INTEGER, value IN CLOB));
```

CONSTRUCT Function

STATIC FUNCTION construct RETURN aq$_jms_stream_message
Creates an empty `aq$_jms_stream_message` object.

JMS Header Methods

set_replyto (replyto IN sys.aq$_agent)
Sets the `replyto` parameter, which corresponds to `JMSReplyTo` in JMS.

set_type (type IN VARCHAR)
Sets the JMS type, which can be any text, and which corresponds to `JMSType` in JMS.

get_replyto RETURN sys.aq$_agent
Returns `replyto`, which corresponds to `JMSReplyTo`.

get_type RETURN VARCHAR
Returns `type`, which corresponds to `JMSType`.

System Properties Methods

set_userid (userid IN VARCHAR)
Sets `userid`, which corresponds to `JMSXUserID` in JMS.

set_appid (appid IN VARCHAR)
Sets `appid`, which corresponds to `JMSXAppID` in JMS.

set_groupid (groupid IN VARCHAR)
Sets `groupid`, which corresponds to `JMSXGroupID` in JMS.

set_groupseq (groupseq IN INT)
Sets `groupseq`, which corresponds to `JMSXGroupSeq` in JMS.

get_userid RETURN VARCHAR
Returns `userid`, which corresponds to `JMSXUserID`.

get_appid RETURN VARCHAR
Returns `appid`, which corresponds to `JMSXAppID`.

get_groupid RETURN VARCHAR
Returns `groupid`, which corresponds to `JMSXGroupID`.

get_groupseq RETURN NUMBER
Returns `groupseq`, which corresponds to `JMSXGroupSeq`.

User Properties Methods

clear_properties
Clears all user properties. This procedure does not affect system properties.

set_boolean_property (property_name IN VARCHAR, property_value IN BOOLEAN)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value` in an internal representation. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_byte_property (property_name IN VARCHAR, property_value IN INT)
Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -128 to 127 (8-bits). This check is necessary because neither PL/SQL nor RDBMS defines the `BYTE` datatype. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_double_property (property_name IN VARCHAR, property_value IN DOUBLE PRECISION)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_float_property (property_name IN VARCHAR, property_value IN FLOAT)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_int_property (property_name IN VARCHAR, property_value IN INT)
Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -2147483648 to 2147483647 (32-bits). This check is necessary because in PL/SQL and Oracle Database, the `INT` datatype is 38 bits. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_long_property (property_name IN VARCHAR, property_value IN NUMBER)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. In PL/SQL and Oracle Database, the `NUMBER` datatype is 38 bits. In Java, the `long` datatype is 64 bits. Therefore, no range check is needed.Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_short_property (property_name IN VARCHAR, property_value IN INT)
Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -32768 to 32767 (16-bits). This check is necessary because neither PL/SQL nor RDBMS defines the `short` datatype. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_string_property (property_name IN VARCHAR, property_value IN VARCHAR)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

get_boolean_property (property_name IN VARCHAR) RETURN BOOLEAN
If the property with the corresponding property name passed in exists, and if it is a `BOOLEAN` property, then this function returns the value of the property. Otherwise it returns a null.

get_byte_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a `BYTE` property, then this function returns the value of the property. Otherwise it returns a null.

get_double_property (property_name IN VARCHAR) RETURN DOUBLE PRECISION
If the property with the corresponding property name passed in exists, and if it is a `DOUBLE` property, then this function returns the value of the property. Otherwise it returns a null.

get_float_property (property_name IN VARCHAR) RETURN FLOAT
If the property with the corresponding property name passed in exists, and if it is a `FLOAT` property, then this function returns the value of the property. Otherwise it returns a null.

get_int_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a `Integer` property, then this function returns the value of the property. Otherwise it returns a null.

get_long_property (property_name IN VARCHAR) RETURN NUMBER
If the property with the corresponding property name passed in exists, and if it is a `long` property, then this function returns the value of the property. Otherwise it returns a null.

get_short_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a `short` property, then this function returns the value of the property. Otherwise it returns a null.

get_string_property (property_name IN VARCHAR) RETURN VARCHAR
If the property with the corresponding property name passed in exists, and if it is a `STRING` property, then this function returns the value of the property. Otherwise it returns a null.

Payload Methods

get_bytes (payload OUT RAW)
Puts the internal payload into a RAW variable without any interpretation. The payload of type `aq$_jms_stream_message` is stored as either `RAW` or `BLOB` in the database. This member function gets a payload as raw bytes without interpreting it. Raises exception ORA-24190 if the length of internal payload is more than 32767.

get_bytes (payload OUT BLOB)
Puts the internal payload into a BLOB variable without any interpretation. The payload of type `aq$_jms_stream_message` is stored as either `RAW` or `BLOB` in the database. This member function gets a payload as a `BLOB` variable without interpreting it.

set_bytes (payload IN RAW)
Sets the internal payload as the RAW variable without any interpretation. The payload of type `aq$_jms_stream_message` is stored as either `RAW` or `BLOB` in the database. This member function sets a payload as raw bytes without interpreting it.

set_bytes (payload IN BLOB)
Sets the internal payload as the `BLOB` variable without any interpretation. The payload of type `aq$_jms_stream_message` is stored as either `RAW` or `BLOB` in the database. This member function sets a payload as a `BLOB` variable without interpreting it.

prepare (id IN PLS_INTEGER) RETURN PLS_INTEGER
Takes the byte array stored in `aq$_jms_stream_message` and decodes it as a Java object in the Java stored procedure. The result of the decoding is stored as a static variable in Jserv session memory. Parameter `id` is used to identify the slot where the Java object is stored in the Oracle Database JVM session memory. If `id` is null, then a new slot is created for this PL/SQL object. Subsequent JMS operations on the payload need to provide this operation ID.

This function also sets the message access mode to `MESSAGE_ACCESS_READONLY`. Subsequent calls of `write_XXX` procedure raise an ORA-24196 error. Users can call the `clear_body` procedure to set the message access mode to `MESSAGE_ACCESS_READONLY`.

This function raises ORA-24197 if the Java stored procedure throws an exception during execution, ORA-24198 if the operation ID is invalid, or ORA-24199 if the Java stored procedure message store overflows.

reset (id IN PLS_INTEGER)
Resets the starting position of the stream to the beginning and puts the stream message in `MESSAGE_ACCESS_READONLY` mode.

flush (id IN PLS_INTEGER)
Takes the static variable in Jserv and synchronizes the content back to `aq$_jms_stream_message`. This procedure will not affect the underlying access mode. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

clear_body (id IN PLS_INTEGER)
Sets the Java stored procedure static variable to empty payload. Parameter `id` is used to identify the slot where the Java object is stored in the Oracle Database JVM session memory. If `id` is null, a new slot is created for this PL/SQL object. Subsequent JMS operations on the payload need to provide this operation ID.

It also sets the message access mode to `MESSAGE_ACCESS_WRITEONLY`. Later calls of `read_XXX` procedure raise ORA-24196 error. Users can call the `reset` or `prepare` procedures to set the message access mode to `MESSAGE_ACCESS_READONLY`. Write-only and read-only modes affect only the payload functions of `AQ$_JMS_BYTES_MESSAGE`. They do not affect the header functions.

This function raises ORA-24197 if the Java stored procedure throws an exception during execution, ORA-24198 if the operation ID is invalid, or ORA-24199 if the Java stored procedure message store overflows.

clean (id IN PLS_INTEGER)
Closes and cleans up the `DataInputStream` or `DataOutputStream` at the Java stored procedure side corresponding to the operation ID. It is very important to call this procedure to avoid memory leaks. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

clean_all
Closes and cleans up all the messages in the corresponding type of message store at the Java stored procedure side. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution.

get_mode (id IN PLS_INTEGER) RETURN PLS_INTEGER
Returns the current mode of this message. The return value is either `SYS.dbms_aqjms.READ_ONLY` or SYS.`dbms_aqjms.WRITE_ONLY`. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

read_boolean (id IN PLS_INTEGER) RETURN BOOLEAN
Reads and returns a Boolean value from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_byte (id IN PLS_INTEGER) RETURN PLS_INTEGER
Reads and returns a byte value from the stream message. If the end of the message stream has been reached, then null is returned. Because there is no `BYTE` type in PL/SQL, `INT` is used to represent a byte. Although PL/SQL users get an `INT`, they are guaranteed that the value is in the Java `BYTE` value range. If this value is issued with a `write_byte` function, then there will not be an out of range error. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_bytes (id IN PLS_INTEGER) RETURN BLOB
Reads and returns a byte array from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid or ORA-24196 if the stream message is in write-only mode.

read_bytes (id IN PLS_INTEGER, value OUT NOCOPY BLOB)
Reads a byte array from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_char (id IN PLS_INTEGER) RETURN CHAR
Reads and returns a character value from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_double (id IN PLS_INTEGER) RETURN DOUBLE PRECISION
Reads and returns a double from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_float (id IN PLS_INTEGER) RETURN FLOAT
Reads and returns a float from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_int (id IN PLS_INTEGER) RETURN PLS_INTEGER
Reads and returns an `INT` from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_long (id IN PLS_INTEGER) RETURN NUMBER
Reads and returns a long from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_short (id IN PLS_INTEGER) RETURN PLS_INTEGER
Reads and returns a short value from the stream message. If the end of the message stream has been reached, then null is returned. Because there is no short type in PL/SQL, `INT` is used to represent a byte. Although PL/SQL users get an `INT`, they are guaranteed that the value is in the Java short value range. If this value is issued with a `write_short` function, then there will not be an out of range error. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type

is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_string RETURN CLOB
Reads and returns a string from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid or ORA-24196 if the stream message is in write-only mode.

read_string (id IN PLS_INTEGER, value OUT NOCOPY CLOB)
Reads a string from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_object (id IN PLS_INTEGER, value OUT NOCOPY AQ$_JMS_VALUE)
Returns a general value ADT `AQ$_JMS_VALUE`. Users can use the type attribute of this ADT to interpret the data. See Table 187–2 on page 187-57 for the correspondence among `dbms_jms_plsql` package constants, Java data type and `AQ$_JMS_VALUE` attribute. This member procedure might bring additional overhead compared to other read member procedures or functions. It is used only if the user does not know the data type of the fields within a message beforehand. Otherwise it is a good idea to use a specific read member procedure or function.

Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_boolean (id IN PLS_INTEGER, value IN BOOLEAN)
Writes a Boolean to the stream message. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_byte (id IN PLS_INTEGER, value IN INT)
Writes a byte to the stream message. Because there is no `BYTE` type in PL/SQL, `INT` is used to represent a byte. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_bytes (id IN PLS_INTEGER, value IN RAW)
Writes a byte array field to the stream message. Consecutively written byte array fields are treated as two distinct fields when the fields are read. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_bytes (id IN PLS_INTEGER, value IN RAW, offset IN INT, length IN INT)
Writes a portion of a byte array as a byte array field to the stream message. Consecutively written byte array fields are treated as two distinct fields when the fields are read. Parameter `offset` is the initial offset within the byte array, and parameter `length` is the number of bytes to use. If the range [offset, offset+length] exceeds the boundary of the byte array value, then a Java `IndexOutOfBounds` exception is thrown in the Java stored procedure. The index starts from 0.

Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_bytes (id IN PLS_INTEGER, value IN BLOB)
Writes a byte array field to the stream message. Consecutively written byte array fields are treated as two distinct fields when the fields are read. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_bytes (id IN PLS_INTEGER, value IN BLOB, offset IN INT, length IN INT)
Writes a portion of a byte array as a byte array field to the stream message. Consecutively written byte array fields are treated as two distinct fields when the fields are read. Parameter `offset` is the initial offset within the byte array, and parameter `length` is the number of bytes to use. If the range [offset, offset+length] exceeds the boundary of the byte array value, then a Java `IndexOutOfBounds` exception is thrown in the Java stored procedure. The index starts from 0.

Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_char (id IN PLS_INTEGER, value IN CHAR)
Writes a character value to the stream message. If this value has multiple characters, then it is the first character that is written. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_double (id IN PLS_INTEGER, value IN DOUBLE PRECISION)
Writes a double to the stream message. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_float (id IN PLS_INTEGER, value IN FLOAT)
Writes a float to the stream message. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_int (id IN PLS_INTEGER, value IN PLS_INTEGER)
Writes an `INT` to the stream message. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_long (id IN PLS_INTEGER, value IN NUMBER)
Writes a long to the stream message. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_short (id IN PLS_INTEGER, value IN PLS_INTEGER)
Writes a short to the stream message. Because there is no short type in PL/SQL, `INT` is used to represent a short. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_string (id IN PLS_INTEGER, value IN VARCHAR2)
Writes a string to the stream message. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_string (id IN PLS_INTEGER, value IN CLOB)
Writes a string to the stream message. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

SYS.AQ$_JMS_OBJECT_MESSAGE Type

This type is the ADT used to store an `ObjectMessage` in an Oracle Streams AQ queue.

Syntax

```
TYPE aq$_jms_object_message AS object(
  header     aq$_jms_header,
  bytes_len  int,
  bytes_raw  raw(2000),
  bytes_lob  blob);
```

SYS.AQ$_JMS_NAMESARRAY Type

This type represents the name array returned by the `get_names` procedure of `aq$_jms_map_message`. The maximum number of names this type can hold is 1024. The maximum length of each name is 200 characters.

Syntax

```
CREATE OR REPLACE TYPE AQ$_JMS_NAMESARRAY AS VARRAY(1024) OF VARCHAR(100);
```

Usage Notes

If the names array in the message payload is greater than 1024, then use the following function to retrieve the names in multiple portions:

```
MEMBER FUNCTION get_names(id IN PLS_INTEGER, names OUT aq$_jms_namearray,
   offset IN PLS_INTEGER, length IN PLS_INTEGER) RETURN PLS_INTEGER;
```

SYS.AQ$_JMS_VALUE Type

This type represents the general data returned by the `get_object` procedure of `aq$_jms_map_message` and the `read_object` procedure of `aq$_jms_stream_message`. The `type` field in this ADT is used to decide which type of data this object is really holding. Table 187–2 lists the mapping between the `sys.dbms_jms_plsql` type constants, the corresponding Java type, and the data field of ADT `aq$_jms_value` which effectively holds the data.

Syntax

```
CREATE OR REPLACE TYPE AQ$_JMS_VALUE AS object(
 type      number(2),
 num_val   number,
 char_val  char(1),
 text_val  clob,
 bytes_val blob);
```

Table 187–2 AQ$_JMS_VALUE Type Fields and Java Fields

Type	Java Type	aq$_jms_value Data Field
DBMS_JMS_PLSQL.DATA_TYPE_BYTE	byte	num_val
DBMS_JMS_PLSQL.DATA_TYPE_SHORT	short	num_val
DBMS_JMS_PLSQL.DATA_TYPE_INTEGER	int	num_val
DBMS_JMS_PLSQL.DATA_TYPE_LONG	long	num_val
DBMS_JMS_PLSQL.DATA_TYPE_FLOAT	float	num_val
DBMS_JMS_PLSQL.DATA_TYPE_DOUBLE	double	num_val
DBMS_JMS_PLSQL.DATA_TYPE_BOOLEAN	boolean	num_val: 0 FALSE, 1 TRUE
DBMS_JMS_PLSQL.DATA_TYPE_CHARACTER	char	char_val
DBMS_JMS_PLSQL.DATA_TYPE_STRING	java.lang.String	text_val
DBMS_JMS_PLSQL.DATA_TYPE_BYTES	byte[]	bytes_val

SYS.AQ$_JMS_EXCEPTION Type

This type represents a Java exception thrown on the Java stored procedure side. The `id` field is reserved for future use. The `exp_name` stores the Java exception name, the `err_msg` field stores the Java exception error message, and the `stack` field stores the stack trace of the Java exception.

Syntax

```
CREATE OR REPLACE TYPE AQ$_JMS_EXCEPTION AS OBJECT (
   id          number, -- Reserved and not used. Right now always return 0.
   exp_name    varchar(200),
   err_msg     varchar(500),
   stack       varchar(4000));
```

188

Logical Change Record TYPEs

This chapter describes the logical change record (LCR) types. In Streams, LCRs are message payloads that contain information about changes to a database. These changes can include changes to the data, which are data manipulation language (DML) changes, and changes to database objects, which are data definition language (DDL) changes.

When you use Streams, the capture process captures changes in the form of LCRs and enqueues them into a queue. These LCRs can be propagated from a queue in one database to a queue in another database. Finally, the apply process can apply LCRs at a destination database. You also have the option of creating, enqueuing, and dequeuing LCRs manually.

> **See Also:** *Oracle Streams Concepts and Administration* for more information about LCRs

This chapter contains these topics:

Summary of Logical Change Record Types

Table 188–1 Logical Change Record (LCR) Types

Type	Description
"LCR$_DDL_RECORD Type" on page 188-3	Represents a data definition language (DDL) change to a database object
"LCR$_ROW_RECORD Type" on page 188-11	Represents a data manipulation language (DML) change to a database object
"LCR$_ROW_LIST Type" on page 188-34	Identifies a list of column values for a row in a table
"LCR$_ROW_UNIT Type" on page 188-35	Identifies the value for a column in a row

These logical change record (LCR) types can be used with the following Oracle-supplied PL/SQL packages:

- DBMS_APPLY_ADM
- DBMS_AQ
- DBMS_AQADM
- DBMS_CAPTURE_ADM
- DBMS_PROPAGATION_ADM
- DBMS_RULE
- DBMS_RULE_ADM
- DBMS_STREAMS
- DBMS_STREAMS_ADM
- DBMS_TRANSFORM

LCR$_DDL_RECORD Type

This type represents a data definition language (DDL) change to a database object.

If you create or modify a DDL logical change record (DDL LCR), then make sure the `ddl_text` is consistent with the `base_table_name`, `base_table_owner`, `object_type`, `object_owner`, `object_name`, and `command_type` attributes.

This section contains information about the constructor for DDL LCRs and information about the member subprograms for this type:

- LCR$_DDL_RECORD Constructor
- Summary of LCR$_DDL_RECORD Subprograms, which also include the subprograms described in "Common Subprograms for LCR$_DDL_RECORD and LCR$_ROW_RECORD" on page 188-26

Note:

- When passing a name as a parameter to an LCR constructor, you can enclose the name in double quotes to handle names that use mixed case or lower case for database objects. For example, if a name contains any lower case characters, then you must enclose it in double quotes.
- The application does not need to specify a transaction identifier or SCN when it creates an LCR because the apply process generates these values and stores them in memory. If a transaction identifier or SCN is specified in the LCR, then the apply process ignores it and assigns a new value.

LCR$_DDL_RECORD Constructor

Creates a `SYS.LCR$_DDL_RECORD` object with the specified information.

```
STATIC FUNCTION CONSTRUCT(
   source_database_name  IN  VARCHAR2,
   command_type          IN  VARCHAR2,
   object_owner          IN  VARCHAR2,
   object_name           IN  VARCHAR2,
   object_type           IN  VARCHAR2,
   ddl_text              IN  CLOB,
   logon_user            IN  VARCHAR2,
   current_schema        IN  VARCHAR2,
   base_table_owner      IN  VARCHAR2,
   base_table_name       IN  VARCHAR2,
   tag                   IN  RAW       DEFAULT NULL,
   transaction_id        IN  VARCHAR2  DEFAULT NULL,
   scn                   IN  NUMBER    DEFAULT NULL)
RETURN SYS.LCR$_DDL_RECORD;
```

LCR$_DDL_RECORD Constructor Function Parameters

Table 188–2 Constructor Function Parameters for LCR$_DDL_RECORD

Parameter	Description
source_database_name	The database where the DDL statement occurred. If you do not include the domain name, then the function appends the local domain to the database name automatically. For example, if you specify DBS1 and the local domain is .NET, then the function specifies DBS1.NET automatically. This parameter should be set to a non-NULL value.
command_type	The type of command executed in the DDL statement. This parameter should be set to a non-NULL value. **See Also:** The "SQL Command Codes" table in the *Oracle Call Interface Programmer's Guide* for a complete list of command types The following command types *are not supported* in DDL LCRs: ALTER MATERIALIZED VIEW ALTER MATERIALIZED VIEW LOG ALTER SUMMARY CREATE SCHEMA CREATE MATERIALIZED VIEW CREATE MATERIALIZED VIEW LOG CREATE SUMMARY DROP MATERIALIZED VIEW DROP MATERIALIZED VIEW LOG DROP SUMMARY RENAME The snapshot equivalents of the materialized view command types are also not supported.
object_owner	The user who owns the object on which the DDL statement was executed
object_name	The database object on which the DDL statement was executed
object_type	The type of object on which the DDL statement was executed. The following are valid object types: CLUSTER FUNCTION INDEX LINK OUTLINE PACKAGE PACKAGE BODY PROCEDURE SEQUENCE SYNONYM TABLE TRIGGER TYPE USER VIEW LINK represents a database link. NULL is also a valid object type. Specify NULL for all object types not listed. The GET_OBJECT_TYPE member procedure returns NULL for object types not listed.

Table 188–2 (Cont.) Constructor Function Parameters for LCR$_DDL_RECORD

Parameter	Description
ddl_text	The text of the DDL statement. This parameter should be set to a non-NULL value.
logon_user	The user whose session executed the DDL statement
current_schema	The schema that is used if no schema is specified explicitly for the modified database objects in ddl_text. If a schema is specified in ddl_text that differs from the one specified for current_schema, then the function uses the schema specified in ddl_text. This parameter should be set to a non-NULL value.
base_table_owner	If the DDL statement is a table related DDL (such as CREATE TABLE and ALTER TABLE), or if the DDL statement involves a table (such as creating a trigger on a table), then base_table_owner specifies the owner of the table involved. Otherwise, base_table_owner is NULL.
base_table_name	If the DDL statement is a table related DDL (such as CREATE TABLE and ALTER TABLE), or if the DDL statement involves a table (such as creating a trigger on a table), then base_table_name specifies the name of the table involved. Otherwise, base_table_name is NULL.
tag	A binary tag that enables tracking of the LCR. For example, this tag can be used to determine the original source database of the DDL statement if apply forwarding is used. **See Also:** *Oracle Streams Replication Administrator's Guide* for more information about tags
transaction_id	The identifier of the transaction
scn	The SCN at the time when the change record for a captured LCR was written to the redo log. The SCN value is meaningless for a user-created LCR.

Summary of LCR$_DDL_RECORD Subprograms

Table 188–3 LCR$_DDL_RECORD Type Subprograms

Subprogram	Description
"EXECUTE Member Procedure" on page 188-6	Executes the LCR under the security domain of the current user
"GET_BASE_TABLE_NAME Member Function" on page 188-6	Returns the base (dependent) table name
"GET_BASE_TABLE_OWNER Member Function" on page 188-6	Returns the base (dependent) table owner
"GET_CURRENT_SCHEMA Member Function" on page 188-6	Returns the default schema (user) name
"GET_DDL_TEXT Member Procedure" on page 188-7	Gets the DDL text in a CLOB
"SET_LOGON_USER Member Procedure" on page 188-9	Returns the logon user name
"GET_OBJECT_TYPE Member Function" on page 188-7	Returns the type of the object involved for the DDL
"SET_BASE_TABLE_NAME Member Procedure" on page 188-8	Sets the base (dependent) table name

Table 188–3 (Cont.) LCR$_DDL_RECORD Type Subprograms

Subprogram	Description
"SET_BASE_TABLE_OWNER Member Procedure" on page 188-8	Sets the base (dependent) table owner
"SET_CURRENT_SCHEMA Member Procedure" on page 188-8	Sets the default schema (user) name
"SET_DDL_TEXT Member Procedure" on page 188-8	Sets the DDL text
"SET_LOGON_USER Member Procedure" on page 188-9	Sets the logon user name
"SET_OBJECT_TYPE Member Procedure" on page 188-9	Sets the object type
Common Subprograms	See "Common Subprograms for LCR$_DDL_RECORD and LCR$_ROW_RECORD" on page 188-26 for a list of subprograms common to the `SYS.LCR$_ROW_RECORD` and `SYS.LCR$_DDL_RECORD` types

EXECUTE Member Procedure

Executes the DDL LCR under the security domain of the current user. Any apply process handlers that would be run for an LCR are not run when the LCR is applied using this procedure.

Note: The `EXECUTE` member procedure can be invoked only in an apply handler for an apply process.

Syntax

```
MEMBER PROCEDURE EXECUTE();
```

GET_BASE_TABLE_NAME Member Function

Returns the base (dependent) table name.

Syntax

```
MEMBER FUNCTION GET_BASE_TABLE_NAME()
RETURN VARCHAR2;
```

GET_BASE_TABLE_OWNER Member Function

Returns the base (dependent) table owner.

Syntax

```
MEMBER FUNCTION GET_BASE_TABLE_OWNER()
RETURN VARCHAR2;
```

GET_CURRENT_SCHEMA Member Function

Returns the current schema name.

Syntax

```
MEMBER FUNCTION GET_CURRENT_SCHEMA()
RETURN VARCHAR2;
```

GET_DDL_TEXT Member Procedure

Gets the DDL text in a CLOB.

The following is an example of a PL/SQL procedure that uses this procedure to get the DDL text in a DDL LCR:

```
CREATE OR REPLACE PROCEDURE ddl_in_lcr (ddl_lcr in SYS.LCR$_DDL_RECORD)
IS
  ddl_text   CLOB;
BEGIN
  DBMS_OUTPUT.PUT_LINE( '  ----------------------------------------' );
  DBMS_OUTPUT.PUT_LINE( '  Displaying DDL text in a DDL LCR: ' );
  DBMS_OUTPUT.PUT_LINE( '  ----------------------------------------' );
  DBMS_LOB.CREATETEMPORARY(ddl_text, true);
  ddl_lcr.GET_DDL_TEXT(ddl_text);
  DBMS_OUTPUT.PUT_LINE('DDL text:' || ddl_text);
  DBMS_LOB.FREETEMPORARY(ddl_text);
END;
/
```

> **Note:** GET_DDL_TEXT is a member procedure and not a member function to make it easier for you to manage the space used by the CLOB. Notice that the previous example creates temporary space for the CLOB and then frees the temporary space when it is no longer needed.

Syntax

```
MEMBER FUNCTION GET_DDL_TEXT
  ddl_text  IN/OUT  CLOB);
```

Parameter

Table 188–4 GET_DDL_TEXT Procedure Parameter

Parameter	Description
ddl_text	The DDL text in the DDL LCR

GET_LOGON_USER Member Function

Returns the logon user name.

Syntax

```
MEMBER FUNCTION GET_LOGON_USER()
RETURN VARCHAR2;
```

GET_OBJECT_TYPE Member Function

Returns the type of the object involved for the DDL.

Syntax

```
MEMBER FUNCTION GET_OBJECT_TYPE()
```

```
RETURN VARCHAR2;
```

SET_BASE_TABLE_NAME Member Procedure

Sets the base (dependent) table name.

Syntax

```
MEMBER PROCEDURE SET_BASE_TABLE_NAME(
   base_table_name  IN  VARCHAR2);
```

Parameter

Table 188–5 SET_BASE_TABLE_NAME Procedure Parameter

Parameter	Description
base_table_name	The name of the base table

SET_BASE_TABLE_OWNER Member Procedure

Sets the base (dependent) table owner.

Syntax

```
MEMBER PROCEDURE SET_BASE_TABLE_OWNER(
   base_table_owner  IN  VARCHAR2);
```

Parameter

Table 188–6 SET_BASE_TABLE_OWNER Procedure Parameter

Parameter	Description
base_table_owner	The name of the base owner

SET_CURRENT_SCHEMA Member Procedure

Sets the default schema (user) name.

Syntax

```
MEMBER PROCEDURE SET_CURRENT_SCHEMA(
   current_schema  IN  VARCHAR2);
```

Parameter

Table 188–7 SET_CURRENT_SCHEMA Procedure Parameter

Parameter	Description
current_schema	The name of the schema to set as the current schema. This parameter should be set to a non-NULL value.

SET_DDL_TEXT Member Procedure

Sets the DDL text.

Syntax

```
MEMBER PROCEDURE SET_DDL_TEXT(
   ddl_text  IN  CLOB);
```

Parameter

Table 188–8 SET_DDL_TEXT Procedure Parameter

Parameter	Description
ddl_text	The DDL text. This parameter should be set to a non-NULL value.

SET_LOGON_USER Member Procedure

Sets the logon user name.

Syntax

```
MEMBER PROCEDURE SET_LOGON_USER(
   logon_user  IN  VARCHAR2);
```

Parameter

Table 188–9 SET_LOGON_USER Procedure Parameter

Parameter	Description
logon_user	The name of the schema to set as the logon user

SET_OBJECT_TYPE Member Procedure

Sets the object type.

Syntax

```
MEMBER PROCEDURE SET_OBJECT_TYPE(
   object_type  IN  VARCHAR2);
```

Parameter

Table 188–10 SET_OBJECT_TYPE Procedure Parameter

Parameter	Description
object_type	The object type. The following are valid object types: CLUSTER FUNCTION INDEX LINK OUTLINE PACKAGE PACKAGE BODY PROCEDURE SEQUENCE SYNONYM TABLE TRIGGER TYPE USER VIEW LINK represents a database link. NULL is also a valid object type. Specify NULL for all object types not listed. The GET_OBJECT_TYPE member procedure returns NULL for object types not listed.

LCR$_ROW_RECORD Type

This type represents a data manipulation language (DML) change to a row in a table. This type uses the `LCR$_ROW_LIST` type.

If you create or modify a row logical change record (row LCR), then make sure the `command_type` attribute is consistent with the presence or absence of old column values and the presence or absence of new column values.

This section contains information about the constructor for DDL LCRs and information about the member subprograms for this type:

- LCR$_ROW_RECORD Constructor
- Summary of LCR$_ROW_RECORD Subprograms, which also include the subprograms described in Common Subprograms for LCR$_DDL_RECORD and LCR$_ROW_RECORD on page 188-26

Note:

- When passing a name as a parameter to an LCR constructor, you can enclose the name in double quotes to handle names that use mixed case or lower case for database objects. For example, if a name contains any lower case characters, then you must enclose it in double quotes.
- The application does not need to specify a transaction identifier or SCN when it creates an LCR because the apply process generates these values and stores them in memory. If a transaction identifier or SCN is specified in the LCR, then the apply process ignores it and assigns a new value.

See Also: "LCR$_ROW_LIST Type" on page 188-34

LCR$_ROW_RECORD Constructor

Creates a `SYS.LCR$_ROW_RECORD` object with the specified information.

```
STATIC FUNCTION CONSTRUCT(
   source_database_name  IN  VARCHAR2,
   command_type          IN  VARCHAR2,
   object_owner          IN  VARCHAR2,
   object_name           IN  VARCHAR2,
   tag                   IN  RAW                DEFAULT NULL,
   transaction_id        IN  VARCHAR2           DEFAULT NULL,
   scn                   IN  NUMBER             DEFAULT NULL,
   old_values            IN  SYS.LCR$_ROW_LIST  DEFAULT NULL,
   new_values            IN  SYS.LCR$_ROW_LIST  DEFAULT NULL)
RETURN SYS.LCR$_ROW_RECORD;
```

LCR$_ROW_RECORD Constructor Function Parameters

Table 188–11 Constructor Function Parameters for LCR$_ROW_RECORD

Parameter	Description
source_database_name	The database where the row change occurred. If you do not include the domain name, then the function appends the local domain to the database name automatically. For example, if you specify DBS1 and the local domain is .NET, then the function specifies DBS1.NET automatically. This parameter should be set to a non-NULL value.
command_type	The type of command executed in the DML statement. This parameter should be set to a non-NULL value. Valid values are the following: INSERT UPDATE DELETE LOB ERASE LOB WRITE LOB TRIM If INSERT, then an LCR should have a new_values collection that is not empty and an empty or NULL old_values collection. If UPDATE, then an LCR should have a new_values collection that is not empty and an old_values collection that is not empty. If DELETE, then an LCR should have a NULL or empty new_values collection and an old_values collection that is not empty. If LOB ERASE, LOB WRITE, or LOB TRIM, then an LCR should have a new_values collection that is not empty and an empty or NULL old_values collection.
object_owner	The user who owns the table on which the row change occurred. This parameter should be set to a non-NULL value.
object_name	The table on which the DML statement was executed. This parameter should be set to a non-NULL value.
tag	A binary tag that enables tracking of the LCR. For example, this tag can be used to determine the original source database of the DML change when apply forwarding is used. **See Also:** *Oracle Streams Replication Administrator's Guide* for more information about tags
transaction_id	The identifier of the transaction
scn	The SCN at the time when the change record was written to the redo log. The SCN value is meaningless for a user-created LCR.
old_values	The column values for the row before the DML change. If the DML statement is an UPDATE or a DELETE statement, then this parameter contains the values of columns in the row before the DML statement. If the DML statement is an INSERT statement, then there are no old values.

Table 188–11 (Cont.) Constructor Function Parameters for LCR$_ROW_RECORD

Parameter	Description
new_values	The column values for the row after the DML change. If the DML statement is an UPDATE or an INSERT statement, then this parameter contains the values of columns in the row after the DML statement. If the DML statement is a DELETE statement, then there are no new values. If the LCR reflects a LOB operation, then this parameter contains the supplementally logged columns and any relevant LOB information.

Summary of LCR$_ROW_RECORD Subprograms

Table 188–12 LCR$_ROW_RECORD Type Subprograms

Subprogram	Description
"ADD_COLUMN Member Procedure" on page 188-14	Adds the value as old or new, depending on the value type specified, for the column
"CONVERT_LONG_TO_LOB_CHUNK Member Procedure" on page 188-15	Converts LONG data in a row LCR into fixed width CLOB, or converts LONG RAW data in a row LCR into a BLOB
"DELETE_COLUMN Member Procedure" on page 188-15	Deletes the old value, the new value, or both, for the specified column, depending on the value type specified
"EXECUTE Member Procedure" on page 188-16	Executes the LCR under the security domain of the current user
"GET_LOB_INFORMATION Member Function" on page 188-17	Gets the LOB information for the column
"GET_LOB_OFFSET Member Function" on page 188-18	Returns the LOB offset for the specified column
"GET_LOB_OPERATION_SIZE Member Function" on page 188-18	Gets the operation size for the LOB column
"GET_LONG_INFORMATION Member Function" on page 188-19	Gets the LONG information for the column
"GET_VALUE Member Function" on page 188-20	Returns the old or new value for the specified column, depending on the value type specified
"GET_VALUES Member Function" on page 188-20	Returns a list of old or new values, depending on the value type specified
"RENAME_COLUMN Member Procedure" on page 188-21	Renames a column in an LCR
"SET_LOB_INFORMATION Member Procedure" on page 188-21	Sets LOB information for the column
"SET_LOB_OFFSET Member Procedure" on page 188-22	Sets the LOB offset for the specified column
"SET_LOB_OPERATION_SIZE Member Procedure" on page 188-22	Sets the operation size for the LOB column
"SET_VALUE Member Procedure" on page 188-23	Overwrites the value of the specified column
"SET_VALUES Member Procedure" on page 188-24	Replaces the existing old or new values for the LCR, depending on the value type specified

Table 188–12 (Cont.) LCR$_ROW_RECORD Type Subprograms

Subprogram	Description
Common Subprograms	See Common Subprograms for LCR$_DDL_RECORD and LCR$_ROW_RECORD on page 188-26 for a list of subprograms common to the SYS.LCR$_ROW_RECORD and SYS.LCR$_DDL_RECORD types

ADD_COLUMN Member Procedure

Adds the value as old or new, depending on the value type specified, for the column. An error is raised if a value of the same type already exists for the column.

Note: To set a column value that already exists, run SET_VALUE.

See Also: "SET_VALUE Member Procedure" on page 188-23

Considerations for LOB Columns

When processing a row LCR with LOB columns with a DML handler or error handler and the handler is using LOB assembly (the assemble_lobs parameter is set to TRUE for the handler), you use this member procedure in the handler procedure to add a LOB column to a row LCR. If assemble_lobs is set to FALSE for the handler, then you cannot use this member procedure to add a LOB column to a row LCR.

To use a DML or error handler to add a LOB column, specify the LOB locator for the column_value parameter in the member procedure. The ADD_COLUMN member procedure verifies that a ANYDATA encapsulated LOB locator is processed with a DML or error handler that is using LOB assembly. An error is raised under the following conditions:

- The handler attempts to enqueue a row LCR with a ANYDATA encapsulated LOB locator.
- An attempt is made to add an LOB column that is set incorrectly.

If an error is raised because of one of these conditions, then the transaction that includes the row LCR is moved to the error queue, and the LOB is represented by the original (nonassembled) row LCRs.

Note:

- Database compatibility must be 10.2.0 or higher to use LOB assembly.
- When you are processing a row LCR with a rule-based transformation, you cannot use this member procedure to add a LOB column.
- When you are processing a row LCR with a rule-based transformation, DML handler, or error handler, you cannot use this member procedure to add a LONG or LONG RAW column.

Syntax

```
MEMBER PROCEDURE ADD_COLUMN(
   value_type    IN  VARCHAR2,
```

```
   column_name   IN  VARCHAR2,
   column_value  IN  ANYDATA);
```

Parameters

Table 188–13 ADD_COLUMN Procedure Parameters

Parameter	Description
value_type	The type of value to add for the column. Specify old to add the old value of the column. Specify new to add the new value of the column.
column_name	The column name. This name is not validated. An error can be raised during application of the LCRs if an invalid name is specified.
column_value	The value of the column. If NULL, then this procedure raises an error. If the member procedure is used in a DML handler or error handler that uses LOB assembly, then a LOB locator can be specified. A NULL column value can be specified by encapsulating the NULL value in a ANYDATA wrapper.

CONVERT_LONG_TO_LOB_CHUNK Member Procedure

Converts LONG data in a row LCR into a CLOB, or converts LONG RAW data in a row LCR into a BLOB.

This procedure can change the operation code from LONG WRITE to LOB WRITE for the row LCR.

This member procedure can be used in rule-based transformations.

The following restrictions apply to this member procedure:

- This member procedure cannot be used in apply handlers.
- LONG data can be sent as a part of a row LCR with one of the following operation codes: INSERT, UPDATE, or LONG_WRITE. Because LONG data can be sent in multiple pieces, make sure that this method is invoked on either none or all LONG pieces.
- LOB to LONG conversion is not supported.
- A row LCR on which this procedure is executed must have been created by a capture process. That is, this procedure does not support user-enqueued row LCRs.

See Also: *Oracle Streams Replication Administrator's Guide*

Syntax

```
MEMBER PROCEDURE CONVERT_LONG_TO_LOB_CHUNK();
```

DELETE_COLUMN Member Procedure

Deletes the old value, the new value, or both, for the specified column, depending on the value type specified.

Syntax

```
MEMBER PROCEDURE DELETE_COLUMN(
```

```
  column_name  IN  VARCHAR2,
  value_type   IN  VARCHAR2  DEFAULT '*');
```

Parameters

Table 188–14 DELETE_COLUMN Procedure Parameters

Parameter	Description
column_name	The column name. An error is raised if the column does not exist in the LCR.
value_type	The type of value to delete for the column. Specify old to delete the old value of the column. Specify new to delete the new value of the column. If * is specified, then the procedure deletes both the old and new values.

EXECUTE Member Procedure

Executes the row LCR under the security domain of the current user. Any apply process handlers that would be run for an LCR are not run when the LCR is applied using this procedure.

This member procedure can be run on a row LCR under any of the following conditions:

- The LCR is being processed by an apply handler.
- The LCR is in a queue and was last enqueued by an apply process, an application, or a user.
- The LCR has been constructed using the LCR$_ROW_RECORD constructor function but has not been enqueued.
- The LCR is in the error queue.

Note: A custom rule-based transformation should not run this member procedure on a row LCR. Doing so could execute the row LCR outside of its transactional context.

Considerations for LOB Columns

When processing a row LCR with LOB columns with a DML handler or error handler, and the handler is using LOB assembly (the assemble_lobs parameter is set to TRUE for the handler), this member procedure executes the assembled row LCR. An assembled row LCR represents a LOB value with a LOB locator or NULL.

If assemble_lobs is set to FALSE for the handler, then this member procedure executes the nonassembled row LCRs. Nonassembled row LCRs represent LOB values with VARCHAR2 and RAW datatypes. These nonassembled row LCRs might have been modified by the handler.

An error is raised under the following conditions:

- A DML or error handler configured with assemble_lobs set to FALSE attempts to execute a row LCR that contains a LOB locator.
- A DML or error handler configured with assemble_lobs set to TRUE attempts to execute a row LCR that contains one or more LOB values represented with VARCHAR2 or RAW datatypes.

If an error is raised because of one of these conditions, then the transaction that includes the row LCR is moved to the error queue, and the LOB is represented by the original (nonassembled) row LCRs.

Syntax

```
MEMBER PROCEDURE EXECUTE(
   conflict_resolution  IN  BOOLEAN);
```

Parameters

Table 188–15 EXECUTE Procedure Parameters

Parameter	Description
conflict_resolution	If TRUE, then any conflict resolution defined for the table using the SET_UPDATE_CONFLICT_HANDLER procedure in the DBMS_APPLY_ADM package is used to resolve conflicts resulting from the execution of the LCR. If FALSE, then conflict resolution is not used. An error is raised if this parameter is not specified or is set to NULL.

GET_LOB_INFORMATION Member Function

Gets the LOB information for the column.

The return value can be one of the following:

```
DBMS_LCR.NOT_A_LOB        CONSTANT NUMBER := 1;
DBMS_LCR.NULL_LOB         CONSTANT NUMBER := 2;
DBMS_LCR.INLINE_LOB       CONSTANT NUMBER := 3;
DBMS_LCR.EMPTY_LOB        CONSTANT NUMBER := 4;
DBMS_LCR.LOB_CHUNK        CONSTANT NUMBER := 5;
DBMS_LCR.LAST_LOB_CHUNK   CONSTANT NUMBER := 6;
```

Returns NULL if the specified column does not exist.

If the command type of the row LCR is UPDATE, then specifying 'Y' for the use_old parameter is a convenient way to get the value of the columns.

Syntax

```
MEMBER FUNCTION GET_LOB_INFORMATION(
  value_type   IN  VARCHAR2,
  column_name  IN  VARCHAR2,
  use_old      IN  VARCHAR2  DEFAULT 'Y')
RETURN NUMBER;
```

Parameters

Table 188–16 GET_LOB_INFORMATION Function Parameters

Parameter	Description
value_type	The type of value to return for the column, either old or new
column_name	The name of the column

Table 188–16 (Cont.) GET_LOB_INFORMATION Function Parameters

Parameter	Description
use_old	If Y and value_type is new, and no new value exists, then the function returns the corresponding old value. If N and value_type is new, then the function does not return the old value if no new value exists. If value_type is old or if the command_type of the row LCR is not UPDATE, then the function ignores the value of the use_old parameter. NULL is not a valid specification for the use_old parameter.

GET_LOB_OFFSET Member Function

Gets the LOB offset for the specified column in the number of characters for CLOB columns and the number of bytes for BLOB columns. Returns a non-NULL value only if all of the following conditions are met:

- The value exists for the column
- The column value is an out-of-line LOB. That is, the information is DBMS_LCR.LAST_LOB_CHUNK or DBMS_LCR.LOB_CHUNK
- The command type is LOB ERASE or LOB WRITE

Otherwise, returns NULL.

Syntax

```
GET_LOB_OFFSET(
   value_type   IN  VARCHAR2,
   column_name  IN  VARCHAR2)
RETURN NUMBER;
```

Parameters

Table 188–17 GET_LOB_OFFSET Procedure Parameters

Parameter	Description
value_type	The type of value to return for the column. Currently, only new can be specified.
column_name	The name of the LOB column

GET_LOB_OPERATION_SIZE Member Function

Gets the operation size for the LOB column in the number of characters for CLOB columns and the number of bytes for BLOB columns. Returns a non-NULL value only if all of the following conditions are met:

- The value exists for the column
- The column value is an out-of-line LOB
- The command type is LOB ERASE or LOB TRIM
- The information is DBMS_LCR.LAST_LOB_CHUNK

Otherwise, returns NULL.

Syntax

```
MEMBER FUNCTION GET_LOB_OPERATION_SIZE(
  value_type   IN  VARCHAR2,
  column_name  IN  VARCHAR2)
RETURN NUMBER,
```

Parameters

Table 188–18 GET_LOB_OPERATION_SIZE Function Parameters

Parameter	Description
value_type	The type of value to return for the column. Currently, only new can be specified.
column_name	The name of the LOB column

GET_LONG_INFORMATION Member Function

Gets the LONG information for the column.

The return value can be one of the following:

```
DBMS_LCR.NOT_A_LONG        CONSTANT NUMBER := 1;
DBMS_LCR.NULL_LONG         CONSTANT NUMBER := 2;
DBMS_LCR.INLINE_LONG       CONSTANT NUMBER := 3;
DBMS_LCR.LONG_CHUNK        CONSTANT NUMBER := 4;
DBMS_LCR.LAST_LONG_CHUNK   CONSTANT NUMBER := 5;
```

Returns NULL if the specified column does not exist.

If the command type of the row LCR is UPDATE, then specifying 'Y' for the use_old parameter is a convenient way to get the value of the columns.

Syntax

```
MEMBER FUNCTION GET_LONG_INFORMATION(
  value_type   IN  VARCHAR2,
  column_name  IN  VARCHAR2,
  use_old      IN  VARCHAR2  DEFAULT 'Y')
RETURN NUMBER;
```

Parameters

Table 188–19 GET_LONG_INFORMATION Function Parameters

Parameter	Description
value_type	The type of value to return for the column, either old or new
column_name	The name of the column
use_old	If Y and value_type is new, and no new value exists, then the function returns the corresponding old value. If N and value_type is new, then the function does not return the old value if no new value exists. If value_type is old or if the command_type of the row LCR is not UPDATE, then the function ignores the value of the use_old parameter. NULL is not a valid specification for the use_old parameter.

GET_VALUE Member Function

Returns the old or new value for the specified column, depending on the value type specified.

If the command type of the row LCR is UPDATE, then specifying 'Y' for the use_old parameter is a convenient way to get the value of a column.

Syntax

```
MEMBER FUNCTION GET_VALUE(
   value_type   IN  VARCHAR2,
   column_name  IN  VARCHAR2,
   use_old      IN  VARCHAR2  DEFAULT 'Y')
RETURN ANYDATA;
```

Parameters

Table 188–20 GET_VALUE Function Parameters

Parameter	Description
value_type	The type of value to return for the column. Specify old to get the old value for the column. Specify new to get the new value for the column.
column_name	The column name. If the column is present and has a NULL value, then the function returns a ANYDATA instance containing a NULL value. If the column value is absent, then the function returns a NULL.
use_old	If Y and value_type is new, and no new value exists, then the function returns the corresponding old value. If N and value_type is new, then the function returns NULL if no new value exists. If value_type is old or if the command_type of the row LCR is not UPDATE, then the function ignores the value of the use_old parameter. NULL is not a valid specification for the use_old parameter.

GET_VALUES Member Function

Returns a list of old or new values, depending on the value type specified.

If the command type of the row LCR is UPDATE, then specifying 'Y' for the use_old parameter is a convenient way to get the values of all columns.

Syntax

```
MEMBER FUNCTION GET_VALUES(
   value_type  IN  VARCHAR2,
   use_old     IN  VARCHAR2  DEFAULT 'Y')
RETURN SYS.LCR$_ROW_LIST;
```

Parameter

Table 188–21 GET_VALUES Function Parameter

Parameter	Description
value_type	The type of values to return. Specify old to return a list of old values. Specify new to return a list of new values.

Table 188–21 (Cont.) GET_VALUES Function Parameter

Parameter	Description
use_old	If Y and value_type is new, then the function returns a list of all new values in the LCR. If a new value does not exist in the list, then the function returns the corresponding old value. Therefore, the returned list contains all existing new values and old values for the new values that do not exist. If N and value_type is new, then the function returns a list of all new values in the LCR without returning any old values. If value_type is old or if the command_type of the row LCR is not UPDATE, then the function ignores the value of the use_old parameter. NULL is not a valid specification for the use_old parameter.

RENAME_COLUMN Member Procedure

Renames a column in an LCR.

Syntax

```
MEMBER PROCEDURE RENAME_COLUMN(
   from_column_name  IN  VARCHAR2,
   to_column_name    IN  VARCHAR2,
   value_type        IN  VARCHAR2  DEFAULT '*');
```

Parameters

Table 188–22 RENAME_COLUMN Procedure Parameters

Parameter	Description
from_column_name	The existing column name
to_column_name	The new column name. An error is raised if a column with the specified name already exists.
value_type	The type of value for which to rename the column. Specify old to rename the old value of the column. An error is raised if the old value does not exist in the LCR. Specify new to rename the new value of the column. An error is raised if the new value does not exist in the LCR. If * is specified, then the procedure renames the column names for both old and new value. The procedure raises an error if either column value does not exist in the LCR.

SET_LOB_INFORMATION Member Procedure

Sets LOB information for the column.

Note: When you are processing a row LCR with a rule-based transformation, DML handler, or error handler, you cannot use this member procedure.

Syntax

```
MEMBER PROCEDURE SET_LOB_INFORMATION(
  value_type       IN  VARCHAR2,
  column_name      IN  VARCHAR2,
```

```
  lob_information  IN  NUMBER);
```

Parameters

Table 188–23 SET_LOB_INFORMATION Procedure Parameters

Parameter	Description
value_type	The type of value to set for the column, either old or new. Specify old only if lob_information is set to DBMS_LCR.NOT_A_LOB.
column_name	The name of the column. An exception is raised if the column value does not exist. You might need to set this parameter for non-LOB columns.
lob_information	Specify one of the following values: DBMS_LCR.NOT_A_LOB CONSTANT NUMBER := 1; DBMS_LCR.NULL_LOB CONSTANT NUMBER := 2; DBMS_LCR.INLINE_LOB CONSTANT NUMBER := 3; DBMS_LCR.EMPTY_LOB CONSTANT NUMBER := 4; DBMS_LCR.LOB_CHUNK CONSTANT NUMBER := 5; DBMS_LCR.LAST_LOB_CHUNK CONSTANT NUMBER := 6;

SET_LOB_OFFSET Member Procedure

Sets the LOB offset for the specified column in the number of characters for CLOB columns and the number of bytes for BLOB columns.

> **Note:** When you are processing a row LCR with a rule-based transformation, DML handler, or error handler, you cannot use this member procedure.

Syntax

```
MEMBER PROCEDURE SET_LOB_OFFSET(
   value_type   IN  VARCHAR2,
   column_name  IN  VARCHAR2,
   lob_offset   IN  NUMBER);
```

Parameters

Table 188–24 SET_LOB_OFFSET Procedure Parameters

Parameter	Description
value_type	The type of value to set for the column. Currently, only new can be specified.
column_name	The column name. An error is raised if the column value does not exist in the LCR.
lob_offset	The LOB offset number. Valid values are NULL or a positive integer less than or equal to DBMS_LOB.LOBMAXSIZE.

SET_LOB_OPERATION_SIZE Member Procedure

Sets the operation size for the LOB column in the number of characters for CLOB columns and bytes for BLOB columns.

Note: When you are processing a row LCR with a rule-based transformation, DML handler, or error handler, you cannot use this member procedure.

Syntax

```
MEMBER PROCEDURE SET_LOB_OPERATION_SIZE(
  value_type          IN  VARCHAR2,
  column_name         IN  VARCHAR2,
  lob_operation_size  IN  NUMBER);
```

Parameters

Table 188–25 SET_LOB_OPERATION_SIZE Procedure Parameters

Parameter	Description
value_type	The type of value to set for the column. Currently, only new can be specified.
column_name	The name of the LOB column. An exception is raised if the column value does not exist in the LCR.
lob_operation_size	If lob_information for the LOB is or will be DBMS_LCR.LAST_LOB_CHUNK, then this parameter can be set to either a valid LOB ERASE value or a valid LOB TRIM value. A LOB ERASE value must be a positive integer less than or equal to DBMS_LOB.LOBMAXSIZE. A LOB TRIM value must be a nonnegative integer less than or equal to DBMS_LOB.LOBMAXSIZE. Otherwise, set to NULL.

SET_VALUE Member Procedure

Overwrites the old or new value of the specified column.

One reason to overwrite an old value for a column is to resolve an error that resulted from a conflict.

Note: To add a column to a row LCR, run ADD_COLUMN.

See Also: "ADD_COLUMN Member Procedure" on page 188-14

Considerations for LOB Columns

When processing a row LCR with LOB columns with a DML handler or error handler and the handler is using LOB assembly (the assemble_lobs parameter is set to TRUE for the handler), you use this member procedure in the handler procedure on a LOB column in a row LCR. If assemble_lobs is set to FALSE for the handler, then you cannot use this member procedure on a LOB column.

To use a DML or error handler to set the value of a LOB column, specify the LOB locator for the column_value parameter in the member procedure. The SET_VALUE member procedure verifies that a ANYDATA encapsulated LOB locator is processed with a DML or error handler that is using LOB assembly. An error is raised under the following conditions:

- The handler attempts to enqueue a row LCR with a ANYDATA encapsulated LOB locator.

- An attempt is made to set a LOB column incorrectly.

If an error is raised because of one of these conditions, then the transaction that includes the row LCR is moved to the error queue, and the LOB is represented by the original (nonassembled) row LCRs.

Note:

- Database compatibility must be `10.2.0` or higher to use LOB assembly.
- When you are processing a row LCR with a rule-based transformation, you cannot use this member procedure on a LOB column.
- When you are processing a row LCR with a rule-based transformation, DML handler, or error handler, you cannot use this member procedure on a `LONG` or `LONG RAW` column.

Syntax

```
MEMBER PROCEDURE SET_VALUE(
   value_type    IN  VARCHAR2,
   column_name   IN  VARCHAR2,
   column_value  IN  ANYDATA);
```

Parameters

Table 188–26 SET_VALUE Procedure Parameters

Parameter	Description
`value_type`	The type of value to set. Specify `old` to set the old value of the column. Specify `new` to set the new value of the column.
`column_name`	The column name. An error is raised if the specified `column_value` does not exist in the LCR for the specified `column_type`.
`column_value`	The new value of the column. If `NULL` is specified, then this procedure raises an error. To set the value to `NULL`, encapsulate the `NULL` in a `ANYDATA` instance. If the member procedure is used in a DML handler or error handler that uses LOB assembly, then specify a LOB locator.

SET_VALUES Member Procedure

Replaces all old values or all new values for the LCR, depending on the value type specified.

Considerations for LOB Columns

When processing a row LCR with LOB columns with a DML handler or error handler and the handler is using LOB assembly (the `assemble_lobs` parameter is set to `TRUE` for the handler), you use this member procedure in the handler procedure on a row LCR that contains one or more LOB columns. If `assemble_lobs` is set to `FALSE` for the handler, then you cannot use this member procedure on a row LCR.

To use a DML or error handler to set the value of one or more LOB columns in a row LCR, specify a LOB locator for each LOB column in the `value_list` parameter. The `SET_VALUES` member procedure verifies that a `ANYDATA` encapsulated LOB locator is

processed with a DML or error handler that is using LOB assembly. An error is raised under the following conditions:

- The handler attempts to enqueue a row LCR with a ANYDATA encapsulated LOB locator.
- An attempt is made to set a LOB column incorrectly.

If an error is raised because of one of these conditions, then the transaction that includes the row LCR is moved to the error queue, and the LOB columns are represented by the original (nonassembled) row LCRs.

Note:

- Database compatibility must be 10.2.0 or higher to use LOB assembly.
- When you are processing a row LCR with a rule-based transformation, you cannot use this member procedure on LOB columns.
- When you are processing a row LCR with a rule-based transformation, DML handler, or error handler, you cannot use this member procedure on LONG or LONG RAW columns.

Syntax

```
MEMBER PROCEDURE SET_VALUES(
   value_type  IN  VARCHAR2,
   value_list  IN  SYS.LCR$_ROW_LIST);
```

Parameters

Table 188–27 SET_VALUES Procedure Parameters

Parameter	Description
value_type	The type of values to replace. Specify old to replace the old values. Specify new to replace the new values.
value_list	List of values to replace the existing list. Use a NULL or an empty list to remove all values. If the member procedure is used in a DML handler or error handler that uses LOB assembly, then specify one or more LOB locators.

Common Subprograms for LCR$_DDL_RECORD and LCR$_ROW_RECORD

The following functions and procedures are common to both the `LCR$_DDL_RECORD` and `LCR$_ROW_RECORD` type.

> **See Also:** For descriptions of the subprograms for these types that are exclusive to each type:
>
> - "Summary of LCR$_DDL_RECORD Subprograms" on page 188-5
> - "Summary of LCR$_ROW_RECORD Subprograms" on page 188-13

Table 188–28 Summary of Common Subprograms for DDL and Row LCR Types

Subprogram	Description
"GET_COMMAND_TYPE Member Function" on page 188-27	Returns the command type of the logical change record (LCR)
"GET_COMMIT_SCN Member Function" on page 188-27	Returns the commit system change number (SCN) of the transaction to which the current LCR belongs
"GET_COMPATIBLE Member Function" on page 188-27	Returns the minimal database compatibility required to support the LCR
"GET_EXTRA_ATTRIBUTE Member Function" on page 188-28	Returns the value for the specified extra attribute in the LCR
"GET_OBJECT_NAME Member Function" on page 188-29	Returns the name of the object that is changed by the LCR
"GET_OBJECT_OWNER Member Function" on page 188-29	Returns the owner of the object that is changed by the LCR
"GET_SCN Member Function" on page 188-29	Returns the system change number (SCN) of the LCR
"GET_SOURCE_DATABASE_NAME Member Function" on page 188-30	Returns the source database name.
"GET_SOURCE_TIME Member Function" on page 188-30	Returns the time when the change in an LCR captured by a capture process was generated in the redo log of the source database, or the time when a user-enqueued LCR was created.
"GET_TAG Member Function" on page 188-30	Returns the tag for the LCR
"GET_TRANSACTION_ID Member Function" on page 188-30	Returns the transaction identifier of the LCR
"IS_NULL_TAG Member Function" on page 188-30	Returns `Y` if the tag for the LCR is `NULL`, or returns `N` if the tag for the LCR is not `NULL`
"SET_COMMAND_TYPE Member Procedure" on page 188-31	Sets the command type in the LCR
"SET_EXTRA_ATTRIBUTE Member Procedure" on page 188-31	Sets the value for the specified extra attribute in the LCR
"SET_OBJECT_NAME Member Procedure" on page 188-32	Sets the name of the object that is changed by the LCR
"SET_OBJECT_OWNER Member Procedure" on page 188-33	Sets the owner of the object that is changed by the LCR

Table 188–28 (Cont.) Summary of Common Subprograms for DDL and Row LCR Types

Subprogram	Description
"SET_SOURCE_DATABASE_NAME Member Procedure" on page 188-33	Sets the source database name of the object that is changed by the LCR
"SET_TAG Member Procedure" on page 188-33	Sets the tag for the LCR

GET_COMMAND_TYPE Member Function

Returns the command type of the LCR.

> **See Also:** The "SQL Command Codes" table in the *Oracle Call Interface Programmer's Guide* for a complete list of command types

Syntax

```
MEMBER FUNCTION GET_COMMAND_TYPE()
RETURN VARCHAR2;
```

GET_COMMIT_SCN Member Function

Returns the commit system change number (SCN) of the transaction to which the current LCR belongs.

The commit SCN for a transaction is available only during apply or during error transaction execution. This function can be used only in a DML handler, DDL handler, or error handler. Such a handler can use the SCN obtained by this procedure to flashback to the transaction commit time for an LCR. In this case, the flashback must be performed at the source database for the LCR.

The commit SCN might not be available for an LCR that is part of an incomplete transaction. For example, user-enqueued LCRs might not have a commit SCN. If the commit SCN is not available for an LCR, then this function returns `NULL`.

Syntax

```
MEMBER FUNCTION GET_COMMIT_SCN()
RETURN NUMBER;
```

GET_COMPATIBLE Member Function

Returns the minimal database compatibility required to support the LCR. You control the compatibility of an Oracle database using the `COMPATIBLE` initialization parameter.

The return value for this function can be one of the following:

Return Value	COMPATIBLE Initialization Parameter Equivalent
`DBMS_STREAMS.COMPATIBLE_9_2`	`9.2.0`
`DBMS_STREAMS.COMPATIBLE_10_1`	`10.1.0`
`DBMS_STREAMS.COMPATIBLE_10_2`	`10.2.0`

DDL LCRs always return `DBMS_STREAMS.COMPATIBLE_9_2`.

You can use the following functions for constant compatibility return values:

- The DBMS_STREAMS.COMPATIBLE_9_2 function returns the DBMS_STREAMS.COMPATIBLE_9_2 constant.
- The DBMS_STREAMS.COMPATIBLE_10_1 function returns DBMS_STREAMS.COMPATIBLE_10_1 constant.
- The DBMS_STREAMS.COMPATIBLE_10_2 function returns DBMS_STREAMS.COMPATIBLE_10_2 constant.

You can use these functions with the GET_COMPATIBLE member function for an LCR in rule conditions and apply handlers.

Note: You can determine which database objects in a database are not supported by Streams by querying the DBA_STREAMS_UNSUPPORTED data dictionary view.

See Also:

- *Oracle Streams Concepts and Administration* for examples of rules that discard changes that are not supported by Streams
- Chapter 105, "DBMS_STREAMS" and Chapter 106, "DBMS_STREAMS_ADM"
- *Oracle Database Reference* and *Oracle Database Upgrade Guide* for more information about the COMPATIBLE initialization parameter

Syntax

```
MEMBER FUNCTION GET_COMPATIBLE()
RETURN NUMBER;
```

GET_EXTRA_ATTRIBUTE Member Function

Returns the value for the specified extra attribute in the LCR. The returned extra attribute is contained within a ANYDATA instance. You can use the INCLUDE_EXTRA_ATTRIBUTE procedure in the DBMS_CAPTURE_ADM package to instruct a capture process to capture one or more extra attributes.

See Also: "INCLUDE_EXTRA_ATTRIBUTE Procedure" on page 20-19

Syntax

```
MEMBER FUNCTION GET_EXTRA_ATTRIBUTE(
   attribute_name  IN  VARCHAR2)
RETURN ANYDATA;
```

Parameters

Table 188–29 GET_EXTRA_ATTRIBUTE Function Parameter

Parameter	Description
attribute_name	The name of the extra attribute to return. Valid names are: ■ row_id The rowid of the row changed in a row LCR. This attribute is not included in DDL LCRs, nor in row LCRs for index-organized tables. The type is UROWID. ■ serial# The serial number of the session that performed the change captured in the LCR. The type is NUMBER. ■ session# The identifier of the session that performed the change captured in the LCR. The type is NUMBER. ■ thread# The thread number of the instance in which the change captured in the LCR was performed. Typically, the thread number is relevant only in a Real Application Clusters environment. The type is NUMBER. ■ tx_name The name of the transaction that includes the LCR. The type is VARCHAR2. ■ username The name of the current user who performed the change captured in the LCR. The type is VARCHAR2. An error is raised if the specified attribute_name is not valid. If no value exists for the specified extra attribute, then the function returns a NULL. **See Also:** *Oracle Database PL/SQL User's Guide and Reference* for more information about the current user

GET_OBJECT_NAME Member Function

Returns the name of the object that is changed by the LCR.

Syntax

```
MEMBER FUNCTION GET_OBJECT_NAME()
RETURN VARCHAR2;
```

GET_OBJECT_OWNER Member Function

Returns the owner of the object that is changed by the LCR.

Syntax

```
MEMBER FUNCTION GET_OBJECT_OWNER()
RETURN VARCHAR2;
```

GET_SCN Member Function

Returns the system change number (SCN) of the LCR.

Syntax

```
MEMBER FUNCTION GET_SCN()
RETURN NUMBER;
```

GET_SOURCE_DATABASE_NAME Member Function

Returns the global name of the source database name. The source database is the database where the change occurred.

Syntax

```
MEMBER FUNCTION GET_SOURCE_DATABASE_NAME()
RETURN VARCHAR2;
```

GET_SOURCE_TIME Member Function

Returns the time when the change in an LCR captured by a capture process was generated in the redo log of the source database, or the time when a user-enqueued LCR was created.

Syntax

```
MEMBER FUNCTION GET_SOURCE_TIME()
RETURN DATE;
```

GET_TAG Member Function

Returns the tag for the LCR. An LCR tag is a binary tag that enables tracking of the LCR. For example, this tag can be used to determine the original source database of the DML or DDL change when apply forwarding is used.

> **See Also:** *Oracle Streams Replication Administrator's Guide* for more information about tags

Syntax

```
MEMBER FUNCTION GET_TAG()
RETURN RAW;
```

GET_TRANSACTION_ID Member Function

Returns the transaction identifier of the LCR.

Syntax

```
MEMBER FUNCTION GET_TRANSACTION_ID()
RETURN VARCHAR2;
```

IS_NULL_TAG Member Function

Returns `Y` if the tag for the LCR is `NULL`, or returns `N` if the tag for the LCR is not `NULL`.

> **See Also:** *Oracle Streams Replication Administrator's Guide* for more information about tags

Syntax

```
MEMBER FUNCTION IS_NULL_TAG()
RETURN VARCHAR2;
```

SET_COMMAND_TYPE Member Procedure

Sets the command type in the LCR. If the command type specified cannot be interpreted, then this procedure raises an error. For example, changing `INSERT` to `GRANT` would raise an error.

See Also:

- The description of the `command_type` parameter in "LCR$_DDL_RECORD Constructor Function Parameters" on page 188-4
- The description of the command_type parameter in "LCR$_ROW_RECORD Type" on page 188-11
- The "SQL Command Codes" table in the *Oracle Call Interface Programmer's Guide* for a complete list of command types

Syntax

```
MEMBER PROCEDURE SET_COMMAND_TYPE(
   command_type  IN  VARCHAR2);
```

Parameter

Table 188–30 SET_COMMAND_TYPE Procedure Parameter

Parameter	Description
`command_type`	The command type. This parameter should be set to a non-`NULL` value.

SET_EXTRA_ATTRIBUTE Member Procedure

Sets the value for the specified extra attribute in the LCR. You can use the `INCLUDE_EXTRA_ATTRIBUTE` procedure in the `DBMS_CAPTURE_ADM` package to instruct a capture process to capture one or more extra attributes.

See Also: "INCLUDE_EXTRA_ATTRIBUTE Procedure" on page 20-19

Syntax

```
MEMBER PROCEDURE SET_EXTRA_ATTRIBUTE(
   attribute_name   IN  VARCHAR2,
   attribute_value  IN  ANYDATA);
```

Parameters

Table 188–31 SET_EXTRA_ATTRIBUTE Procedure Parameter

Parameter	Description
attribute_name	The name of the extra attribute to set. Valid names are: ■ row_id The rowid of the row changed in a row LCR. This attribute is not included in DDL LCRs, nor in row LCRs for index-organized tables. The type is VARCHAR2. ■ serial# The serial number of the session that performed the change captured in the LCR. The type is NUMBER. ■ session# The identifier of the session that performed the change captured in the LCR. The type is NUMBER. ■ thread# The thread number of the instance in which the change captured in the LCR was performed. Typically, the thread number is relevant only in a Real Application Clusters environment. The type is NUMBER. ■ tx_name The name of the transaction that includes the LCR. The type is VARCHAR2. ■ username The name of the current user who performed the change captured in the LCR. The type is VARCHAR2. An error is raised if the specified attribute_name is not valid. **See Also:** *Oracle Database PL/SQL User's Guide and Reference* for more information about the current user
attribute_value	The value to which the specified extra attribute is set. If set to NULL, then this procedure removes the specified extra attribute from the LCR. To set to NULL, encapsulate the NULL in a ANYDATA instance.

SET_OBJECT_NAME Member Procedure

Sets the name of the object that is changed by the LCR.

Syntax

```
MEMBER PROCEDURE SET_OBJECT_NAME(
   object_name  IN  VARCHAR2);
```

Parameter

Table 188–32 SET_OBJECT_NAME Procedure Parameter

Parameter	Description
object_name	The name of the object

SET_OBJECT_OWNER Member Procedure

Sets the owner of the object that is changed by the LCR.

Syntax

```
MEMBER PROCEDURE SET_OBJECT_OWNER(
   object_owner  IN  VARCHAR2);
```

Parameter

Table 188–33 SET_OBJECT_OWNER Procedure Parameter

Parameter	Description
object_owner	The schema that contains the object

SET_SOURCE_DATABASE_NAME Member Procedure

Sets the source database name of the object that is changed by the LCR.

Syntax

```
MEMBER PROCEDURE SET_SOURCE_DATABASE_NAME(
   source_database_name  IN  VARCHAR2);
```

Parameter

Table 188–34 SET_SOURCE_DATABASE_NAME Procedure Parameter

Parameter	Description
source_database_name	The source database of the change. If you do not include the domain name, then the procedure appends the local domain to the database name automatically. For example, if you specify DBS1 and the local domain is .NET, then the procedure specifies DBS1.NET automatically. This parameter should be set to a non-NULL value.

SET_TAG Member Procedure

Sets the tag for the LCR. An LCR tag is a binary tag that enables tracking of the LCR. For example, this tag can be used to determine the original source database of the change when apply forwarding is used.

> **See Also:** *Oracle Streams Replication Administrator's Guide* for more information about tags

Syntax

```
MEMBER PROCEDURE SET_TAG(
   tag  IN  RAW);
```

Parameter

Table 188–35 SET_TAG Procedure Parameter

Parameter	Description
tag	The binary tag for the LCR. The size limit for a tag value is two kilobytes.

LCR$_ROW_LIST Type

Identifies a list of column values for a row in a table.

This type uses the `LCR$_ROW_UNIT` type and is used in the `LCR$_ROW_RECORD` type.

See Also:

- "LCR$_ROW_UNIT Type" on page 188-35
- "LCR$_ROW_RECORD Type" on page 188-11

Syntax

```
CREATE TYPE SYS.LCR$_ROW_LIST AS TABLE OF SYS.LCR$_ROW_UNIT
/
```

LCR$_ROW_UNIT Type

Identifies the value for a column in a row.

This type is used in the LCR$_ROW_LIST type.

See Also: "LCR$_ROW_LIST Type" on page 188-34

Syntax

```
CREATE TYPE LCR$_ROW_UNIT AS OBJECT (
  column_name         VARCHAR2(4000),
  data                ANYDATA,
  lob_information     NUMBER,
  lob_offset          NUMBER,
  lob_operation_size  NUMBER
  long_information    NUMBER);
/
```

Attributes

Table 188–36 LCR$_ROW_UNIT Attributes

Attribute	Description
column_name	The name of the column
data	The data contained in the column
lob_information	Contains the LOB information for the column and contains one of the following values: DBMS_LCR.NOT_A_LOB CONSTANT NUMBER := 1; DBMS_LCR.NULL_LOB CONSTANT NUMBER := 2; DBMS_LCR.INLINE_LOB CONSTANT NUMBER := 3; DBMS_LCR.EMPTY_LOB CONSTANT NUMBER := 4; DBMS_LCR.LOB_CHUNK CONSTANT NUMBER := 5; DBMS_LCR.LAST_LOB_CHUNK CONSTANT NUMBER := 6;
lob_offset	The LOB offset specified in the number of characters for CLOB columns and the number of bytes for BLOB columns. Valid values are NULL or a positive integer less than or equal to DBMS_LOB.LOBMAXSIZE.
lob_operation_size	If lob_information for the LOB is DBMS_LCR.LAST_LOB_CHUNK, then this parameter can be set to either a valid LOB ERASE value or a valid LOB TRIM value. A LOB ERASE value must be a positive integer less than or equal to DBMS_LOB.LOBMAXSIZE. A LOB TRIM value must be a nonnegative integer less than or equal to DBMS_LOB.LOBMAXSIZE. If lob_information is not DBMS_LCR.LAST_LOB_CHUNK and for all other operations, is NULL.
long_information	Contains the LONG information for the column and contains one of the following values: DBMS_LCR.not_a_long CONSTANT NUMBER := 1; DBMS_LCR.null_long CONSTANT NUMBER := 2; DBMS_LCR.inline_long CONSTANT NUMBER := 3; DBMS_LCR.long_chunk CONSTANT NUMBER := 4; DBMS_LCR.last_long_chunk CONSTANT NUMBER := 5;

189

*inter*Media ORDAudio TYPE

The *inter*Media ORDAudio object type supports the storage and management of audio data.

Audio data can have different formats, encoding types, compression types, numbers of channels, sampling rates, sample sizes, and playing times (duration) depending upon how the audio data is digitally recorded. Oracle *inter*Media ORDAudio can store and retrieve audio data of any data format. Oracle *inter*Media ORDAudio can automatically extract metadata from audio data of a variety of popular audio formats. Oracle *inter*Media ORDAudio can also extract application attributes and store them in the comments attribute of the object in XML form.

- Documentation of ORDAudio

Documentation of ORDAudio

For a complete description of this package within the context of Oracle *inter*Media, see `ORDAudio` in the *Oracle interMedia Reference*.

190

*inter*Media ORDDoc TYPE

The *inter*Media ORDDoc object type supports the storage and management of heterogeneous media data including image, audio, and video.

Heterogeneous media data can have different formats depending upon the application generating the media data. Oracle *inter*Media can store and retrieve media data of any data format. The *inter*Media ORDDoc data type can be used in applications that require you to store different types of heterogeneous media data in the same column so you can build a common metadata index on all the different types of media data. Using this index, you can search across all the different types of heterogeneous media data. Note that you cannot use this same search technique if the different types of heterogeneous media data are stored in different types of objects in different columns of relational tables.

- Documentation of ORDDoc

Documentation of ORDDoc

For a complete description of this package within the context of Oracle *inter*Media, see `ORDDoc` in the *Oracle interMedia Reference*.

191

*inter*Media ORDImage TYPE

The *inter*Media ORDImage object type supports the storage, management, and manipulation of image data.

Digitized images consist of the image data (digitized bits) and attributes that describe and characterize the image data.

The image data (pixels) can have varying depths (bits for each pixel) depending on how the image was captured, and can be organized in various ways. The organization of the image data is known as the data format. Oracle *inter*Media ORDImage can store and retrieve image data of any data format. Oracle *inter*Media ORDImage can process (cut, scale, and generate thumbnails) of images, convert the format of images, and automatically extract properties of images of a variety of popular data formats.

- Documentation of ORDImage

Documentation of ORDImage

For a complete description of this package within the context of Oracle *inter*Media, see `ORDImage` in the *Oracle interMedia Reference*.

192

*inter*Media ORDImageSignature TYPE

The *inter*Media `ORDImageSignature` object type supports content-based retrieval of images (image matching).

The `interMedia` `ORDImageSignature` object type supports the extraction of color, texture, and shape information from an image. This extracted information, referred to as the image signature, is stored in an `ORDImageSignature` object. You can then use object methods to find matching images based on their extracted signatures.

- Documentation of ORDImageSignature

Documentation of ORDImageSignature

For a complete description of this package within the context of Oracle *inter*Media, see `ORDImageSignature` in the *Oracle interMedia Reference*.

193

*inter*Media SQL/MM Still Image TYPE

Oracle *inter*Media provides support for the SQL/MM Still Image Standard, which supports the storage, retrieval, and modification of images in the database and the ability to locate images using visual predicates.

The following object relational types for images and image characteristics are included in this support: `SI_StillImage`, `SI_AverageColor`, `SI_Color`, `SI_ColorHistogram`, `SI_FeatureList`, `SI_PositionalColor`, and `SI_Texture`.

- Documentation of SQL/MM Still Image

Documentation of SQL/MM Still Image

For a complete description of this package within the context of Oracle *inter*Media, see SQL/MM Still Image in the *Oracle interMedia Reference.*

194

*inter*Media ORDVideo TYPE

The *inter*Media ORDVideo object type supports the storage and management of video data.

Digitized video consists of the video data (digitized bits) and the attributes that describe and characterize the video data. Video applications sometimes associate application-specific information, such as the description of the video training tape, date recorded, instructor's name, producer's name, and so forth, within the video data.

The video data can have different formats, compression types, frame rates, frame sizes, frame resolutions, playing times, compression types, number of colors, and bit rates depending upon how the video data was digitally recorded. Oracle *inter*Media ORDVideo can store and retrieve video data of any data format. Oracle *inter*Media ORDVideo can automatically extract metadata from video data of a variety of popular video formats. Oracle *inter*Media ORDVideo can also extract application attributes and store them in the comments attribute of the object in XML form identical to what is provided by the *inter*Media Annotator utility.

- Documentation of ORDVideo

Documentation of ORDVideo

For a complete description of this package within the context of Oracle *inter*Media, see `ORDVideo` in the *Oracle interMedia Reference*.

195

Rules Manager Types

Rules Manager is supplied with one predefined type and a public synonym for this type.

> **See Also:** *Oracle Database Application Developer's Guide - Rules Manager and Expression Filter* for more information.

This chapter contains the following topics:

Summary of Rule Manager Types

Table 195–1 describes the Rules Manager object type.

Table 195–1 Rules Manager Object Types

Object Type Name	Description
RLM$EVENTIDS Object Type	Specifies a list of event identifiers to the `CONSUME_PRIM_EVENTS` procedure

RLM$EVENTIDS Object Type

The RLM$EVENTIDS type is defined as a table of VARCHAR2 values as follows:

Syntax

```
CREATE OR REPLACE TYPE RLM$EVENTIDS is table of VARCHAR2(38);
```

Attributes

None.

Usage Notes

RLM$EVENTIDS type is used to pass a list of event identifiers to the CONSUME_PRIM_EVENTS procedure. These event identifiers are ROWIDs for the corresponding events in the database and their values are available through the arguments of the action callback procedure and rule class results view columns, when the rule class is configured for RULE consumption policy.

Examples

The following commands show the body of the action callback procedure for a rule class configured for RULE consumption policy. This demonstrates the use of RLM$EVENTDIDS type to consume the events before executing the action for the matched rules.

```
CREATE OR REPLACE PROCEDURE PromoAction (
      Flt        AddFlight,
      Flt_EvtId  ROWID,    --- rowid for the fligt primitive event
      Car        AddRentalCar,
      Car_EvtId  ROWID,
      rlm$rule   TravelPromotions%ROWTYPE) is
 evtcnsmd   NUMBER;
BEGIN
 evtcnsmd := dbms_rlmgr.consume_prim_events(
              rule_class   => 'TravelPromotions',
              event_idents => RLM$EVENTIDS(Flt_EvtId, Car_EvtId));

 IF (evtcnsmd = 1) THEN
   -- consume operation was successful; perform the action ---
   OfferPromotion (Flt.CustId, rlm$rule.PromoType, rlm$rule.OfferedBy);
 END IF;
END;
```

196

Rule TYPEs

This chapter describes the types used with rules, rule sets, and evaluation contexts.

See Also:

- Chapter 91, "DBMS_RULE"
- Chapter 92, "DBMS_RULE_ADM"

This chapter contains the following topic:

- Summary of Rule Types

Summary of Rule Types

Table 196–1 Rule Types

Type	Description
"RE$ATTRIBUTE_VALUE Type" on page 196-4	Specifies the value of a variable attribute
"RE$ATTRIBUTE_VALUE_LIST Type" on page 196-5	Identifies a list of attribute values
"RE$COLUMN_VALUE Type" on page 196-6	Specifies the value of a table column
"RE$COLUMN_VALUE_LIST Type" on page 196-7	Identifies a list of column values
"RE$NAME_ARRAY Type" on page 196-8	Identifies a list of names
"RE$NAME_ARRAY Type" on page 196-8	Identifies a list of name-value pairs
"RE$NV_LIST Type" on page 196-10	Identifies an object containing a list of name-value pairs and methods that operate on this list. This object type is used to represent the event context and the action context for a rule
"RE$NV_NODE Type" on page 196-12	Identifies a name-value pair
"RE$RULE_HIT Type" on page 196-13	Specifies a rule found as a result of evaluation
"RE$RULE_HIT_LIST Type" on page 196-14	Identifies a list of rules found as a result of evaluation
"RE$TABLE_ALIAS Type" on page 196-15	Provides the table corresponding to an alias used in a rule evaluation context
"RE$TABLE_ALIAS_LIST Type" on page 196-16	Identifies a list of table aliases used in a rule evaluation context
"RE$TABLE_VALUE Type" on page 196-17	Specifies the value of a table row using a `ROWID`
"RE$TABLE_VALUE_LIST Type" on page 196-18	Identifies a list of table values
"RE$VARIABLE_TYPE Type" on page 196-19	Provides the type of a variable used in a rule evaluation context
"RE$VARIABLE_TYPE_LIST Type" on page 196-21	Identifies a list of variables and their types used in a rule evaluation context
"RE$VARIABLE_VALUE Type" on page 196-22	Specifies the value of a variable
"RE$VARIABLE_VALUE_LIST Type" on page 196-23	Identifies a list of variable values

Rule types are used with the following Oracle-supplied PL/SQL packages:

- `DBMS_RULE`
- `DBMS_RULE_ADM`

You can use the DBMS_RULE_ADM package to create and administer rules, rule sets, and evaluation contexts, and you can use the DBMS_RULE package to evaluate rules.

When you use Streams, rules determine which changes are captured by a capture process, which messages are propagated by a propagation, which messages are applied by an apply process, and which messages are dequeued by a messaging client. The following Streams packages use rules:

- DBMS_APPLY_ADM
- DBMS_CAPTURE_ADM
- DBMS_PROPAGATION_ADM
- DBMS_STREAMS
- DBMS_STREAMS_ADM
- DBMS_STREAMS_AUTH

See Also: *Oracle Streams Concepts and Administration*

RE$ATTRIBUTE_VALUE Type

Specifies the value of a variable attribute.

> **Note:** Enclose the variable name and attribute name in double quotation marks (") if the name contains special characters.

Syntax

```
TYPE SYS.RE$ATTRIBUTE_VALUE (
   variable_name    VARCHAR2(32),
   attribute_name   VARCHAR2(4000),
   attribute_value  ANYDATA);
```

Attributes

Table 196–2 RE$ATTRIBUTE_VALUE Attributes

Attribute	Description
`variable_name`	Specifies the variable used in a rule
`attribute_name`	Specifies the attribute name. The attribute name can be a multi-component name, such as `a1.b2.c3`.
`attribute_value`	Specifies the attribute value

RE$ATTRIBUTE_VALUE_LIST Type

Identifies a list of attribute values.

Syntax

```
TYPE SYS.RE$ATTRIBUTE_VALUE_LIST AS VARRAY(1024) OF SYS.RE$ATTRIBUTE_VALUE;
```

RE$COLUMN_VALUE Type

Specifies the value of a table column.

> **Note:** Enclose the column name in double quotation marks (") if the name contains special characters.

Syntax

```
TYPE SYS.RE$COLUMN_VALUE (
   table_alias   VARCHAR2(32),
   column_name   VARCHAR2(4000),
   column_value  ANYDATA);
```

Attributes

Table 196–3 RE$COLUMN_VALUE Attributes

Attribute	Description
`table_alias`	Specifies the alias used for the table in a rule
`column_name`	Specifies the column name
`column_value`	Specifies the column value

RE$COLUMN_VALUE_LIST Type

Identifies a list of column values.

Syntax

```
TYPE SYS.RE$COLUMN_VALUE_LIST AS VARRAY(1024) OF SYS.RE$COLUMN_VALUE;
```

RE$NAME_ARRAY Type

Identifies a list of names.

Syntax

```
TYPE SYS.RE$NAME_ARRAY AS VARRAY(1024) OF VARCHAR2(30);
```

RE$NV_ARRAY Type

Identifies a list of name-value pairs.

Syntax

```
TYPE SYS.RE$NV_ARRAY AS VARRAY(1024) OF SYS.RE$NV_NODE;
```

RE$NV_LIST Type

Identifies an object containing a list of name-value pairs and methods that operate on this list. This object type is used to represent the event context for rule set evaluation and the action context for a rule.

Syntax

```
TYPE SYS.RE$NV_LIST AS OBJECT(
   actx_list  SYS.RE$NV_ARRAY);
```

Attributes

Table 196–4 RE$NV_LIST Attributes

Attribute	Description
actx_list	The list of name-value pairs

RE$NV_LIST Subprograms

This section describes the following member procedures and member functions of the SYS.RE$NV_LIST type:

- ADD_PAIR Member Procedure
- GET_ALL_NAMES Member Function
- GET_VALUE Member Function
- REMOVE_PAIR Member Procedure

ADD_PAIR Member Procedure

Adds a name-value pair to the list of name-value pairs.

> **Note:** Enclose the name in double quotation marks (") if the name contains special characters.

Syntax

```
MEMBER PROCEDURE ADD_PAIR(
   name   IN  VARCHAR2,
   value  IN  ANYDATA);
```

Parameters

Table 196–5 ADD_PAIR Procedure Parameters

Parameter	Description
name	The name in the name-value pair being added to the list. If the name already exists in the list, then this procedure raises an error.
value	The value in the name-value pair being added to the list

GET_ALL_NAMES Member Function

Returns a list of all the names in the name-value pair list.

Syntax

```
MEMBER FUNCTION GET_ALL_NAMES()
RETURN SYS.RE$NAME_ARRAY;
```

GET_VALUE Member Function

Returns the value for the specified name in a name-value pair list.

> **Note:** Enclose the name in double quotation marks (") if the name contains special characters.

Syntax

```
MEMBER FUNCTION GET_VALUE(
   name  IN  VARCHAR2)
RETURN ANYDATA;
```

Parameters

Table 196–6 GET_VALUE Procedure Parameters

Parameter	Description
name	The name whose value to return

REMOVE_PAIR Member Procedure

Removes the name-value pair with the specified name from the name-value pair list.

> **Note:** Enclose the name in double quotation marks (") if the name contains special characters.

Syntax

```
MEMBER PROCEDURE REMOVE_PAIR(
   name  IN  VARCHAR2);
```

Parameters

Table 196–7 REMOVE_PAIR Procedure Parameters

Parameter	Description
name	The name of the pair to remove

RE$NV_NODE Type

Identifies a name-value pair.

> **Note:** Enclose the name in double quotation marks (") if the name contains special characters.

Syntax

```
TYPE SYS.RE$NV_NODE (
   nvn_name   VARCHAR2(30),
   nvn_value  ANYDATA);
```

Attributes

Table 196–8 RE$NV_NODE Attributes

Attribute	Description
nvn_name	Specifies the name in the name-value pair
nvn_value	Specifies the value in the name-value pair

RE$RULE_HIT Type

Specifies a rule found as a result of an evaluation.

See Also:

- "CREATE_RULE Procedure" on page 92-14
- "ALTER_RULE Procedure" on page 92-10

Syntax

```
TYPE SYS.RE$RULE_HIT (
   rule_name            VARCHAR2(65),
   rule_action_context  RE$NV_LIST);
```

Attributes

Table 196–9 RE$RULE_HIT Attributes

Attribute	Description
`rule_name`	The rule name in the form *`schema_name.rule_name`*. For example, a rule named `employee_rule` in the `hr` schema is returned in the form `"hr"."employee_rule"`.
`rule_action_context`	The rule action context as specified in the `CREATE_RULE` or `ALTER_RULE` procedure of the `DBMS_RULE_ADM` package

RE$RULE_HIT_LIST Type

Identifies a list of rules found as a result of an evaluation.

Syntax

```
TYPE SYS.RE$RULE_HIT_LIST AS VARRAY(1024) OF SYS.RE$RULE_HIT;
```

RE$TABLE_ALIAS Type

Provides the table corresponding to an alias used in a rule evaluation context. A specified table name must satisfy the schema object naming rules.

Note: Enclose the table name in double quotation marks (") if the name contains special characters.

See Also: *Oracle Database SQL Reference* for information about schema object naming rules

Syntax

```
TYPE SYS.RE$TABLE_ALIAS IS OBJECT(
   table_alias  VARCHAR2(32),
   table_name   VARCHAR2(194));
```

Attributes

Table 196–10 RE$TABLE_ALIAS Attributes

Attribute	Description
`table_alias`	The alias used for the table in a rule
`table_name`	The table name referred to by the alias. A synonym can be specified. The table name is resolved in the evaluation context schema. The format is one of the following: *`schema_name.table_name`* *`table_name`* For example, if the *`schema_name`* is `hr` and the *`table_name`* is `employees`, then enter the following: `hr.employees`

RE$TABLE_ALIAS_LIST Type

Identifies a list of table aliases used in a rule evaluation context.

Syntax

```
TYPE SYS.RE$TABLE_ALIAS_LIST AS VARRAY(1024) OF SYS.RE$TABLE_ALIAS;
```

RE$TABLE_VALUE Type

Specifies the value of a table row using a `ROWID`.

Syntax

```
TYPE SYS.RE$TABLE_VALUE(
   table_alias  VARCHAR2(32),
   table_rowid  VARCHAR2(18));
```

Attributes

Table 196–11 RE$TABLE_VALUE Attributes

Attribute	Description
`table_alias`	Specifies the alias used for the table in a rule
`table_rowid`	Specifies the rowid for the table row

RE$TABLE_VALUE_LIST Type

Identifies a list of table values.

Note: Each table alias in the list in the list must be unique.

Syntax

```
TYPE SYS.RE$TABLE_VALUE_LIST AS VARRAY(1024) OF SYS.RE$TABLE_VALUE;
```

RE$VARIABLE_TYPE Type

Provides the type of a variable used in a rule evaluation context. A specified variable name must satisfy the schema object naming rules.

> **Note:** Enclose the variable name in double quotation marks (") if the name contains special characters.

> **See Also:** *Oracle Database SQL Reference* for information about schema object naming rules

Syntax

```
TYPE SYS.RE$VARIABLE_TYPE (
   variable_name             VARCHAR2(32),
   variable_type             VARCHAR2(4000),
   variable_value_function   VARCHAR2(228),
   variable_method_function  VARCHAR2(228));
```

Attributes

Table 196–12 RE$VARIABLE_TYPE Attributes

Attribute	Description
`variable_name`	The variable name used in a rule
`variable_type`	The type that is resolved in the evaluation context schema. Any valid Oracle built-in datatype, user-defined type, or Oracle-supplied type can be specified. See the *Oracle Database SQL Reference* for more information about these types.
`variable_value_function`	A value function that can be specified for implicit variables. A synonym can be specified. The function name is resolved in the evaluation context schema. It is executed on behalf of the owner of a rule set using the evaluation context or containing a rule that uses the evaluation context. See the "Usage Notes" for more information.
`variable_method_function`	Specifies a value function, which can return the result of a method invocation. Specifying such a function can speed up evaluation, if there are many simple rules that invoke the method on the variable. The function can be a synonym or a remote function. The function name is resolved in the evaluation context schema. It is executed on behalf of the owner of a rule set using the evaluation context or containing a rule that uses the evaluation context. See the "Usage Notes" for more information.

Usage Notes

The functions for both the for the `variable_value_function` parameter and `variable_method_function` parameter have the following format:

`schema_name.package_name.function_name@dblink`

Any of the following parts of the format can be omitted: *schema_name*, *package_name*, and *@dblink*.

For example, if the *schema_name* is hr, the *package_name* is var_pac, the *function_name* is func_value, and the *dblink* is dbs1.net, then enter the following:

```
hr.var_pac.func_value@dbs1.net
```

The following sections describe the signature of the functions.

Signature for variable_value_function

The function must have the following signature:

```
FUNCTION variable_value_function_name(
  evaluation_context_schema  IN VARCHAR2,
  evaluation_context_name    IN VARCHAR2,
  variable_name              IN VARCHAR2,
  event_context              IN SYS.RE$NV_LIST )
RETURN SYS.RE$VARIABLE_VALUE;
```

Signature for variable_method_function

This function must have the following signature:

```
FUNCTION variable_method_function_name(
  evaluation_context_schema  IN VARCHAR2,
  evaluation_context_name    IN VARCHAR2,
  variable_value             IN SYS.RE$VARIABLE_VALUE,
  method_name                IN VARCHAR2,
  event_context              IN SYS.RE$NV_LIST)
RETURN SYS.RE$ATTRIBUTE_VALUE;
```

RE$VARIABLE_TYPE_LIST Type

Identifies a list of variables and their types used in a rule evaluation context.

Syntax

```
TYPE SYS.RE$VARIABLE_TYPE_LIST AS VARRAY(1024) OF SYS.RE$VARIABLE_TYPE;
```

RE$VARIABLE_VALUE Type

Specifies the value of a variable.

> **Note:** Enclose the variable name in double quotation marks (") if the name contains special characters.

Syntax

```
TYPE SYS.RE$VARIABLE_VALUE (
   variable_name  VARCHAR2(32),
   variable_data  ANYDATA);
```

Attributes

Table 196–13 RE$VARIABLE_VALUE Attributes

Attribute	Description
`variable_name`	Specifies the variable name used in a rule
`variable_data`	Specifies the data for the variable value

RE$VARIABLE_VALUE_LIST Type

Identifies a list of variable values.

Syntax

```
TYPE SYS.RE$VARIABLE_VALUE_LIST AS VARRAY(1024) OF SYS.RE$VARIABLE_VALUE;
```

197

XMLTYPE

`XMLType` is a system-defined opaque type for handling XML data. It as predefined member functions on it to extract XML nodes and fragments.

You can create columns of `XMLType` and insert XML documents into it. You can also generate XML documents as `XMLType` instances dynamically using the `SYS_XMLGEN` and `SYS_XMLAGG SQL` functions.

This chapter contains the following topics:

See Also:

- *Oracle XML DB Developer's Guide*

Summary of XMLType Subprograms

Table 197–1 summarizes functions and procedures of the `XMLType`.

Table 197–1 XMLTYPE Subprograms

Method	Description
CREATENONSCHEMABASEDXML on page 197-4	Creates a non schema based XML from the input schema based instance.
CREATESCHEMABASEDXML on page 197-5	Creates a schema based `XMLType` instance from the non-schema based instance using the input schema URL.
CREATEXML on page 6	Static function for creating and returning an `XMLType` instance.
EXISTSNODE on page 197-8	Takes a `XMLType` instance and a `XPath` and returns 1 or 0 indicating if applying the `XPath` returns a non-empty set of nodes.
EXTRACT on page 197-9	Takes a `XMLType` instance and an `XPath`, applies the XPath expression and returns the results as an `XMLType`.
GETBLOBVAL on page 197-10	Returns the value of the `XMLType` instance as a `BLOB`
GETCLOBVAL on page 197-11	Returns the value of the `XMLType` instance as a `CLOB`.
GETNAMESPACE on page 197-12	Returns the namespace for the top level element in a schema based document.
GETNUMBERVAL on page 197-13	Returns the value of the `XMLType` instance as a `NUMBER`. This is only valid if the input `XMLType` instance contains a simple text node and is convertible to a number.
GETROOTELEMENT on page 197-14	Returns the root element of the input instance. Returns `NULL` if the instance is a fragment
GETSCHEMAURL on page 197-15	Returns the XML schema URL if the input is an XML Schema based.
GETSTRINGVAL on page 197-16	Returns the value of the `XMLType` instance as a string.
ISFRAGMENT on page 197-17	Checks if the input `XMLType` instance is a fragment or not. A fragment is a XML instance, which has more than one root element.
ISSCHEMABASED on page 197-18	Returns 1 or 0 indicating if the input `XMLType` instance is a schema based one or not.
ISSCHEMAVALID on page 197-19	Checks if the input instance is schema valid according to the given schema URL.
ISSCHEMAVALIDATED on page 197-20	Checks if the instance has been validated against the schema.
SCHEMAVALIDATE on page 197-21	Validates the input instance according to the XML Schema. Raises error if the input instance is non-schema based.
SETSCHEMAVALIDATED on page 197-22	Sets the schema valid flag to avoid costly schema validation.
TOOBJECT on page 197-23	Converts the `XMLType` instance to an object type.
TRANSFORM on page 197-24	Takes an `XMLType` instance and an associated stylesheet (which is also an `XMLType` instance), applies the stylesheet and returns the result as XML.

Table 197–1 (Cont.) XMLTYPE Subprograms

Method	Description
XMLTYPE on page 197-25	*Constructs an instance of the* `XMLType` *datatype. The constructor can take in the XML as a* `CLOB`*,* `VARCHAR2` *or take in a object type.*

CREATENONSCHEMABASEDXML

Member function. Creates a non-schema based XML document from a schema based instance.

Syntax

```
MEMBER FUNCTION CREATENONSCHEMABASEDXML
return XMLType deterministic;
```

CREATESCHEMABASEDXML

Member function. Creates a schema based `XMLType` instance from a non-schema based `XMLType` value. It uses either the supplied `SCHEMA` URL, or the `SCHEMALOCATION` attribute of the instance.

Syntax

```
MEMBER FUNCTION createSchemaBasedXML(
schema IN varchar2 := NULL)
return XMLType deterministic;
```

Parameter	Description
`schema`	Optional XMLSchema URL used to convert the value to the specified schema..

CREATEXML

Static function for creating and returning an XMLType instance. The string and clob parameters used to pass in the date must contain well-formed and valid XML documents. The options are described in the following table.

Syntax	Description

`STATIC FUNCTION createXML(`
`  xmlData IN varchar2)`
`RETURN XMLType deterministic;`
	Creates the XMLType instance from a string.

`STATIC FUNCTION createXML(`
`  xmlData IN clob)`
`RETURN XMLType  deterministic;`
	Creates the XMLType instance from a CLOB.

`STATIC FUNCTION createXML (`
`  xmlData IN clob,`
`  schema IN varchar2,`
`  validated IN number := 0,`
`  wellformed IN number := 0 )`
`RETURN XMLType deterministic;`
	This static function creates a schema-based XMLType instance using the specified schema and xml data parameters.

`STATIC FUNCTION createXML (`
`  xmlData IN varchar2,`
`  schema IN varchar2,`
`  validated IN number := 0,`
`  wellformed IN number := 0)`
`RETURN XMLType deterministic;`
	This static function creates a schema-based XMLType instance using the specified schema and xml data parameters.

`STATIC FUNCTION createXML (`
`  xmlData IN "<ADT_1>",`
`  schema IN varchar2 := NULL,`
`  element IN varchar2 := NULL,`
`  validated IN NUMBER := 0)`
`RETURN XMLType deterministic;`
	Creates an XML instance from an instance of an user-defined type.

`STATIC FUNCTION createXML (`
`  xmlData IN SYS_REFCURSOR,`
`  schema in varchar2 := NULL,`
`  element in varchar2 := NULL,`
`  validated in number := 0)`
`RETURN XMLType deterministic;`
	Creates an XML instance from a cursor reference. You can pass in any arbitrary SQL query as a CURSOR.

Syntax	Description
STATIC FUNCTION createXML (xmlData IN AnyData, schema in varchar2 := NULL, element in varchar2 := NULL, validated in number := 0) RETURN sys.XMLType deterministic parallel_enable	Creates an XML instance from ANYDATA. If the ANYDATAinstance contains an ADT, the XMLTypereturned is the same as would be returned for a call directly on the ADT. If the ANYDATAcontains a scalar, the XMLType contains a leaf node with the scalar value. The element name for this node is taken from the optional element string if present, and is "ANYDATA" if it is not.
STATIC FUNCTION createXML (xmlData IN blob, csid IN number, schema IN varchar2, validated IN number := 0, wellformed IN number := 0) return sys.XMLType deterministic	Creates an XML instance from a BLOB.
STATIC FUNCTION createXML (xmlData IN bfile, csid IN number, Schema IN varchar2, validated IN number := 0, wellformed IN number := 0) return sys.XMLType deterministic	Creates an XML instance from a BFILE.

Parameter	Description
xmlData	The actual data in the form of a BFILE, BLOB, CLOB, REF cursor, VARCHAR2 or object type.
schema	Optional Schema URL to be used to make the input conform to the given schema.
validated	Flag to indicate that the instance is valid according to the given XML Schema. (Default is 0)
wellformed	Flag to indicate that the input is well formed. If set, then the database would not do well formed check on the input instance. (Default is 0)
element	Optional element name in the case of the ADT_1 or REF CURSOR constructors. (Default is NULL)
CSID	The character set id of input XML data.

EXISTSNODE

Member function. Checks if the node exists. If the XPath string is `NULL` or the document is empty, then a value of 0 is returned, otherwise returns 1. The options are described in the following table.

Syntax	Description

`MEMBER FUNCTION existsNode(`
`  xpath IN varchar2)`
`RETURN number deterministic;`
	Given an XPath expression, checks if the XPath applied over the document can return any valid nodes.

`MEMBER FUNCTION existsNode(`
`  xpath in varchar2,`
`  nsmap in varchar2)`
`RETURN number deterministic;`
	This member function uses the XPath expression with the namespace information and checks if applying the XPath returns any nodes or not.

Parameter	Description
`xpath`	The XPath expression to test.
`nsmap`	Optional namespace mapping.

EXTRACT

Member function. Extracts an `XMLType` fragment and returns an `XMLType` instance containing the result node(s). If the XPath does not result in any nodes, then returns `NULL`. The options are described in the following table.

Syntax	Description

`MEMBER FUNCTION extract(`
`  xpath IN varchar2)`
`RETURN XMLType deterministic;`
	Given an XPath expression, applies the XPath to the document and returns the fragment as an `XMLType`.

`MEMBER FUNCTION extract(`
`  xpath IN varchar2,`
`  nsmap IN varchar2)`
`RETURN XMLType deterministic;`
	This member function applies the XPath expression and namespace mapping, over the XML data to return a `XMLType` instance containing the resultant fragment.

Parameter	Description
`xpath`	The XPath expression to apply.
`nsmap`	Optional prefix to namespace mapping information.

GETBLOBVAL

Member function. Returns a `BLOB` containing the serialized XML representation; if the returns is a temporary `BLOB`, then it must be freed after use.

Syntax

```
MEMBER FUNCTION getBlobVal()
RETURN Blob deterministic;
```

GETCLOBVAL

Member function. Returns a CLOB containing the serialized XML representation; if the returns is a temporary CLOB, then it must be freed after use.

Syntax

```
MEMBER FUNCTION getClobVal()
RETURN clob deterministic;
```

GETNAMESPACE

Member function. Returns the namespace of the top level element in the instance. Returns `NULL` if the input is a fragment or is a non-schema based instance.

Syntax

```
MEMBER FUNCTION getNamespace
return varchar2 deterministic;
```

GETNUMBERVAL

Member function. Returns a numeric value, formatted from the text value pointed to by the `XMLType` instance. The `XMLType` must point to a valid text node that contains a numerical value. The options are described in the following table.

Syntax

```
MEMBER FUNCTION getNumberVal()
RETURN number deterministic;
```

GETROOTELEMENT

Member function. Gets the root element of the `XMLType` instance. Returns `NULL` if the instance is a fragment.

Syntax

```
MEMBER FUNCTION getRootElement
return varchar2 deterministic;
```

GETSCHEMAURL

Member function. Returns the XML Schema URL corresponding to the `XMLType` instance, if the `XMLType` instance is a schema-based document. Otherwise returns `NULL`.

Syntax

```
MEMBER FUNCTION getSchemaURL
return varchar2 deterministic;
```

GETSTRINGVAL

Member function. Returns the document as a string. Returns s string containing the seralized XML representation, or in case of text nodes, the text itself. If the XML document is bigger than the maximum size of the `VARCHAR2`, which is 4000, then an error is raised at run time.

Syntax

```
MEMBER FUNCTION getStringVal()
RETURN varchar2 deterministic;
```

ISFRAGMENT

Determines if the `XMLType` instance corresponds to a well-formed document, or a fragment. Returns 1 or 0 indicating if the `XMLType` instance contains a fragment or a well-formed document.

Syntax

```
MEMBER FUNCTION isFragment()
RETURN number deterministic;
```

ISSCHEMABASED

Member function. Determines whether the `XMLType` instance is schema-based or not. Returns 1 or 0 depending on whether the `XMLType` instance is schema-based.

Syntax

```
MEMBER FUNCTION isSchemaBased
return number deterministic;
```

ISSCHEMAVALID

Member function. Checks if the input instance is conformant to a specified schema. Does not change the validation status of the XML instance. If a XML Schema URL is not specified and the xml document is schema based, the conformance is checked against the `XMLType` instance's own schema.

Syntax

```
member function isSchemaValid(
schurl IN VARCHAR2 := NULL,
elem IN VARCHAR2 := NULL)
return NUMBER deterministic;
```

Parameter	IN / OUT	Description
`schurl`	`(IN)`	The URL of the XML Schema against which to check conformance.
`elem`	`(IN)`	Element of a specified schema, against which to validate. This is useful when we have a XML Schema which defines more than one top level element, and we want to check conformance against a specific one of these elements.

ISSCHEMAVALIDATED

Member function. Returns the validation status of the `XMLType` instance -- tells if a schema based instance has been actually validated against its schema. Returns 1 if the instance has been validated against the schema, 0 otherwise.

Syntax

```
MEMBER FUNCTION isSchemaValidated
return NUMBER deterministic;
```

SCHEMAVALIDATE

Member procedure. Validates the XML instance against its schema if it hasn't already been done. For non-schema based documents an error is raised. If validation fails an error is raised; else, the document's status is changed to validated.

Syntax

```
MEMBER PROCEDURE schemaValidate(
   self IF OUT NOCOPY XMLType);
```

Parameter	IN / OUT	Description
self	(OUT)	XML instance being validated against the schema.

SETSCHEMAVALIDATED

Member function. Sets the `VALIDATION` state of the input XML instance.

Syntax

```
MEMBER PROCEDURE setSchemaValidated(
self IF OUT NOCOPY XMLType,
   flag IN BINARY_INTEGER := 1);
```

Parameter	IN / OUT	Description
`self`	`(OUT)`	XML instance.
`flag`	`(IN)`	0 - `NOT VALIDATED`; 1 - `VALIDATED` (Default)

TOOBJECT

Member procedure. Converts the XML value to an object type using the XMLSCHEMA mapping, if available. If a SCHEMA is not supplied or the input is a non-schema based XML, the procedure uses cannonical mapping between elements and object type attributes.

See Also:

- An in-depth discussion of this topic inside *Oracle XML DB Developer's Guide*

Syntax

```
MEMBER PROCEDURE toObject(
SELF in XMLType,
object OUT "<ADT_1>",
schema in varchar2 := NULL,
element in varchar2 := NULL);
```

Parameter	IN / OUT	Description
SELF	(IN)	Instance to be converted. Implicit if used as a member procedure.
object	(IN)	Converted object. An object instance of the required type may be passed in to this function
schema	(IN)	Schema URL. The mapping of the XMLType instance to the converted object instance may be specified using a schema.
element	(IN)	Top-level element name. An XML Schema document does not specify the top-level element for a conforming XML instance document without this parameter.

TRANSFORM

Member function. This member function transforms the XML data using the XSL stylesheet argument and the top-level parameters passed as a string of name=value pairs. If any of the arguments other than the parammap is NULL, then a NULL is returned.

Syntax

```
MEMBER FUNCTION transform(
xsl IN XMLType,
parammap in varchar2 := NULL)
RETURN XMLType deterministic;
```

Parameter	IN / OUT	Description
xsl	(IN)	The XSL stylesheet describing the transformation
parammap	(IN)	Top level parameters to the XSL - string of name=value pairs

XMLTYPE

XMLType constructor. The options are described in the following table.

Syntax	Description
constructor function XMLType(xmlData IN clob, schema IN varchar2 := NULL, validated IN number := 0, wellformed IN Number := 0) return self as result deterministic;	This constructor function creates an optionally schema-based XMLType instance using the specified schema and xml data parameters.
constructor function XMLType(xmlData IN varchar2, schema IN varchar2 := NULL, validated IN number := 0, wellformed IN number := 0) return self as result deterministic;	This constructor function creates an optionally schema-based XMLType instance using the specified schema and xml data parameters.
constructor function XMLType (xmlData IN "w<ADT_1>", schema IN varchar2 := NULL, element IN varchar2 := NULL, validated IN number := 0) return self as result deterministic;	This constructor function creates an optionally schema-based XMLType instance from the specified object type parameter.
constructor function XMLType(xmlData IN SYS_REFCURSOR, schema in varchar2 := NULL, element in varchar2 := NULL, validated in number := 0) return self as result deterministic;	This constructor function creates an optionally schema-based XMLType instance from the specified REF CURSOR parameter.
constructor function XMLType(xmlData IN AnyData, schema IN varchar2 := NULL, element IN varchar2 := NULL, validated IN number := 0) return self as result deterministic parallel_enable	This constructor function creates an optionally schema-based XMLType instance from the specified ANYDATA parameter. If the ANYDATA instance contains an ADT, the XMLType returned is the same as would be returned for a call directly on the ADT. If the ANYDATA contains a scalar, the XMLType contains a leaf node with the scalar value. The element name for this node is taken from the optional element string if present, and is "ANYDATA" if it is not.

Syntax	Description

`constructor function XMLType(`
`  xmlData IN blob, csid IN number,`
`  schema IN varchar2 := NULL,`
`  validated IN number := 0,`
`  wellformed IN number := 0)`
`return self as result deterministic`
	This constructor function creates an optionally schema-based `XMLType` instance from the specified `BLOB` parameter.

`constructor function XMLType(`
`  xmlData IN bfile,`
`  csid IN number,`
`  schema IN varchar2 := NULL,`
`  validated IN number := 0,`
`  wellformed IN number := 0)`
`return self as result deterministic`
	This constructor function creates an optionally schema-based `XMLType` instance from the specified `BFILE` parameter.

Parameter	Description
`xmlData`	The data in the form of a `BFILE`, `BLOB`, `CLOB`, `REF`s, `VARCHAR2` or object type.
`schema`	Optional Schema URL to be used to make the input conform to the given schema.
`validated`	Indicates that the instance is valid to the given XML Schema.
`wellformed`	Indicates that the input is well formed. If set, then the database would not do well formed check on the input instance.
`element`	Optional element name in the case of the `ADT_1` or `REF CURSOR` constructors. (Default is `NULL`)
`CSID`	The character set id of input XML data.

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z