Oracle Procedural Gateway® for APPC

User's Guide

10*g* Release 2 (10.2) for Microsoft Windows (32-Bit)

B16212-01

August 2005

ORACLE®

Oracle Procedural Gateway for APPC User's Guide, 10g Release 2 (10.2) for Microsoft Windows (32-Bit)

B16212-01

Primary Author: Maitreyee Chaliha

Contributing Author: Shahrokh Talmoud

Contents

2   Procedural Gateway Administration Utility

3   Creating a TIP

4 Client Application Development (SNA Only)

5 Implementing Commit-Confirm (SNA Only)

6   PG4TCPMAP Commands (TCP/IP Only)

7   Client Application Development (TCP/IP Only)

8   Problem Determination

A   Procedural Gateway for APPC Data Dictionary

B   Gateway RPC Interface

C   The UTL_PG and UTL_RAW Interface

D   Datatype Conversions

E   National Language Support

F   Tip Internals

G   Administration Utility Samples

Index

x

Preface

The Oracle Procedural Gateway for APPC provides Oracle applications seamless access to virtually any APPC-enabled system, including IBM mainframe data and services through Remote Procedure Call (RPC) processing. This document is intended for Windows users.

Intended Audience

Read this guide if you are responsible for tasks such as:

- determining hardware and software requirements

- installing, configuring, or administering an Oracle Procedural Gateway for APPC

- developing applications that access remote host databases through the Oracle Procedural Gateway for APPC using the SNA Communication Protocol or the TCP/IP communication protocol

- determining security requirements

- determining and resolving problems

Before using this guide to administer the gateway, you should understand the fundamentals of your Microsoft Windows operating system and Procedural Gateways.

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.

Related Documents

The *Oracle Procedural Gateway for APPC User's Guide for Microsoft Windows* is included as part of your product shipment. Also included is:

- *Oracle Procedural Gateway for APPC Messages Guide for UNIX and Microsoft Windows*
- *Oracle Procedural Gateway for APPC Installation and Configuration Guide*

You might also need Oracle Database 10*g* and Oracle Net documentation. The following is a useful list of the Oracle publications that may be referenced in this book:

- *Oracle Database Installation Guide 10g* Release 2 (10.2) *for Microsoft Windows (32-Bit)*
- *Oracle Database Administrator's Guide 10g* Release 2 (10.2) *for Microsoft Windows (32-Bit)*
- *Oracle Database Application Developer's Guide*
- *Oracle Database Server Concepts*
- *Oracle Database Server Distributed Systems*
- *Oracle Database Error Messages*
- *Oracle Net Services Administrator's Guide*

Refer to the *Oracle Technical Publications Catalog and Price Guide* for a complete list of documentation provided for Oracle products.

Conventions

Examples of input and output for the gateway and Oracle environment are shown in a special font:

```
C:\> mkdir \ORACLE\your_name
```

All output is shown as it appears. For input, refer to the following list of conventions and their meanings:

- `example text:` Words or phrases, such as `mkdir` and `ORACLE`, must be entered exactly as spelled and in the letter case shown. In this example, `mkdir` must be entered in lowercase letters and `ORACLE` in uppercase letters.
- *italic text:* Italicized uppercase or lowercase, such as *your_name*, indicates that you must substitute a word or phrase, such as the actual directory name.
- **BOLD text** or *bold italic TEXT:* Bold words or phrases refer to a file or directory structure, such as a directory, path, or file ID.
- ... Ellipses indicate that the preceding item can be repeated. You can enter an arbitrary number of similar items.

- **{ }**   Curly braces indicate that one of the enclosed arguments is required. Do not enter the braces themselves.

- **|**   Vertical lines separate choices

- **[ ]**   Square brackets enclose optional clauses from which you can choose one or none. Do not enter the brackets themselves.

Other punctuation, such as commas, quotation marks or the pipe symbol (|) must be entered as shown unless otherwise specified. Directory names, file IDs and so on appear in the required letter case in examples. The same convention is used when these names appear in text, and the names are highlighted in **bold**. The use of *italics* indicates that those portions of a file ID that appear in *italics* can vary.

Gateway commands, file IDs and reserved words, MS-DOS commands, keywords and environment variables appear in uppercase in examples and text. Reserved words and keywords must always be entered as shown; they have reserved meanings within the Oracle system.

SQL*Plus Prompts

The SQL*Plus prompt, SQL>, appears in SQL statement and SQL*Plus command examples. Enter your response at the prompt. Do not enter the text of the prompt, SQL>, in your response.

MS-DOS Prompts

The MS-DOS prompt, C:\>, appears in MS-DOC command examples. Enter your response at the prompt. Do not enter the text of the prompt, C:\>, in your response.

PGAU Prompts

The PGAU prompt, PGAU>, appears in PGAU command examples. Enter your response at the prompt. Do not enter the text of the prompt, PGAU>, in your response.

Directory Names

Throughout this document, there are references to the directories in which product-related files reside. **%ORACLE_HOME%** is used to represent the Oracle home directory. This is the default location for Oracle products. If you have installed into a location other than **%ORACLE_HOME%**, replace all references to **%ORACLE_HOME%** with the drive and path specification you have used.

Storage Measurements

Storage measurements use the following abbreviations:

- K, for kilobyte which equals 1024 bytes

- M, for megabyte which equals 1 048 576 bytes

- G, for gigabyte which equals 1 073 741 824 bytes

Accessing Installed Documentation

Documentation for Oracle Procedural Gateway for APPC for Windows

Documentation for this product includes this guide and the *Oracle Procedural Gateway for APPC Installation and Configuration Guide 10g* Release 2 (10.2) *for Microsoft Windows (32-Bit)* and the *Oracle Procedural Gateway for APPC Messages Guide 10g* Release 2 (10.2) *for UNIX and Windows*.

To access the documentation in HTML and PDF formats, use a browser to open the top level of the Gateway Documentation CD-ROM. This level contains links to product and Windows-specific documentation.

Oracle Product Documentation

Oracle Database 10*g* product documentation is on the Oracle Database 10*g* Platform-Specific Documentation CD-ROM. Instructions for accessing and installing the documents on the CD-ROM are found in the README file on the top level directory of the CD-ROM.

Oracle Services and Support

Oracle's corporate web page is at the following address:

http://www.oracle.com

Oracle offers a wide range of services to help facilitate corporate system solutions, including Oracle Education courses, Oracle Consulting services, and Oracle Support Services from the Web site. In addition, Oracle provides free trial software, updates on Oracle products and service, and technical brochures and data sheets.

Oracle Support Services

Technical Support registration and contact information worldwide is available at the following address:

http://www.oracle.com/support

At Oracle's support site, you will find templates to help you prepare information about your problem before you call so that you may be helped more quickly. You will also need your CSI number (if applicable) or complete contact details, including any special project information.

Oracle Technology Network

OTN delivers all product documentation, as well as technical papers, code samples, self-service developer support, and Oracle's key developer products to enable rapid development and deployment of applications built on Oracle technology.

Register with the Oracle Technology Network (OTN) at:

http://otn.oracle.com

All Oracle product documentation can be found at:

http://otn.oracle.com/documentation

Oracle*MetaLink*

Oracle*MetaLink* is Oracle's web service for technical information. Members of Oracle*MetaLink* can search for updates, alerts, patches, and other information about products, releases, and operating systems, or set preferences to be notified automatically of new information. Oracle*MetaLink* offers a variety of services to assist in setting up and administrating Oracle products, including procedures, scripts, commentary, and tuning and configuration best-practices bulletins. Please logon to Oracle*MetaLink* before installing or administrating your product to search for up to date information about Oracle Database 10*g* Release 2 (10.2) for Microsoft Windows (32-Bit).

In addition, Oracle*MetaLink* offers forums for information sharing among Oracle customers, and direct communication with Oracle Support Services. Oracle*MetaLink* is available to Product Support Customers at no extra cost. Sign up for free membership for this service at the following site:

http://www.oracle.com/support/metalink

Use your Support Access Code (SAC) number to register.

Oracle Products and Other Documentation

For U.S.A. customers, the Oracle Store is at:

http://store.oracle.com

Links to Stores in other countries are provided from this site.

Customer Service

Oracle Support Services contacts are listed at:

http://www.oracle.com/support

Support for Hearing and Speech Impaired Customers

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 technical questions, call:

  1.800.446.2398

- For non-technical questions, call:

  1.800.464.2330

Education and Training

Training information and worldwide schedules are available from:

http://education.oracle.com

1

Introduction to Oracle Procedural Gateway for APPC

This guide is intended for for Microsoft Windows (32-Bit) users.

Refer to the *Oracle Database Installation Guide 10g* Release 2 (10.2) *for Microsoft Windows (32-Bit)* and to the certification matrix on the Oracle*MetaLink* Web site for the most up-to-date list of certified hardware platforms and operating system versions. The Oracle*MetaLink* Web site can be found at:

http://metalink.oracle.com

The Oracle Procedural Gateway for APPC (the "gateway") enables users to initiate transaction program execution on remote online transaction processors (OLTPs). The Oracle Procedural Gateway for APPC can establish connection with OLTP using the SNA communication protocol. The gateway can also use TCP/IP for IMS Connect to establish communication with IMS/TM through TCP/IP. The gateway provides Oracle applications with seamless access to IBM mainframe data and services through Remote Procedural Call (RPC) processing.

Read this chapter to learn more about the architecture, uses, and features of the Oracle Procedural Gateway for APPC.

This chapter contains the following sections:

-

-

-

1.1 Overview of the Gateway

The Oracle Procedural Gateway for APPC extends the Remote Procedural Call (RPC) facilities available with the Oracle server. The gateway enables any client application to use PL/SQL to request execution of a remote transaction program (RTP) residing on a host. The gateway provides RPC processing to systems using the SNA APPC (Advanced Program-to-Program Communication) protocol and to IMS/TM systems using TCP/IP support for IMS Connect. This architecture allows efficient access to data and transactions available on the IBM mainframe and IMS, respectively.

The gateway requires no Oracle software on the remote host system. Thus, the gateway uses existing transactions with little or no programming effort on the remote host.

❏ **For gateways using SNA only:** The use of a generic and standard protocol, APPC, allows the gateway to access a multitude of systems. The gateway can communicate with virtually any APPC-enabled system, including IBM Corporation's CICS on any platform, IBM Corporation's IMS and APPC/MVS, and Computer Associates' IDMS. These transaction monitors provide access to a broad range of systems, allowing the gateway to access many datastores, including VSAM, DB2 (static SQL), IMS, IDMS, ADABAS and others.

The gateway can access any application capable of using the CPI-C API, either directly or through a TP monitor such as CICS.

1.2 Features of the Gateway

The Oracle Procedural Gateway for APPC provides the following benefits:

- Fast interface

  The gateway is optimized so that remote execution of a program is achieved with minimum network traffic. The interface to the gateway is an optimized PL/SQL stored procedure specification (called the "TIP" or "transaction interface package") precompiled in the Oracle Integrating Server. Because there are no additional software layers on the remote host, overhead occurs only when your program executes.

- Location transparency

  Client applications need not be operating system-specific. For example, your application can call a program in a CICS Transaction Server for z/OS. If you move the program to a CICS region on AIX, then you need not change the application.

- Application transparency

  Users calling applications that execute a remote transaction program are unaware that a request is sent to a host.

- Flexible interface

  You can use the gateway to interface with existing procedural logic or to integrate new procedural logic into an Oracle Integrating Server environment.

- Oracle server integration

The integration of the Oracle server with the gateway enables the gateway to benefit from existing and future Oracle features. For example, the gateway can be called from an Oracle stored procedure or database trigger.

- Transactional support

  The gateway and the Oracle Integrating Server allow remote transfer updates and Oracle server updates to be performed in a coordinated fashion.

- Wide selection of tools

  The gateway supports any tool or application that supports PL/SQL.

- PL/SQL code generator

  The Oracle Procedural Gateway for APPC provides a powerful development environment, including:

  - a data dictionary to store information relevant to the remote transaction

  - a tool to generate the PL/SQL Transaction Interface Package, or TIP

  - a report utility to view the information stored in the gateway dictionary

  - a complete set of tracing and debugging facilities

  - a wide set of samples to demonstrate the use of the product against datastores such as DB2, IMS, IDMS, CICS, and ADABAS

- Site autonomy and security

  The gateway provides site autonomy, allowing you to do such things as authenticate users. It also provides role-based security compatible with any security package running on your mainframe computer.

- Automatic conversion

  Through the TIP, the following conversions are performed:

  - ASCII to and from EBCDIC

  - remote transaction program datatypes to and from PL/SQL datatypes

  - national language support for many languages

- TCP/IP support for IMS Connect

  This release of the gateway includes TCP/IP support for IMS Connect, giving users a choice of whether to use an SNA or TCP/IP communication protocol. IMS Connect is an IBM product which allows TCP/IP clients to trigger execution of IMS transactions. The gateway can use a TCP/IP communication protocol to access IMS Connect, which triggers execution of IMS transactions. There is no SNA involvement with this configuration.

  Related to this new feature of the gateway is:

  - The gateway mapping tool. This release of the gateway includes a tool (pg4tcpmap) whose purpose is to map the information from your SNA Side Profile Name to the TCP/IP host name and Port Number. For more information about the pg4tcpmap tool, refer to Chapter 6 of this guide and to Chapter 10 of the *Oracle Procedural Gateway for APPC Installation and Configuration Guide* regarding configuration of the gateway using TCP/IP.

    Note: When your communications protocol is TCP/IP, only IMS is supported as the Online Transaction Processor (OLTP).

1.3 Terms

The following terms and definitions are used throughout this guide:

Gateway Initialization File

This file is known as **init*sid*.ora** and it contains parameters that govern the operation of the gateway. If you are using the SNA protocol, refer to Appendix A, "Gateway Initialization Parameters for SNA Protocol" in the *Oracle Procedural Gateway for APPC Installation and Configuration Guide* for more information. If your protocol is TCP/IP, refer to Appendix B, "Gateway Initialization Parameters for TCP/IP Communication Protocol" in the *Oracle Procedural Gateway for APPC Installation and Configuration Guide*.

Gateway Remote Procedure

The Oracle Procedural Gateway for APPC provides prebuilt remote procedures. In general, the following three remote procedures are used:

- PGAINIT, which initializes transactions

- PGAXFER, which transfers data

- PGATERM, which terminates transactions

Refer to Appendix B, "Gateway RPC Interface" in this guide and to "Remote Procedural Call Functions" in Chapter 1 of the *Oracle Procedural Gateway for APPC Installation and Configuration Guide* for more information about gateway remote procedures.

tg4pwd

 tg4pwd is a utility which encrypts passwords that are normally stored in the gateway initialization file. Passwords are stored in an encrypted form in the password file, making the information more secure. Refer to "Passwords in the Gateway Initialization File" in the security requirements chapter of the *Oracle Procedural Gateway for APPC Installation and Configuration Guide* for detailed information about how the **tg4pwd** utility works.

pg4tcpmap Tool

This tool is applicable only when the gateway is using TCP/IP support for IMS Connect. Its function is to map SNA parameters (such as Side Profile Name) to TCP/IP parameters (such as OLTP host name, IMS Connect port number and IMS destination ID).

PGA (Procedural Gateway Administration)

PGA is a general reference within this guide to all or most components comprising the Oracle Procedural Gateway for APPC. This term is used when references to a specific product or component are too narrow.

PGDL (Procedural Gateway Definition Language)

PGDL is the collection of statements used to define transactions and data to the PGAU.

PL/SQL Stored Procedure Specification (PL/SQL package)

This is a precompiled PL/SQL procedure that is stored in Oracle Integrating Server.

UTL_RAW PL/SQL Package (the UTL_RAW Functions)

This component of the gateway represents a series of data conversion functions for PL/SQL RAW variables and remote host data. The types of conversions performed depend on the language of the remote host data. Refer to Appendix D, "Datatype Conversions" in this guide for more information.

UTL_PG PL/SQL Package (the UTL_PG Functions)

This component of the gateway represents a series of COBOL numeric data conversion functions. Refer to "NUMBER_TO_RAW and RAW_TO_NUMBER Argument Values" in Appendix C of this guide for supported numeric datatype conversions.

Oracle Integrating Server

This is any Oracle server instance that communicates with the gateway for purposes of performing remote procedural calls to execute remote transaction programs (RTP). The Oracle Integrating Server can be on the same system as the gateway or on a different system. If it is on a different system, then Oracle Net is required on both systems. Refer to Figure 1–2, "Gateway Architecture Featuring SNA or TCP/IP Protocol" for a view of the gateway architecture.

OLTP (Online Transaction Processor)

OLTP is any of a number of online transaction processors available from other vendors, including CICS Transaction Server for z/OS, IMS/TM, and IDMS-DC.

> **Note:** When your communications protocol is TCP/IP, only IMS is supported as the Online Transaction Processor (OLTP).

PGAU (Procedural Gateway Administration Utility)

PGAU is the tool that is used to define and generate PL/SQL transaction interface packages (TIPs). Refer to Chapter 2, "Procedural Gateway Administration Utility" in this guide for more information about PGAU.

PG DD (Data Dictionary)

This component of the gateway is a repository of remote host transaction definitions and data definitions. PGAU accesses definitions in the PG DD when generating TIPs. The PG DD has datatype dependencies because it supports the PGAU and is not intended to be directly accessed by the customer. Refer to Appendix A, "Procedural Gateway for APPC Data Dictionary" in this guide for a list of PG DD tables.

RPC (Remote Procedural Call)

RPC is a programming call that executes program logic on one system in response to a request from another system. Refer to "Gateway Remote Procedure" in Appendix C of the *Oracle Procedural Gateway for APPC Installation and Configuration Guide*, and to Appendix B, "Gateway RPC Interface" in this guide for more information.

RTP (Remote Transaction Program)

A remote transaction program is a customer-written transaction, running under the control of an OLTP, which the user invokes remotely using a PL/SQL procedure. To execute a remote transaction program through the gateway, you must use RPC to execute a PL/SQL program to call the gateway functions.

TIP (Transaction Interface Package)

A TIP is an Oracle PL/SQL package that exists between your application and the remote transaction program. The transaction interface package, or TIP, is a set of PL/SQL stored procedures that invoke the remote transaction program through the gateway. TIPs perform the conversion and reformatting of remote host data using PL/SQL and UTL_RAW/UTL_PG functions.

Figure 1–1 illustrates where the terminology discussed in the preceding sections applies to the gateway's architecture.

Figure 1–1   Relationship of Gateway and Oracle Integrating Server on Windows
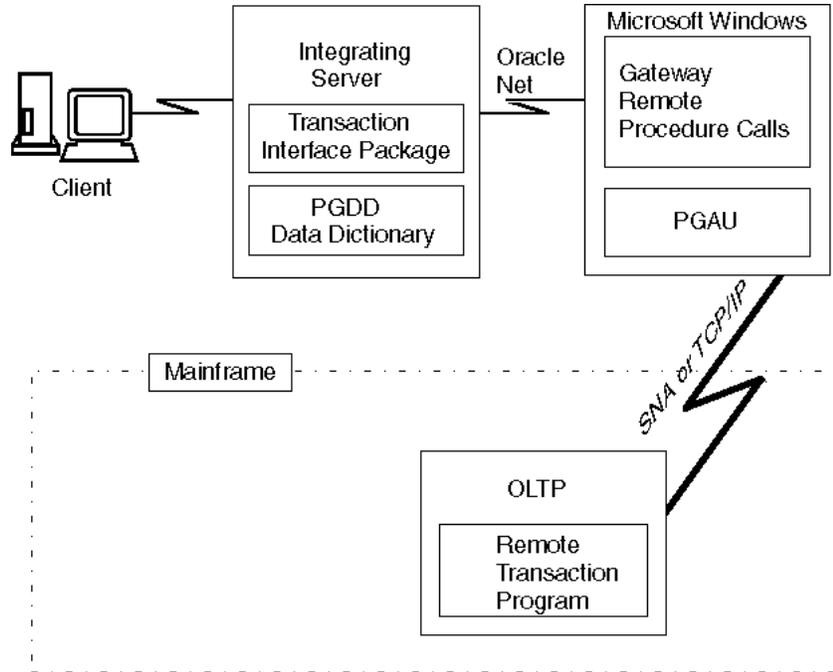
Figure 1–1 provides an illustration of the relationship between the gateway and the Oracle Integrating Server on a Windows host. The client is connected to the Oracle Integrating Server, which is connected to the gateway, which holds the gateway remote procedure calls and PGAU, via Oracle Net . They are connected to the OLTP in the Mainframe via TCP/IP or SNA. Figure 1–1 is adequately described in the text surrounding it.

End of description.

****

1.4 Examples and Sample Files for the Gateway

The following sample files and examples are referred to for illustration purposes throughout this guide. There are different example and sample files for a gateway using the SNA protocol than for a gateway using TCP/IP for IMS Connect.

Examples and Sample Files for Gateway Using SNA

For gateways using the SNA communication protocol, this guide uses a CICS-DB2 inquiry as an example. Transaction Interface Packages (TIPs) **pgadb2i.pkb** and **pgadb2i.pkh** send an employee number, empno, to a DB2 application and receive an employee record, emprec.

The CICS-DB2 inquiry sample and its associated PGAU commands are also available in the **%ORACLE_HOME%\pg4appc\demo\CICS** directory. The sample CICS-DB2 inquiry used as an example in this chapter is in files **pgadb2i.pkh** and **pgadb2i.pkb**. Refer to the **README.doc** file in the same directory for information about installing and using the samples. It can be found in the **%ORACLE_HOME%pg4appc\demo\CICS** directory.

Examples and Sample Files for Gateway Using TCP/IP

If your gateway is using the TCP/IP communication protocol, this guide uses an IMS inquiry as an example. Transaction Interface Packages (TIPs) **pgtflip.pkh** and **pgtflip.pkb** send input to IMS, through IMS Connect, and receive the flipped input as the output.

The IMS inquiry sample (**FLIP**) and its associated PGAU commands are located in the **%ORACLE_HOME%\pg4appc\demo\IMS** directory. The sample IMS inquiry used as an example for a gateway using TCP/IP is located in files **pgtflip.pkh** and **pgtflip.pkb**.

Refer to the **README.doc** file for more information about installing and using other IMS samples. It can be found in the **%ORACLE_HOME%\pg4appc\demo\IMS** directory.

1.5 Architecture of the Gateway

The architecture of Oracle Procedural Gateway for APPC consists of several components:

1. Oracle Integrating Server

Refer to the configuration chapter appropriate to your communications protocol in the *Oracle Procedural Gateway for APPC Installation and Configuration Guide* for a description of the various methods for establishing the gateway-Oracle Integrating Server relationship.

The Oracle Integrating Server can also be used for non-gateway applications.

2. The gateway

Oracle Procedural Gateway for APPC must be installed on a server that can run the required version of the operating system.

3. An OLTP (online transaction processor)

The OLTP must be accessible from the gateway using your SNA or TCP/IP communication protocol. Multiple Oracle Integrating Servers can access the same gateway. A single system gateway installation can be configured to access more than one OLTP.

– **For gateway using TCP/IP for IMS Connect:** The only OLTP that is supported through TCP/IP is IMS through IMS Connect.

– The OLTP must be accessible to the system using the TCP/IP protocol. Multiple Oracle Integrating Servers can access the same gateway. A single system gateway installation can be configured to access more than one OLTP. Multiple IMS can be accessed from an IMS Connect. If you have a number of IMS Connect systems available, any of these may be connected to one or more IMS systems.

Figure 1–2 illustrates the architecture of the Oracle Procedural Gateway for APPC using SNA or TCP/IP, as described in the previous section.

Figure 1–2   Gateway Architecture Featuring SNA or TCP/IP Protocol
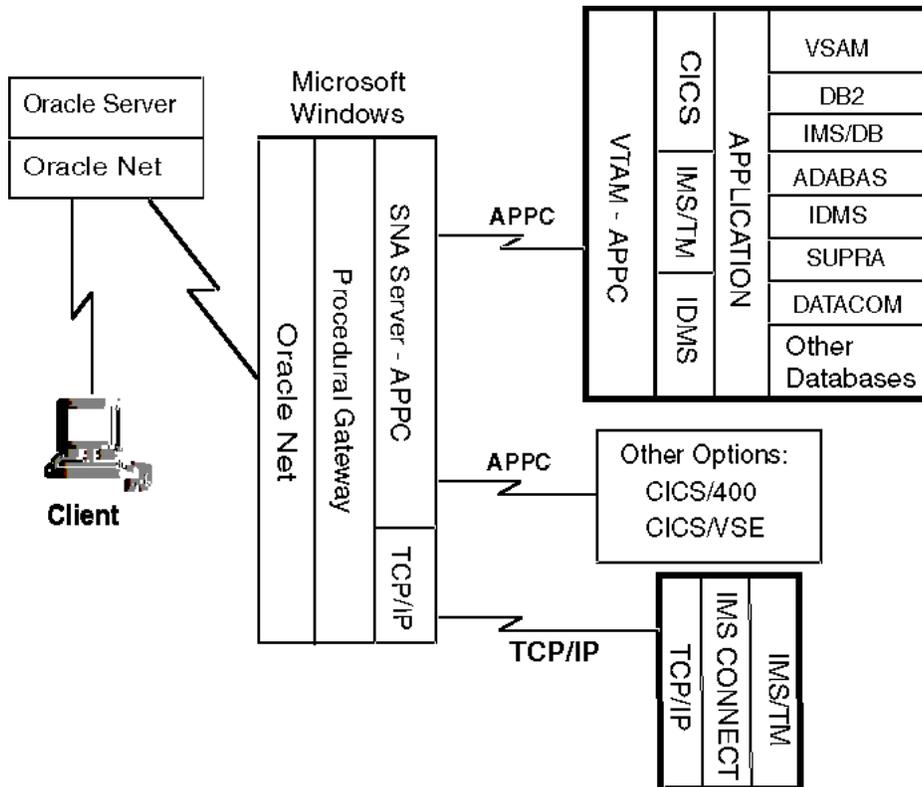
Figure 1–2 illustrates the architecture of the gateway when using either the SNA or TCP/IP communication protocol. It is adequately described in the text surrounding the figure.

End of description.

****

1.6 Starting the Gateway

Because the gateway does not have background processes and doesn't need a management utility such as Oracle Enterprise Manager, you needn't "start" the gateway. Each Oracle Integrating Server user session that accesses a particular gateway creates an independent process on Windows that runs the gateway server and executes either the SNA or TCP/IP functions to communicate with an OLTP, depending upon your protocol.

1.7 Communication With the Gateway

All of the communication between the user or client program and the gateway is handled through a transaction interface package (TIP) which executes on an Oracle Integrating Server. The TIP is a standard PL/SQL package that provides the following functions:

- declares the PL/SQL variables that can be exchanged with a remote transaction program;

- calls the gateway packages that handle the communications for starting the conversation, exchanging data and terminating the conversation;

■ handles all datatype conversions between PL/SQL datatypes and the target program datatypes.

The Procedural Gateway Administration Utility (PGAU), provided with the gateway, automatically generates the TIP specification.

The gateway is identified to the Oracle Integrating Server using a database link. The database link is the same construct used to identify other Oracle server databases. The functions in the gateway are referenced in PL/SQL as:

function_name@dblink_name

1.8 Remote Procedural Call Functions

The Oracle Procedural Gateway for APPC provides a set of functions that are invoked by the client through remote procedural call (RPC). These functions direct the gateway to initiate, transfer data with, and terminate remote transaction programs running under an OLTP on another system.

Table 1–1 lists the remote procedural call functions and the correlating commands that are invoked in the gateway and remote host.

Table 1–1   RPC Functions and Commands in the Gateway and Remote Host

Applications	Oracle TIP	Gateway	Remote Host
call tip_init	tip_init call pgainit@gateway	PGAINIT	Initiate program
call tip_main	tip_main call pgaxfer@gateway	PGAXFER	Exchange data
call tip_term	tip_term call pgaterm@gateway	PGATERM	Terminate program

1.8.1 TIP Function

The following sections describe how a TIP works by first establishing a connection to the remote host, then exchanging data from the target transaction program and finally, terminating a conversation.

1.8.1.1 Remote Transaction Initiation

The TIP initiates a connection to the remote host using one of the gateway functions, PGAINIT.

When the communication protocol is SNA: PGAINIT provides, as input, the required SNA parameters to start a conversation with the target transaction program. These parameters are sent across the SNA network, which returns a conversation identifier to PGAINIT. Any future calls to the target program use the conversation identifier as an INPUT parameter.

When the communication protocol is TCP/IP: PGAINIT provides, as input, the required TCP/IP parameters.These parameters are sent across the TCP/IP network to start the conversation with the target transaction program; the TCP/IP network returns a socket file descriptor to PGAINIT. Any future calls, such as PGAXFER and PGATERM, use this same socket file descriptor as an input parameter.

1.8.1.2 Data Exchange

After the conversation is established, a procedural gateway function called PGAXFER can exchange data in the form of input and output variables. PGAXFER sends and receives buffers to and from the target transaction program. The gateway sees a buffer as only a RAW stream of bytes. The TIP that is residing in the Oracle Integrating Server is responsible for converting the application's PL/SQL datatypes to RAW before sending the buffer to the gateway. It is also responsible for converting RAW to the PL/SQL datatypes before returning the results to the application.

1.8.1.3 Remote Transaction Termination

When communication with the remote program is complete, the gateway function PGATERM terminates the conversation between the gateway and the remote host.

When the communication protocol is SNA: PGATERM uses the conversation identifier as an INPUT parameter to request conversation termination.

When the communication protocol is TCP/IP: PGATERM uses the socket file descriptor for TCP/IP as an INPUT parameter to request conversation termination.

❏ At this point, if your communication protocol is SNA, then proceed to the following section, Section 1.9, "Overview of a Gateway Using SNA" on page 1-10.

❏ If your gateway communication protocol is TCP/IP, then proceed to Section 1.12, "Overview of a Gateway Using TCP/IP" on page 1-14.

1.9 Overview of a Gateway Using SNA

If you are using the SNA communication protocol, read the following sections to develop an understanding of how the gateway communicates with the Oracle Integrating Server and with the mainframe, as well as transaction types unique to your gateway and writing TIPs.

1.9.1 Transaction Types for a Gateway Using SNA

The Oracle Procedural Gateway for APPC supports three types of transactions that read data from and write data to remote host systems:

■ **one-shot**

In a one-shot transaction, the application initializes the connection, exchanges data and terminates the connection, all in a single call.

■ **persistent**

In a persistent transaction, multiple calls to exchange data with the remote transaction can be executed before terminating the conversation.

■ **multi-conversational**

In a multi-conversation transaction, the procedural gateway server can be used to exchange multiple records in one call to the remote transaction program.

Refer to "Remote Host Transaction Types" in Chapter 4, "Client Application Development (SNA Only)" of this guide for more information about transaction types.

The following list demonstrates examples of the power of the Oracle Procedural Gateway for APPC:

■ You can initiate a CICS transaction on the mainframe to retrieve data from a VSAM file for a PC application.

- You can modify and monitor the operation of a remote process control computer.

- You can initiate an IMS/TM transaction that executes static SQL in DB2.

- You can initiate a CICS transaction that returns a large number of records in a single call.

1.10 Simple Gateway Communication With the Oracle Server (SNA)

This section describes simple communication between the mainframe and the Oracle Integrating Server on a gateway using the SNA communication protocol. The Oracle Procedural Gateway for APPC lets you write your own procedures to begin transferring information between the Oracle Integrating Server and a variety of programs on an IBM mainframe, including IBM CICS, IMS, and APPC/MVS.

For an illustration of the communications function of the Oracle Procedural Gateway for APPC, refer to **%ORACLE_HOME%\pg4appc\demo\CICS\pgacics.sql**, which is a simple sample communication between the Oracle server and CICS Transaction Server for z/OS. Executing the simple PL/SQL procedure **pgacics.sql** causes the Oracle Integrating Server to invoke the procedural gateway, which uses SNA to converse with the **FLIP** transaction in CICS. These steps are described in detail in Section 1.10.1, "Steps to Communication Between Gateway and Mainframe, Using SNA". Note that you will already have compiled and linked the stored procedure when you configured the gateway.

1.10.1 Steps to Communication Between Gateway and Mainframe, Using SNA

The following steps describe the Windows-to-mainframe communications process illustrated in Figure 1–3, "Communication Between the Oracle Server and the Mainframe, Using SNA" when your communication protocol is SNA to communicate between the gateway and the mainframe:

1. From SQL*Plus, execute **pgacics**; this invokes the PL/SQL stored procedure in the Oracle Integrating Server.

   ```
   C:\> sqlplus <userid>/<password>@<database_specification_string>
   SQL> execute pgacics('==< .SCIC htiw gnitacinummoc si yawetag ruoy
   ,snoitalutargnoC >==');
   ```

2. The **pgacics** PL/SQL stored procedure will start up the gateway. The gateway will start up communication with CICS Transaction Server for z/OS through SNA and will call **FLIP**.

3. **FLIP** processes the input, generates the output and sends the output back to the procedural gateway.

4. Finally, the procedural gateway will send the output back to the PL/SQL stored procedure in the Oracle Integrating Server. The result is displayed in SQL*Plus:

   ```
   ==> Congratulations, your gateway is communicating with CICS. <==
   PL/SQL procedure successfully completed.
   ```

   Figure 1–3, "Communication Between the Oracle Server and the Mainframe, Using SNA" illustrates the communications process described in steps 1 through 4 in the previous section.

Figure 1–3   Communication Between the Oracle Server and the Mainframe, Using SNA
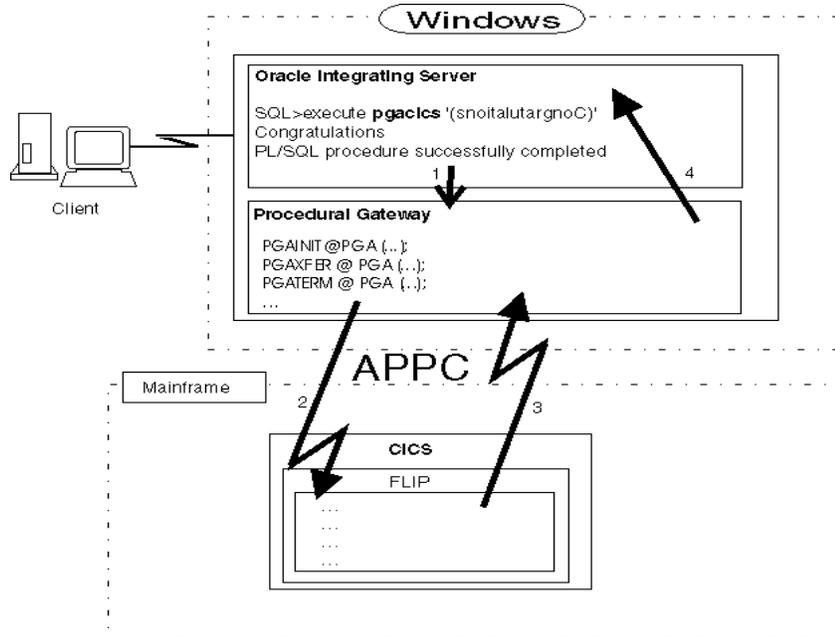
Figure 1–3 describes an illustration of the communication flow between the gateway and the mainframe, using SNA. This figure is adequately described in the sections preceding it.

End of description.

****

1.11 Writing TIPs to Generate PL/SQL Programs on Gateway Using SNA

Most transactions using SNA communication protocol are much larger and more complex than the simple sample **pgacics.sql** file referred to in Figure 1–3, "Communication Between the Oracle Server and the Mainframe, Using SNA". Additionally, communication with a normal-sized RTP (remote transaction program) would require you to create an extremely long PL/SQL file. Oracle Procedural Gateway for APPC's PGAU function generates the PL/SQL procedure for you.

The following is a brief description of the four steps necessary for you to generate a TIP. Refer to Chapter 3, "Creating a TIP" for detailed information about this procedure, and refer to Chapter 2, "Procedural Gateway Administration Utility" for more information about PGAU.

All parameter names in this section are taken from a file called **pgadb2i.ctl** in the **%ORACLE_HOME%\pga4appc\demo\CICS** directory.

1.11.1 Steps to Writing a TIP on a Gateway Using SNA

Follow these steps to write a TIP.

Step 1   Create a control file:

The user writes the control files. The control file has four main types of PGAU commands:

1. DEFINE DATA. This is used to define input and output fields, using COBOL data definitions.

    - Sample define data:

    ```
    define data empno plsdname(empno) usage(pass) language(ibmvscobolii)
                     infile("empno.cob");
    ```

2. DEFINE CALL. This is used to define PL/SQL functions calls to be generated as part of the package.

    - Sample define call:

    ```
    define call db2imain pkgcall(pgadb2i_main)
                         parms((empno in),(emprec out));
    ```

3. DEFINE TRANSACTION. This is used to group the above functions and specify other parameters on which the TIP depends.

    - Sample define transaction:

    ```
    define transaction db2i call(db2imain,db2idiag)
                             sideprofile(CICSPGA)
                             tpname(DB2I)
                             logmode(oraplu62)
                             synclevel(0)
                             nls_language("american_america.we8ebcdic37c");
    ```

4. GENERATE. This is used to generate the TIP specification files from the previously stored transaction, call and data definitions.

    - Sample generate transaction:

    ```
    generate db2i pkgname(pgadb2i) pganode(pga) outfile("pgadb2i");
    ```

Step 2   Execute the control file within PGAU

Running the control file within PGAU will create PG DD entries for the data, call, and transaction definitions, and will generate the specification files (For example, **pgadb2i.pkh** and **pgadb2i.pkb)**:

```
C:\> pgau
PGAU> CONNECT<userid>/<password>@<database>_specification_string>
PGAU> @pgadb2i.ctl
```

Step 3   Execute the specification files

Running the specification files will create the PL/SQL stored procedures. Note that the header specification file (for example, **pgadb2i.pkh**) must be run first:

```
C:\> sqlplus<userid>/<password>@<database_specification_string>
SQL> @pgadb2i.pkh;
SQL> @pgadb2i.pkb;
```

Step 4 Create a driver procedure to run the TIP

The TIP is now ready for use. For convenience, it will usually be called using a driver procedure (for example, **db2idriv**). This driver will then call the individual stored procedures in the correct order. Create the driver procedure and run it:

```
C:\> sqlplus <userid>/<password>@<database_specification string>
SQL> @pgadb2id.sql
SQL> execute db2idriv('000320');
```

1.12 Overview of a Gateway Using TCP/IP

If you are using the TCP/IP communication protocol, read the following sections to develop an understanding of how the gateway communicates with the Oracle Integrating Server and with the mainframe, as well as transaction types unique to your gateway and writing TIPs.

1.12.1 Transaction Types for a Gateway Using TCP/IP

The Oracle Procedural Gateway for APPC using TCP/IP support for IMS Connect supports three types of transaction socket connections:

- **transaction socket**

  The socket connection lasts across a single transaction.

- **persistent socket**

  The socket connection lasts across multiple transactions.

- **non-persistent socket**

  The socket connection lasts across a single exchange consisting of one input and one output.

  > **Note:**   Do not use the non-persistent socket type if you plan on implementing conversational transactions because multiple connects and disconnects will occur.

Refer to the section about pg4tcpmap commands in Chapter 6, "PG4TCPMAP Commands (TCP/IP Only)" of this guide for more information about the function and use of these parameters.

You can initiate an IMS/TM transaction that executes static SQL in DB2; this illustrates the power of the Oracle Procedural Gateway for APPC's feature supporting TCP/IP for IMS Connect.

1.13 Simple Gateway Communication with the Oracle Server (TCP/IP)

This section describes simple communication between IMS and the Oracle Integrating Server whenTCP/IP for IMS Connect is being used as the communication protocol between the gateway and the remote host (IMS). The Oracle Procedural Gateway for APPC lets you write your own procedures to begin transferring information between the Oracle Integrating Server and I/O PCB programs on IMS.

For an illustration of the communications function of the gateway using TCP/IP for IMS Connect, refer to **%ORACLE_HOME%\pg4appc\demo\IMS\pgaims.sql** file.

Executing the simple PL/SQL procedure **pgaims.sql** causes the Oracle Integrating Server to invoke the gateway, which uses TCP/IP to converse with the sample transaction **FLIP** in IMS. The communication steps that take place when you execute the PL/SQL procedure are described in detail in Section 1.13.2, "Steps to Communication Between the Gateway and IMS, Using TCP/IP". Note that you will already have compiled and linked the stored procedure when you configured the gateway.

1.13.1 Preparing the Gateway to Communicate Using TCP/IP

If your gateway is using TCP/IP support for IMS Connect, then you must use the pg4tcpmap tool to create the required mapping between PGAINIT parameters and the target system network address information. The pg4tcpmap tool maps the Side Profile Name specified in a DEFINE TRANSACTION to TCP/IP and IMS Connect attributes, such as port number, IP address (host name) and IMS subsystem ID. The TCP/IP parameters are used to start a conversation with the target transaction program.

The pg4tcpmap tool must be run in order to populate the PGA_TCP_IMSC table before executing any TIPs which rely on TPC/IP support for IMS Connect.

- Refer to Chapter 6, "PG4TCPMAP Commands (TCP/IP Only)" in this guide for complete instructions for setting up and executing pg4tcpmap commands to populate the PGA_TCP_IMSC table. Chapter 6 also explains the content of the PGA_TCP_IMSC table and an example of how to use the table.

- For more information about the role of the pg4tcpmap tool in configuring the gateway, refer to Chapter 10 in the *Oracle Procedural Gateway for APPC Installation and Configuration Guide* regarding gateway configuration using the TCP/IP protocol.

- A trace file from a sample pg4tcpmap execution is located in Chapter 8, "Problem Determination" in this guide.

- A screen output file is located in Appendix B, "Gateway Initialization Parameter File Using TCP/IP" in the *Oracle Procedural Gateway for APPC Installation and Configuration Guide*.

1.13.2 Steps to Communication Between the Gateway and IMS, Using TCP/IP

The following steps describe the Windows-to-IMS on a mainframe communications process, as illustrated in Figure 1–4 when your communication protocol is TCP/IP:

1. From SQL*Plus, execute **pgaims.sql**; this invokes the PL/SQL stored procedure in the Oracle Integrating Server.

   ```
   C:\> sqlplus <userid>/<password>@<database_specification_string>
   SQL> execute pgaims 'snoitalutargnoC';
   ```

   The **pgaims.sql** stored procedure will start up the gateway.

2. The gateway which has the APPC information will call the mapping table (PGA_TCP_IMSC). The mapping table will map the information so that it will have the host name (TCP/IP address) and the port number.

   > **Note:** Rather than insert, delete or update the PGA_TCP_IMSC mapping table manually, you should use the pg4tcpmap tool to do so. You may use the select statement to query the rows.

3. When the gateway has the port number and host name, it will initiate communication with IMS through TCP/IP, and will call **FLIP** through IMS Connect.

4. **FLIP** processes the input, generates the output and sends the output back to the gateway.

5. Finally, the gateway will send the output back to the PL/SQL stored procedure in the Oracle Integrating Server. The result is displayed in SQL*Plus:

```
Congratulations
PL/SQL procedure successfully completed.
```

Figure 1–4, "Communication Between Oracle Server and Mainframe, Using TCP/IP" illustrates the communications process described in the previous Steps 1 through 5.

Figure 1–4   Communication Between Oracle Server and Mainframe, Using TCP/IP
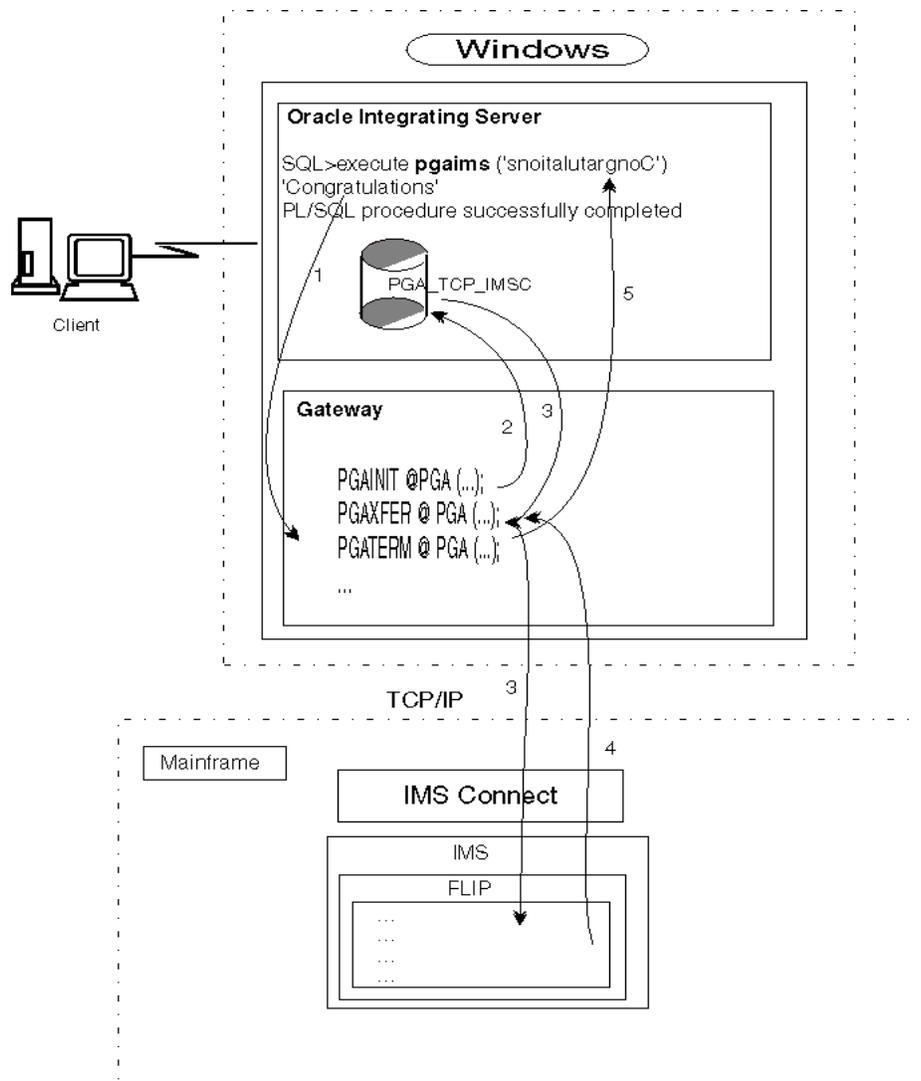
Figure 1–4 illustrates the communication flow between the gateway and the mainframe, when using the TCP/IP communication protocol. This figure is adequately described in the section preceding it.

End of description.

****

1.14 Writing TIPs to Generate PL/SQL Programs on Gateway Using TCP/IP

Most transactions are much larger and more complex than the simple sample **pgaims.sql** file referred to in Figure 1–4, "Communication Between Oracle Server and Mainframe, Using TCP/IP". Additionally, communication with a normal-sized RTP (remote transaction program) would require you to create an extremely long PL/SQL file. Oracle Procedural Gateway for APPC's TIP function generates the PL/SQL procedure for you.

The following is a brief description of the four steps necessary for you to generate a TIP. Refer to Chapter 3, "Creating a TIP" for detailed information about this procedure, and refer to Chapter 2, "Procedural Gateway Administration Utility" for more information about PGAU.

All parameter names in this section are taken from a file called **pgtflip.ctl** in the **%ORACLE_HOME%\pga4appc\demo\IMS** directory.

1.14.1 Steps to Writing a TIP on a Gateway Using TCP/IP

Follow these steps to write a TIP.

Step 1   Create a control file:

The user writes the control files. The control file has four main types of PGAU commands:

1. DEFINE DATA. This is used to define input and output fields, using COBOL data definitions.

    ■   Sample define data:

    ```
    define data flipin plsdname(flipin) usage(pass) language(ibmvscobolii)
    (
      01 msgin pic x(20).
    )

    define data flipout plsdname(flipout) usage(pass) language(ibmvscobolii)
    (
      01 msgout pic x(20).
    )
    ```

2. DEFINE CALL. This is used to define PL/SQL functions calls to be generated as part of the package.

    ■   Sample define call:

    ```
    define call flipmain pkgcall(pgtflip_main)
                        parms((flipin in),(flipout out));
    ```

3. DEFINE TRANSACTION. This is used to group the above functions and specify other parameters on which the TIP depends.

■ Sample define transaction:

```
define transaction imsflip call(flipmain)
                          sideprofile(pgatcp)
                          tpname(flip)
                          nls_language("american_america.us7ascii");
```

> **Note:** On a gateway using TCP/IP, the side profile name value is actually the TCP/IP unique name that was defined when the user specified the value, hostname, port number and many other IMS Connect values during configuration of the network.
>
> Refer to Chapter 6, "PG4TCPMAP Commands (TCP/IP Only)" in this guide and to Chapter 10 of the *Oracle Procedural Gateway for APPC Installation and Configuration Guide* for detailed information about the pg4tcpmap tool.

4. GENERATE. This is used to generate the TIP specification files from the previously stored transaction, call and data definitions.

■ Sample generate transaction:

```
generate imsflip pkgname(pgtflip) pganode(pga10ia) outfile("pgtflip")
diagnose(pkgex(dc,dr));
```

Step 2 Execute the control file within PGAU

Running the control file within PGAU will create PG DD entries for the data, call, and transaction definitions, and will generate the specification files (For example, **pgtflip.pkh** and **pgtflip.pkb**):

```
C:\> cd %ORACLE_HOME%\pg4appc\demo\IMS
C:\> pgau
PGAU> CONNECT userid/password@database_specification_string
PGAU> @pgtflip.ctl
```

Step 3 Execute the specification files

Running the specification files will create the PL/SQL stored procedures. Note that the header specification file (for example, **pgtflip.pkh**) must be run first:

```
C:\> sqlplus userid/password@database_specification_string
SQL> @pgtflip.pkh;
SQL> @pgtflip.pkb;
```

Step 4 Create a driver procedure to run the TIP

The TIP is now ready for use. For convenience, it will usually be called using a driver procedure (for example, **pgtflipd**). This driver will then call the individual stored procedures in the correct order. Create the driver procedure and run it:

```
C:\> sqlplus <userid>/<password>@<database_specification string>
SQL> @pgtflip.sql
SQL> execute pgtflipd('hello');
```

2

Procedural Gateway Administration Utility

The Procedural Gateway Administration Utility (PGAU) is a utility that assists the PGA administrator or user to define the data which is to be exchanged with remote transaction programs. It generates the PL/SQL Transaction Interface Packages (TIPs) discussed in Chapter 3, "Creating a TIP", Appendix F, "Tip Internals" and, depending upon your communication protocol, either Chapter 4, "Client Application Development (SNA Only)" or Chapter 7, "Client Application Development (TCP/IP Only)".

This chapter contains the following sections:

- "Overview of PGAU" on page 2-2
- "COMMIT/ROLLBACK Processing" on page 2-3
- "Invoking PGAU" on page 2-3
- "Definitions and Generation in PGAU" on page 2-4
- "Process to Define and Test a TIP" on page 2-5
- "PGAU Commands" on page 2-7

2.1 Overview of PGAU

> **Note:** If you have existing TIPs that were generated previously on a gateway using the SNA protocol and you want to utilize the new TCP/IP feature, then the TIPs will have to be regenerated by PGAU with mandatory NLS_LANGUAGE and Side Profile Settings. Specify the appropriate ASCII character set in the DEFINE TRANSACTION command.
>
> This is due to the fact that the gateway assumes that the appropriate "user exit" in IMS Connect is being used, which would translate between the appropriate ASCII and EBCDIC character sets.

PGAU maintains a data dictionary, PG DD, which is a collection of tables in an Oracle database. These tables hold the definitions of the remote transaction data and how that data is to be exchanged with the remote transaction program. Refer to "Ensuring TIP and Remote Transaction Program Correspondence" on page 4-12 for a discussion of the correlation between TIPs and their respective remote transaction programs. The PG DD contents define this correlation.

The PGA administrator or user defines the correlation between TIPs and the remote transaction program using the following PGAU commands (also called "statements"):

- PGAU DEFINE DATA statements, which describe the data to be exchanged.

- PGAU DEFINE CALL statements, which describe the exchange sequences.

- PGAU DEFINE TRANSACTION statements, which group the preceding CALL and DATA commands together and describe certain aspects unique to the remote transaction program, such as its network name or location.

- PGAU GENERATE statement, which the PGA administrator or user uses to specify and create the TIP specifications, after the TIP/transaction correlation has been defined in the PG DD. Additional PGAU commands needed to alter and delete definitions in the PG DD are described in "PGAU Commands" later in this chapter.

The PGAU commands are known collectively as Procedural Gateway Definition Language (PGDL). Any references to PGDL are to the collection of PGAU commands defined in this chapter.

PGAU provides editing and spooling facilities and the ability to issue SQL commands.

> **Caution:** Do not use PGAU instead of SQL*Plus for general database administration.

Alternatively, PGAU commands can be supplied in a control file. The control file contains one or more PGAU commands for manipulating the PG DD or generating TIP specifications.

PGAU issues status messages on each operation. The message text is provided through Oracle NLS message support. PGAU processes each command in sequence. An error on a single command causes PGAU to skip that command.

To run PGAU, the PG Data Dictionary tables must already have been created. Refer to the gateway configuration chapters pertinent to your communications protocol in the *Oracle Procedural Gateway for APPC Installation and Configuration Guide*.

2.2 COMMIT/ROLLBACK Processing

The following sections provide information on COMMIT/ROLLBACK processing.

2.2.1 COMMIT Processing

PGAU never issues COMMIT commands. As the user, it is your responsibility to COMMIT PG DD changes when all the changes are implemented. Otherwise Oracle issues a COMMIT command by default when you exit the PGAU session. If PG DD changes are not to be committed, you must run a ROLLBACK command before exiting.

2.2.2 ROLLBACK Processing

PGAU sets a savepoint at the beginning of each PGAU command that alters the PG DD and at the beginning of a PGAU GROUP. PGAU rolls back to the savepoint upon any PGAU command or group failure.

You can code COMMIT or ROLLBACK commands within PGAU scripts, or interactively in PGAU, but not within a GROUP.

Neither COMMIT nor ROLLBACK is issued for PGAU GENERATE or REPORT commands.

For information about grouping PGAU commands together to roll back changes in case of failure, refer to the discussion of the PGAU "GROUP" command on page 2-21 later in this chapter.

2.3 Invoking PGAU

Before you can invoke PGAU, your Oracle Integrating Server should already be set up. If it is not, refer to the chapter on configuring your Oracle Procedural Gateway for APPC, in the *Oracle Procedural Gateway for APPC Installation and Configuration Guide*.

Before executing PGAU, you must set the ORACLE_HOME environment variable to the directory into which the gateway server was installed.

If you want to receive PGAU messages in a language other than English, set the LANGUAGE environment variable to the appropriate value. For a list of supported languages and the syntax for the LANGUAGE setting, refer to Appendix E, "National Language Support".

PGAU is invoked by entering the `pgau` command. You can run prepared scripts of PGAU commands directly from the operating system prompt by specifying a command string on the command line using the following syntax:

```
C:\> pgau @command_file
C:\> pgau command=@command_file
C:\> pgau command="@command_file"
```

The default extension is **.sql**. Use the last form if the command filename contains non-alphanumeric characters.

To perform PG DD maintenance and PL/SQL package generation, you must connect to the Oracle Integrating Server from PGAU as user PGAADMIN, using the

CONNECT command. The "PGAU Commands" section on page 2-7 discusses how to use the "CONNECT" command.

2.4 Definitions and Generation in PGAU

This version of PGAU supports the definition of remote transaction data in COBOL, entered interactively or in a file. File input is supported for the DEFINE and REDEFINE DATA commands, and standard COBOL data division macros or "copybooks" can be supplied.

PGAU and the PG DD support different versions of user data and remote transaction definitions. This facilitates alteration and testing of data formats and transactions without affecting production usage.

Multiple versions of any data or transaction definitions might exist. You must ensure that versions stored and used in the PG DD are synchronized with the remote transactions. Neither the gateway, PGAU, nor generated TIPs provide this synchronization, but they will issue messages as error conditions are detected.

Data definitions must exist before being referenced by call definitions. Call definitions must exist before being referenced by transaction definitions.

> **Note:** It is your responsibility to ensure that the data transaction definition versions that are stored and used in the PG DD are synchronized with the remote transactions. The gateway, PGAU and generated TIPs do not provide this synchronization, but issue messages as error conditions are detected.

2.5 Process to Define and Test a TIP

The general process for defining and testing a TIP for a given transaction is as follows:

1. Define input and output using COBOL data definitions.

2. Redefine the default datanames and PL/SQL variable names created by the above process (optional).

3. Define PL/SQL FUNCTION calls to be generated as part of the PL/SQL package.

4. Define a transaction that groups the above functions.

5. Generate the TIP specifications from the previously stored transaction, call, and data definitions.

6. Generate the TIP PL/SQL stored procedures.

7. Test the TIP by calling it from a high-level application.

Refer to Chapter 3, "Creating a TIP" for more information about TIPs.

2.5.1 Definition Names

Definition names are unique identifiers that you designate through PGAU. The name is a string of 1 to 30 bytes. If punctuation or white space is included, the name must be specified within double quotes.

Names are assumed to be unique within the PG DD, except when duplicate names are intentionally distinguished by a unique version number. It is your responsibility to ensure name uniqueness.

Valid characters for PG DD definition names are:

- A through Z

- a through z

- 0 through 9

- #

- $

- _ (underscore)

Note that unless defaults are overridden, transaction definition names might be PL/SQL package names, and transaction call names might be PL/SQL procedure names. Therefore, choose names that are syntactically correct for PL/SQL, making certain that they are also unique names within that system. As the user, it is your responsibility to ensure PL/SQL name compatibility.

2.5.2 Definition Versioning

The PG Data Dictionary tables contain the descriptions of transactions and data structures. There might be more than one version of a definition. Old versions are retained indefinitely.

In all PG DD operations, a definition or package is referred to by its name. That name can be qualified by a specific version number.

All version numbers:

- are supplied by Oracle Sequence Objects

- are purely numeric

- must be free from user alteration, suffixing, or prefixing

Refer to Appendix A, "Procedural Gateway for APPC Data Dictionary" and the **pgddcr8.sql** file in the **%ORACLE_HOME%\pg4appc\admin**directory for the specific names of the Oracle Sequence Objects used for version number generation.

If an explicit version number is specified, it is presumed to be the version number of an existing definition, not a new definition. Such explicit references are used when:

- generating a TIP from a specific remote transaction version

- defining a remote transaction based on a specific data version

If no explicit version is specified:

- The latest (highest number) is assumed when a definition is being referenced. This is the MAX value selected from the VERSION column for all rows with the same definition name, not the CURRVAL number.

- The next (NEXTVAL number) is assumed when a definition is being added.

Version numbers might not be contiguous. Although version numbers are always increasing, multiple versions of a given definition might skip numbers. This is because the sequence object is shared for all definitions of the same type (Transaction, Call, or Data), and sequence object NEXTVAL is not restored in event of an Oracle database transaction ROLLBACK. Thus, NEXTVAL might be assigned to a different definition before the next version of the same definition.

Examples of valid definition names:

```
DEFINE TRANSACTION|CALL|DATA
    payroll                         (new or latest definition)
    payroll_xaction                 (new or latest definition)
    payroll_xaction VERSION(3)...(an existing definition)
```

No attempt is made by PGAU to synchronize versions. Although the existence of dependent items is assured at definition time, deletion is done without reference to dependencies. For example, generating a TIP requires prior definition of the transaction, which requires prior definition of the calls, which require prior definition of the data. But nothing prevents PGAU from deleting an active data definition while a call definition still references it.

2.5.3 Keywords

All PGAU keywords can be specified in upper or lower case and are not reserved words. Reservation is not necessary because all keywords have known spelling and appear in predictable places, and because all data is delimited by parentheses, apostrophes, quotes, or blanks.

Note that all unquoted values specified by keywords are stored in the PG Data Dictionary in uppercase unless otherwise specified in the keyword description.

2.6 PGAU Commands

PGAU allows you to enter Procedural Gateway Administration commands (commands), such as DEFINE, UNDEFINE, REDEFINE, and GENERATE, in addition to normal SQL commands. The SET and SHOW commands are also implemented. In addition, the PGAU commands listed in the following section are available to you.

2.6.1 CONNECT

Purpose

This command enables you to make a connection to PGAU. Use the CONNECT command to log on to an Oracle database, optionally specifying the user ID and password in addition to the Oracle instance. The CONNECT command has the following syntax:

Syntax

```
CONNECT [username|username/password|username@connect-string|username\password@connect-string
```

Parameters

username/password is the username and password used to connect to PGAU,

and

connect-string specifies the service name of the remote database.

Refer to the *Oracle Net Services Administrator's Guide* for more information about specifying remote databases.

Examples

```
CONNECT
CONNECT SCOTT/TIGER
CONNECT SCOTT@OTHERSYS
```

CONNECT Usage Notes

- Before connecting, you must set ORACLE_SID to the database `SIDname`.

- If you want to connect to a remote database, you must set TNS_ADMIN to the full pathname of the directory in which the file `tnsnames.ora` is stored.

- You do not need to place a semi colon (;) at the end of the command.

2.6.2 DEFINE CALL

Purpose

This command creates a new version of the PL/SQL call definition in the PG Data
Dictionary.

Syntax

```
DEFINE CALL cname
    [PKGCALL(pcname)]
    [PARMS( (dname
            {IN | OUT | IN OUT}
            [VERSion(datavers)]), ...)];
```

Where Table 2–1 describes the parameters in this syntax:

Table 2–1   DEFINE CALL Parameter Descriptions

Parameter	Definition
CALL cname	is a mandatory parameter. It is the name of the call definition to be created.
PKGCALL (pcname)	is an optional parameter. It specifies the name of the PL/SQL package procedure or function by which the application might invoke the call. The default value, cname, is assumed if this operand is omitted, in which case cname must also be valid in PL/SQL syntax and unique within the transactions and TIPs referencing this call.
PARMS((dname {IN|OUT|IN OUT}[VERSION(datavers)]), . . .)	is an optional parameter. It specifies a list of previously defined data input to and output from this PL/SQL function call, and the type of each parameter (input to the call, output from, or both). The order in which the parameters are specified determines the order in which they must appear in subsequent calls to the TIP from an application.
	Each dname specifies a previously defined data item, and is mandatory. {IN | OUT | IN OUT} specifies the PL/SQL call mode of the parameter and indicates whether the dname data is sent, received, or both in the exchange with the remote transaction program. One must be chosen. VERS(datavers) is an optional specific version number of the dname data definition, if not the latest. If this operand is omitted, it is assumed that the call takes no parameters.

Examples

Refer to "Sample PGAU DEFINE CALL Statements" on page G-3 in Appendix G for
examples of DEFINE CALL commands.

DEFINE CALL Usage Notes

- Version of the CALL definition is not specified and defaults to NEXTVAL of the
  Oracle Sequence Object for CALL.

- PKGCALL and PARMS can be specified in either order.

- You need to place a semi colon (;) at the end of the command.

2.6.3 DEFINE DATA

Purpose
This command creates a new version of the data definition in the PG Data Dictionary.

Syntax
```
DEFINE DATA dname
   [PLSDNAME(plsdvar)]
   [USAGE({PASS|ASIS|SKIP})]
   [COMPOPTS ('options')]
   LANGUAGE(language)
   {(definition)|INFILE("filespec")};
```

Parameters
Table 2–2 describes the DEFINE DATA parameters:

Table 2–2   DEFINE DATA Parameter Descriptions

Parameter	Description
DATA *dname*	is a mandatory parameter. It is the name of the data definition to be created.
PLSDNAME (*plsdvar*)	is an optional parameter. It is the name of the PL/SQL variable associated with dname. It becomes the name of a PL/SQL variable if the dname item is atomic data, or a PL/SQL record variable if the dname item is aggregate data (such as a record or structure), when the TIP is generated.
USAGE({PASS|ASIS| SKIP})	is an optional parameter. It specifies the way the TIP handles the data items when exchanged in calls with the remote transaction.
	PASS indicates that the item should be translated and exchanged with the transaction.
	ASIS indicates the item is binary and, though exchanged, should not be translated.
	SKIP indicates the item should be deleted from all exchanges.
	The default value, PASS, is assumed if this parameter is omitted.
	The USAGE(NULL) keyword on DEFINE or REDEFINE DATA PGAU statements is not supported.
COMPOPTS ('options')	is an optional parameter. It specifies the compiler options used when compiling the data definition on the remote host. The only option currently supported is 'TRUNC(BIN)'. Note that the options must be enclosed in apostrophes (') or quotes ("). TRUNC(BIN) is a COBOL option that affects the way halfword and fullword binary values are handled.
	Refer to "DEFINE DATA Usage Notes" on page 2-10 for further information on this option.
LANGUAGE (*language*)	is a mandatory parameter. It specifies the name of the programming language in the supplied definition. PGAU presently supports only COBOL.

Table 2–2   (Cont.) DEFINE DATA Parameter Descriptions

Parameter	Description
(definition)	is mutually exclusive with the INFILE parameter. It is an inline description of the data. The description must be provided in COBOL syntax, as indicated above. This inline description must begin with an opening parenthesis and end with a closing parenthesis. The opening parenthesis must be the last non-blank character on the line and the COBOL data definition must start on a new line, following the standard COBOL rules for column usage and continuations. The closing parenthesis and terminating semicolon must be on a separate line following the last line of the COBOL data definition. In COBOL, the specification is a COBOL data item or structure, defined in accordance with COBOL. Margins are assumed to be standard, and explicit or implicit continuation is supported. Datanames containing invalid characters (for example, "-") for PL/SQL use are translated to their closest equivalent and truncated as required.
INFILE (*"filespec"*)	is mutually exclusive with the *(definition)* parameter. It indicates that the definition is to be read from the user disk file described by *filespec,* instead of an inline definition described by (definition).
	Note that *filespec* must be enclosed in double quotes.

Examples

Refer to "Sample PGAU DEFINE DATA Statements" in Appendix G for examples of DEFINE DATA commands.

DEFINE DATA Usage Notes

- Version of the DATA definition is not specified and defaults to NEXTVAL of the Oracle Sequence Object for DATA.

- PLSDNAME, USAGE, and LANGUAGE can be specified in any order.

- INFILE ("*filespec*") is a platform-specific designation of a disk file.

- COMPOPTS ('TRUNC(BIN)') should be used only when the remote host transaction was compiled using COBOL with the TRUNC(BIN) compiler option specified. When this option is used, binary data items defined as PIC 9(4) or PIC S9(4) can actually contain values with 5 digits, and binary data items defined as PIC 9(9) or PIC S9(9) can actually contain values with 10 digits. Without COMPOPTS ('TRUNC(BIN)'), PGAU generates NUMBER(4,0) or NUMBER(9,0) fields for these data items, resulting in possible truncation of the values.

  When COMPOPTS ('TRUNC(BIN)') is specified, PGAU generates NUMBER(5,0) or NUMBER(10, 0) fields for these data items, avoiding any truncation of the values. Care must be taken when writing the client application to ensure that invalid values are not sent to the remote host transaction.

  For a PIC 9(4) the value must be within the range 0 to 32767, for a PIC S9(4) the value must be within the range -32767 to +32767, for a PIC 9(9) the value must be within the range 0 to 2,147,483,647, and for a PIC S9(9) the value must be within the range -2,147,483,647 to +2,147,483,647. COBOL always reserves the high-order bit of binary fields for a sign, so the value ranges for unsigned fields are limited to the absolute values of the value ranges for signed fields. For further information, refer to the appropriate IBM COBOL programming manuals.

- Refer to "USAGE(PASS)" in Appendix D, "Datatype Conversions" for information about how PGAU converts COBOL statements.

- You need to place a semi colon (;) at the end of the command.

2.6.4 DEFINE TRANSACTION

Purpose

This command creates a new version of the transaction definition in the PG Data Dictionary.

Syntax

```
DEFINE TRANSACTION tname
CALL(cname [VERS(callvers)], ...
     [ENVIRONMENT(name)]
     {SIDEPROFILE(name) [LUNAME(name)] [TPNAME(name)]
                                       [LOGMODE(name)] |
     LUNAME(name) TPNAME(name) LOGMODE(name)}
     [SYNCLEVEL(0|1|2)]
     [NLS_LANGUAGE("nlsname")];
     [REMOTE_MBCS("nlsname")]
     [LOCAL_MBCS("nlsname")];
```

Parameters

Table 2–3 describes the DEFINE TRANSACTION parameters:

Table 2–3   DEFINE TRANSACTION Parameter Descriptions

Parameter	Description
TRANSACTION tname	A mandatory parameter. It is the name of the transaction definition to be created. If you do not specify a package name (TIP name) in the GENERATE statement, the transaction name specified here will become the package name, by default. In that case, the tname must be unique and must be in valid PL/SQL syntax within the database containing the PL/SQL packages.
CALL(cname [VERS(callvers)], ...)	A mandatory parameter. It specifies a list of previously defined calls (created with DEFINE CALL) which, taken together, comprise this transaction. The order in which the calls are specified here determines the order in which they are created by GENERATE, but not necessarily the order in which they might be called by an application. VERS(callvers) is an optional specific version number of the call definition, if not the latest.
	The relative position of each cname in its left-to-right sequence is the seq# column in pga_trans_calls. For example:
| | ```
CALL        (cname1, cname2,cname3)
pga_trans_calls(seq#) = 1
2        3
``` |
| ENVIRONMENT (name) | Specifies the name of the host environment for this transaction, for example, "IBM370". If this parameter is omitted, IBM370 is assumed. IBM370 is the only environment supported by this version of PGAU. |

Table 2–3   (Cont.) DEFINE TRANSACTION Parameter Descriptions

Parameter	Description
SIDEPROFILE (*name*)	**This parameter is optional for a gateway using SNA, but if omitted, the user must specify** the parameters for LUNAME, TPNAME, and LOGMODE. It specifies the name of an SNA Side Information Profile which directs the APPC connection to the transaction manager. This name can be 1 to 8 characters in length. Name values can be alphanumeric with '@', '#', and '$' characters only if unquoted. Quoted values can contain any character, and delimited by quotes ("), or apostrophes ('). Case is preserved for all values.
	This parameter is mandatory for a gateway using the TCP/IP connection. It has no comparable SNA meaning.
	You need to run the pg4tcpmap tool to map this name to the hostname, port number, subsystem ID and any other desired attribute of IMS Connect.
	This name represents a group of IMS transactions with similar IMS Connect attributes. You can re-use the same name as long as they share the same IMS Connect attributes, such as subsystem ID, TIME delay or socket type. Refer to Chapter 6, "PG4TCPMAP Commands (TCP/IP Only)" for details.
LUNAME(*name*)	**This parameter is optional on a gateway using SNA:** Overrides the LUNAME within the Side Information Profile, if the Side Information Profile was specified. It specifies the SNA Logical Unit name of the transaction manager (OLTP).
	This is either the fully-qualified LU name, 3 to 17 characters in length, or an LU alias 1 to 8 characters in length (when the SNA software on your gateway system supports LU aliases).
	Name values can be alphanumeric with '@', '#', and '$' characters and a single period '.', to delimit the network from the LU, as in netname.luname, if fully qualified. Quoted values can contain any character, and delimited by quotes ("), or apostrophes ('). Case is preserved for all values.
	This parameter is not applicable when using the TCP/IP communication protocol.
TPNAME (*name*)	**This parameter is optional on a gateway using SNA:** Overrides the TPNAME within the Side Profile, if the Side profile was specified. It specifies the partner Transaction Program name to be invoked.
	■ For CICS, this must be the CICS Transaction ID and is 1 to 4 characters in length.
	■ For IDMS, this must be the IDMS Task Code and is 1 to 8 characters in length.
	■ For IMS, this must be the IMS Transaction Name and is 1 to 8 characters in length.
	■ For AS/400, this must be specified as "library/program" and cannot exceed 21 bytes.
	Name values can be alphanumeric with '@', '#', and '$' characters only if unquoted. Quoted values can contain any character, and delimited by quotes ("), or apostrophes ('). Case is preserved for all values.
	This parameter is required for a gateway using TCP/IP support for IMS Connect. It must be the IMS Transaction Name.
	■ The IMS Transaction Name must be 1 to 8 characters in length.

Table 2–3   (Cont.) DEFINE TRANSACTION Parameter Descriptions

Parameter	Description
LOGMODE(*name*)	**This parameter is optional on a gateway using SNA:** Overrides the LOGMODE within the Side Information Profile, if the Side Information Profile was specified. It specifies the name of a VTAM logmode table entry to be used to communicate with this transaction, and is 1-8 characters in length.
	Name values can be alphanumeric with '@', '#', and '$' characters only. Values cannot be quoted. Case is not preserved and always translated to upper case.
	This parameter is not applicable when using the TCP/IP communication protocol.
SYNCLEVEL (0|1)	**This parameter is optional on a gateway using SNA:** It specifies the APPC sync level of this transaction ('0' or '1'). The default value of 0 is assumed if this operand is omitted, indicating the remote transaction program does not support synchronization. A value of '1' indicates that CONFIRM is supported.
	On a gateway using TCP/IP: The default of this parameter is '0', which is the only accepted value.
NLS_LANGUAGE (*"nlsname"*)	This is an optional parameter. The default value is "american_america.we8ebcdic37c". It is an Oracle NLS name in the language_territory.charset format. It specifies the Oracle NLS name in which the remote host data for all single-byte character set fields in the transaction are encoded. Refer to Appendix E, "National Language Support" for more information.
	Note that if you are using TCP/IP, make sure that you set this parameter to "american_america.us7ascii".
REMOTE_MBCS (*"nlsname"*)	This is an optional parameter. The default value is "japanese_japan.jal6dbcs". It is an Oracle NLS name in the language_territory.charset format. It specifies the Oracle NLS name in which the remote host data for all multi-byte character set fields in the transaction are encoded.
	Refer to Appendix E, "National Language Support" for more information.
LOCAL_MBCS (*"nlsname"*)	This is an optional parameter. The default value is "japanese_japan.jal6dbcs". It is an Oracle NLS name in the language_territory.charset format. It specifies the Oracle NLS name in which the local host data for all multi-byte character set fields in the transaction are encoded.
	Refer to Appendix E, "National Language Support" for more information.

Examples

Refer to "Sample PGAU DEFINE TRANSACTION Statement" in Appendix G for examples of DEFINE TRANSACTIONS commands.

DEFINE TRANSACTION Usage Notes:

■   NLS_LANGUAGE and the Oracle Integrating Server's LANGUAGE specify default character sets to be used for conversion of all single-byte character fields for the entire transaction. These defaults can be overridden for each SBCS field by the REDEFINE DATA REMOTE_LANGUAGE or LOCAL_LANGUAGE parameters.

- The version of the TRANSACTION definition is not specified and defaults to NEXTVAL of the Oracle Sequence Object for TRANS.

- REMOTE_MBCS and LOCAL_MBCS specify the default multi-byte character sets to be used for conversion of all DBCS or MBCS fields for the entire transaction. This default can be overridden for each DBCS or MBCS field by the REDEFINE DATA REMOTE_LANGUAGE or LOCAL_LANGUAGE parameters.

- You must place ";" at the end of the command.

2.6.5 DESCRIBE

Purpose

Use this command to describe a table, view, stored procedure, or function. If neither TABLE, VIEW, nor PROCEDURE are explicitly specified, the table or view with the specified name is described.

Syntax

The DESCRIBE command has the following syntax:

```
DESCRIBE [TABLE table|VIEW view|PROCEDURE proc|some_name]
```

Parameters

Table 2–4 describes the DESCRIBE parameter:

Table 2–4   DESCRIBE Parameter Descriptions

Parameter	Description
table	is the tablename
view	is the viewname
proc	is the procedurename

Examples

```
DESCRIBE PROCEDURE SCOTT.ADDEMP
DESCRIBE SYS.DUAL
DESCRIBE TABLE SCOTT.PERSONNEL
DESCRIBE VIEW SCOTT.PVIEW
```

DESCRIBE Usage Notes

- You do not need to place ";" at the end of the command.

2.6.6 DISCONNECT

Purpose

Use this command to disconnect from an Oracle database.

Syntax

The DISCONNECT command has the following syntax:

```
DISCONNECT
```

Parameters

None

Examples

None

DISCONNECT Usage Notes

- You do not need to place ";" at the end of the command.

2.6.7 EXECUTE

Purpose
Use this command to execute a one-line PL/SQL statement.

Syntax
The EXECUTE command has the following syntax:

```
EXECUTE pl/sql block
```

Parameters
pl/sql block is any valid pl/sql block. Refer to the *PL/SQL User's Guide and Reference* for more information.

Examples
```
EXECUTE :balance := get_balance(333)
```

EXECUTE Usage Notes
- You do not need to place ";" at the end of the command

2.6.8 EXIT

Purpose
Use this command to terminate PGAU.

Syntax
The EXIT command has the syntax:

```
EXIT
```

Parameters
None

Examples
None

EXIT Usage Notes
- You do not need to place ";" at the end of the command.
- The "quit" command is not a valid statement in PGAU.

2.6.9 GENERATE

Purpose

A PL/SQL package is built and written to the indicated output files. The PG Data Dictionary is not updated by this command.

Syntax

```
GENERATE tname
  [VERSion(tranvers)]
  [PKGNAME(pname)]
  [PGANODE(dblink_name)]
  [OUTFILE("[specpath]{specname}[.{spectype}]")]
          [,"[bodypath]{bodyname}[.{bodytype}]]")]
  [DIAGNOSE ({[TRACE({[SE] [,IT] [,QM] [,IO] [,OC] [,DD] [,TG] })]
              [PKGEX({[DC][,DR]})])]};
```

Parameters

Table 2–5 describes the GENERATE parameters:

Table 2–5   GENERATE Parameter Descriptions

Parameter	Description
tname	is a mandatory parameter. It is the transaction name defined in a DEFINE TRANSACTION statement.
VERSion(*transvers*)	is an optional parameter. It specifies which transaction definition is to be used. The VERsion parameter defaults to highest numbered transaction if not specified.
PKGNAME(*pname*)	is an optional parameter. It specifies the name of the PL/SQL package to be created. If this operand is omitted, the package name is assumed to be the same as the transaction name.
PGANODE (*dblink_name*)	is an optional parameter. It specifies the Oracle database link name to the gateway server. If this operand is omitted, "PGA" is assumed to be the *dblink_name*.
OUTFILE	is an optional parameter. If this parameter is specified, *specname* must also be specified.
specpath	is the optional directory path of the TIP specification and the TIP content documentation. It defaults to the current directory. The value must end with a backward slash (\).
specname	is the filename of the TIP specification and the TIP content documentation. It defaults to *pname*, if specified, or else **pgau**.
spectype	is the optional file extension of the TIP specification and defaults to **pkh**.
bodypath	is the optional directory path of the TIP body. It defaults to *specpath*, if specified, or else the current directory. The value must end with a backward slash (\).
bodyname	is the optional file name of the TIP body. It defaults to *specname*, if specified, or else *pname*, if specified, or else *pgau*. If *bodyname* defaults to *specname*, the leftmost period of *specname* is used to extract *bodyname* when *specname* contains multiple qualifiers.

Table 2–5   (Cont.) GENERATE Parameter Descriptions

Parameter	Description
bodytype	is the optional file extension of the TIP body and defaults to **pkb**.
	The TIP Content output path defaults to specpath or else the current directory. The file id defaults to *specname*, if specified, or else *pname*, if specified, or else **pgau,** and always has an extension of **doc**.
	Refer to the "GENERATE Usage Notes:" on page 2-20 for more examples, and Appendix F, "Tip Internals" for more information.
DIAGNOSE	is an optional parameter with two options, TRACE and PKGEX.
TRACE	specifies that an internal trace of the execution of PGAU is written to output file **pgau.trc** in the user's current directory.
	TRACE suboptions are delimited by commas.
	Trace messages are provided as a diagnostic tool to Oracle Support Services and other Oracle representatives to assist them in diagnosing customer problems when generating TIPs. They are part of an Oracle reserved function for which the usage, interface, and documentation might change without notice at Oracle's sole discretion. This information is provided so customers might document problem symptoms.
	■ SE - Subroutine Entry/Exit
	Messages are written tracing subroutine name and arguments upon entry, and subroutine name and conditions at exit.
	■ IT - Initialization/Termination
	Messages are written tracing PGAU initialization and termination functions.
	■ QM - Queue Management
	Messages are written tracing control block allocation, queuing, searching, dequeuing, and deletion.
	■ IO - Input/Output
	Messages are written tracing input, output, and control operations for **.dat** input files and **.wrk** and package output files.
	■ DD - PG DD Definitions
	Messages are written tracing the loading of transaction, call, data parameter, field, attribute, environment and compiler information from the PG DD.
	■ OC - Oracle Calls
	Messages are written tracing the Oracle UPI call results for SQL statement processing and SELECTs from the PG DD.
	■ TG - TIP Generation
	Messages are written tracing steps completed in TIP Generation, typically a record for each call, parameter, and data field for which a PL/SQL code segment has been generated.

Table 2–5   (Cont.) GENERATE Parameter Descriptions

Parameter	Description
PKGEX	causes additional TIP execution time diagnostic logic to be included within the generated PL/SQL package.
	PKGEX suboptions are delimited by commas.
	■ DC - Data Conversion
	Enables runtime checking of repeating group limits and the raising of exceptions when such limits are exceeded.
	Enables warning messages to be passed from the UTL_PG data conversion functions:
	■ NUMBER_TO_RAW
	■ RAW_TO_NUMBER
	■ MAKE_NUMBER_TO_RAW_FORMAT
	■ MAKE_RAW_TO_NUMBER_FORMAT
	The additional logic checks for the existence of warnings and, if present, causes them to be displayed using DBMS_OUTPUT calls.
	The TIP generation default is to suppress such warnings on the presumption that a TIP has been tested with production data and that data conversion anomalies either do not exist, or are known and to be ignored.
	If errors occur which might be due to data conversion problems, regeneration of the TIP with PKGEX(DC) enabled might provide additional information.
	Note: A runtime switch is also required to execute the warning logic. PKGEX(DC) only causes the warning logic to be included in the TIP. Refer to "Controlling TIP Runtime Conversion Warnings" on page 8-9 in Chapter 8, "Problem Determination".
	Additional messages are written to a named pipe for tracing the data conversion steps performed by the TIP as it executes.
	This option only causes the trace logic to be generated in the TIP. It must be enabled when the TIP is initialized.
	Refer to"Controlling TIP Runtime Conversion Warnings" in Chapter 8, "Problem Determination" for more information.
	■ DR - Dictionary Reference
	PL/SQL single line Comments are included in TIPs which reference the PG DD id numbers for the definitions causing the TIP function calls and conversions.

Examples

Refer to "Sample PGAU GENERATE Statement" in Appendix G for examples of GENERATE commands.

The following list describes the TIP output file ids resulting from various combinations of GENERATE parameters:

```
GENERATE tran
   GENERATE tran OUTFILE("dirpath\")
   dirpath\pgau.pkh - TIP specification
   dirpath\pgau.doc - TIP Content
   dirpath\pgau.pkb - TIP body
GENERATE tran PKGNAME tipname OUTFILE("dirpath\")
```

```
    dirpath\tipname.pkh - TIP specification
    dirpath\tipname.doc - TIP Content
    dirpath\tipname.pkb - TIP body
GENERATE tran PKGNAME tipname OUTFILE("sdirpath\",
 "bdir\path\")
    sdirpath\pgau.pkh  - TIP specification
    sdirpath\pgau.doc  - TIP Content
    bdir\path\pgau.pkb - TIP body
GENERATE tran PKGNAME tipname OUTFILE("spath\sname","bpath\bname")
    spath\sname.pkh  - TIP specification
    spath\sname.doc  - TIP Content
    bpath\bname.pkb  - TIP body
GENERATE tran PKGNAME tipname
    tipname.pkh - TIP specification
    tipname.doc - TIP Content
    tipname.pkb - TIP body
GENERATE tran PKGNAME tipname OUTFILE("fileid")
    fileid.pkh - TIP specification
    fileid.doc - TIP Content
    fileid.pkb - TIP body
GENERATE tran PKGNAME tipname OUTFILE("specname","bodyname")
    specname.pkh - TIP specification
    specname.doc - TIP Content
    bodyname.pkb - TIP body
GENERATE tran PKGNAME tipname OUTFILE("specname.spectype")
    specname.spectype - TIP specification
    specname.doc      - TIP Content
    specname.pkb      - TIP body
GENERATE tran PKGNAME tipname OUTFILE("sname.stype","bname")
    sname.stype - TIP specification
    sname.doc   - TIP Content
    bname.pkb   - TIP body
GENERATE tran PKGNAME tipname
 OUTFILE("sname.stype","bname.btype")
    sname.stype - TIP specification
    sname.doc   - TIP Content
    bname.btype - TIP body
GENERATE tran PKGNAME tipname
 OUTFILE("spath\sname.sext","bpath\bname.bext")
    spath\sname.sext  - TIP specification
    spath\sname.doc   - TIP Content
    bpath\bname.bext  - TIP body
```

GENERATE Usage Notes:

- All PGAU GENERATE trace messages are designated PGU-39*nnn*. Refer to the **%ORACLE_HOME%\pg4appc\mesg\pguus.msg** file for further information on any given trace message.

- The **pgau.trc** trace message output file is overwritten by the next invocation of GENERATE, regardless of the TRACE specification. A trace header record is always written to the **pgau.trc** file. If a particular trace file is to be saved, it must be copied to another file before the next invocation of GENERATE.

- TRACE options can be specified in any order or combination, and can also be specified with PKGEX operand on the same GENERATE statement.

- You must place ";" at the end of the command.

2.6.10 GROUP

Purpose

Multiple PGAU commands can be grouped together for purposes of updating the PG DD, and for rolling back all changes resulting from the commands in the group, if any one statement fails.

No COMMIT processing is performed, even if all commands within the group succeed. You perform the COMMIT either by coding COMMIT commands in the PGAU script, outside of GROUPs, or by issuing COMMIT interactively to PGAU.

PGAU issues a savepoint ROLLBACK to conditions before processing the group if any statement within the group fails.

Syntax

```
GROUP (pgaustmt1; pgaustmt2; ... pgaustmtN);
```

Parameters

pgaustmtN: is a PGAU DEFINE, REDEFINE, or UNDEFINE statement

Examples

```
GROUP (
      DEFINE DATA EMPNO
            PLSDNAME (EMPNO)
            USAGE (PASS)
            LANGUAGE (IBMVSCOBOLII)
            (
            01 EMP-NO PIC X(6).
            );

      DEFINE CALL DB2IMAIN
            PKGCALL (PGADB2I_MAIN)
            PARMS ( (EMPNO      IN ),
                    (EMPREC     OUT) );

      DEFINE TRANSACTION DB2I
         CALL (   DB2IMAIN,
                  DB2IDIAG   )
         SIDEPROFILE(CICSPROD)
         TPNAME(DB2I)
         LOGMODE(ORAPLU62)
   SYNCLEVEL(0)
   NLS_LANGUAGE("AMERICAN_AMERICA.WE8EBCDIC37C");

GENERATE DB2I
   PKGNAME(PGADB2I)
   OUTFILE("pgadb2i"););
```

GROUP Usage Notes:

- No non-PGAU commands, such as ORACLE or SQL, can be placed inside the parentheses delimiting the group.

- A PGAU script can contain multiple GROUPs. Each GROUP can be interspersed with SQL commands, such as COMMIT or SELECT or with PGAU commands, such as GENERATE or REPORT.

- The first failing PGAU statement within the group causes a savepoint ROLLBACK to conditions at the beginning of the group. All subsequent commands within the group are flushed and not examined. PGAU execution resumes with the statement following the group. If that statement is a COMMIT, all PG DD changes made before the failing group are committed.

- You must place ";" at the end of the command.

2.6.11 HOST

Purpose
Use this command to execute an operating system command without exiting PGAU.

Syntax
The HOST command has the syntax:

```
HOST host_command
```

Parameters
host_command is any valid MS-DOS command.

Examples
```
HOST  vi log.out
HOST  ls -la
HOST  pwd
```

HOST Usage Notes
- Using the HOST command starts a new command shell under which to execute the specified operating system command. This means that any environment changes caused by the executed command affect only the new command shell started by PGAU, and not the command shell under which PGAU itself is executing. For example, a "cd" command executed by the HOST command does not change the current directory in the PGAU execution environment.

- You do not need to place ";" at the end of the command.

2.6.12 PRINT

Purpose
Use this command to print the value of a variable defined with the VARIABLE command.

Syntax
The PRINT command has the syntax:

```
PRINT varname
```

Parameters
varname is a variable name which is defined by a variable command.

Examples
```
PRINT ename
PRINT balance
```

PRINT Usage Notes
- You do not need to place ";" at the end of the command.

2.6.13 REDEFINE DATA

Purpose

The existing data definition in the PG Data Dictionary is modified. PG DD column values for DATA#, FLD#, and POS remain the same for redefined data items. This permits existing CALL and DATA definitions to utilize the redefined data. REDEFINE does not create a different version of a data definition and the version number is not updated.

Syntax

```
REDEFINE DATA dname
    [VERSion(datavers)]
    [PLSDNAME(plsdvar)]
    [FIELD(fname) [PLSFNAME(plsfvar)]]
    [USAGE({PASS|ASIS|SKIP})]
    [COMPOPTS ('options')]
    [REMOTE_LANGUAGE("nlsname")]
    [LOCAL_LANGUAGE("nlsname")]
    LANGUAGE(language)
    <(definition) | INFILE("filespec")>;
```

Parameters

Table 2–6 describes the REDEFINE DATA parameters:

Table 2–6   REDEFINE DATA Parameter Descriptions

Parameter	Description
DATA *dname*	is a mandatory parameter. It is the name of the data definition to be modified.
VERSion(*datavers*)	is an optional parameter. It specifies which version of *dname* is to be modified, and if specified, the updated *dname* information retains the same version number; a new version is not created. It defaults to the highest version if omitted.
PLSDNAME(*plsdvar*)	is an optional parameter. It is the name of the PL/SQL variable associated with the *dname* above. It becomes the name of a PL/SQL variable if the *dname* item is atomic data, or a PL/SQL record variable if the *dname* item is aggregate data (such as a record or structure), when the TIP is generated. This name replaces any *plsdvar* name previously specified by DEFINE DATA into pga_data(plsdvar) of the PG DD.
FIELD(*fname*)	is an optional parameter. It is the name of a field or group within the *dname* item, if aggregate data is being redefined (such as changing a field within a record).
PLSFNAME (*plsfvar*)	is an optional parameter if FIELD is specified. It is the name of the PL/SQL variable associated with the *fname* above. It becomes the name of a PL/SQL field variable within a PL/SQL record variable when the TIP is generated. This name replaces any *plsfvar* name previously specified by REDEFINE DATA into pga_data(*plsfvar*) of the PG DD.

Table 2–6   (Cont.) REDEFINE DATA Parameter Descriptions

Parameter	Description
USAGE({PASS|ASIS |SKIP})	is optional. If omitted, the last usage specified is retained. It specifies the way the TIP handles the data items when exchanged in calls with the remote transaction: ■ PASS indicates that the item should be translated and exchanged with the transaction. ■ ASIS indicates the item is binary and, though exchanged, should not be translated. ■ SKIP indicates the item should be deleted from all exchanges. If specified, all affected fields are updated with the same USAGE value. (Refer to the notes pertaining to single or multiple field redefinition, under FIELD). The USAGE(NULL) keyword on DEFINE or REDEFINE DATA PGAU statements is not supported.
COMPOPTS ('options')	is optional. If omitted, the last options specified are retained. If specified as a null string ('') then the last options specified are removed. If a non-null value is specified, then the last options specified are all replaced with the new options. The only option currently supported is 'TRUNC(BIN)'. Note that the options must be enclosed in apostrophes (') or quotes ("). TRUNC(BIN) is a COBOL option that affects the way halfword and fullword binary values are handled. Refer to "REDEFINE DATA Usage Notes:" on page 2-26 for further information on this option.
REMOTE_LANGUAGE ("nlsname")	is an optional parameter. The default value is "american_america.we8ebcdic37c" or as overridden by the NLS_LANGUAGE parameter of DEFINE TRANSACTION. It is an Oracle NLS name in the language_territory.charset format. It specifies the Oracle NLS name in which the remote host data for the specific character field being redefined is encoded. The field can be single byte or multi-byte character data. Refer to Appendix E, "National Language Support" for more information.
LOCAL_LANGUAGE ("nlsname")	is an optional parameter. The default value is initialized from the LANGUAGE variable of the local Oracle server when the TIP executes. It is an Oracle NLS name in the language_territory.charset format. It specifies the Oracle NLS name in which the local Oracle data for the specific character field being redefined is encoded. The field can be single byte or multi-byte character data. Refer to Appendix E, "National Language Support" for more information.
LANGUAGE ("language")	is a mandatory parameter if definition input is specified. It specifies the name of the programming language in the supplied definition. PGAU presently supports only COBOL.

Table 2–6 (Cont.) REDEFINE DATA Parameter Descriptions

Parameter	Description
(definition)	is mutually exclusive with the INFILE parameter. It is an inline description of the data. The description must be provided in COBOL syntax. This inline description must begin with an opening parenthesis and end with a closing parenthesis. The opening parenthesis must be the last non-blank character on the line and the COBOL data definition must start on a new line, following the standard COBOL rules for column usage and continuations. The closing parenthesis and terminating semicolon must be on a separate line following the last line of the COBOL data definition. If in COBOL, the specification is a COBOL data item or structure, defined according to the rules for COBOL. Margins are assumed to be standard, explicit or implicit continuation is supported. Datanames containing invalid characters (for example, "-") for PL/SQL use are translated to their closest equivalent and truncated as required.
INFILE (*"filespec"*)	is mutually exclusive with the *(definition)* parameter. It indicates that the definition is to be read from the operating system file described by *filespec*, instead of an inline definition described by *(definition)*. Note that "*filespec*" must be enclosed in double quotes.

Examples

Refer to "Sample PGAU REDEFINE DATA Statements" in Appendix G for examples of REDEFINE commands.

REDEFINE DATA Usage Notes:

- Specification of either PLSDNAME, FIELD, or PLSFNAME allows redefinition of a single data item's names while the *(definition)* parameter redefines the named data item's content.

- The presence of FIELD denotes only a single data field (single PG DD row uniquely identified by *dname*, *fname*, and *version*) is updated. The absence of FIELD denotes that multiple data fields (multiple PG DD rows identified by *dname* and *version*) are updated or replaced by the definition input.

- REMOTE_LANGUAGE and LOCAL_LANGUAGE override the character sets used for conversion of any individual SBCS, DBCS, or MBCS character data field.

- LANGUAGE *(language)* and *(definition)*|INFILE(*"filespec"*) are mandatory as a group. If data definitions are to be supplied, then a LANGUAGE parameter must be specified and then either the inline definition or INFILE must also be specified.

- The presence of *(definition)* | INFILE(*"filespec"*) denotes that multiple data fields (those PG DD rows identified by *dname* and version) are updated or replaced by the definition input. Fewer, equal, or greater numbers of fields might result from the replacement.

- INFILE(*"filespec"*) is a platform-specific designation of a disk file.

- COMPOPTS ('TRUNC(BIN)') should be used only when the remote host transaction was compiled using COBOL with the TRUNC(BIN) compiler option specified. When this option is used, binary data items defined as PIC 9(4) or PIC S9(4) can actually contain values with 5 digits, and binary data items defined as PIC 9(9) or PIC S9(9) can actually contain values with 10 digits. Without COMPOPTS ('TRUNC(BIN)'), PGAU generates NUMBER(4,0) or NUMBER(9,0)

fields for these data items, resulting in possible truncation of the values. When COMPOPTS ('TRUNC(BIN)') is specified, PGAU generates NUMBER(5,0) or NUMBER(10, 0) fields for these data items, avoiding any truncation of the values. Care must be taken when writing the client application to ensure that invalid values are not sent to the remote host transaction. For a PIC 9(4) the value must be within the range 0 to 32767, for a PIC S9(4) the value must be within the range -32767 to +32767, for a PIC 9(9) the value must be within the range 0 to 2,147,483,647, and for a PIC S9(9) the value must be within the range -2,147,483,647 to +2,147,483,647. COBOL always reserves the high-order bit of binary fields for a sign, so the value ranges for unsigned fields are limited to the absolute values of the value ranges for signed fields. For further information, refer to the appropriate IBM COBOL programming manuals.

- Refer to "USAGE(PASS)" in Appendix D, "Datatype Conversions" for information about how PGAU converts COBOL statements.

- You must place ";" at the end of the command.

2.6.14 REM

Purpose

Comments can either be introduced by the REM command or started with the two-character sequence /* and terminated with the two-character sequence */.

Use the REM command to start a Comment line.

Syntax

The REM command has the syntax:

```
REM Comment
```

Parameters

Comment is any strings.

Examples

```
REM This is a Comment....
```

REM Usage Notes

You do not need to place ";" at the end of the command.

2.6.15 REPORT

Purpose

This command produces a report of selected data from the PG Data Dictionary. Selection criteria might determine that:

- a single TRANSACTION, CALL, or DATA entity (with or without an explicit version) is reported, or

- that all TRANSACTION, CALL, or DATA entities with a given name be reported or that all entities in the PG DD be reported, or

- that all invalid TRANSACTIONs or CALLs and all unreferenced CALLs, or DATA entities be reported.

Syntax

```
REPORT { { TRANSACTION tname | CALL cname | DATA dname } [VERSION(ver1...)]
         | ALL { TRANSACTIONS [tname] | CALLS [cname] | DATA [dname] } }
         [WITH { CALLS | DATA | DEBUG } ... ]
         | ISOLATED;
```

Parameters

Table 2–7 describes the REPORT parameter:

Table 2–7   REPORT Parameters Descriptions

Parameter	Description
TRANSACTION *tname*	Reports the PG DD contents for the latest or selected versions of the transaction *tname*.
CALL *cname*	Reports the PG DD contents for the latest or selected versions of the call *cname*.
DATA *dname*	Reports the PG DD contents for the latest or selected versions of the data *dname*.
VERSION(*ver1*, [*ver2 ...*])	Reports selected versions of the indicated entry and is mutually exclusive with ALL.
ALL TRANSACTIONS [*tname*]	Reports the PG DD contents for all existing versions of every transaction entry or optionally a specific transaction *tname*, and is mutually exclusive with TRANSACTION.
ALL CALLS [*cname*]	Reports the PG DD contents for all existing versions of every call entry or optionally a specific call *cname*, and is mutually exclusive with CALL.
ALL DATA [*dname*]	Reports the PG DD contents for all existing versions of every data entry or optionally a specific data *dname*, and is mutually exclusive with DATA.
WITH CALLS	Reports call entries associated with the specified transactions.
WITH DATA	Reports data entries associated with the specified calls, and when specified for transactions, implies WITH CALLS.
WITH DEBUG	Reports PG DD column values for tran#, call#, parm#, data#, and attr# as appropriate, depending on the type of items being reported.
	This report is useful with TIPs generated with PG DD Diagnostic references. Refer to the GENERATE DIAGNOSE PGEX(OR) option for more information.

Table 2–7   (Cont.) REPORT Parameters Descriptions

Parameter	Description
ISOLATED	Mutually exclusive with all other parameters. All unreferenced CALL and DATA entries are reported along with TRANSACTIONs that reference missing CALLs and DATA and CALLs that reference missing DATA.

Examples

Refer to "Sample PGAU REPORT Output" in Appendix G for sample REPORT commands.

REPORT Usage Notes:

- Report output is to the terminal and can be spooled, saved, and printed.

- Data reports are formatted according to their original compiler language, and preceded by a PGAU DEFINE DATA command which defines the data to the PG DD.

- Call and Transaction reports are formatted as PGAU DEFINE CALL or TRANSACTION commands (also called "statements"), which effectively define the entry to the PG DD.

- The following command reports the single most recent data definition specified by data name dname, or optionally, for those specific versions given.

```
REPORT DATA dname;
REPORT DATA dname VERSION(version#1,version#2);
```

  This command reports all data definitions specified by data name dname:

```
REPORT ALL DATA dname;
```

- The following command reports the single most recent call definitions specified by call name cname, or optionally for those specific versions given.

```
REPORT CALL cname;
REPORT CALL cname VERSION(version#1,version#2) WITH DATA;
```

  This command reports all call definitions specified by call name cname:

```
REPORT ALL CALLS cname WITH DATA;
```

  This command reports all call definitions in the PG DD:

```
REPORT ALL CALLS WITH DATA;
```

  When WITH DATA is specified, all the data definitions associated with each selected call are also reported. The data definitions precede each corresponding selected call in the report output.

- The following command reports the single most recent transaction definitions specified by transaction name tname, or optionally for those specific versions given.

```
REPORT TRANSACTION tname
REPORT TRANSACTION tname VERSION(version#1,version#2)
WITH DATA WITH CALLS;
```

This command reports all transaction definitions specified by transaction name tname:

```
REPORT ALL TRANSACTIONS tname WITH DATA WITH CALLS;
```

This command reports all transaction definitions in the PG DD:

```
REPORT ALL TRANSACTIONS WITH DATA WITH CALLS;
```

When WITH CALLS option is specified, all call definitions associated with each selected transaction are also reported (the call definitions precede each corresponding selected transaction in the report output).

When WITH DATA is specified, all the data definitions associated with each selected call are also reported (the data definitions precede each corresponding selected call in the report output).

For transaction reports, specification of WITH DATA implies specification of WITH CALL.

■ The following command reports any unreferenced CALL or DATA definitions. It also reports any TRANSACTION or CALL definitions that reference missing CALL or DATA definitions respectively.

```
REPORT ISOLATED;
```

■ The following command reports all definitions in the PG DD.

```
REPORT ALL;
```

Data definitions are reported, followed by their associated call definitions, followed by the associated transaction definition.

This sequence is repeated for every defined call and transaction in the PG DD.

■ You must place ";" at the end of the command.

2.6.16 SET

Parameters

Table 2–8 describes the SET parameters:

Table 2–8   SET Parameter Descriptions

Parameter	Description
ARRAYSIZE *[n]*	Sets the number of rows fetched at a time from the database. The default is 20.
CHARWIDTH *[n]*	Sets the column display width for CHAR data. If entered with no argument, it returns the setting to 9, which is the default.
DATEWIDTH	Sets the column display width for DATE data. If entered with no argument, it returns the setting to 9, which is the default.
ECHO {ON|OFF}	Sets echoing of commands entered from command files to ON or OFF. The default is OFF.
FETCHROWS *[n]*	Sets the number of rows returned by a query. This is useful with ordered queries for finding a certain number of items in a category, the top ten items for example. It is also useful with unordered queries for finding the first *n* records that satisfy a certain criteria.
LONGWIDTH *[n]*	Sets the column display width for LONG data. If entered with no argument, it returns the setting to 80, which is the default.
MAXDATA *[n]*	Sets the maximum data size. It indicates the maximum data that can be received in a single fetch during a SELECT command. The default is 20480 bytes (20K).
NUMWIDTH *[n]*	Sets the column display width for NUMBER data. If entered with no argument, it returns the setting to 10, which is the default.
SERVEROUTPUT {OFF|ON [SIZE *n*|*n*]}	Sets debugging output from stored procedures that use DBMS_OUTPUT PUT and PUT_LINE commands to ON or OFF. You can specify the size in bytes of the message buffer using SIZE *n*. The size specified is the total number of bytes of all messages sent that can be accumulated at one time. The minimum is 2000 bytes. If the buffer fills before calls to the get-message routines make room for additional message bytes, an error is returned to the program sending the message. SERVEROUTPUT with no parameters is the same as SERVEROUTPUT ON.
STOPONERROR {ON|OFF}	Indicates whether execution of a command file should stop if an error occurs. Specifying OFF disables STOPONERROR.
TERMOUT {ON|OFF}	Enables or disables terminal output for SQL commands. It is useful for preventing output to the terminal when spooling output to files. The default is OFF, which disables terminal output.
TIMING {ON|OFF}	Enables or disables display of parse, execute, and fetch times (both CPU and elapsed) for each executed SQL statement. The default is OFF, which disables the TIMING display.

Examples

```
PGAU> set arraysize 30

PGAU> set CHARWIDTH
```

SET Usage Notes

■ You do not need to place ";" at the end of the command.

2.6.17 SHOW

Parameters

Table 2–9 describes the SHOW parameters:

Table 2–9   SHOW Parameter Descriptions

Parameters	Description
ALL	Shows all valid SET parameters
ARRAYSIZE	Shows the number of rows fetched at a time from the database.
CHARWIDTH	Shows the column display width for CHAR data.
DATEWIDTH	Shows the column display width for DATE data.
ECHO	Shows echoing of commands entered from command files to ON or OFF.
FETCHROWS	Shows the number of rows returned by a query.
LONGWIDTH	Shows the column display width for LONG data.
MAXDATA	Shows the maximum data size.
NUMWIDTH	Shows the column display width for NUMBER data.
SERVEROUTPUT	Shows debugging output from stored procedures that use DBMS_OUTPUT PUT and PUT_LINE commands.
STOPONERROR	Indicates whether execution of a command file should stop if an error occurs.
TERMOUT	Shows whether the terminal output for SQL commands is enabled or disabled.
TIMING	Shows whether display of parse, execute, and fetch times (both CPU and elapsed) for each executed SQL statement is enabled or disabled.
VAR	Is the same as the PRINT command; in addition, it shows all variables and their datatypes.

Examples

Note that when you issue a SET command, there will be no output if it is successful. If you want to check whether your statement was executed successfully, issue a SHOW command like the following:

```
PGAU> show arraysize
Arraysize                30

PGAU> show CHARWIDTH
Charwidth                80

PGAU> show all
Instance                 local
Spool                    OFF
Timing                   OFF
Termout                  ON
Echo                     OFF
Stoponerror              OFF
```

```
Maxdata                        20480
Arraysize                      20
Fetchrows                      100

Numwidth                       10
Charwidth                      80
Longwidth                      80
Datewidth                      9
ServerOutput                   OFF
```

SHOW Usage Notes

■   You do not need to place ";" at the end of the command.

2.6.18 SPOOL

Purpose

Use this command to specify a filename that captures PGAU output. All output is directed to the terminal unless TERMOUT is off.

Syntax

The SPOOL command has the syntax:

```
SPOOL [filename|OFF]
```

Parameters

If a simple filename is specified, with no periods, then **.log** is appended to the filename.

filename is where the output of your executed commands is placed.

Examples

```
SPOOL log.outfile
SPOOL out
SPOOL OFF
```

SPOOL Usage Notes

- You do not need to place ";" at the end of the command.

2.6.19 UNDEFINE CALL

Purpose

Use this command to remove an occurrence of the CALL definition from PG DD.

Syntax

```
UNDEFINE CALL cname          [VERSion(callvers|ALL)];
```

Parameters

Table 2–10 describes the UNDEFINE CALL parameters:

Table 2–10   UNDEFINE CALL Parameter Descriptions

Parameter	Description
CALL *cname*|	A mandatory parameter. It specifies the name associated with the item to be dropped; if no version is specified only the latest (highest numbered) version is removed.
VERSion(*{datavers*| *callvers*| *transvers*|ALL})	An optional parameter. It specifies which singular version of a definition is to be removed, or if ALL, then all definitions are removed, for the given definition named. The default of the highest numbered version of the named definition is assumed if VERSION is omitted.

Examples

Refer to "Sample PGAU UNDEFINE Statements" in Appendix G for examples of UNDEFINE CALL commands.

UNDEFINE CALL Usage Notes:

- Removing definitions only prevents PL/SQL packages from being subsequently generated. TIPs can still be recreated if the **.pkh** and **.pkb** specification files exist and those previous TIPS can be invoked if they remain in the database of the Oracle Integrating Server. Whether such TIPs execute successfully depends on whether the corresponding remote transaction programs are still active.

- Remove a CALL definition only after all TRANSACTIONs which reference it are removed. No integrity checking is done.

- You must place ";" at the end of the command.

2.6.20 UNDEFINE DATA

Purpose

Use this command to remove an occurrence of the DATA definition in the PG Data
Dictionary.

Syntax

```
UNDEFINE DATA dname        [VERSion(datavers|ALL)];
```

Parameters

Table 2–11 describes the UNDEFINE DATA parameters:

Table 2–11   UNDEFINE DATA Parameter Descriptions

Parameter	Description
DATA *dname* |	A mandatory parameter. It specifies the name associated with the item to be dropped. If no version is specified, only the latest (highest numbered) version is removed.
VERSion(*{datavers* | *callvers* | *transvers* | *ALL})*	An optional parameter. It specifies which singular version of a definition is to be removed, or if ALL, then all definitions are removed, for the given definition named. The default of the highest numbered version of the named definition is assumed if VERSION is omitted.

Examples

Refer to "Sample PGAU UNDEFINE Statements" in Appendix G for examples of
UNDEFINE DATA commands.

UNDEFINE DATA Usage Notes

- Removing definitions only prevents PL/SQL packages (TIPs) from being
  subsequently generated. Previously generated TIPs can still be recreated if the
  .pkh and **.pkb** specification files remain in existence. Previously created TIPs can
  still be invoked if they remain in the database of the Oracle Integrating Server.
  Whether such TIPs execute successfully depends on whether the corresponding
  remote transaction programs are still active.

- Remove a DATA definition only after all CALLs and all TRANSACTIONs which
  reference it are removed. No integrity checking is done.

- You must place ";" at the end of the command.

2.6.21 UNDEFINE TRANSACTION

Purpose

This command removes an occurrence of the TRANSACTION definition in the PG Data Dictionary.

Syntax

```
UNDEFINE TRANSACTION tname [VERSion(tranvers|ALL)];
```

Parameters

Table 2–12 describes the UNDEFINE TRANSACTION parameters:

Table 2–12   UNDEFINE TRANSACTION Parameter Descriptions

Parameter	Description
TRANSACTION *tname}*	Mandatory parameter. It specifies the name associated with the item to be dropped. If no version is specified, only the latest (highest numbered) version is removed.
VERSion({*datavers*| *callvers*| *transvers*|*ALL*})	Optional parameter. It specifies which singular version of a definition is to be removed, or if ALL, then all definitions are removed, for the given definition named. The default of the highest numbered version of the named definition is assumed if VERSION is omitted.

Examples

Refer to "Sample PGAU UNDEFINE Statements" in Appendix G for examples of UNDEFINE TRANSACTION commands.

UNDEFINE TRANSACTION Usage Notes

- Removing definitions only prevents PL/SQL packages from being subsequently generated. TIPs can still be recreated if the **.pkh** and **.pkb** specification files remain in existence. Previously created TIPs can be invoked if they remain in the database of the Oracle Integrating Server. Whether such TIPs execute successfully depends on whether the corresponding remote transaction programs are still active.

- A TRANSACTION definition can be removed at any time.

- You must place ";" at the end of the command.

2.6.22 VARIABLE

Purpose

Use this command to declare a bind variable for use in the current session with the EXECUTE or PRINT command, or for use with a PL/SQL block.

Syntax

The VARIABLE command has the syntax:

```
VARIABLE name type
```

Parameters

Table 2–13 describes the VARIABLE parameters.

Table 2–13   VARIABLE Parameter Descriptions

Parameter	Description
name	Is a variable name.
type	Is the variable datatype

Examples

```
VARIABLE balance NUMBER
VARIABLE emp_name VARCHAR2
```

VARIABLE Usage Notes

- You do not need to place ";" at the end of the command.

3

Creating a TIP

This chapter shows in detail how you can define, generate and compile a Transaction Interface Package (TIP). It assumes that a remote host transaction program (RTP) already exists. This transaction program has operational characteristics that dictate how the TIP is defined and how the TIP is used by the client application.

This chapter contains the following sections:

- "Granting Privileges for TIP Creators" on page 3-2
- "Evaluating the RHT" on page 3-2
- "Defining and Generating the TIP" on page 3-7
- "Compiling the TIP" on page 3-8
- "TIP Content Documentation (tipname.doc)" on page 3-8

The following steps create a TIP for use with a remote host transaction (RHT):

- evaluating the RHT

- preparing the PGAU statements

- defining and generating the TIP

- compiling the TIP

This chapter also discusses the generated TIP content file.

3.1 Granting Privileges for TIP Creators

Every TIP developer requires access to the following PL/SQL packages, which are shipped with the Oracle server:

- **DBMS_PIPE** in **%ORACLE_HOME%\rdbms\admin**

- **UTL_RAW** in **%ORACLE_HOME%\rdbms\admin**

- **UTL_PG** in **%ORACLE_HOME%\rdbms\admin**

If anyone other than user PGAADMIN will be developing TIPs, they will need explicit grants to perform these operations. Refer to the "Optional Configuration Steps" section in the configuration chapter appropriate to your communication protocol in the *Oracle Procedural Gateway for APPC Installation and Configuration Guide* for more information about private and public grants.

3.2 Evaluating the RHT

Follow the steps below to identify and become familiar with your remote host transaction data exchanges.

3.2.1 Identify the Remote Host Transaction

You must first identify the RHT data exchange steps. These are the send and receive calls embedded within the RHT program.

If your gateway is using the SNA communication protocol:

The RHT data exchange steps are identified under the following languages:

- You may use COBOL for:
  - CICS
  - IMS

- You may use IBM 370 Assembler for:
  - CICS
  - IDMS
  - IMS

- You may use IBM REXX for:
  - CICS
  - IDMS
  - IMS
  - z/OS

If your gateway is using the TCP/IP communication protocol:

IMS is the only OLTP that is supported when the gateway is using TCP/IP support for IMS Connect. The RHT programs must use embedded I/O PCB function calls. The function is identified only under the COBOL and Assembler languages.

3.2.2 PGAU DEFINE CALL Command

Make a call list of every data exchange. This list dictates a series of PGAU DEFINE CALL statements. Refer to "DEFINE CALL" in Chapter 2, "Procedural Gateway Administration Utility" for more information about this PGAU command.

The three important parameters that you will use for each call are:

- `cname`: the name of the call definition to be created;

- `dname`: the name of the data structure to be exchanged; and

- whether it is send (`OUT`) or receive (`IN`)

RHT send corresponds to a TIP OUT and RHT receive corresponds to a TIP IN.

If your communication protocol is SNA: Refer to Section 4.6.2.1, "Flexible Call Sequence" on page 4-14 for more information about PGAU DEFINE CALL commands.

If your communication protocol is TCP/IP: Refer to Section 7.3.2.1, "Flexible Call Sequence" on page 7-7 for more information about PGAU DEFINE CALL commands.

PGAU call entries are only defined once, so eliminate any duplicates.

This call list defines the TIP function calls, not the order in which they are used. Note that the order in which each call is made is a behavior of the transaction and dictates the order of calls made by the high-level application to the TIP, which then calls the RHT through the Procedural Gateway server. While this calling sequence is critical to maintaining the synchronization between the application and the RHT, the TIP is only an access method for the application and has no knowledge of higher level sequencing of calls.

3.2.3 PGAU DEFINE DATA Command

For each call in the RHT call list, identify the RHT data structures being sent or received in the call buffers.

Make a data list of every such structure. This list dictates a series of PGAU DEFINE DATA statements.

The two important parameters that you will use for DEFINE DATA are:

- `dname:` the name of the data definition to be created; and

- `dname.ext:` the file in which the data definition is stored.

PGAU data entries are only defined once, so eliminate any duplicates.

> **Note:** Move COBOL record layouts (copybooks) to the gateway system.
>
> PGAU can use copybooks as input when defining the data items. Once you have identified the data items to be exchanged, use a file transfer program to download the copybooks to the gateway system. The copybooks are later used to define the data items. The sample copybook used in the example is documented in Appendix G, "Administration Utility Samples".

3.2.4 PGAU DEFINE TRANSACTION Command on a Gateway Using SNA

Determine the network address information for the RHT program. Your network or OLTP system programmer can provide you with this information.

The five important parameters that you will use for PGAU DEFINE TRANSACTION are:

- Side Profile name

- TP name

- LU name

- LOGMODE

- SYNCLEVEL

You must also identify the Oracle NLS character set (`charset`) for the language in which the OLTP expects the data.

❏ At this point, if your gateway is using SNA, then proceed to Section 3.2.6, "Writing the PGAU Statements".

3.2.5 PGAU DEFINE TRANSACTION Command on a Gateway Using TCP/IP

Before you use this command, you will need to know the IMS Connect hostname (or TCP/IP address), port number and the other IMS Connect parameters that are defined as columns within the PGA_TCP_IMSC table. Refer to Chapter 6, "PG4TCPMAP Commands (TCP/IP Only)" for complete information about preparation for mapping parameters to TCP/IP using the pg4tcpmap tool.

When you run the pg4tcpmap tool you need to specify a unique name (Side Profile Name). That name must be the same name that you are using here to create your TIP.

If you are converting your gateway from the SNA to a TCP/IP communications protocol to invoke IMS transactions: You need to regenerate the TIPs. Refer to Chapter 2, "Procedural Gateway Administration Utility" for details.

3.2.6 Writing the PGAU Statements

After evaluating the RHT, define the TIP to PGAU for placement in the PG DD.

1. Write a DEFINE DATA statement for each entry in your data list. If, for example, your RHT had three different data structures, your data definitions might be:

```
DEFINE DATA dname1 LANGUAGE(IBMVSCOBOLII) INFILE(dnamel.ext);
DEFINE DATA dname2 LANGUAGE(IBMVSCOBOLII) INFILE(dname2.ext);
DEFINE DATA dname3 LANGUAGE(IBMVSCOBOLII) INFILE(dname3.ext);
```

Then you must copy or transfer the source file containing these data definitions to the directory where PGAU can read them as input.

2. Write a DEFINE CALL statement for each entry in your call list. If, for example, your RHT had a receive send receive send sequence, your call definitions would be:

```
DEFINE CALL cname1 PARMS((dnamel IN));
DEFINE CALL cname2 PARMS((dname2 OUT));
DEFINE CALL cname3 PARMS((dname3 IN));
DEFINE CALL cname4 PARMS((dname2 OUT));
```

> **Note:** Optionally, you can rewrite your call definitions to consolidate the data transmission into fewer exchanges, as long as you do not alter the data transmission sequence. For example:
>
> ```
> DEFINE CALL cname1 PARMS((dname1 IN),
>
>                                   (dname2 OUT));
>
> DEFINE CALL cname3 PARMS((dname3 IN),
>
>                                   (dname2 OUT));
> ```
>
> This reduces the calls between the application and the TIP from four calls to two calls passing an IN and OUT parameter on each call. Because TIPs always process IN parameters before OUT parameters, the data transmission sequence is unchanged. However, this consolidation is not always possible.
>
> **If your communication protocol is SNA**: Refer to Section 4.6.2.1, "Flexible Call Sequence" on page 4-14 for more information about PGAU DEFINE CALL commands.
>
> **If your communication protocol is TCP/IP**: Refer to Section 7.3.2.1, "Flexible Call Sequence" on page 7-7 for more information about PGAU DEFINE CALL commands.

3. Write a DEFINE TRANSACTION statement that contains every call, specifying the network address and NLS information:

```
DEFINE TRANSACTION tname CALLS(cname1
                               cname2, ....
                               cnameN)
                    ENVIRONMENT(IBM370)
                    SIDEPROF(profname) |
                      TPNAME(tpid) LUNAME(luname) LOGMODE(mode)
                    SYNCLEVEL(n)
                    NLS_LANGUAGE(charset);
```

4. You can add a GENERATE statement to create the TIP specification:

```
GENERATE tname
```

> **Note:** You can also add a REPORT statement to list the PG DD entries for `tname`:
>
> ```
> REPORT TRANSACTION tname with CALLS with DATA;
> ```
>
> Also annotate the script with Comments:
>
> ```
> REM this is a Comment
> ```

3.2.7 Writing a PGAU Script File

The previous section describes the steps you need to follow in order to execute PGAU statements via your PGAU command line processor. As a time saving measure, you can choose to write all of the statements (DEFINE DATA, DEFINE CALL and DEFINE TRANSACTION) into a single PGAU script file named **tname.ctl**, in the following order:

- define data
- define call
- define transaction
- generate

> **Caution:** Because you will probably run this script more than once, you should include UNDEFINE statements first to remove any previous entries in the PG DD.

This is an example of a **tname.ctl** PGAU script file:

```
UNDEFINE TRANSACTION tname Version(all);
UNDEFINE CALL cname1 Version(all);
UNDEFINE CALL cname2 Version(all);
UNDEFINE DATA dname1 Version(all);
UNDEFINE DATA dname2 Version(all);
UNDEFINE DATA dname3 Version(all);
DEFINE DATA dname1 LANGUAGE(IBMVSCOBOLII) INFILE(dname1.ext);
DEFINE DATA dname2 LANGUAGE(IBMVSCOBOLII) INFILE(dname2.ext);
DEFINE DATA dname3 LANGUAGE(IBMVSCOBOLII) INFILE(dname3.ext);
DEFINE CALL cname1 PARMS(dname1 IN),
                       (dname2 OUT));
DEFINE CALL cname2 PARMS(dname3 IN),
                      (dname2 OUT));
DEFINE TRANSACTION tname CALLS(cname1,
                          cname2, ....
                          cnameN)
                  ENVIRONMENT(IBM370)
                  SIDEPROF(profname) |
                    TPNAME(tpid) LUNAME(luname) LOGMODE(mode)
                  SYNCLEVEL(n)
                  NLS_LANGUAGE(charset);
Generate tname
```

3.3 Defining and Generating the TIP

After you have created your control file, use PGAU to create the PG DD entries and the TIP specification files.

Note: The user ID under which you run PGAU must have:

- write access to output the specification files (**pgau.pkh, pgau.pkb**, and **pgau.doc**), where **pgau** is the default name; and

- read access to the data definition source files (`dname.ext`), where `dname.ext` will be specified in PGAU DEFINE DATA statement(s).

Invoke PGAU against your PG DD stored in the Oracle Procedural Gateway for APPC Administrator's user ID:

```
C:\> pgau
PGAU> connect pgaadmin\pw@database_specification_string
```

Issue the following commands:

```
PGAU> set echo on
PGAU> spool tname.def
PGAU> @tname.ctl
PGAU> spool off
```

The TIP is now ready to be compiled. By default, the GENERATE statement writes your TIP specifications to the following output files in your current directory:

```
pgau.pkh (TIP Header)
pgau.pkb (TIP Body)
pgau.doc (TIP content documentation)
```

Note: You can optionally add **spool** and **echo** to your script (`tname.ctl`) or make other enhancements, such as using PG DD roles and the PGAU GROUP statement for shared PG DDs.

- **If your gateway is using SNA**: Refer to Chapter 4, "Client Application Development (SNA Only)" for more information.

- **If your gateway is using TCP/IP support for IMS Connect**: Refer to Chapter 7, "Client Application Development (TCP/IP Only)" for more information.

3.4 Compiling the TIP

Exit PGAU. Remain in your current directory and invoke SQL*Plus.

```
C:\> sqlplus userid/pw@database_specification_string
SQL> set echo on
SQL> @pgau.pkh
SQL> @pgau.pkb
```

The last two commands compile the TIP specification and body, respectively.

You have now compiled a TIP which can be called by your client application. If your client application is already written you can begin testing.

For more information about designing your client application and compiling a TIP, refer to Chapter 1, "Introduction to Oracle Procedural Gateway for APPC" and Appendix F, "Tip Internals".

If your gateway is using SNA: Refer to Chapter 4, "Client Application Development (SNA Only)" for information about PGAU statement syntax and usage.

If your gateway is using TCP/IP support for IMS Connect: Refer to Chapter 7, "Client Application Development (TCP/IP Only)" for information about PGAU statement syntax and usage.

3.5 TIP Content Documentation (tipname.doc)

This section discusses the TIP documentation file that is produced when the user issues a PGAU GENERATE command. This TIP content file describes the function calls and PL/SQL variables and datatypes available in the TIP.

PGAU GENERATE always produces a TIP content file named *tipname***.doc**. The filename is the name of the transaction that was specified in the PGAU GENERATE command, and the filetype is always **.doc**. This TIP content file contains the following sections:

- GENERATION Status

  This section contains the status under which the TIP is generated.

- TIP Transaction

  This section identifies the defined transaction attributes. These result from the PGAU DEFINE TRANSACTION definition.

- TIP Default Calls

  This section identifies the syntax of the calls made by the user's application to initialize and terminate the transaction. PGAU generates these calls into every TIP regardless of how the TIP or transaction is defined.

- TIP User Calls

  This section identifies the syntax of the calls which the user defines for the application to interact with the transaction.

- TIP User Declarations

  This section identifies the TIP package public datatype declarations, implied by the user's data definition specified in each call parameter.

- TIP User Variables

  This section contains TIP variables that can be referred to by applications or referenced by applications.

4

Client Application Development (SNA Only)

This chapter discusses how you will call a TIP and control a remote host transaction. It also provides you with the steps for preparing and executing a gateway transaction. This chapter assumes:

- a remote host transaction (RHT) has already been written;
- a TIP corresponding to the RHT has already been defined using the steps described in Chapter 3, "Creating a TIP".

> **Note:**   If your gateway uses the TCP/IP support for IMS Connect, refer to Chapter 7, "Client Application Development (TCP/IP Only)" for information about calling a TIP and controlling a remote host transaction.

This chapter contains the following sections:

- "Overview of Client Application" on page 4-2
- "Preparing the Client Application" on page 4-3
- "Understanding the Remote Host Transaction Requirements" on page 4-3
- "Customized TIPs for Each Remote Host Transaction" on page 4-7
- "Client Application Requirements" on page 4-8
- "Ensuring TIP and Remote Transaction Program Correspondence" on page 4-12
- "Calling the TIP from the Client Application" on page 4-16
- "Exchanging Data" on page 4-22
- "Executing the Application" on page 4-23
- "APPC Conversation Sharing" on page 4-23
- "Application Development with Multi-Byte Character Set Support" on page 4-29
- "Modifying a Terminal-Oriented Transaction to Use APPC" on page 4-30
- "Privileges Needed to Use TIPs" on page 4-31

4.1 Overview of Client Application

The Procedural Gateway Administration Utility (PGAU) generates a complete TIP using definitions you provide. The client application can then call the TIP to access the remote host transaction. Chapter 2, "Procedural Gateway Administration Utility", discusses the use of PGAU in detail.

This overview explains what you must do in order to call a TIP and control a remote host transaction.

The gateway receives PL/SQL calls from the Oracle Integrating Server and issues APPC calls to communicate with a remote transaction program. The following three application programs make this possible:

1. an APPC-enabled remote host transaction program

2. a Transaction Interface Package, or TIP. A TIP is a PL/SQL package that handles communication between the client and the gateway and performs datatype conversions between COBOL and PL/SQL.

   PGAU generates the TIP specification for you. In the shipped samples, the PGAU-generated package is called **pgadb2i.pkb**. This generated TIP includes at least three function calls that map to the remote transaction program:

   – **pgadb2i_init** initializes the conversation with the remote transaction program

   – **pgadb2i_main** exchanges application data with the remote transaction program

   – **pgadb2i_term** terminates the conversation with the remote transaction program

   Refer to Appendix F, "Tip Internals" for more information about TIPs, if you are writing your own TIP or debugging.

3. a client application that calls the TIP.

   The client application calls the three TIP functions with input and output arguments. In the example, the client application passes `empno`, an employee number to the remote transaction and the remote transaction sends back `emprec` an employee record.

Table 4–1 demonstrates the logic flow between the PL/SQL driver, the TIP, and the gateway using the example CICS-DB2 transaction.

Table 4–1   Logic Flow of CICS-DB2 Example

Client Application	Oracle TIP	Procedures Established Between the Gateway and the Remote Transaction (mainframe)
calls tip_init	Calls PGAINIT	Gateway sets up control blocks and issues APPC ALLOCATE. Mainframe program initiates.
calls tip_main	Calls PGAXFER to send `empno` and receive `emprec`	Gateway issues APPC SEND to the mainframe. Mainframe RECEIVE completes. Mainframe performs application logic and issues APPC SEND back to gateway. The gateway- issues APPC RECEIVE; receive completes. Mainframe issues APPC TERM.
calls tip_term	Call PGATERM	Gateway cleans up control blocks.

A client application which utilizes the gateway to exchange data with a remote host transaction performs some tasks for itself and instructs the TIP to perform other tasks on its behalf. The client application designer must consequently know the behavior of the remote transaction and how the TIP facilitates the exchange.

The following sections provide an overview of remote host transaction behavior, how this behavior is controlled by the client application and how TIP function calls and data declarations support the client application to control the remote host transaction. These sections also provide background information about what the TIP does for the client application and how the TIP calls exchange data with the remote host transaction.

4.2 Preparing the Client Application

To prepare the client application for execution you must understand the remote host transaction requirements and then perform these steps:

1. Move relevant COBOL records layout (copybooks) to the gateway system for input to PGAU.

2. Describe the remote host transaction data and calls to the PG Data Dictionary (PG DD) with DEFINE DATA, DEFINE CALL, and DEFINE TRANSACTION statements.

3. Generate the TIP in the Oracle Integrating Server, using GENERATE.

4. Create the client application that calls the TIP public functions.

5. Grant privileges on the newly created package.

4.3 Understanding the Remote Host Transaction Requirements

Browse through the remote host transaction program (RTP) to determine:

- the PL/SQL parameters required on the various client application to TIP calls

- the order in which the calls are made

Identify the remote host transaction program (RTP) facilities to be called and the data to be exchanged on each call. You will then define the following, and store them in the PG DD:

- DEFINE DATA

- DEFINE CALL

- DEFINE TRANSACTION

Refer to Chapter 3, "Creating a TIP" for specific definition steps and for the actual creation and generation of a TIP.

4.3.1 TIP Content and Purpose

The content of a PGAU-generated TIP reflects the calls available to the remote host transaction and the data that has been exchanged. Understanding this content helps when designing and debugging client applications that call the TIP.

A TIP is a PL/SQL package, and accordingly has two sections:

1. A Package Specification containing:

   - Public function prototypes and parameters, and

2. A Package Body containing:

- Private functions and internal control variables

- Public functions

- Package initialization following the last public function.

The purpose of the TIP is to provide a PL/SQL callable public function for every allowed remote transaction program interaction. A remote transaction program interaction is a logically related group of data exchanges through one or more PGAXFER RPC calls. This is conceptually similar to a screen or menu interaction in which several fields are filled in, the enter key is pressed, and several fields are returned to the user. Carrying the analogy further:

- the user might be likened to the TIP or client application

- fields to be filled in are IN parameters on the TIP function call

- fields returned are OUT parameters on the TIP function call

- screen or menu is the group of IN and OUT parameters combined

- a pressed enter key is likened to the PGAXFER remote procedural call (RPC)

The actual grouping of parameters that constitute a transaction call is defined by the user. The gateway places no restrictions on how a remote transaction program might correspond to a collection of TIP function calls, each call having many IN and OUT parameters.

PGA users typically have one TIP per remote transaction program. How the TIP function calls are grouped and what data parameters are exchanged on each call depends on the size, complexity and behavior of the remote transaction program.

Refer to Oracle's *PL/SQL User's Guide and Reference* for a discussion of how PL/SQL packages work. The following discussion covers the logic that must be performed within a TIP. Refer to the sample TIP and driver supplied in the **%ORACLE_HOME%\pg4appc\demo\CICS** directory in files **pgadb2i.pkh, pgadb2i.pkb**, and **pgadb2id.sql**.

4.3.2 Remote Host Transaction Types

From a procedural gateway application perspective, there are three main types of remote host transactions:

- one-shot

- persistent

- multi-conversational

4.3.2.1 One-Shot Transactions

A simple remote transaction program which receives one employee number and returns the employee record could have a TIP which provides one call, passing the employee number as an IN parameter and returning the employee record as an OUT parameter. An additional two function calls must be provided by this and every TIP:

- a remote transaction program init function call

- a remote transaction program terminate function call

The most simple TIP has three public functions, such as tip_init, tip_main, and tip_term.

The client application calls tip_init, tip_main, and tip_term in succession. The corresponding activity at the remote site is remote transaction program start, data exchange, and remote transaction program end.

The remote transaction program might even terminate itself before receiving a terminate signal from the gateway. This sequence is usual and is handled normally by gateway logic. This kind of remote transaction program is termed one-shot.

4.3.2.2 Persistent Transactions

A more complex remote transaction program has two modes of behavior: an INQUIRY or reporting mode, and an UPDATE mode. These modes can have two TIP data transfer function calls: one for INQUIRY and one for UPDATE. Such a TIP might have five public functions. For example:

- tip_init

  This initializes communications with the remote transaction program.

- tip_mode

  This accepts a mode selection parameter and puts the transaction program into either inquiry or update mode.

- tip_inqr

  This returns an employee record for a given employee number.

- tip_updt

  This accepts an employee record for a given employee number.

- tip_term

  This terminates communications with the remote transaction program.

The client application calls tip_init and then tip_mode to place the remote transaction program in inquiry mode which then scans employee records, searching for some combination of attributes (known to the client application and end-user).   Some parameter on an inquiry call is then set to signal a change to update mode and the client application calls tip_updt to update some record. The client application finally calls tip_term to terminate the remote transaction program.

The corresponding activity at the remote site is:

- remote transaction program start

- mode selection exchange

- loop reading records

- switch to update mode

- update one record

- remote transaction program end

Such a remote transaction program is called persistent because it interacts until it is signalled to terminate.

The remote transaction program can be written to permit a return to inquiry mode and repeat the entire process indefinitely.

4.3.2.3 Multi-Conversational Transactions

A client application might need to get information from one transaction, tran_A, and subsequently write or lookup information from another, tran_B. This is possible with a properly written client application and TIPs for tran_A and tran_B. In fact, any number of transactions might be concurrently controlled by a single client application. All transactions could be read-only, with the client application retrieving data from each and consolidating it into a local Oracle database or displaying it in an Oracle Form.

Alternatively, a transaction could be capable of operating in different modes or performing different services depending on what input selections were supplied by the client application. For example, one instance of tran_C can perform one service while a second instance of tran_C performs a second service. Each instance of tran_C would have its own unique conversation with the client application and each instance could have its own behavior (one-shot or persistent) depending on the nature of the service being performed.

4.4 Customized TIPs for Each Remote Host Transaction

Each remote host system might have hundreds of remote transaction programs (RTPs) which a user might want to call. Each remote transaction program is different, passes different data, and performs different functions. The interface between the user and each remote transaction program must consequently be specialized and customized to the user's requirements for each remote transaction program. The Transaction Interface Package provides this customized interface.

Example

Assume that the remote site has a transaction program which manages employee information in an employee database or other file system. The remote transaction program's name, in the remote host, is EMPT for EMPloyee Tracking. EMPT provides both inquiry and update facilities, and different Oracle users are required to access and use these EMPT facilities.

Some users might be restricted to inquiry-only use of EMPT, while others might have update requirements. In support of the Oracle users' client applications, at least three possible TIPs could exist:

1. EMP_MGMT to provide access to all facilities of the EMPT remote transaction program.

2. EMP_UPDT to access only the update functions of the EMPT remote transaction program.

3. EMP_INQR to access only the lookup functions of the EMPT remote transaction program.

End-user access to these TIPs is controlled by Oracle privileges. Additional security might be imposed on the end-user by the remote host.

Each TIP also has encoded within it the name of the remote transaction program (EMPT) and network information sufficient to establish an APPC conversation with EMPT.

4.5 Client Application Requirements

Using the TIP, the client application must correspond with and control the remote host transaction. This involves:

1. client application initialization

2. user input and output

3. remote host transaction initialization using the TIP initialization functions (with and without overrides)

4. remote host transaction control and data exchange using the TIP user functions

5. remote host transaction termination using the TIP termination function

6. exception handling

7. client application termination

Steps 3, 4 and 5 vary, based on the requirements of the remote host transaction.

One-shot remote host transaction client applications must:

- Declare RHT/TIP datatypes to be exchanged. All client applications must declare variables to be exchanged with the RHTs using TIPs. PL/SQL datatypes for such variables have already been defined in the TIP corresponding to each RHT and the client application need only reference the TIP datatype in its declaration. Refer also to "Declaring TIP Variables" on page 4-16 for more information. Also refer to the TIP content documentation file for the specific TIP/RHT for more information about the exact usage of these variables.

- Initialize the RHT using the TIP initialization function. The TIP directs the gateway server to initialize a conversation with the desired RHT, specifying either default RHT identifying parameters (supplied when the RHT was defined in the PG DD and encoded within the TIP when it was generated) or override RHT identifying parameters supplied by the user or client application when the TIP initialization function is called. Refer to "Initializing the Conversation" on page 4-17 and "Overriding TIP Initializations" on page 4-19 for more details.

- Exchange data with the RHT using the TIP user function (one call). As previously discussed, a one-shot remote host transaction only accommodates a single data exchange and upon completion of that exchange, the RHT terminates on its own. The client application consequently needs only to execute a single call to the user-defined TIP function to cause the data exchange.

  Refer to the TIP content documentation file in **%ORACLE_HOME%\pg4appc\demo\CICS** for the specific TIP/RHT for the exact syntax of this call.

  The client application should initialize values into IN or IN OUT parameter values before calling the TIP function call. These are the same variables that were declared above, when you declared the RHT/TIP datatypes to be exchanged.

  All TIP function calls return a 0 return code value and all returned user gateway data values are exchanged in the function parameters. Any exception conditions are raised as required and can be intercepted in an exception handler.

  Upon return from the TIP function call, the client application can analyze and operate on the IN OUT or OUT parameter values. These are the same variables that were declared above, when you declared the RHT/TIP datatypes to be exchanged.

Refer to Appendix D, "Datatype Conversions" for details about how TIPs convert the various types and formats of remote host data.

■ Terminate the RHT using the TIP termination function. Regardless of the type of RHT being accessed, the TIP terminate function should be called to clean up and terminate the conversation with the RHT. Conversations with one-shot RHTs can be terminated from the gateway server before the RHT terminates. The TIP must perform its cleanup as well. Cleanup is only performed at the termination request of the client application.

The client application can request a normal or an aborted termination.

Refer to "Terminating the Conversation" on page 4-22 for more information.

Persistent remote host transaction client applications must:

■ Declare RHT/TIP datatypes to be exchanged.   All client applications must declare variables to be exchanged with the RHTs using TIPs. PL/SQL datatypes for such variables have already been defined in the TIP corresponding to each RHT; the client application need only reference the TIP datatype in its declaration. Refer to "Declaring TIP Variables" on page 4-16 for more information. Refer also to the TIP content documentation file for the specific TIP/RHT for more information about the exact usage of these variables.

■ Initialize the RHT using the TIP initialization function. The TIP directs the gateway server to initialize a conversation with the desired RHT, specifying either default RHT identifying parameters (supplied when the RHT was defined in the PG DD and encoded within the TIP when it was generated) or override RHT identifying parameters supplied by the user or client application when the TIP initialization function is called. Refer to "Initializing the Conversation" on page 4-17 and "Overriding TIP Initializations" on page 4-19 for more details.

■ Repetitively exchange data with RHT using the TIP user function(s). Remote host transactions that provide or require ongoing or repetitive control sequences should be controlled by the client application in the same manner that the RHT would be operated by an interactive user or other control program. The intercession of the TIP and gateway server does not alter the RHT behavior; instead, it extends control of that behavior to the client application using the various function calls defined in the TIP.

A persistent RHT can be controlled with one or more TIP function calls. The RHT might be designed, for example, to loop and return output for every input until the conversation is explicitly terminated. Or it could have been designed to accept as input a count or list of operations to perform and return the results in multiple exchanges for which the TIP function has only OUT parameters.

A persistent RHT can also be interactive, each output being specified by a previous input selection and ending only when the conversation has been explicitly terminated by the client application.

The TIP function calls available to the client applications and their specific syntax is documented in the TIP Content documentation file for the specific TIP/RHT.

The manner in which the RHT interprets the TIP IN parameters and returns TIP OUT parameters must be determined from the RHT or explained by the RHT programmer. The TIP provides the function calls and the exchanged parameter datatypes to facilitate the client application's control of the RHT and imposes no limitations or preconditions on the sequence of operations the RHT is directed to perform. The TIP provides the client application with the calls and data parameters the RHT was defined to accept in the PG DD.

■ Terminate the RHT using the TIP termination function. Regardless of the type of RHT being accessed, the TIP terminate function should be called to clean up and terminate the conversation with the RHT. Conversations with persistent RHTs can be terminated from the gateway server before the RHT terminates, or the RHT might have already terminated. The TIP must perform its cleanup as well and this cleanup is only performed at the termination request of the client application.

The client application can request a normal or an aborted termination.

Refer to "Terminating the Conversation" on page 4-22 for more information.

Multi-conversational remote host transaction client applications must:

■ Declare RHT/TIP datatypes to be exchanged. All client applications must declare variables to be exchanged with the RHTs using TIPs. PL/SQL datatypes for such variables have already been defined in the TIP corresponding to each RHT, and the client application need only reference the TIP datatype in its declaration. Refer to "Declaring TIP Variables" on page 4-16 for more information. Also refer to the TIP content documentation file for the specific TIP/RHT for more information about the exact usage of these variables.

■ Initialize each RHT involved, using the TIP initializing function. A specific customized TIP exists for each RHT as defined in the PG DD. Client applications that control multiple RHTs are multi-conversational and must start each RHT and its associated conversation. This is done by calling each TIP initialization function as before; but multiple TIPs are initialized.

If a single RHT is designed to perform multiple services for one or more callers and if the client application is designed to use this RHT, the TIP corresponding to that RHT can be initialized multiple times by the client application.

The client application subsequently distinguishes from active RHTs under its control using:

– TIP schema `tipname.callname` when multiple TIP/RHTs are being controlled. By encoding the same TIP schema name on TIP user calls, the client application specifies to which RHT the call is being made.

– `tranuse` IN OUT parameter value when multiple instances of the same TIP/RHT are being controlled. This is the value returned on the TIP initialization function call and subsequently passed as an IN parameter on the user-defined TIP function calls. The returned `tranuse` value corresponds to that conversation connected to a given instance of an RHT. By supplying the same `tranuse` value on TIP user calls, the client application specifies to which RHT instance the given RHT call is being made.

Client application logic must keep track of which RHTs have been started and which TIPs and `tranuse` values correspond to started RHTs.

■ Exchange data with each RHT, using the TIP user function(s), either once or repetitively if the RHT is one-shot or persistent. Client application logic must sequence the RHTs though their allowed steps in accordance with proper RHT operation, as does a user operating the RHTs interactively.

Client application logic must also perform any cross-RHT result analysis or data transfer that might be required. All TIPs execute in isolation from each other.

Output from one RHT intended as input to another RHT must be received in the client application as an IN or IN OUT parameter from the first RHT and sent as an IN or IN OUT parameter from the client application to the second RHT. All TIP-to-RHT function calls must be performed by the client application and data

parameters exchanged must have been declared as variables by the client application. The TIPs provide both the required datatype definitions and the RHT function calls for the client application.

Refer to the TIP content documentation file for each specific TIP/RHT for the exact syntax of the TIP function calls and definitions of the parameter datatypes exchanged.

■ Terminate each initialized RHT, using the TIP termination function. To terminate an RHT, its corresponding TIP termination function must be called to terminate the RHT and its conversation and to initiate TIP cleanup. The RHT to be terminated is specified by its TIP schema name (the same schema as for its data exchange function calls) and the `tranuse` value when multiple instances of the same RHT are being terminated.

RHTs and their corresponding TIPs can be terminated in any sequence desired by the client application and do not have to be terminated in the same order in which they are initialized.

Note: The specific syntax of the various TIP data exchange variables function calls is the same as was previously defined in the PG DD for the particular RHT and can be researched by examining the TIP content documentation file (**tipname.doc**) or the TIP specification file produced when the TIP was generated. If a TIP has not yet been generated for the RHT being accessed, refer to Chapter 3, "Creating a TIP", and "DATA Correspondence" on page 4-12, "CALL Correspondence" on page 4-13, and "TRANSACTION Correspondence" on page 4-15 for more information. It is preferable to define and generate the TIP first, however, so that the client application reference documentation is available to you when needed.

4.6 Ensuring TIP and Remote Transaction Program Correspondence

A remote host transaction program and its related TIP with client application must correspond on two key requirements:

- Parameter datatype conversion, which results from the way in which transaction DATA is defined. Refer to Appendix D, "Datatype Conversions" for a discussion of how PGAU-generated TIPs convert data based on the data definitions.

- APPC send/receive synchronization, which results from the way in which transaction CALLs are defined

These DATA and CALL definitions are then included by reference in a TRANSACTION definition.

4.6.1 DATA Correspondence

Using data definitions programmed in the language of the remote host transaction, the PGAU DEFINE DATA command stores in the PG DD the information needed for PGAU GENERATE to create the TIP function logic to perform:

- all data conversion from PL/SQL IN parameters supplied by the receiving remote host transaction

- all buffering into the format expected by the receiving remote host transaction

- all data unbuffering from the format supplied by the sending remote host transaction

- all data conversion to PL/SQL OUT parameters supplied by the sending remote host transaction

PGAU determines the information needed to generate the conversion and buffering logic from the data definitions included in the remote host transaction program. PGAU DEFINE DATA reads this information from files, such as COBOL copy books, or in-stream from scripts and saves it in the PG DD for repeated use. The gateway Administrator needs to transfer these definition files from the remote host to the Oracle host where PGAU runs.

From the data definitions stored in the PG DD, PGAU GENERATE determines the remote host datatype and matches it to an appropriate PL/SQL datatype. It also determines data lengths and offsets within records and buffers and generates the needed PL/SQL logic into the TIP. Refer to the PGAU "DEFINE DATA" statement on page 2-2 in Chapter 2, "Procedural Gateway Administration Utility" and "Sample PGAU DEFINE DATA Statements" in Appendix G, "Administration Utility Samples" for more information.

All data that are referenced as parameters by subsequent calls must first be defined using PGAU DEFINE DATA. Simple data items, such as single numbers or character strings, and complex multi-field data aggregates, such as records or structures, can be defined. PGAU automatically generates equivalent PL/SQL variables and records of fields or tables for the client application to reference in its calls to the generated TIP.

As discussed, a parameter might be a simple data item, such as an employee number, or a complex item, such as an employee record. PGAU DEFINE DATA automatically extracts the datatype information it needs from the input program data definition files.

In this example, `empno` and `emprec` are the arguments to be exchanged.

```
pgadb2i_main(trannum,empno,emprec)
```

A PGAU DEFINE DATA statement must therefore be issued for each of these parameters:

```
DEFINE DATA EMPNO
       PLSDNAME (EMPNO)
       USAGE (PASS)
       LANGUAGE (IBMVSCOBOLII)
       (
       01 EMP-NO PIC X(6).
       );

DEFINE DATA EMPREC
       PLSDNAME (DCLEMP)
       USAGE (PASS)
       LANGUAGE (IBMVSCOBOLII)
       INFILE("emp.cob");
```

Note that a definition is not required for the `trannum` argument. This is the APPC conversation identifier and does not require a definition in PGAU.

4.6.2 CALL Correspondence

The requirement to synchronize APPC SENDs and RECEIVEs means that when the remote transaction program expects data parameters to be input, it issues APPC RECEIVEs to read the data parameters. Accordingly, the TIP must cause the gateway to issue APPC SENDs to write the data parameters to the remote transaction program. The TIP must also cause the gateway to issue APPC RECEIVEs when the remote transaction program issues APPC SENDs.

The PGAU DEFINE CALL statement specifies how the generated TIP is to be called by the client application and which data parameters are to be exchanged with the remote host transaction for that call. Each PGAU DEFINE CALL statement might specify the name of the TIP function, one or more data parameters, and the IN/OUT mode of each data parameter. Data parameters must have been previously defined with PGAU DEFINE DATA statements. Refer to "DEFINE CALL" on page 2-2 in Chapter 2, "Procedural Gateway Administration Utility" and "Sample PGAU DEFINE CALL Statements" in Appendix G, "Administration Utility Samples" for more information.

PGAU DEFINE CALL processing stores the specified information in the PG DD for later use by PGAU GENERATE. PGAU GENERATE then creates the following in the TIP package specification:

- declarations of public PL/SQL functions for each CALL defined with PL/SQL parameters for each DATA definition specified on the CALL
- declarations of the public PL/SQL data parameters

The client application calls the TIP public function as a PL/SQL function call, using the function name and parameter list specified in the PGAU DEFINE CALL statement. The client application might also declare, by reference, private variables of the same datatype as the TIP public data parameters to facilitate data passing and handling within the client application, thus sharing the declarations created by PGAU GENERATE.

In this example, the following PGAU DEFINE CALL statement must be issued to define the TIP public function:

```
DEFINE CALL DB2IMAIN
       PKGCALL (pgadb2i_main)
       PARMS ((empno IN),(emprec OUT));
```

4.6.2.1 Flexible Call Sequence

The number of data parameters exchanged between the TIP and the gateway on each call can vary at the user's discretion, as long as the remote transaction program's SEND/RECEIVE requests are satisfied. For example, the remote transaction program data exchange sequence might be:

```
APPC SEND    5 fields  (field1-field5)
APPC RECEIVE 1 fields  (field6)
APPC SEND    1 field   (field7)
APPC RECEIVE 3 fields  (field8 - field10)
```

The resulting TIP/application call sequence could be:

```
tip_call1(parm1 OUT,  <-- APPC SEND field1 from remote TP
          parm2 OUT,  <-- APPC SEND field2 from remote TP
          parm3 OUT); <-- APPC SEND field3 from remote TP

tip_call2(parm4 OUT,  <-- APPC SEND field4 from remote TP
          parm5 OUT); <-- APPC SEND field5 from remote TP
tip_call3(parm6 IN OUT); --> APPC RECEIVE field6 in remote TP
                         <-- APPC SEND field7 from remote TP

tip_call4(parm8 IN,   --> APPC RECEIVE field8 into remote TP
          parm9 IN,   --> APPC RECEIVE field9 into remote TP
          parm10 IN); --> APPC RECEIVE field10 into remote TP
```

To define these four public functions to the TIP, four PGAU DEFINE CALL statements must be issued, each specifying its unique public function name (tip_callx) and the data parameter list to be exchanged. Once a data item is defined using DEFINE DATA, it can be referenced in multiple calls in any mode (IN, OUT, or IN OUT). For example, `parm5` could be used a second time in place of `parm6`. This implies the same data is being exchanged in both instances, received into the TIP and application on `tip_call2` and returned, possibly updated, to the remote host in `tip_call4`.

Notice also that the remote transaction program's first five written fields are read by two separate TIP function calls, `tip_call1` and `tip_call2`. This could also have been equivalently accomplished with five TIP function calls of one OUT parameter each or a single TIP function call with five OUT parameters. Then the remote transaction program's first read field (`field6`) and subsequent written field (`field7`) correspond to a single TIP function call (`tip_call3`) with a single IN OUT parameter (`parm6`).

This use of a single IN OUT parameter implies that the remote transaction program's datatype for `field6` and `field7` are both the same and correspond to the conversion performed for the datatype of `parm6`. If `field6` and `field7` were of different datatypes, then they have to correspond to different PL/SQL parameters (for example, `parm6` IN and `parm7` OUT). They could still be exchanged as two parameters on a single TIP call or one parameter each on two TIP calls, however.

Lastly, the remote transaction program's remaining three RECEIVE fields are supplied by `tip_call4` parameters 8-10. They also could have been done with three TIP calls passing one parameter each or two TIP calls passing one parameter on one call and two parameters on the other, in either order. This flexibility permits the user to define the correspondence between the remote transaction program's operation and the TIP function calls in whatever manner best suits the user.

4.6.2.2 Call Correspondence Order Restrictions

Each TIP public function first sends all IN parameters, before it receives any OUT parameters. Thus, a remote transaction program expecting to send one field and then receive one field must correspond to separate TIP calls.

For example:

```
tip_callO( parmO OUT); <-- APPC SEND outfield from remote TP
```

PGAXFER RPC checks first for parameters to send, but finds none and proceeds to receive parameters:

```
tip_callI( parmI IN);  --> APPC RECEIVE infield to remote TP
```

PGAXFER RPC processes parameters to send and then checks for parameters to receive, but finds none and completes; therefore, a single TIP public function with an OUT parameter followed by an IN parameter does not work, because the IN parameter is processed first--regardless of its position in the parameter list.

4.6.3 TRANSACTION Correspondence

The remote host transaction is defined with the PGAU DEFINE TRANSACTION statement with additional references to prior definitions of CALLs that the transaction supports.

You specify the remote host transaction attributes, such as:

- transaction ID or name

- network address or location

- system type (such as IBM370)

- Oracle National Language of the remote host

Note: The PL/SQL package name is specified when the transaction is defined; this is the name by which the TIP is referenced and which the public function calls to be included within the TIP. Each public function must have been previously defined with a PGAU DEFINE CALL statement, which has been stored in the PG DD.   If you do not specify a package name (TIP name) in the GENERATE statement, the transaction name you specified will become the package name by default. In that case, the transaction name (*tname)* must be unique and must be in valid PL/SQL syntax within the database containing the PL/SQL packages.

For more information, refer to "DEFINE TRANSACTION" in Chapter 2, "Procedural Gateway Administration Utility" and "Sample PGAU DEFINE TRANSACTION Statement" in Appendix G, "Administration Utility Samples".

In this example, the following DEFINE TRANSACTION statements are used to define a remote CICS transaction called DB2I:

```
DEFINE TRANSACTION DB2I
   CALL (  DB2IMAIN,
           DB2IDIAG  )
   SIDEPROFILE(CICSPROD)
```

```
                    TPNAME(DB2I)
                    LOGMODE(ORAPLU62)
                    SYNCLEVEL(0)
                    NLS_LANGUAGE("AMERICAN_AMERICA.WE8EBCDIC37C");
```

4.7 Calling the TIP from the Client Application

Once a TIP is created, a client application must be written to interface with the TIP. A
client application that calls the TIP functions must include five logical sections:

- declaring TIP variables

- initializing the conversation

- exchanging data

- terminating the conversation

- error handling

4.7.1 Declaring TIP Variables

The user declarations section of the **tipname.doc** file documents the required
declarations.

When passing PL/SQL parameters on calls to TIP functions, the client application
must use the exact same PL/SQL datatypes for TIP function arguments as are defined
by the TIP in its specification section. Assume, for example, the following is in the TIP
specification, or **tipname.doc:**

```
FUNCTION tip_call1    tranuse,   IN      BINARY_INTEGER,
                      tip_var1   io_mode pls_type1,
                      tip_record io_mode tran_rectype)
RETURN INTEGER;

TYPE  tran_rectype is RECORD
      (rec_field1 pls_type1,
      ...
      rec_fieldN pls_typeN);
```

Table 4–2 provides a description of the function declarations:

Table 4–2   Function Declarations

Item	Description
tip_call1	The TIP function name as defined in the package specification.
tranuse	The remote transaction instance parameter returned from the TIP init function identifying the conversation on which this TIP call is to exchange data.
tran_rectype	The PL/SQL record datatype declared in the tipname TIP specification. This is the same value as in the TYPE tran_rectype is RECORD statement.
pls_typeN	Is a PL/SQL atomic datatype.
rec_fieldN	Is a PL/SQL record field corresponding to a remote transaction program record field.

In the client application PL/SQL atomic datatypes should be defined as the exact same
datatype of their corresponding arguments in the TIP function definition. The
following should be coded in the client application before the BEGIN command:

```
appl_var  pls_type1;    /* declare appl variable for ....  */
```

TIP datatypes need not be redefined. They must be declared locally within the client application, appearing in the client application before the BEGIN:

```
appl_record tipname.tran_rectype;  /* declare appl record */
```

Table 4–3 describes the command line arguments:

Table 4–3   Command Line Arguments

Item	Description
tip_call1	The TIP function name as defined in the package specification.
tranuse	The remote transaction instance parameter returned from the TIP init function identifying the conversation on which this TIP call is to exchange data.
tran_rectype	The PL/SQL record datatype declared in the `tipname` TIP specification. This is the same value as in the TYPE *tran_rectype* is `RECORD` statement.

Refer to the *tipname*.**doc** content file for a complete description of the user declarations you can reference.

The client application calls the TIP public function as if it were any local PL/SQL function:

```
rc = tip_call1( tranuse,
                appl_var,
                appl_record);
```

In the CICS-DB2 inquiry example, the PL/SQL driver **pgadb2id.sql,** which is located in **%ORACLE_HOME%\pg4appc\demo\CICS** directory, is the client application and includes the following declaration:

```
...
...
CREATE or REPLACE PROCEDURE db2idriv(empno IN CHAR) IS
tranuse INTEGER :=0                /* transaction usage number   */
DCLEMP PGADB2I.DCLEMP_typ;         /* DB2 EMP row definition     */
DB2 PGADB2I.DB2_typ;              /* DB2 diagnostic information  */
rc INTEGER :=0                     /* PGA RPC return codes        */
line VARCHAR2(132);                /* work buffer for output     */
term INTEGER :=0;                  /* 1 if pgadb2i_term called   */
...
...
```

4.7.2 Initializing the Conversation

The call to initialize the conversation serves several purposes:

- To cause the PL/SQL package, the TIP, to be loaded and to perform the initialization logic programmed in the TIP initialization section.

- To cause the TIP init function to call the PGAINIT remote procedural call (RPC), which in turn establishes communication with the remote transaction program (RTP), and returns a transaction instance number to the application.

Optionally, calls to initialize the conversation can be used to:

- Override default RHT/OLTP identification, network address attributes, and conversation security user ID and password.

- Specify what diagnostic traces the TIP is to produce. Refer to Chapter 8, "Problem Determination" for more information about diagnostic traces.

PGAU-generated TIPs provide four different initialization functions that client applications can call. These are overloaded functions which all have the same name, but vary in the types of parameters passed.

Three initialization parameters are passed:

- The transaction instance number for RHT conversation identification. The `tranuse` parameter is required on all TIP initializations.

- TIP diagnostic flags for TIP runtime diagnostic controls. The `tipdiag` parameter is optional. Refer to Chapter 8, "Problem Determination" for a discussion of TIP diagnostics.

- TIP default overrides for overriding OLTP and network attributes. The `override` parameter is optional.

The following four functions are shown as they might appear in the TIP Content documentation file. Examples of client application use are provided later.

```
TYPE override_Typ IS RECORD (
        tranname  VARCHAR2(255),  /* Transaction Program   */
        transync  BINARY_INTEGER, /* RESERVED              */
        trannls   VARCHAR2(50),   /* RESERVED              */
        oltpname  VARCHAR2(255),  /* Logical Unit          */
        oltpmode  VARCHAR2(255),  /* LOG Mode Entry        */
        netaddr   VARCHAR2(255),  /* Side Profile          */
        oltpuser  VARCHAR2(8),    /* userid for OLTP access */
        oltppass  VARCHAR2(8));   /* password for OLTP access*/

FUNCTION pgadb2i_init(               /* init standard */
        tranuse IN OUT BINARY_INTEGER)
        RETURN INTEGER;

FUNCTION pgadb2i_init(               /* init override */
        tranuse IN OUT BINARY_INTEGER,
        override IN override_Typ)
        RETURN INTEGER;

FUNCTION pgadb2i_init(               /* init diagnostic */
        tranuse IN OUT BNARY_INTEGER,
        tipdiag IN CHAR)
        RETURN INTEGER;

FUNCTION pgadb2i_init(               /* init over-diag */
        tranuse IN OUT BINARY_INTEGER,
        override IN override_Typ,
        tipdiag IN CHAR)
        RETURN INTEGER;
```

4.7.2.1 Transaction Instance Parameter

This transaction instance number (shown in examples as `tranuse`) must be passed to subsequent TIP exchange and terminate functions. It identifies to the gateway on which APPC conversation--and therefore which iteration of a remote transaction program--the data is to be transmitted or communication terminated.

A single client application might control multiple instances of the same remote transaction program or multiple different remote transaction programs, all concurrently. The transaction instance number is the TIP's mechanism for routing the client application call through the gateway to the intended remote transaction program.

It is the responsibility of the client application to save the transaction instance number of each active transaction and pass the correct one to each TIP function called for that transaction.

The client application calls the TIP initialization function as if it were any local PL/SQL function. For example:

```
...
...
tranuse INTEGER := 0;/* transaction usage number*/
...
...
BEGIN
 rc := pgadb2i.pgadb2i_init(tranuse);
...
...
```

4.7.2.2 Overriding TIP Initializations

Note that in the preceding example the client application did not specify any remote transaction program name, network connection, or security information. The TIP has such information internally coded as defaults and the client application simply calls the appropriate TIP for the chosen remote transaction program. The client application can, however, optionally override some TIP defaults and supply security information.

You do not need to change any client applications that do not require overrides.

When the remote host transaction was defined in the PG DD, the DEFINE TRANSACTION statement specified certain default OLTP and network identification attributes which can be overridden:

- TPname

- LUname

- LOGMODE

- Side Profile

Refer to "DEFINE TRANSACTION" in Chapter 2, "Procedural Gateway Administration Utility" for more information about the DEFINE TRANSACTION statement.

These PG DD-defined transaction attributes are generated into TIPs as defaults and can be overridden at TIP initialization time. This facilitates the use of one TIP, which can be used with a test transaction or system, and can later be used with a production transaction or system, without having to regenerate the TIP.

The `override_Typ` record datatype describes the various transaction attributes that can be overridden by the client application. The following overrides are currently supported:

- `tranname` can be set to override the value that was specified by the TPNAME parameter of the DEFINE TRANSACTION statement

- `oltpname` can be set to override the value that was specified by the LUNAME parameter of the DEFINE TRANSACTION statement

- `oltpmode` can be set to override the value that was specified by the LOGMODE parameter of the DEFINE TRANSACTION statement

- `netaddr` can be set to override the value that was specified by the SIDEPROFILE parameter of the DEFINE TRANSACTION statement

In addition to the transaction attributes defined in the PG DD, there are two security-related parameters, conversation security user ID and conversation security password, that can be overridden at TIP initialization time. The values for these parameters normally come from either the database link used to access the gateway or the Oracle database session. There are cases when the Oracle database user ID is not sufficient for accessing the OLTP system. The user ID and password overrides provide a way to specify those parameters to the OLTP system.

The following overrides are currently supported:

- `oltpuser` can be set to override the user ID used to initialize the conversation with the OLTP

- `oltppass` can be set to override the password used to initialize the conversation with the OLTP

The security overrides have an effect only if PGA_SECURITY_TYPE=PROGRAM is specified in the gateway initialization file, and the OLTP system is configured to accept a user ID and password on incoming conversation requests.

The `transync` (APPC synclevel) and `trannls` (NLS character set) are defined in the override record datatype, but are reserved for future use. The RHT synclevel and NLS name cannot be overridden.

The client application might override the default attributes at TIP initialization for the following reasons:

- to start a different version of the RHT (such as production instead of test)

- to change the location of the OLTP containing the RHT (if the OLTP was moved due to migration or a switch to backup configuration)

Client applications requiring overrides can use any combination of override and initialization parameters and might alter the combination at any time without regenerating the TIP or affecting applications that do not override parameters.

To override the TIP defaults, an additional client application record variable must be declared as `override_Typ` datatype, values must be assigned to the override subfields, and the override record variable must be passed on the TIP initialization call from the client application.

For example:

```
...
...
my_overrides pgadb2i.override_Typ;   -- declaration
...
...
my_overrides.oltpname := 'CICSPROD'; -- swap to production CICS
my_overrides.tranname := 'TNEW';     -- new transaction name

BEGIN
   rc := pgadb2i.pgadb2i_init(tranuse,my_overrides); -- init
   ...
   ...
```

Within the TIP, override attributes are checked for syntax problems and passed to the gateway server.

4.7.2.3 Security Considerations

The security requirements of the default and overridden OLTPs must be the same because the same gateway server is used in either conversation, as dictated by the database link names in the PGA RPC calls. The gateway server startup security mode is set at gateway server initialization time and passed unchanged to the OLTP at TIP or conversation initialization time.

4.8 Exchanging Data

The client application should pass the transaction instance number, returned from a previous tip_init call, to identify which remote transaction program is affected and to identify any client application data parameters to be exchanged with the remote transaction program.

In this CICS-DB2 inquiry example, we pass an employee number and receive an employee record back:

```
rc = pgadb2i.pgadb2i_main(tranuse, /* transfer data          */
                          empno,  /* employee number       */
                          DCLEMP); /* return employee record*/
```

4.8.1 Terminating the Conversation

The client application calls the TIP termination function as if it were any local PL/SQL function. For example:

```
...
...
term := 1;    /* indicate term called* */
 rc := pgadb2i.pgadb2i_term(tranuse,0); /* terminate normally  */
...
...
```

After a transaction instance number has been passed on a TIP terminate call to terminate the transaction, or after the remote transaction program has abended, that particular transaction instance number may be forgotten.

4.8.2 Error Handling

The client application should include an exception handler that can clean up any active APPC conversations before the client application terminates. The sample client application provided in **pgadb2id.sql** contains an example of exception handling.

Gateway exceptions are reported in the range PGA-20900 to PGA-20999. When an exception occurs, the TIP termination function should be called for any active conversations that have been started by prior calls to the TIP initialization function.

For example:

```
EXCEPTION
 WHEN OTHERS THEN
  IF term = 0 THEN         /* terminate function not called yet */
   rc := pgadb2i.pgadb2i_term(tranuse,1); /*terminate abnormally*/
  END IF;
 RAISE;
...
...
```

The remote transaction should also include provisions for error handling and debugging, such as writing debugging information to the CICS temporary storage queue area. Refer to the *PL/SQL User's Guide and Reference* for a discussion of how to intercept and handle Oracle exceptions.

4.8.3 Granting Execute Authority

The TIP is a standard PL/SQL package and execute authority must be granted to users who call the TIP from their client application. In this example, we grant execute on the PGADB2I package to user SCOTT:

```
GRANT EXECUTE ON PGADB2I TO SCOTT
```

Refer to the *Oracle Database Administrator's Guide* for further information.

4.9 Executing the Application

Before executing the client application, ensure that a connection to the host is established and that the receiving partner is available. In this example we use PL/SQL driver DB2IDRIV to execute the CICS-DB2 inquiry. To execute this client application, enter from SQL*Plus:

```
set serveroutput on
execute DB2IDRIV('nnnnnn');
```

4.10 APPC Conversation Sharing

Multiple TIPs can share the same APPC conversation with one or more Remote Host Transactions (RHTs) which are also sharing that same conversation. Two benefits derive from this feature:

- Existing RHTs which rely upon passing control of a conversation are supported by Oracle Procedural Gateway for APPC.

- TIPs otherwise too large for PL/SQL compilation can be separated into multiple smaller TIPs, each with fewer user-defined functions, providing the client application with the same set of function calls and data definitions without any change to the RHT.

4.10.1 APPC Conversation Sharing Concepts

Mainframe OLTPs, such as IMS, allow transactions to share a single APPC conversation by passing it when the transaction calls another transaction. RHTs are defined to PGAU as single transactions with calls, inputs and outputs for which PGAU generates a single TIP with initialization, transfer and termination functions corresponding to that specific RHT.

Logic generated into every TIP allows that TIP either:

- to initiate a new conversation when its init function is called, or

- to transfer data on an existing conversation when its user-defined functions are called, or

- to terminate an existing conversation when its "term" function is called.

An APPC conversation is treated as a resource shared and managed by multiple TIPs. There is no requirement for any TIP to be the sole user of an APPC conversation.

Any TIP generated at 3.4.0 or later can perform any of the following combinations of service:

- initiate

- initiate and transfer

- initiate, transfer, and terminate (standard operation)

- transfer

- transfer and terminate

- terminate

- initiate and terminate (assumes other TIPs perform transfer)

A single APPC conversation can be shared in the following ways:

- from one TIP to multiple RHTs

- from multiple TIPs to one RHT

- from multiple TIPs to multiple RHTs

Without APPC conversation sharing, a single TIP must be defined which contains all functions and data for all RHTs which a client application might need to call. Creating TIPs with a superset of RHTs often causes such TIPs to be too large for PL/SQL to compile.

Conversely, with APPC conversation sharing, each RHT (or even each RHT data exchange for those RHTs which perform multiple, different data exchange operations) can be defined in a single TIP which is smaller and less likely to exceed PL/SQL compilation limits.

4.10.2 APPC Conversation Sharing Usage

APPC conversation sharing is automatically available in every TIP generated at 3.4.0 or later. No TIPs generated before 3.4.0 can participate in APPC conversation sharing. TIPs generated before 3.4.0 must be regenerated using PGAU 3.4.0. or later to participate in APPC conversation sharing. PGAU is upward compatible and regeneration should be transparent, provided only the regenerated TIP body (**tipname**.pkb) is recompiled. If the TIP specification is also recompiled, the client application needs recompilation as well. Refer to Appendix F, "Tip Internals" for more detailed information.

Definition and generation of TIPs is accomplished as previously discussed in Chapters 1, 2, and 3. No additional options or parameters need be specified.

Run-time use of APPC conversation sharing is under the control of the client application. It is accomplished simply by calling the init function of one of the TIPs that share a conversation and passing the `tranuse` value returned to the other TIP functions as each is called in its desired order. Any TIP init function can be used, provided that all TIPs were defined with the same DEFINE TRANSACTION TPNAME or SIDEPROFILE value. The TPNAME or SIDEPROFILE value specifies which RHT to initialize.

When the init function of an APPC conversation sharing-capable TIP is called to initialize a conversation, the `tranuse` value returned indicates conversation sharing is enabled. By passing that same `tranuse` value when calling functions in other TIPs, those other TIPs perform their transfers on the same conversation already initialized, provided that all TIPs involved were generated at Version 3.4.0 or later.

4.10.3 APPC Conversation Sharing TIP Compatibility

TIPs generated at 3.4.0 or later of the procedural gateway use and expect different values for `tranuse` than do pre-3.4.0 TIPs. If a pre-3.4.0 TIP is used to initialize a conversation and its `tranuse` value is passed to a 3.4.0 or later generated TIP, the following exception is raised:

```
ORA-20704 PGA_TIP: tranuse value cannot be shared
```

Pre-3.4.0 generated TIPs do not detect the different `tranuse` value for shared conversations, however, and this can result in unpredictable errors.

> **Caution:**   All TIPs called in a shared conversation must have been generated at 3.4.0 or later.
>
> No TIPs generated before 3.4.0 can participate in APPC conversation sharing.

The `tranuse` values are incompatible between pre-3.4.0 and 3.4.0 or later releases. This should not pose a problem for you for the following reason: before 3.4.0, all RHT functions defined in a TIP had to be called through that TIPs functions, and the init function of that same TIP had to be called first to initialize the conversation. The `tranuse` value was only valid for the TIP which initialized it. Thus, unless you make programming changes, it is not possible for an existing application to accidentally mix `tranuse` values.

Pre-3.4.0 TIPs and client applications can continue to be used without change and old client applications can call new 3.4.0 or later TIPs without change. This is made possible when an old TIP body is regenerated and compiled; the TIP now becomes capable of APPC conversation sharing, even though the old client application has not changed.

None of the functions of a pre-3.4.0 TIP can share an APPC conversation. However, once a TIP is regenerated at 3.4.0 or later, any of its functions can share APPC conversations.

4.10.4 APPC Conversation Sharing for TIPs That Are Too Large

You can use conversation sharing to circumvent a TIP that is too large to compile. This is identified by 'PLS-00123 - package too large to compile', or some other problem symptom such as PL/SQL compilation hanging. In this case you must choose which function calls to remove from the former TIP and define into new TIPs.

Specifically, you must decide which PGAU DEFINE CALL statements and their related DEFINE DATA statements should be moved from the old PGAU control file (**.ctl**) into one or more new PGAU control files. In addition, you must decide which PGAU DEFINE TRANSACTION statements should be included in each new PGAU control file defining each new TIP.

You must consider several PGAU statements; refer to Table 4–4 for a list of the PGAU statements and their descriptions:

Table 4–4   PGAU Statements

Statement	Description
DEFINE DATA statements	Must be unique. They can be shared by all affected PGAU control files, provided they are defined to the Procedural Gateway Data Dictionary (PG DD) before being referenced by DEFINE CALL statements. No changes are needed to these statements.
DEFINE CALL statements	Must be unique. They need only be referenced by the new DEFINE TRANSACTION statement of the TIP in which they are included, provided they are defined to the PG DD before being referenced by a DEFINE TRANSACTION statement. The DEFINE CALL statements can optionally be moved to the new PGAU control file of the TIP in which they are included.
DEFINE TRANSACTION statements	Specified for each new TIP desired and will reference those call definitions moved from the former large TIP to the new small TIPs. No transaction attributes will change. This allows any new TIP to perform the same initialization or termination with the same RHT as the former large TIP. The old DEFINE TRANSACTION statement (of the former large TIP) should now exclude any call definitions which are being moved to new small TIPs.

4.10.5 APPC Conversation Sharing Example

Assume the existence of RHTs A, B and C, and that RHT A performs a menu selection and calls RHT B for a query function or RHT C for an update followed by a select function.

You could define the following DATA and CALLs:

- DEFINE DATA choice ...

- DEFINE DATA input ...

- DEFINE DATA answer ...

- DEFINE DATA record ...

- DEFINE CALL menu_A callname(pick) parms(choice in);

- DEFINE CALL query_B callname(query) parms((input in),

                                 (answer out));

- DEFINE CALL update_C callname(update) parms(record in);

- DEFINE CALL select_C callname(select) parms(record out);

The following example TIPs could be defined:

Example 1

This example does not use APPC conversation sharing, but is a valid TIP definition created before release 3.4.0, combining the functions of RHTs A, B and C.

```
DEFINE TRANSACTION rhtABC calls(menu_A,
                               query_B,
                               update_C,
                               select_C)
                          tpname(RHTA);
```

This TIP includes all data definitions and calls, and might be too large to compile. This TIP does not use APPC conversation sharing as there is only the one TIP, `rhtABC`. The RHTs do, however, perform their normal sharing of the conversation at the remote host. If the TIP was small enough to compile, the client application calls TIP functions as follows:

```
rc := rhtABC.rhtABC_init(tranuse);
rc := rhtABC.pick(tranuse, choice);
rc := rhtABC.query(tranuse, input, answer);
rc := rhtABC.update(tranuse, record);
rc := rhtABC.select(tranuse, record);
rc := rhtABC.rhtABC_term(tranuse);
```

Example 2

This example demonstrates defining a set of TIPs with APPC conversation sharing, separating the functions of RHTs A, B and C into three TIPs:

```
DEFINE TRANSACTION rhtA calls(menu_A)    tpname(RHTA);
DEFINE TRANSACTION rhtB calls(query_B)   tpname(RHTA);
DEFINE TRANSACTION rhtC calls(update_C,
                             select_C) tpname(RHTA);
```

Each TIP includes only the call and data it requires, and each TIP automatically performs APPC conversation sharing. The client application calls these functions as follows:

```
rc := rhtA.rhtA_init(tranuse);
rc := rhtA.pick(tranuse, choice);
rc := rhtB.query(tranuse, input, answer);
rc := rhtC.update(tranuse, record);
rc := rhtC.select(tranuse, record);
rc := rhtB.rhtB_term(tranuse);
```

The only client application difference between the two examples is in the schema qualifier on each of the TIP calls. This is because the function being called is in a different TIP which has a different package name in the database.

Only new DEFINE TRANSACTION statements were needed to make use of APPC conversation sharing. The CALL and DATA definitions were used as-is. This means the old TIP `rhtABC` is still defined as it was and might still be too large to compile.

Example 3

If you performed Sample 2 but you still believe that the TIP may be too large to compile, try this:

```
DEFINE TRANSACTION rhtABC calls(menu_A)    tpname(RHTA);
DEFINE TRANSACTION rhtB    calls(query_B)   tpname(RHTA);
DEFINE TRANSACTION rhtCU   calls(update_C) tpname(RHTA);
DEFINE TRANSACTION rhtCS   calls(select_C) tpname(RHTA);
```

TIP `rhtABC` has had three functions removed so it is now smaller and more likely to compile. TIP `rhtB` has one function and TIP `rhtC` has been separated into two TIPs even though the corresponding host functions remain in a single RHT.

The client application calls these functions as follows:

```
rc := rhtB.rhtB_init(tranuse);
rc := rhtABC.pick(tranuse, choice);
rc := rhtB.query(tranuse, input);
rc := rhtCU.update(tranuse, record);
```

```
rc := rhtCS.select(tranuse, record);
rc := rhtABC.rhtABC_term(tranuse);
```

A different TIP is used for initialization, illustrating that all TIPs contain the init and term functions, and because the DEFINE TRANSACTION statements all specified the same tpname (RHTA), the same remote host transaction is always called for initialization.

4.10.6 APPC Conversation Sharing Overrides and Diagnostics

TIP default override parameters are processed in the TIP init function which was called to perform initialization. Once the APPC conversation is established, no further sharing of overriding parameters is necessary. You need do nothing more than pass the overrides to the TIP init function.

TIP diagnostic parameters are shared among all TIPs sharing a given conversation. In effect, requesting diagnostics of the TIP performing initialization causes the same diagnostics to be requested of all TIPs sharing the conversation. Requesting diagnostics from only one TIP of several sharing a conversation is not possible. The application designer or user need only pass the TIP runtime trace controls to the TIP init function.

4.11 Application Development with Multi-Byte Character Set Support

COBOL presently only supports double byte character sets (DBCS) for PIC G datatypes.

PGAU processes COBOLII PIC G datatypes as PL/SQL VARCHAR2 variables and generates TIPs which automatically convert the data according to the Oracle NLS_LANGUAGEs specified for the remote host data and the local Oracle data.

These Oracle NLS_LANGUAGEs can be specified as defaults for all PIC G data exchanged by the TIP with the remote transaction (see DEFINE TRANSACTION ... REMOTE_MBCS or LOCAL_MBCS). The Oracle NLS_LANGUAGEs for any individual PIC G data item can be further overridden (see REDEFINE DATA ... REMOTE or LOCAL_LANGUAGE).

DBCS data can be encoded in any combination of supported DBCS character sets. For example, a remote host application which allows different codepages for each field of data in a record is supported by the Oracle Procedural Gateway MBCS support.

Use of REDEFINE DATA ... REMOTE_LANGUAGE or LOCAL_LANGUAGE on PIC X items is also supported. Thus a TIP can perform DBCS or MBCS conversions for specified PIC X data fields, in addition to SBCS conversions by default for the remaining PIC X data fields. Default SBCS conversion is according to the DEFINE TRANSACTION... NLS_LANGUAGE and local Oracle default LANGUAGE environment values.

When PGAU is generating a TIP, the PIC G datatypes are converted to PL/SQL VARCHAR2 datatypes. After conversion by the TIP, received 'PIC G' VARCHAR2 datatypes can have a length less then the maximum due to deletion of shift-out and shift-in meta characters, and sent 'PIC G' RAWs will have the shift-out and shift-in characters inserted as required by the remote host character set specified.

This is different from the conversions performed for PIC X data which is always a known fixed-length and hence CHAR datatypes are used in TIPs for PIC X data fields. However, even when the PIC X field contains DBCS or MBCS data, a CHAR variable is still used and padded with blanks if needed.

Some remote host applications bracket a PIC G field with PIC X bytes used for shift-out, shift-in meta-character insertion. Such a COBOL definition might look like:

```
01 MY_RECORD.
   05 SO PIC X.
   05 MY_SBCS_DATA PIC G(52).
   05 SI PIC X.
```

This is not processed correctly by PGAU, because all three fields are defined, and consequently treated, as separate data items when conversion is performed.

To be properly processed, the definition input to PGAU should be:

```
01 MY_RECORD.
   05 MY_MBCS_DATA PIC G(51).
```

The PGAU REDEFINE DATA statement can redefine the 3-field definition to the 1-field definition by specifying USAGE(SKIP) on fields SO and SI, and '05 MY_MBCS_DATA PIC G(51).' to redefine MY_MBCS_DATA. The three REDEFINE statements can be placed in the PGAU input control file, and thus the remote host definition need not be altered.

4.12 Modifying a Terminal-Oriented Transaction to Use APPC

The remote transaction program must include mapped APPC verbs to initiate, communicate, and terminate the APPC conversation. However, when the remote transaction program is terminal-oriented, the following options are available:

- You can separate the terminal logic from the application and I/O logic. Once this separation is achieved, a small front end remote transaction program can be written to interface between the gateway calls and the transaction application logic. For example, in CICS the CICS LINK is used to implement this technique.

- You can modify your existing program so that APPC calls are embedded. In the example, PGADB2I, we use CICS and its associated mapped APPC verbs as follows:

  - EXEC CICS ASSIGN accepts the conversation initiated by the gateway.

  - EXEC CICS RECEIVE receives the arguments.

  - EXEC CICS SEND ends the results.

  - EXEC CICS RETURN terminates the conversation.

- If you do not want to modify your terminal-oriented transaction, you can insert an APPC-capable interface, such as IBM Corporation's FEPI for CICS Transaction Server for z/OS, between the terminal-oriented program and the gateway.

- With IMS/TM, existing unmodified IMS transactions can be accessed with the gateway using the implicit APPC facility. With implicit APPC, the standard DLI GU, GN, and ISRT calls using the I/O PCB are automatically converted to appropriate APPC send or receive calls when the IMS transaction is invoked through APPC.

4.13 Privileges Needed to Use TIPs

Execute privileges must be explicitly granted to callers of TIPs or procedures. This privilege cannot be granted through a role.

Any TIP user wanting to trace a TIP must be granted execute privileges on the rtrace and ptrace procedures. Refer to the "Configuring PGAU" section in the chapter appropriate for your communications protocol in the *Oracle Procedural Gateway for APPC Installation and Configuration Guide*, and the *Oracle Database Application Developer's Guide* for more information.

For example:

```
C:\> sqlplus pgaadmin\pw@database_specification_string
SQL> grant execute on pgaadmin.purge_trace to tip_user_userid;
SQL> grant execute on pgaadmin.read_trace to tip_user_userid;
```

After a TIP has been developed, the TIP user must be granted execute privileges on the TIP by the TIP owner. The TIP owner is usually PGAADMIN, but can be another user who has been granted either the PGDDDEF or PGDDGEN roles.

For example:

```
C:\> sqlplus tip_owner\pw@database_specification_string
SQL> grant execute on tipname to tip_user_userid;
```

where *database_specification_string* is the Oracle Net identifier for the Oracle Integrating Server where the gateway UTL_RAW and UTL_PG components were installed. This is the same Oracle Integrating Server where the TIPs are executed and where grants on the TIPs are performed from the TIP owner user ID.

A SQL script for performing these grants is provided in the **%ORACLE_HOME%\pg4appc\admin** directory. The **pgddausr.sq**l script performs the grants for private access to the packages by a single TIP user. If private grants are to be used, the **pgddausr.sql** script must be run once for each TIP user's user ID.

To run these scripts, use SQL*Plus to connect to the Oracle Integrating Server as user PGAADMIN. From SQL*Plus, run the **pgddausr.sql** script from the **%ORACLE_HOME%\pg4appc\admin** directory. The script performs the necessary grants as previously described. You are prompted for the required user IDs, passwords, and database specification strings. If you are using private grants, repeat this step for each user ID requiring access to the packages.

No script has been provided to perform public grants. To do this, issue the following commands:

```
C:\> sqlplus tip_owner\pw@database_specification_string
SQL> grant execute on tipname to PUBLIC;
```

5

Implementing Commit-Confirm (SNA Only)

Commit-confirm allows the updating of local Oracle resources to occur in the same Oracle transaction as the updating of non-Oracle resources accessed through the Oracle Procedural Gateway for APPC.

Read this chapter to familiarize yourself with the elements and functions of commit-confirm.

❏ You will find instructions for configuring gateway components for commit-confirm on an SNA environment in the *Oracle Procedural Gateway for APPC Installation and Configuration Guide*. Refer to Chapter 5, "Configuring Your Network" and Chapter 6, "Gateway Configuration Using the SNA Communications Protocol" of the installation and configuration guide for specific information.

This chapter includes the following sections:

- "Overview of Commit-Confirm" on page 5-2
- "Supported OLTPs" on page 5-3
- "Components Required to Support Commit-Confirm" on page 5-4
- "Application Design Requirements" on page 5-6
- "Commit-Confirm Architecture" on page 5-7
- "Commit-Confirm Flow" on page 5-8

5.1 Overview of Commit-Confirm

> **Important:** If you are planning to implement commit-confirm, then you should already have configured the components. Refer to Chapter 9, "Gateway Configuration Using SNA Communication Protocol" in the *Oracle Procedural Gateway for APPC Installation and Configuration Guide* for instructions on its configuration.

Commit-confirm is a special implementation of two-phase commit that allows a database or gateway that does not support full two-phase commit to participate in distributed update transactions with other databases or gateways that do support full two-phase commit. In this implementation, the commit-confirm site is always the first to be committed, after all other sites have been prepared.This allows all sites to be kept in sync, because if the commit-confirm site fails to commit successfully, all other sites can be rolled back.

Within an Oracle distributed transaction, all work associated with that transaction is assigned a common identifier, known as the Oracle Global Transaction ID. This identifier is guaranteed to be unique, so that it can be used to exclusively identify a particular distributed transaction. The key requirement for commit-confirm support is the ability for the commit-confirm site (in this case, the Oracle Procedural Gateway for APPC) to be able to log the Oracle Global Transaction ID as part of its unit of work, so that if a failure occurs, the gateway's recovery processing can determine the status of a particular Oracle Global Transaction ID by the presence or absence of a log entry for that transaction. A new Oracle Global Transaction ID is generated after every commit or rollback operation.

The Oracle Procedural Gateway for APPC implements commit-confirm using LU6.2 synclevel 1. This is similar to the implementation of single-site update, with the added advantage that resources on both the Oracle site and the OLTP being accessed by the gateway can be updated and kept in sync. The main difference is that the commit-confirm implementation requires some additional programming in the OLTP transaction to perform the transaction logging necessary for recovery support.

5.2 Supported OLTPs

Since commit-confirm uses LU6.2 synclevel 1, it can be supported by any OLTP that supports APPC, including CICS Transaction Server for z/OS and IMS/TM. The Oracle Procedural Gateway for APPC provides sample commit-confirm applications for both CICS Transaction Server for z/OS and IMS/TM.

With CICS Transaction Server for z/OS, the standard command-level EXEC CICS interface can be used for all APPC communications. In addition, the CPI-C interface can be used if it is preferred. A sample DB2 update transaction written in COBOL using the EXEC CICS interface is provided with the gateway. Any language supported by CICS Transaction Server for z/OS can be used for writing commit-confirm transactions.

With IMS/TM, the CPI-C interface must be used, making the IMS transaction an "explicit APPC transaction," as referred to in the IBM IMSCICS Transaction Server for z/OS manuals. This is necessary because it is the only way that the LU6.2 synclevel 1 control flows are accessible to the IMS transaction. When using "implied APPC" where "GU" from the IOPCB and "ISRT" to the IOPCB are used for receiving and sending data, there is no way for the IMS transaction to access the LU6.2 synclevel 1 control flow, making it impossible to use this method for commit-confirm. A sample DLI database update transaction written in COBOL using the CPI-C APPC interface is provided with the gateway. Any language supported by IMS and CPI-C can be used for writing commit-confirm transactions.

5.3 Components Required to Support Commit-Confirm

The following components are required to support commit-confirm:

■   Oracle Procedural Gateway for APPC Server

The gateway server supports commit-confirm when PGA_CAPABILITY=COMMIT_CONFIRM is specified in the gateway initialization file. When the gateway server is running with commit-confirm enabled, it will connect to a local Oracle Integrating Server where it maintains a commit-confirm transaction log, similar to the Oracle two-phase commit log stored in the DBA_2PC_PENDING table. The gateway's transaction log is stored in the PGA_CC_PENDING table. A row is stored in this table for each in-flight transaction and remains there until the transaction has completed. The life span of rows in PGA_CC_PENDING is normally quite short, lasting only from the time the commit is received by the gateway until the time the Oracle Integrating Server completes all commit processing and tells the gateway to forget the transaction.

The commit-confirm gateway SID should be reserved for use only to invoke update transactions that implement commit-confirm. There is some extra overhead involved in the setup for logging when PGA_CAPABILITY is set to COMMIT_CONFIRM. Read-only transactions should be invoked through a separate gateway SID with PGA_CAPABILITY set to READ_ONLY so that they will not incur the extra overhead.

■   Oracle Logging Server

An Oracle server must be available for use by the gateway server for storing the PGA_CC_PENDING table. For maximum performance and reliability, Oracle recommends that this Oracle server reside on the same system as the gateway server.

■   OLTP Commit-Confirm Transaction Log

A commit-confirm transaction log database must be defined to the OLTP   system being accessed. This database must be recoverable and must be   accessible by the OLTP as part of the same unit of work as the OLTP   application's databases, so that updates to the transaction log database will be kept in sync with updates to the application's databases in a single unit of work.

The commit-confirm transaction log database need contain only the Oracle Global Transaction ID and a date/time stamp. The Oracle Global Transaction ID is 169 bytes long and must be the key field. The date/time stamp is used for purging old entries that can be left in the log after certain failure scenarios.

For simplicity, all commit-confirm applications under a particular OLTP   should share the same commit-confirm transaction log.

■   OLTP Transaction Logging Code

Code must be added to each OLTP transaction invoked by a commit-confirm gateway to perform the transaction logging required by the gateway's commit-confirm implementation. This code must receive the Oracle Global Transaction ID from the gateway and write that information into the OLTP commit-confirm transaction log database. For maximum flexibility and ease of use, this code can be written as a subroutine callable from any commit-confirm transaction on your OLTP system.

This code must be executed at the beginning of each commit-confirm   transaction prior to the first APPC receive and then immediately after each COMMIT or ROLLBACK in the transaction. This ensures that the logging is done at the beginning of each unit of work.

- OLTP Forget/Recovery Transaction

  A separate APPC transaction must be created on the OLTP system that can be started by the gateway to forget a transaction once it has been successfully committed and to query a transaction's state during recovery processing. This transaction deletes the entry for a particular Oracle Global Transaction ID from the OLTP commit-confirm transaction log database during forget processing and queries the entry for a particular Oracle Global Transaction ID from the OLTP commit-confirm transaction log database during recovery processing.

  > **Note:**   Make sure that the gateway initialization parameters and the OLTP parameters are properly configured, as described in Chapter 8 of the *Oracle Procedural Gateway for APPC Installation and Configuration Guide*, "Configuring the OLTP."

5.4 Application Design Requirements

When designing commit-confirm applications for use with the Oracle Procedural Gateway for APPC, there are some requirements you must meet to provide the ability for the gateway to determine the state of a transaction in the event of a failure. If these requirements are not met, attempting to use an application with a commit-confirm gateway will produce unpredictable results.

The first thing that must be done by an OLTP transaction invoked by a commit-confirm gateway is to receive the Oracle Global Transaction ID from the gateway and log it into the OLTP commit-confirm transaction log database. This must be done before the normal data flow between the OLTP transaction and the Oracle application begins. The gateway always sends the Oracle Global Transaction ID as the very first data item.

If the OLTP transaction is a one-shot transaction, this is the only change needed. If the transaction is a persistent transaction that performs more than one unit of work (issues more than one commit or rollback), then a new Oracle Global Transaction ID must be received and logged after every commit or rollback.

The Oracle Global Transaction ID is sent by the gateway in a variable-length record with a maximum length of 202 bytes. The first 32 bytes contain a special binary string used to verify that the data came from the gateway and not from some other application. The next 1 byte is a reserved field. The Oracle Global Transaction ID is next, with a maximum length of 169 bytes. You must log the reserved field and the Oracle Global Transaction ID, as well as a date/time stamp and any other information you wish to log. Note that the Oracle Global Transaction ID must be the key field for the log database so that the forget/recovery transaction can use the Oracle Global Transaction ID to directly access a log entry.

> **Note:**   If your OLTP is IMS/TM, you must add a PCB for the commit-confirm transaction log database to the PSB for each transaction that you will use with a commit-confirm gateway. This PCB must be the first PCB in the PSB.

5.5 Commit-Confirm Architecture

The architecture of the commit-confirm implementation in the Oracle Procedural Gateway for APPC consists of three main components:

- Oracle Integrating Server

- Oracle Procedural Gateway for APPC server (gateway server)

- Oracle logging server

This section describes the role each component plays in the operation of commit-confirm and how these components interact.

5.5.1 Components

The Oracle Integrating Server is the controlling component in the commit-confirm architecture. It tells the gateway server when to commit a transaction and when to rollback a transaction. It does the same with all other servers participating in a distributed transaction. When a failure has occurred, it is the integrating server which drives the recovery process in each participating server, including the gateway server.

The gateway server performs the task of converting instructions from the Oracle Integrating Server into LU6.2 operations and then logs the transaction into the Oracle logging server. The gateway server stores the log information in a table called PGA_CC_PENDING on the logging server. If a failure occurs during transaction processing, the gateway server determines which error should be returned to the integrating server.

The Oracle logging server is an Oracle server available to the gateway server for storing and accessing its commit-confirm log information. The logging server need not be the same Oracle server as the integrating server, but can be. Because the logging server is an integral component of gateway commit-confirm operations, the best place for it to reside is on the same system as the gateway server. This allows the communication between the gateway server and the logging server to use interprocess communications, providing a high-speed, low overhead, local connection between the components.

5.5.2 Interactions

There is a specific set of interactions that occur between the components. They are:

- Oracle Integrating Server <--> Gateway Server

  The Oracle Integrating Server drives all actions by the gateway server. At the request of the Oracle application, the integrating server can instruct the gateway server to begin a new Oracle transaction, start a commit sequence, start a rollback sequence, or start a forget sequence. It can also call gateway remote procedural call (RPC) functions (PGAINIT, PGAXFER, PGATERM) on behalf of the Oracle application.

- Gateway Server <--> Oracle Logging Server

  The gateway server calls the Oracle logging server to insert and delete rows from its PGA_CC_PENDING table. This is actually done by calling a PL/SQL stored procedure, PGA_CC_LOG, in the logging server to reduce the number of open cursors required by the gateway server for performing its logging. Only a single cursor is needed by the gateway server for logging.

5.6 Commit-Confirm Flow

The flow of control for a successful commit between an Oracle application and an OLTP transaction is described in the following section and illustrated in Figure 5–1, "Commit-Confirm Flow with Synclevel 1". The figure assumes that both Oracle and OLTP resources have been updated. The following steps in Section 5.6.1 outline the commit-confirm logic flow.

5.6.1 Commit-Confirm Logic Flow, Step by Step

1. The application issues a COMMIT to the Oracle Integrating Server.

2. The Oracle Integrating Server sends PREPARE to each participant in the distributed transaction other than the gateway.

3. Each participant prepares its database updates and responds PREPARE OK to the Oracle Integrating Server.

4. The Oracle Integrating Server sends COMMIT to the gateway. The gateway receives the COMMIT from the Oracle Integrating Server and inserts a new pending transaction row into the PGA_CC_PENDING table.

5. The gateway sends an APPC CONFIRM to the OLTP application. The OLTP application receives the CONFIRM request in the form of a status from the last APPC RECEIVE.

6. The OLTP application issues a COMMIT using an appropriate OLTP function. The OLTP commits all database updates made by the application since the last COMMIT, including the commit-confirm transaction log update.

7. Once the database updates have been committed, the OLTP returns control to the application with a return code indicating the status of the COMMIT.

8. The OLTP application sends an APPC CONFIRMED to the gateway.

9. The gateway receives the CONFIRMED and returns COMMIT OK to the Oracle Integrating Server.

10. The Oracle Integrating Server sends COMMIT to each participant in the distributed transaction other than the gateway.

11. Each participant commits its database updates and responds COMMIT OK to the Oracle Integrating Server.

12. The Oracle Integrating Server sends a FORGET to the gateway.

13. The gateway receives the FORGET and starts a new APPC conversation with the FORGET/RECOVERY transaction at the OLTP, sends it a FORGET request and an APPC CONFIRM. The FORGET/RECOVERY transaction receives the FORGET request and deletes the entry from the commit-confirm transaction log for the current Oracle transaction, and commits the delete.

14. The FORGET/RECOVERY transaction sends an APPC CONFIRMED to the gateway to indicate that the FORGET was processed, and then terminates. The gateway receives the CONFIRMED and deletes the pending transaction row from the PGA_CC_PENDING table.

15. The gateway returns FORGET OK to the Oracle Integrating Server.

16. The Oracle Integrating Server returns control to the Oracle application.

Figure 5–1, "Commit-Confirm Flow with Synclevel 1" illustrates the Commit-Confirm logic flow described in the previous section.

Figure 5–1   Commit-Confirm Flow with Synclevel 1

Figure 5–1 illustrates the commit-confirm logic flow described in detail in the prior sections.

End of description.

****

5.6.2 Gateway Server Commit-Confirm Transaction Log

The commit-confirm transaction log consists of a single table, PGA_CC_PENDING. This table contains a row for each in-flight Oracle transaction that includes the commit-confirm gateway. The table is maintained by the gateway server and is similar in function to the Oracle server's DBA_2PC_PENDING table. Note that a row is not inserted into this table until a COMMIT is received by the gateway and the row is deleted when a FORGET is received by the gateway. There is no involvement by the gateway during the PREPARE phase.

The PGA_CC_PENDING table contains the following columns:

- GLOBAL_TRAN_ID

  This is the Oracle Global Transaction ID for the transaction. It is identical to the corresponding column in the DBA_2PC_PENDING table.

- SIDE_NAME

This is the Side Information Profile name that was used by the gateway to allocate the APPC conversation with the target LU. It corresponds to the SIDENAME parameter passed to the PGAINIT gateway function.

- LU_NAME

  This is the fully-qualified partner LU name of the target LU. This value is either the LU name from the Side Information Profile or the LUNAME parameter passed to the PGAINIT gateway function. This name fully identifies the OLTP system on which the transaction was executed.

- MODE_NAME

  This is the Mode name that was used by the gateway to allocate the APPC conversation with the target LU. The value is either the Mode name from the Side Information Profile or the MODENAME parameter passed to the PGAINIT gateway function.

- TP_NAME

  This is the transaction program name executed at the target LU. The value is either the TP name from the Side Information Profile or the TPNAME parameter passed to the PGAINIT gateway function. This name fully identifies the OLTP transaction program that was executed.

6
PG4TCPMAP Commands (TCP/IP Only)

This chapter contains the commands and instructions necessary to operate the pg4tcpmap tool. This tool allows relevant parameters to map to a gateway using TCP/IP support for IMS Connect. The tool will be used to populate the PGA_TCP_IMSC table.

This chapter contains the following sections:

- "Preparation for Populating the PGA_TCP_IMSC Table" on page 6-2
- "Overview" on page 6-2
- "Populating the PGA_TCP_IMSC Table" on page 6-3
- "Before You Run the pg4tcpmap Tool" on page 6-4
- "pg4tcpmap Tool Commands" on page 6-6

6.1 Preparation for Populating the PGA_TCP_IMSC Table

If your gateway is using TCP/IP support for IMS Connect, then you must use the pg4tcpmap tool to prompt PGAINIT to provide the required TCP/IP parameters as input.

The pg4tcpmap tool must be run before executing any PL/SQL gateway statements in order to populate the PGA_TCP_IMSC table, which utilizes the corresponding TIPs.

Note that you do not need to rerun the pg4tcpmap tool for additional IMS transactions if they share the same IMS Connect attributes.

The PGA_TCP_IMSC table was created when you executed the **%ORACLE_HOME%\pg4appc\admin\pgaimsc.sql** script during your gateway configuration. If you need further information about creating the PGA_TCP_IMSC table, then refer to Chapter 10 of the *Oracle Procedural Gateway for APPC Installation and Configuration Guide*.

6.2 Overview

In a PGAINIT procedure call, the user must specify a Side Profile Name and TP Name. The values of these parameters will be inserted into a table named PGA_TCP_IMSC.

Configure userid and password before running gateway mapping tool

Before executing the pg4tcpmap tool, you must configure a valid userid and password and TNSNAMES alias for the Oracle Integrating Server where the PGA_TCP_IMSC table resides. You must specify the userid, password, and database in the PGA_TCP_USER, PGA_TCP_PASS, and PGA_TCP_DB parameters, respectively, located in the gateway initialization file **%ORACLE_HOME%\pg4appc\admin\init*sid*.ora**.

6.3 Populating the PGA_TCP_IMSC Table

Table 6–1 describes the parameter information contained in the column names, types and contents column found in the PGA_TCP_IMSC table.

Table 6–1   PGA_TCP_IMSC Table Columns

Column Name	Type	Content
SideProfileName	varchar2(8)	**This parameter has no SNA implication**. It is simply a name that is defined in the .ctl file for the PGAU utility. It represents a group of IMS transactions with similar IMS Connect attributes, such as time delay, socket type and IMS subsystem ID. Unique index.
HostName	varchar2(169) NOT NULL	The OLTP TCP/IP address or the hostname.
PortNumber	varchar2(17) NOT NULL	The OLTP port number.
ANDRS	char(1) NOT NULL	ANDRS specifies whether the client is sending: A = ACK: Positive Acknowledgement; N = NAK: Negative Acknowledgement; D = DEALLOCATE: Deallocate Connection; R = RESUME: Resume TPIPE; S = SENDONLY: Send only Acknowledgment or Deallocate. blank: no request for Acknowledgement or Deallocate. The default is "blank".
TIMER	char(1) NOT NULL	Time delay for the receive to the datastore after an ACK or RESUME TPIPE: D = default value X'00' .25 second; S = short wait X'01' through X'19': 01 to .25 second N = No Wait occurs I = Receive waits indefinitely. The default is "D".
SOCK	char(1) NOT NULL	Socket Connection Type T = Transaction Socket: P = Persistent Socket N = Non-persistent Socket The default is "T".
CLIENTID	char(8) NOT NULL	Specifies the name of the client ID that is used by IMS Connect. The default is 'null'.

Table 6–1   (Cont.) PGA_TCP_IMSC Table Columns

Column Name	Type	Content
COMMITMODE	char(1) NOT NULL	It specifies the commit mode:
		0 = the commit mode is 0;
		1 = the commit mode is 1
		The default is "1".
IMSDESTID	char(8) NOT NULL	Specifies the datastore names (IMS subsystem ID) 8 bytes.
		This parameter must be specified.
LTERM	char(8) NOT NULL	Specifies the IMS LTERM override. The default is "blank".
RACFGRPNAM	char(8) NOT NULL	Specifies the RACF group name.
		The default is "blank".
		You need to specify the RACF group name if you have set PGA_SECURITY_TYPE to PROGRAM.
		Refer to "PGA_SECURITY_TYPE" in Table B-1 "PGA Parameters on Gateway Using TCP/IP for IMS Connect" in the *Oracle Procedural Gateway for APPC Installation and Configuration Guide*.
		Refer to "TCP/IP Security Option SECURITY=PROGRAM" in Chapter 11 of the *Oracle Procedural Gateway for APPC Installation and Configuration Guide* to learn more about how to set the RACF userid and RACF password.

6.4 Before You Run the pg4tcpmap Tool

Follow these steps to prepare for running the pg4tcpmap tool before you run the gateway.

1. Set the ORACLE_HOME and ORACLE_SID for the Oracle Integrating Server.

2. Make certain that the user, PGAADMIN, has been created in the Oracle Integrating Server and you can talk to the database. Issue

   %ORACLE_HOME%\pg4appc\admin\pgacr8au.sql

3. The **init*sid*.ora** file must contain appropriate parameters. Set the following parameters:

   – PGA_TCP_USER

   – PGA_TCP_PASS

   – PGA_TCP_DB

   – If you intend to enable the tracing, you will also need to set the following parameters:

     – TRACE_LEVEL=255

     – LOG_DESTINATION=<valid directory>

Refer to Chapter 8, "Problem Determination" for information about tracing.

4. Make certain that the PGA_TCP_IMSC table has been created. Issue:

%ORACLE_HOME%\pg4appc\admin\pgaimsc.sql

Refer to Chapter 10, "Gateway Configuration Using the TCP/IP Communication Protocol" in the *Oracle Procedural Gateway for APPC Installation and Configuration Guide* for more information on the PGA_TCP_IMSC table and on creating the user PGAADMIN.

Figure 6–1 illustrates the relationship between the gateway, the database and the pg4tcpmap tool in mapping the Side Profile Name to TCP/IP and IMS Connect attributes in the PGA_TCP_IMSC table.

Figure 6–1 Mapping SNA Parameters to TCP/IP Using the pg4tcpmap Tool

Figure 6–1 illustrates how the pg4tcpmap tool maps the Side Profile Name to TCP/IP and IMS Connect attributes from the gateway into the PGA_TCP_IMSC table. The process of mapping information to TCP/IP is described in the surrounding text.

End of description.

****

A copy of the screen output file for the pg4tcpmap tool is located in Appendix B of the *Oracle Procedural Gateway for APPC Installation and Configuration Guide*.

An example of a trace file from a sample pg4tcpmap execution can be found in Chapter 8, "Problem Determination".

6.5 pg4tcpmap Tool Commands

There are two commands for the pg4tcpmap tool:

- one command inserts a row into the PGA_TCP_IMSC table;

- the other command deletes a row from the table, and the user must specify the predicate as "Side Profile Name".

6.5.1 Inserting a Row into the PGA_TCP_IMSC Table

Issue the following command from the gateway Oracle home **%ORACLE_HOME%\bin** directory:

```
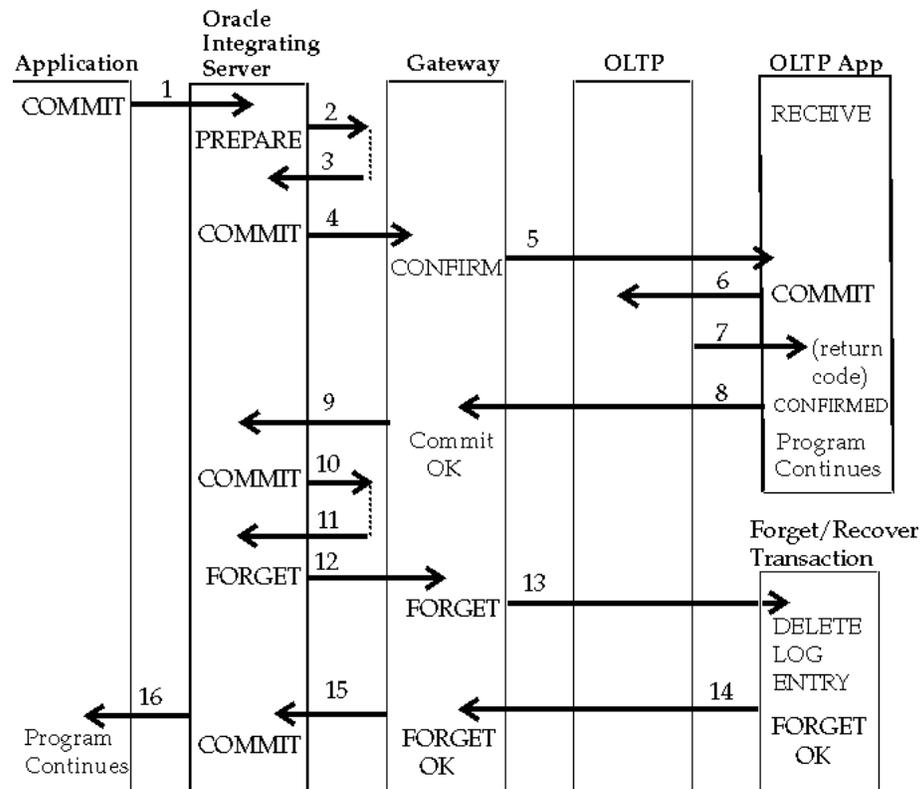C:\> pg4tcpmap
```

The following text appears:

```
PG4TCPMAP: Release 10.2.0.1.0 - Production on Wed Nov 19 17:43:15 2003
Copyright (c) 1982, 2003, Oracle. All rights reserved.
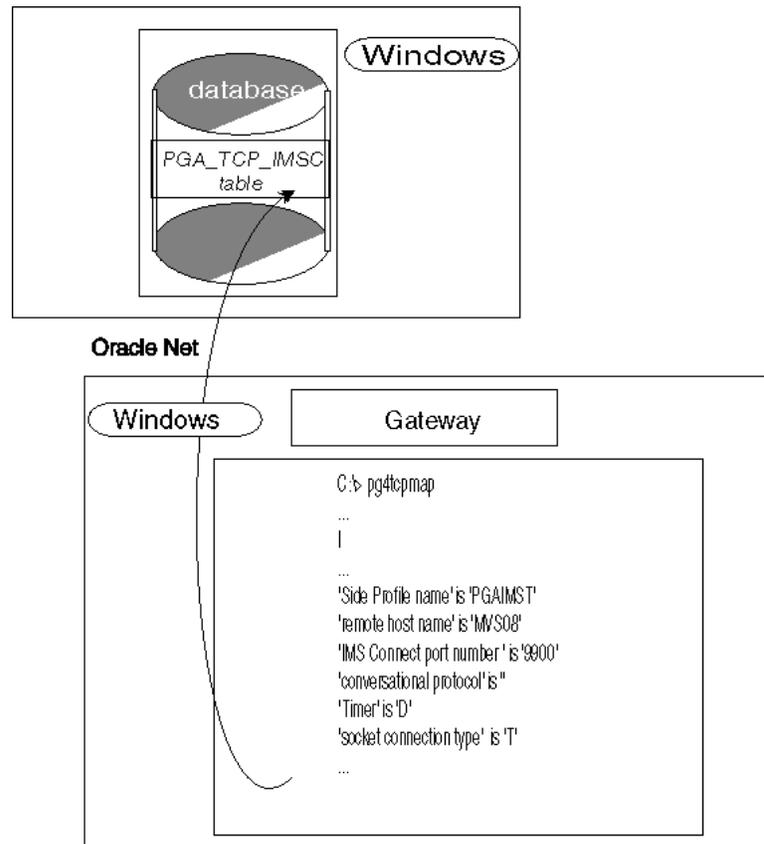
This tool takes the IMS Connect TCP/IP information, such as host name and port
number, and maps them to your TIPs.

You may use this tool to insert or delete IMS Connect TCP/IP information.
If you want to insert a row, Type "I"
If you want to delete a row, type "D"
```

Enter <i>, and after that, you need only enter the required parameters.

6.5.2 Deleting Rows from the PGA_TCP_IMSC Table

Issue the following command from the gateway Oracle home **%ORACLE_HOME%\bin** directory:

```
C:\> pg4tcpmap
```

The following text appears:

```
PG4TCPMAP: Release 10.2.0.1.0 - Production on Wed Nov 19 17:43:15 2003
Copyright (c) 1982, 2003, Oracle. All rights reserved.

This tool takes the IMS Connect TCP/IP information, such as host name and port
number, and maps them to your TIPs.

You may use this tool to insert or delete IMS Connect TCP/IP information.
If you want to insert a row, Type "I"
If you want to delete a row, type "D"
```

Enter <d>, and the pg4tcpmap tool will ask you what Side Profile Name you want to delete.

If the row does not exist, you will receive an ORA-1403 error message.

> **Note:** Do not use SQL*Plus to update the PGA_TCP_IMSC table. If you have problems or incorrect data in the table, use **%ORACLE_HOME%\pg4appc\admin\pgaimsc.sql** to re-create the table and its index.

6.5.3 Querying the PGA_TCP_IMSC Table

Use the regular SQL*Plus select statement to query the table. For example:

```
C:\> sqlplus userid/password@databasename
SQL> column hostname format A22
SQL> column portnumber format A6
SQL> select sideprofilename, hostname,portnumber,imsdestid,commitmode from
     pga_tcp_imsc;

SIDEPROF          HOSTNAME                PORTNU   IMSDESTI   C
---------------   ----------------------  ------   --------   -
IMSPGA            MVS08.US.ORACLE.COM     9900     IMSE       1
```

7

Client Application Development (TCP/IP Only)

This chapter discusses how you will call a TIP and control a remote host transaction if your gateway uses TCP/IP support for IMS Connect. It also provides you with the steps for preparing and executing a gateway transaction.

This chapter assumes:

- a remote host transaction (RHT) has already been written

- a TIP corresponding to the RHT has already been defined using the steps described in Chapter 3, "Creating a TIP".

- the PGA_TCP_IMSC mapping table has been populated, using the pg4tcpmap tool, with the SIDE PROFILE name, TCP/IP hostname, port number and other IMS Connect parameters.

This chapter contains the following sections:

7.1 Overview of Client Application

The Procedural Gateway Administration Utility (PGAU) generates a complete TIP using definitions you provide. The client application can then call the TIP to access the remote host transaction. Chapter 2, "Procedural Gateway Administration Utility", discusses the use of PGAU in detail.

This overview explains what you must do in order to call a TIP and control a remote host transaction.

The gateway receives PL/SQL calls from the Oracle Integrating Server and issues TCP/IP calls to communicate with a remote transaction program.

The following application programs make this possible:

1. an I/O PCB-enabled remote host transaction program

2. the PGA_TCP_IMSC mapping table that has been populated, using the pg4tcpmap tool, with the SIDE PROFILE name as well as the TCP/IP hostname, port number and other IMS Connect parameters.

3. a Transaction Interface Package (TIP). A TIP is a PL/SQL package that handles communication between the client and the gateway and performs datatype conversions between COBOL and PL/SQL.

4. PGAU generates the TIP specification for you. In the shipped samples, the PGAU-generated package is called **pgtflip.pkb**. This generated TIP includes at least three function calls that map to the remote transaction program:

   – **pgtflip_init** initializes the conversation with the remote transaction program

   – **pgtflip_main** exchanges application data with the remote transaction program

   – **pgtflip_term** terminates the conversation with the remote transaction program

   Refer to Appendix F, "Tip Internals" for more information about TIPs, if you are writing your own TIP or debugging.

5. a client application that calls the TIP

   The client application calls the three TIP functions with input and output arguments. In the example, the client application passes an input and the remote transaction and the remote transaction sends back the flipped input as an output.

Table 7–1 demonstrates the logic flow between the PL/SQL driver, the TIP, and the gateway using the example IMS Connect-IMS transaction.

Table 7–1   Logic Flow of IMS Connect-IMS Example

Client Application	Oracle TIP	Procedures Established Between the Gateway and the Remote Transaction (mainframe IMS)
calls tip_init	Calls PGAINIT	Gateway issues TCP/IP socket and connect to initiate the conversation with IMS Connect.
calls tip_main	Calls PGAXFER to send the input and receive the output	Gateway issues TCP/IP send() to IMS Connect. IMS Connect, through OTMA and XCF, talks to the IMS instance. IMS RECEIVE completes. IMS performs application logic and issues SEND back to gateway. The gateway issues TCP/IP receive(); receive completes.
calls tip_term	Call PGATERM	Gateway issues TCP/IP close().

A client application which utilizes the gateway to exchange data with a remote host transaction performs some tasks for itself and instructs the TIP to perform other tasks on its behalf. The client application designer must consequently know the behavior of the remote transaction and how the TIP facilitates the exchange.

The following sections provide an overview of remote host transaction behavior, how this behavior is controlled by the client application and how TIP function calls and data declarations support the client application to control the remote host transaction. These sections also provide background information about what the TIP does for the client application and how the TIP calls exchange data with the remote host transaction.

7.2 Preparing the Client Application

To prepare the client application for execution you must understand the remote host transaction requirements and then perform these steps:

1. Make sure that the pg4tcpmap tool has been used to map the SIDEPROFILE name, defined in the **.ctl** file for the PGAU utility, to TCP/IP and IMS Connect attributes.

   Refer to Chapter 6, "PG4TCPMAP Commands (TCP/IP Only)" in this guide, and to Chapter 10 of the *Oracle Procedural Gateway for APPC Installation and Configuration Guide* for detailed information about mapping parameters.

2. Make certain that you have identified the remote host transaction program facilities to be called.

3. Move relevant COBOL records layout (copybooks) to the gateway system for input to PGAU.

4. Describe the remote host transaction data and calls to the PG Data Dictionary (PG DD) with DEFINE DATA, DEFINE CALL, and DEFINE TRANSACTION statements.

5. Generate the TIP in the Oracle Integrating Server, using GENERATE.

6. Create the client application that calls the TIP public functions.

7. Grant privileges on the newly created package.

7.2.1 TIP Content and Purpose

The content of a PGAU-generated TIP reflects the calls available to the remote host transaction and the data that has been exchanged. Understanding this content helps when designing and debugging client applications that call the TIP.

A TIP is a PL/SQL package, and accordingly has two sections:

1. A Package Specification containing:

   - Public function prototypes and parameters, and

2. A Package Body containing:

   - Private functions and internal control variables

   - Public functions

   - Package initialization following the last public function.

The purpose of the TIP is to provide a PL/SQL callable public function for every allowed remote transaction program interaction. A remote transaction program interaction is a logically related group of data exchanges through one or more

PGAXFER RPC calls. This is conceptually similar to a screen or menu interaction in which several fields are filled in, the enter key is pressed, and several fields are returned to the user. Carrying the analogy further:

- the user might be likened to the TIP or client application

- fields to be filled in are IN parameters on the TIP function call

- fields returned are OUT parameters on the TIP function call

- screen or menu is the group of IN and OUT parameters combined

- a pressed enter key is likened to the PGAXFER remote procedural call (RPC)

The actual grouping of parameters that constitute a transaction call is defined by the user. The gateway places no restrictions on how a remote transaction program might correspond to a collection of TIP function calls, each call having many IN and OUT parameters.

PGA users typically have one TIP per remote transaction program. How the TIP function calls are grouped and what data parameters are exchanged on each call depends on the size, complexity and behavior of the remote transaction program.

Refer to Oracle's *PL/SQL User's Guide and Reference* for a discussion of how PL/SQL packages work. The following discussion covers the logic that must be performed within a TIP. Refer to the sample TIP and driver supplied in the **%ORACLE_HOME%\pg4appc\demo\IMS** directory in files **pgtflip.pkh, pgtflip.pkb**, and **pgtflipd.sql**.

7.2.2 Remote Host Transaction Types

From a procedural gateway application perspective, there are three main types of remote host transactions:

- transaction socket

- persistent socket

- non-persistent socket

You should be familiar with the remote host transaction types. Refer to the IBM *IMS Connect Guide and Reference* for a full description of these transaction types.

7.3 Ensuring TIP and Remote Transaction Program Correspondence

A remote host transaction program and its related TIP with client application must correspond on two key requirements:

- Parameter datatype conversion, which results from the way in which transaction DATA is defined. Refer to Appendix D, "Datatype Conversions" for a discussion of how PGAU-generated TIPs convert data based on the data definitions.

- TCP/IP send/receive synchronization, which results from the way in which transaction CALLs are defined

These DATA and CALL definitions are then included by reference in a TRANSACTION definition.

Make certain that the SIDEPROFILE name has been mapped to TCP/IP and IMS Connect attributes, using the pg4tcpmap tool.

7.3.1 DATA Correspondence

Using data definitions programmed in the language of the remote host transaction, the PGAU DEFINE DATA command stores in the PG DD the information needed for PGAU GENERATE to create the TIP function logic to perform:

- all data conversion from PL/SQL IN parameters supplied by the receiving remote host transaction

- all buffering into the format expected by the receiving remote host transaction

- all data unbuffering from the format supplied by the sending remote host transaction

- all data conversion to PL/SQL OUT parameters supplied by the sending remote host transaction

PGAU determines the information needed to generate the conversion and buffering logic from the data definitions included in the remote host transaction program. PGAU DEFINE DATA reads this information from files, such as COBOL copy books, or in-stream from scripts and saves it in the PG DD for repeated use. The Gateway Administrator needs to transfer these definition files from the remote host to the Oracle host where PGAU runs.

From the data definitions stored in the PG DD, PGAU GENERATE determines the remote host datatype and matches it to an appropriate PL/SQL datatype. It also determines data lengths and offsets within records and buffers and generates the needed PL/SQL logic into the TIP. Refer to the PGAU "DEFINE DATA" statement on page 2-9 in Chapter 2, "Procedural Gateway Administration Utility" and "Sample PGAU DEFINE DATA Statements" in Appendix G, "Administration Utility Samples" for more information.

All data that are referenced as parameters by subsequent calls must first be defined using PGAU DEFINE DATA. Simple data items, such as single numbers or character strings, and complex multi-field data aggregates, such as records or structures, can be defined. PGAU automatically generates equivalent PL/SQL variables and records of fields or tables for the client application to reference in its calls to the generated TIP.

As discussed, a parameter might be a simple data item, such as an employee number, or a complex item, such as an employee record. PGAU DEFINE DATA automatically extracts the datatype information it needs from the input program data definition files.

In this example, `FLIPIN` and `FLIPOUT` are the arguments to be exchanged.

```
PGTFLIP_MAIN(trannum,FLIPIN,FLIPOUT)
```

A PGAU DEFINE DATA statement must therefore be issued for each of these parameters:

```
DEFINE DATA FLIPIN
       PLSDNAME (FLIPIN)
       USAGE (PASS)
       LANGUAGE (IBMVSCOBOLII)
       (
       01 MSGIN PIC X(20).
       );

DEFINE DATA FLIPOUT
       PLSDNAME (flipout)
       USAGE (PASS)
       LANGUAGE (IBMVSCOBOLII)
       (
       01 MSGOUT PIC X(20).
       );
```

Note that a definition is not required for the `trannum` argument. This is the APPC conversation identifier and does not require a definition in PGAU.

7.3.2 CALL Correspondence

The requirement to synchronize TCP/IP send() and receive() means that when the remote transaction program expects data parameters to be input, it issues TCP/IP receive() to read the data parameters. Accordingly, the TIP must cause the gateway to issue TCP/IP send() to write the data parameters to the remote transaction program. The TIP must also cause the gateway to issue TCP/IP receive() when the remote transaction program issues TCP/IP send().

The PGAU DEFINE CALL statement specifies how the generated TIP is to be called by the client application and which data parameters are to be exchanged with the remote host transaction for that call. Each PGAU DEFINE CALL statement might specify the name of the TIP function, one or more data parameters, and the IN/OUT mode of each data parameter. Data parameters must have been previously defined with PGAU DEFINE DATA statements. Refer to "DEFINE CALL" on page 2-8 in Chapter 2, "Procedural Gateway Administration Utility" and "Sample PGAU DEFINE CALL Statements" in Appendix G for more information.

PGAU DEFINE CALL processing stores the specified information in the PG DD for later use by PGAU GENERATE. PGAU GENERATE then creates the following in the TIP package specification:

- declarations of public PL/SQL functions for each CALL defined with PL/SQL parameters for each DATA definition specified on the CALL

- declarations of the public PL/SQL data parameters

The client application calls the TIP public function as a PL/SQL function call, using the function name and parameter list specified in the PGAU DEFINE CALL statement. The client application might also declare, by reference, private variables of the same datatype as the TIP public data parameters to facilitate data passing and handling within the client application, thus sharing the declarations created by PGAU GENERATE.

In this example, the following PGAU DEFINE CALL statement must be issued to define the TIP public function:

```
DEFINE CALL FLIPMAIN
       PKGCALL (pgtflip_main)
       PARMS ((FLIPIN IN),(FLIPOUT OUT));
```

7.3.2.1 Flexible Call Sequence

The number of data parameters exchanged between the TIP and the gateway on each call can vary at the user's discretion, as long as the remote transaction program's SEND/RECEIVE requests are satisfied. For example, the remote transaction program data exchange sequence might be:

```
TCP/IP SEND      5 fields (field1-field5)
TCP/IP RECEIVE   1 fields (field6)
TCP/IP SEND      1 field (field7)
TCP/IP RECEIVE   3 fields (field8 - field10)
```

The resulting TIP/application call sequence could be:

```
tip_call1(parm1 OUT, <-- TCP/IP SEND field1 from remote TP
          parm2 OUT, <-- TCP/IP SEND field2 from remote TP
          parm3 OUT); <-- TCP/IP SEND field3 from remote TP

tip_call2(parm4 OUT, <-- TCP/IP SEND field4 from remote TP
          parm5 OUT); <-- TCP/IP SEND field5 from remote TP
tip_call3(parm6 IN OUT); --> TCP/IP RECEIVE field6 in remote TP
                        <-- TCP/IP SEND field7 from remote TP

tip_call4(parm8 IN, --> TCP/IP RECEIVE field8 into remote TP
          parm9 IN, --> TCP/IP RECEIVE field9 into remote TP
          parm10 IN); --> TCP/IP RECEIVE field10 into remote TP
```

To define these four public functions to the TIP, four PGAU DEFINE CALL statements must be issued, each specifying its unique public function name (tip_callx) and the data parameter list to be exchanged. Once a data item is defined using DEFINE DATA, it can be referenced in multiple calls in any mode (IN, OUT, or IN OUT). For example, parm5 could be used a second time in place of parm6 This implies the same data is being exchanged in both instances, received into the TIP and application on tip_call2 and returned, possibly updated, to the remote host in tip_call4.

Notice also that the remote transaction program's first five written fields are read by two separate TIP function calls, tip_call1 and tip_call2. This could also have been equivalently accomplished with five TIP function calls of one OUT parameter each or a single TIP function call with five OUT parameters. Then the remote transaction program's first read field (field6) and subsequent written field (field7) correspond to a single TIP function call (tip_call3) with a single IN OUT parameter (parm6).

This use of a single IN OUT parameter implies that the remote transaction program's datatype for field6 and field7 are both the same and correspond to the conversion performed for the datatype of parm6. If field6 and field7 were of different datatypes, then they have to correspond to different PL/SQL parameters (for example, parm6 IN and parm7 OUT). They could still be exchanged as two parameters on a single TIP call or one parameter each on two TIP calls, however.

Lastly, the remote transaction program's remaining three RECEIVE fields are supplied by tip_call4 parameters 8-10. They also could have been done with three TIP calls passing one parameter each or two TIP calls passing one parameter on one call and two parameters on the other, in either order. This flexibility permits the user to define

the correspondence between the remote transaction program's operation and the TIP function calls in whatever manner best suits the user.

7.3.2.2 Call Correspondence Order Restrictions

Each TIP public function first sends all IN parameters, before it receives any OUT parameters. Thus, a remote transaction program expecting to send one field and then receive one field must correspond to separate TIP calls.

For example:

```
tip_callO( parmO OUT); <-- TCP/IP SEND outfield from remote TP
```

PGAXFER RPC checks first for parameters to send, but finds none and proceeds to receive parameters:

```
tip_callI( parmI IN); --> TCP/IP RECEIVE infield to remote TP
```

PGAXFER RPC processes parameters to send and then checks for parameters to receive, but finds none and completes; therefore, a single TIP public function with an OUT parameter followed by an IN parameter does not work, because the IN parameter is processed first--regardless of its position in the parameter list.

7.3.3 TRANSACTION Correspondence

The remote host transaction is defined with the PGAU DEFINE TRANSACTION statement with additional references to prior definitions of CALLs that the transaction supports.

You specify the remote host transaction attributes, such as:

- transaction ID or name
- network address or location
- system type (such as IBM370)
- Oracle National Language of the remote host

> **Note:** The PL/SQL package name is specified when the transaction is defined; this is the name by which the TIP is referenced and which the public function calls to be included within the TIP. Each public function must have been previously defined with a PGAU DEFINE CALL statement, which has been stored in the PG DD.   If you do not specify a package name (TIP name) in the GENERATE statement, the transaction name you specified will become the package name by default. In that case, the transaction name (*tname)* must be unique and must be in valid PL/SQL syntax within the database containing the PL/SQL packages.
>
> For more information, refer to "DEFINE TRANSACTION" on page 2-11 in Chapter 2, "Procedural Gateway Administration Utility" and "Sample PGAU DEFINE TRANSACTION Statement" on page G-3 in Appendix G, "Administration Utility Samples".

In this example, the following DEFINE TRANSACTION statement is used to match this information with the inserted row in the PGA_TCP_IMSC table.

```
DEFINE TRANSACTION IMSFLIP
    CALL (FLIPMAIN)
    SIDEPROFILE(PGATCP)
    TPNAME(FLIP)
    NLS_LANGUAGE("american_america.us7ascii");
```

7.4 Calling the TIP from the Client Application

Once a TIP is created, a client application must be written to interface with the TIP. A client application that calls the TIP functions must include five logical sections:

- declaring TIP variables

- initializing the conversation

- exchanging data

- terminating the conversation

- error handling

7.4.1 Declaring TIP Variables

The user declarations section of the **tipname.doc** file documents the required declarations.

When passing PL/SQL parameters on calls to TIP functions, the client application must use the exact same PL/SQL datatypes for TIP function arguments as are defined by the TIP in its specification section. Assume, for example, the following is in the TIP specification, or **tipname.doc:**

```
FUNCTION tip_call1     tranuse,   IN      BINARY_INTEGER,
                       tip_var1   io_mode pls_type1,
                       tip_record io_mode tran_rectype)
RETURN INTEGER;

TYPE  tran_rectype is RECORD
      (rec_field1 pls_type1,
      ...
      rec_fieldN pls_typeN);
```

Where Table 7–2 provides a description of each of the parameters:

Table 7–2   Function Declarations

Parameter	Description
tip_call1	The TIP function name as defined in the package specification.
tranuse	The remote transaction instance parameter returned from the TIP init function identifying the conversation on which this TIP call is to exchange data.
tran_rectype	The PL/SQL record datatype declared in the tipname TIP specification. This is the same value as in the TYPE tran_rectype is RECORD statement.
pls_typeN	Is a PL/SQL atomic datatype.
rec_fieldN	Is a PL/SQL record field corresponding to a remote transaction program record field.

In the client application PL/SQL atomic datatypes should be defined as the exact same datatype of their corresponding arguments in the TIP function definition. The following should be coded in the client application before the BEGIN command:

```
appl_var  pls_type1;   /* declare appl variable for .... */
```

TIP datatypes need not be redefined. They must be declared locally within the client application, appearing in the client application before the BEGIN:

```
appl_record tipname.tran_rectype;  /* declare appl record */
```

Table 7–3 describes the meaning of each procedure declaration:

Table 7–3   Procedure Declarations

Item	Description
`appl_record`	Is a PL/SQL record exchanged with the TIP and used within the client application.
`tipname`	Is the PL/SQL package (TIP) name as stored in Oracle database. This is the same value as in the statement CREATE or REPLACE PACKAGE `tipname` in the TIP specification.
`tran_rectype`	Is the PL/SQL record datatype declared in the `tipname` TIP specification. This is the same value as in the TYPE `tran_rectype` is RECORD statement.

Refer to the *tipname*.**doc** content file for a complete description of the user declarations you can reference.

The client application calls the TIP public function as if it were any local PL/SQL function:

```
rc = tip_call1( tranuse,
                appl_var,
                appl_record);
```

In the TCP/IP IMS Connect example, the PL/SQL driver **pgtflipd.sql,** which is located in **%ORACLE_HOME%\pg4appc\demo\IMS** directory, is the client application and includes the following declaration:

```
...
...
CREATE or REPLACE PROCEDURE pgtflipd(mesgin IN CHAR) IS
trannum INTEGER :=0              /* transaction usage number    */
mesgout VARCHAR2(254);          /* the output parameter        */
rc INTEGER :=0                  /* PGA RPC return codes         */
term INTEGER :=0;               /* 1 if pgtflip_term called    */
...
...
```

7.4.2 Initializing the Conversation

The call to initialize the conversation serves several purposes:

- To cause the PL/SQL package, the TIP, to be loaded and to perform the initialization logic programmed in the TIP initialization section.

- To cause the TIP init function to call the PGAINIT remote procedural call (RPC), which in turn establishes communication with the remote transaction program (RTP), and returns a transaction instance number to the application.

Optionally, calls to initialize the conversation can be used to:

- Override default RHT/OLTP identification, network address attributes, and conversation security user ID and password.

- Specify what diagnostic traces the TIP is to produce. Refer to Chapter 8, "Problem Determination" for more information about diagnostic traces.

PGAU-generated TIPs provide four different initialization functions that client applications can call. These are overloaded functions which all have the same name, but vary in the types of parameters passed.

Three initialization parameters are passed:

- The transaction instance number for RHT socket file descriptor. The `tranuse` parameter is required on all TIP initializations.

- TIP diagnostic flags for TIP runtime diagnostic controls. The `tipdiag` parameter is optional. Refer to Chapter 8, "Problem Determination" for a discussion of TIP diagnostics.

- TIP default overrides for overriding OLTP and network attributes. The `override` parameter is optional.

The following four functions are shown as they might appear in the TIP Content documentation file. Examples of client application use are provided later.

```
TYPE override_Typ IS RECORD (
        tranname  VARCHAR2(2000),  /* Transaction Program   */
        transync  BINARY_INTEGER, /* RESERVED              */
        trannls   VARCHAR2(50),   /* RESERVED              */
        oltpname  VARCHAR2(2000),  /* Logical Unit          */
        oltpmode  VARCHAR2(2000),  /* LOG Mode Entry        */
        netaddr   VARCHAR2(2000),  /* Side Profile          */
        tracetag  VARCHAR2(2000),   /* gateway trace idtag  */

FUNCTION pgtflip_init(              /* init standard */
        tranuse IN OUT BINARY_INTEGER)
        RETURN INTEGER;

FUNCTION pgtflip_init(              /* init override */
        tranuse IN OUT BINARY_INTEGER,
        override IN override_Typ)
        RETURN INTEGER;

FUNCTION pgtflip_init(              /* init diagnostic */
        tranuse IN OUT BNARY_INTEGER,
        tipdiag IN CHAR)
        RETURN INTEGER;

FUNCTION pgtflip_init(              /* init over-diag */
        tranuse IN OUT BINARY_INTEGER,
        override IN override_Typ,
        tipdiag IN CHAR)
        RETURN INTEGER;
```

7.4.2.1 Transaction Instance Parameter

This transaction instance number (shown in examples as `tranuse`) must be passed to subsequent TIP exchange and terminate functions. It identifies to the gateway on which TCP/IP conversation--and therefore which iteration of a remote transaction program--the data is to be transmitted or communication terminated.

A single client application might control multiple instances of the same remote transaction program or multiple different remote transaction programs, all concurrently. The transaction instance number is the TIP's mechanism for routing the client application call through the gateway to the intended remote transaction program.

It is the responsibility of the client application to save the transaction instance number of each active transaction and pass the correct one to each TIP function called for that transaction.

The client application calls the TIP initialization function as if it were any local PL/SQL function. For example:

```
...
...
trannum INTEGER := 0;/* transaction usage number*/
...
...
BEGIN
 rc := pgtflip.pgtflip_init(trannum);
...
...
```

7.4.2.2 Overriding TIP Initializations

Note that in the preceding example the client application did not specify any remote transaction program name, network connection, or security information. The TIP has such information internally coded as defaults and the client application simply calls the appropriate TIP for the chosen remote transaction program. The client application can, however, optionally override some TIP defaults and supply security information.

You do not need to change any client applications that do not require overrides.

When the remote host transaction was defined in the PG DD, the DEFINE TRANSACTION statement specified certain default OLTP and network identification attributes which can be overridden:

- TPname

- Side Profile

Refer to "DEFINE TRANSACTION" in Chapter 2, "Procedural Gateway Administration Utility" for more information about the DEFINE TRANSACTION statement.

These PG DD-defined transaction attributes are generated into TIPs as defaults and can be overridden at TIP initialization time. This facilitates the use of one TIP, which can be used with a test transaction or system, and can later be used with a production transaction or system, without having to regenerate the TIP.

The `override_Typ` record datatype describes the various transaction attributes that can be overridden by the client application. The following overrides are currently supported:

- `tranname` can be set to override the value that was specified by the TPNAME parameter of the DEFINE TRANSACTION statement

- `netaddr` can be set to override the value that was specified by the SIDEPROFILE parameter of the DEFINE TRANSACTION statement

In addition to the transaction attributes defined in the PG DD, there are two security-related parameters, conversation security user ID and conversation security password, that can be overridden at TIP initialization time. The values for these parameters normally come from either the database link used to access the gateway or the Oracle database session. There are cases when the Oracle database user ID is not sufficient for accessing the OLTP system. The user ID and password overrides provide a way to specify those parameters to the OLTP system.

The following overrides are currently supported:

- `oltpuser` can be set to override the user ID used to initialize the conversation with the OLTP

- `oltppass` can be set to override the password used to initialize the conversation with the OLTP

The security overrides have an effect only if PGA_SECURITY_TYPE=PROGRAM is specified in the gateway initialization file, and the OLTP system is configured to accept a user ID and password on incoming conversation requests.

The `transync` (IMS Connect synclevel) and `trannls` (NLS character set) are defined in the override record datatype, but are reserved for future use. The RHT synclevel and NLS name cannot be overridden.

The client application might override the default attributes at TIP initialization for the following reasons:

- to start a different version of the RHT (such as production instead of test)

- to change the location of the OLTP containing the RHT (if the OLTP was moved due to migration or a switch to backup configuration)

Client applications requiring overrides can use any combination of override and initialization parameters and might alter the combination at any time without regenerating the TIP or affecting applications that do not override parameters.

To override the TIP defaults, an additional client application record variable must be declared as `override_Typ` datatype, values must be assigned to the override subfields, and the override record variable must be passed on the TIP initialization call from the client application. For example:

```
...
...
my_overrides pgtflip.override_Typ;   -- declaration
...
...
my_overrides.oltpname := 'IVTNO'; -- swap to production IMS
my_overrides.tranname := 'IVTNV';     -- new transaction name

BEGIN
  rc := pgtflip.pgtflip_init(trannum,my_overrides); -- init
  ...
  ...
```

Within the TIP, override attributes are checked for syntax problems and passed to the gateway server.

7.4.2.3 Security Considerations

The security requirements of the default and overridden OLTPs must be the same because the same gateway server is used in either conversation, as dictated by the database link names in the PGA RPC calls. The gateway server startup security mode is set at gateway server initialization time and passed unchanged to the OLTP at TIP or conversation initialization time.

7.5 Exchanging Data

The client application should pass the transaction instance number, returned from a previous tip_init call, to identify which remote transaction program is affected and to identify any client application data parameters to be exchanged with the remote transaction program.

In this IMS Connect inquiry example, we pass an employee number and receive an employee record back:

```
rc = pgtflip.pgtflip_main(trannum, /* transfer data         */
                          mesgin,   /* input parameter       */
                          mesgout); /* output parameter*/
```

7.5.1 Terminating the Conversation

The client application calls the TIP termination function as if it were any local PL/SQL function. For example:

```
...
...
term := 1;    /* indicate term called */
 rc := pgtflip.pgtflip_term(trannum,0); /* terminate normally  */
...
...
```

After a transaction instance number has been passed on a TIP terminate call to terminate the transaction, or after the remote transaction program has abended, that particular transaction instance number might be forgotten.

7.5.2 Error Handling

The client application should include an exception handler that can clean up any active TCP/IP conversations before the client application terminates. The sample client application provided in **pgtflipd.sql** contains an example of exception handling.

Gateway exceptions are reported in the range PGA-20900 to PGA-20999 and PGA-22000 to PGA 22099. When an exception occurs, the TIP termination function should be called for any active conversations that have been started by prior calls to the TIP initialization function.

For example:

```
EXCEPTION
 WHEN OTHERS THEN
  IF term = 0 THEN          /* terminate function not called yet */
   rc := pgtflip.pgtflip_term(trannum,1); /*terminate abnormally*/
  END IF;
 RAISE;
```

The remote transaction should also include provisions for error handling and debugging, such as writing debugging information to the IMS temporary storage queue area. Refer to the *PL/SQL User's Guide and Reference* for a discussion of how to intercept and handle Oracle exceptions.

7.5.3 Granting Execute Authority

The TIP is a standard PL/SQL package and execute authority must be granted to users who call the TIP from their client application. In this example, we grant execute on the **pgtflip** package to user SCOTT:

```
GRANT EXECUTE ON PGTFLIP TO SCOTT
```

Refer to the *Oracle Database Administrator's Guide* for further information.

7.6 Calling PG4TCPMAP

PGAU need not be modified in order to have a conversation on a gateway using TCP/IP. You use the APPC format of PGAU, but you will map parameters to TCP/IP using the pg4tcpmap tool.

To map the DEFINE TRANSACTION parameters using TCP/IP, you must have a valid input within the PGA_TCP_IMSC table before executing the application. Refer to Chapter 6, "PG4TCPMAP Commands (TCP/IP Only)" for information about setting up and using the mapping tool.

7.7 Executing the Application

Before executing the client application, ensure that a connection to the host is established and that the receiving partner is available. In this example we use PL/SQL driver PGTFLIPD to execute the IMS/IMS Connect inquiry. To execute this client application, enter from SQL*Plus:

```
set serveroutput on
execute pgtflipd('hello');
```

7.8 Application Development with Multi-Byte Character Set Support

COBOL presently only supports double byte character sets (DBCS) for PIC G datatypes.

PGAU processes IBM VS COBOLII PIC G datatypes as PL/SQL VARCHAR2 variables and generates TIPs which automatically convert the data according to the Oracle NLS_LANGUAGEs specified for the remote host data and the local Oracle data.

These Oracle NLS_LANGUAGEs can be specified as defaults for all PIC G data exchanged by the TIP with the remote transaction (see DEFINE TRANSACTION ... REMOTE_MBCS or LOCAL_MBCS). The Oracle NLS_LANGUAGEs for any individual PIC G data item can be further overridden (see REDEFINE DATA ... REMOTE or LOCAL_LANGUAGE).

DBCS data can be encoded in any combination of supported DBCS character sets. For example, a remote host application which allows different codepages for each field of data in a record is supported by the Oracle Procedural Gateway MBCS support.

Use of REDEFINE DATA ... REMOTE_LANGUAGE or LOCAL_LANGUAGE on PIC X items is also supported. Thus a TIP can perform DBCS or MBCS conversions for specified PIC X data fields, in addition to SBCS conversions by default for the remaining PIC X data fields. Default SBCS conversion is according to the DEFINE TRANSACTION... NLS_LANGUAGE and local Oracle default LANGUAGE environment values.

When PGAU is generating a TIP, the PIC G datatypes are converted to PL/SQL VARCHAR2 datatypes. After conversion by the TIP, received 'PIC G' VARCHAR2s can have a length less then the maximum due to deletion of shift-out and shift-in meta characters, and sent 'PIC G' RAW datatypes will have the shift-out and shift-in characters inserted as required by the remote host character set specified.

This is different from the conversions performed for PIC X data which is always a known fixed-length and hence CHAR datatypes are used in TIPs for PIC X data fields. However, even when the PIC X field contains DBCS or MBCS data, a CHAR variable is still used and padded with blanks if needed.

Some remote host applications bracket a PIC G field with PIC X bytes used for shift-out, shift-in meta-character insertion. Such a COBOL definition might look like:

```
01 MY_RECORD.
   05 SO PIC X.
   05 MY_MBCS_DATA PIC G(50).
   05 SI PIC X.
```

This is not processed correctly by PGAU, because all three fields are defined, and consequently treated, as separate data items when conversion is performed.

To be properly processed, the definition input to PGAU should be:

```
01 MY_RECORD.
   05 MY_MBCS_DATA PIC G(51).
```

The PGAU REDEFINE DATA statement can redefine the 3-field definition to the 1-field definition by specifying USAGE(SKIP) on fields SO and SI, and '05 MY_MBCS_DATA PIC G(51).' to redefine MY_MBCS_DATA. The three REDEFINE statements can be placed in the PGAU input control file, and thus the remote host definition need not be altered.

7.9 Privileges Needed to Use TIPs

Execute privileges must be explicitly granted to callers of TIPs or procedures. This privilege cannot be granted through a role.

Any TIP user wanting to trace a TIP must be granted execute privileges on the rtrace and ptrace procedures. Refer to the "Configuring PGAU" chapter appropriate for your communications protocol in the *Oracle Procedural Gateway for APPC Installation and Configuration Guide*, and the *Oracle Database 10g Application Developer's Guide* for more information. For example:

```
C:\> sqlplus pgaadmin\pw@database_specification_string
SQL> grant execute on pgaadmin.purge_trace to tip_user_userid;
SQL> grant execute on pgaadmin.read_trace to tip_user_userid;
```

After a TIP has been developed, the TIP user must be granted execute privileges on the TIP by the TIP owner. The TIP owner is usually PGAADMIN, but can be another user who has been granted either the PGDDDEF or PGDDGEN roles. For example:

```
C:\> sqlplus tip_owner\pw@database_specification_string
SQL> grant execute on tipname to tip_user_userid;
```

where `database_specification_string` is the Oracle Net identifier for the Oracle Integrating Server where the gateway UTL_RAW and UTL_PG components were installed. This is the same Oracle Integrating Server where the TIPs are executed and where grants on the TIPs are performed from the TIP owner user ID.

A SQL script for performing these grants is provided in the **%ORACLE_HOME%\pg4appc\admin** directory. The **pgddausr.sq**l script performs the grants for private access to the packages by a single TIP user. If private grants are to be used, the **pgddausr.sql** script must be run once for each TIP user's user ID.

To run these scripts, use SQL*Plus to connect to the Oracle Integrating Server as user PGAADMIN. From SQL*Plus, run the **pgddausr.sql** script from the **%ORACLE_HOME%\pg4appc\admin** directory. The script performs the necessary grants as previously described. You are prompted for the required user IDs, passwords, and database specification strings. If you are using private grants, repeat this step for each user ID requiring access to the packages.

No script has been provided to perform public grants. To do this, issue the following commands:

```
C:\> sqlplus tip_owner\pw@database_specification_string
SQL> grant execute on tipname to PUBLIC;
```

8

Problem Determination

This chapter discusses diagnostic techniques and aids for determining and resolving problems with data conversion, truncation, and conversation startup. It also describes how to collect the data when the debugging (trace) option is on.

You will want to trace the PL/SQL stored procedures only when you suspect problems. Do not run with tracing enabled during normal operations, because it will affect performance.

This chapter contains the following sections:

- "TIP Definition Errors" on page 8-2
- "Problem Analysis with PG DD Diagnostic References" on page 8-3
- "Problem Analysis with PG DD Select Scripts" on page 8-4
- "Data Conversion Errors" on page 8-5
- "Problem Analysis of Data Conversion and Truncation Errors" on page 8-6
- "Problem Analysis with TIP Runtime Traces" on page 8-8
- "TIP Runtime Trace Controls" on page 8-9
- "Suppressing TIP Warnings and Tracing" on page 8-11
- "Gateway Server Tracing" on page 8-12
- "Sample Gateway Server Initialization and Trace Output Files" on page 8-14

8.1 TIP Definition Errors

TIP definition errors occur when a TRANSACTION, CALL, or DATA entry in the PG DD is not properly defined.

Use the REPORT with DEBUG statement to list the PG DD contents and GENERATE DIAGNOSE(PKGEX(DR)) option to include corresponding ID numbers in the TIP.

Table 8–1 shows the mnemonic used to represent ID numbers and their correspondence with the following:

- PGAU REPORT with debug listings, GENERATE traces and TIPs

- PG DD tables and columns from which ID numbers are selected

- Oracle sequence objects from which ID numbers originate

Table 8–1   PGDD ID Numbers in Correspondence

PGAU REPORT/TIP	PDGG table(col)	Sequence Object
v# transaction version	pga_trans(version)	pga.transvers
v# call version	pga_call(version)	pga.callvers
v# data version	pga_data(version)	pga.datavers
t# transaction id#	pga_trans(trans#)	pga.transeq
c# call id#	pga_call(call#)	pga.callseq
	pga_call_parm(call#)	
d# data id#	pga_call_parm(data#)	pga.dataseq
	pga_data(data#)	
	pga_fields(data#)	
f# field id#	pga_fields(fld#)	pga.fieldseq
q# qualifier id#	pga_data_values(qual#)	pga.fieldseq
a# trans attribute id#	pga_trans_values(attr#)	pga.tattrseq
	pga_trans_attr(attr#)	
a# field attribute id#	pga_data_values(attr#)	pga.dtattseq
	pga_data_attr(attr#)	
e# environment	pga_environments(env#)	pga.envrseq
l# compiler/language	pga_compilers(comp#)	pga.compseq

These ID numbers can be used to associate the conversions performed in the TIP with the definitions stored in the PG DD.

The PG DD diagnostic references appear in TIPs generated with the PKGEX(DR) option as single line Comments:

```
-- PG DD type idno=nnn ...
```

The PG DD diagnostic references appear in REPORT with DEBUG listings before or to the right of their related definition entry as end-delimited Comments:

```
/* idno=nnn */
```

Refer to Appendix A, "Procedural Gateway for APPC Data Dictionary" for more information about PG DD, including a complete list of dictionary tables.

8.2 Problem Analysis with PG DD Diagnostic References

TIPs should be generated by the PGAU GENERATE command with the PKGEX(DR) diagnostic option, to include PG DD reference Comments in the TIP. These diagnostic references are Comments only and do not affect the runtime overhead of the TIP. Refer to Section 2.6.9, "GENERATE" on page 2-17 in Chapter 2, "Procedural Gateway Administration Utility" for a description of the PKGEX (DR) parameter.

1. Before defining the PL/SQL package, identify the transaction name, ID number (t#), and version (v#) from the TIP specification within the TIP.

2. Invoke PGAU REPORT WITH DEBUG specifying the same transaction name and version.

    REPORT selects definitions from the PG DD and produces a listing showing the DATA, CALL, and TRANSACTION definitions and the ID number of each user-supplied definition.

3. Compare the reported definitions with those used in the remote transaction program and identify all corresponding exchanges and the data formats transmitted.

4. Look for and investigate any mismatches, such as:

    - different numbers of send/receive calls

    - different sequence of send/receive calls

    - different parameter lists on send/receive calls

    - different data fields within each exchanged parameter

    - different lengths for each exchanged parameter

    - unsupported datatypes for each exchanged parameter

    - improperly initialized control fields for:
        - repeating group counts

          IBMVSCOBOLII affected clauses include

          OCCURS n TIMES DEPENDING ON field
        - remapped group criteria

          IBMVSCOBOLII affected clauses include

          REDEFINES field1 WHEN field2 = criteria

8.3 Problem Analysis with PG DD Select Scripts

PGAU GENERATE error messages and TRACE(OC) entries reference SQL SELECT
statements. Refer to Table 8–2 for the meaning of the name designations for each entry.

Table 8–2   Meaning of TRACE(OC) Output

Name	Entry
SED	Select Environment Data
STL	Select Transaction (latest version)
STV	Select Transaction (specific version)
STC	Select Transaction Calls
SPD	Select Parameter Data
SF	Select Fields
SFA	Select Field Attributes
SXF	Select conversion Formats
SXA	Select Attribute conversions

The SQL*Plus test scripts in Table 8–3 are provided to perform the identical SELECTS
as GENERATE performs to determine which PG DD rows are being used when the
TIP is generated. These files are loaded into the
%ORACLE_HOME%\pg4appc\admin directory during installation.

Table 8–3   SQL*Plus Test Scripts and Their Corresponding Entries

Script	Entry
pgddsed.sql	Select Environment Data
pgddstl.sql	Select Transaction (latest version)
pgddstv.sql	Select Transaction (specific version)
pgddstc.sql	Select Transaction Calls
pgddspd.sql	Select Parameter Data
pgddsf.sql	Select Fields
pgddsfa.sql	Select Field Attributes
pgddsxf.sql	Select Conversion Formats
pgddsxa.sql	Select Attribute conversions

The scripts are shown in the same order used by GENERATE and each script prompts
the SQL*Plus user for the required input. The information retrieved from a previous
select is often used as input to a subsequent select. If a you suspect that a PG DD field
entry has produced inaccurate data, browse the **.sql** files listed above to determine the
source of the problem. These files are loaded into the
%ORACLE_HOME%\pg4appc\admin directory during installation.

8.4 Data Conversion Errors

Data conversion errors are usually the result of:

- incorrect determination of datatype

  or

- incorrect specification of data position.

PGAU determination of the datatype is based on the values found in the PG DD, pga_fields(mask), and pga_fields(maskopts) columns. PGAU generates PL/SQL code to perform conversions based on the mask value:

- PIC X converted to CHAR with the same character length
- PIC G converted to CHAR with the same character length
- PIC 9 converted to NUMBER

Character datatype is presumed for all PIC X and PIC G mask values and conversion errors are more likely the result of position, length, and justification errors.

Determination of numeric datatype depends on several factors, including the combination of mask and maskopts values and how they apply to the actual remote host data in its internal format. Values for mask, maskopts, and data might conflict in unexpected ways. For example, an option such as USAGE IS COMP might be overridden if the data is in display format. While compilers occasionally perform such overrides correctly, they can cause unexpected results when exchanging data with systems coded in other languages.

To notify the user of such overrides, a warning function has been included in the following UTL_PG functions:

- MAKE _NUMBER_TO_RAW_FORMAT
- MAKE_RAW_TO_NUMBER_FORMAT
- NUMBER_TO_RAW
- RAW_TO_NUMBER

8.5 Problem Analysis of Data Conversion and Truncation Errors

Procedural Gateway for APPC data lengths are limited by PL/SQL to 32,763 bytes per APPC exchange and PL/SQL variable.

The following steps can be used to diagnose data conversion or truncation errors.

Refer to Chapter 3, "Creating a TIP" to review the proper values and definitions referenced in items 1 through 4 below:

1. Ensure that the COBOL definitions used in the RHT match the input to PGAU;

2. Ensure the RHT transmission buffers are of sufficient length;

3. **If your gateway uses SNA**: Ensure the RHT APPC call addresses the correct transmission buffer and uses the correct data length;

    If your gateway uses TCP/IP: Ensure the RHT I/O PCB call addresses the correct transmission buffer and uses the correct data length

4. Ensure the client application has declared the correct TIP datatypes used as arguments in the TIP calls.

5. Ensure that the client application is calling the TIP functions in the proper sequence (init, user-defined..., term), and that any input data to the RHT is correct. Also ensure that if multiple user-defined functions exist, they are being called in the proper sequence and passed the correct input values, if any.

    DBMS_OUTPUT calls can be inserted in the client application to trace its behavior.

    For more information about calling TIP functions in proper sequence, refer to the chapter on configuring the Oracle Integrating Server for first time installations, in the *Oracle Procedural Gateway for APPC Installation and Configuration Guide*.

6. Optionally, regenerate the TIP with diagnostic traces included and enable them. The following traces are particularly useful:

    - data conversion trace

    - function entry/exit trace

    - gateway exchange trace

    Refer to "Problem Analysis with TIP Runtime Traces" on page 8-8 for more information about traces; refer also to GENERATE on page 2-17 in Chapter 2, "Procedural Gateway Administration Utility".

    Note that the output of the trace is different for a gateway using SNA than for a gateway using TCP/IP. However, the method of invoking the trace is the same regardless of which communication protocol you are using.

    The gateway server tracing must also be enabled in **%ORACLE_HOME%\pg4appc\admin\init***sid***.ora**. Set the parameters SET TRACE_LEVEL=255 and SET LOG_DESTINATION=C:\oracle\pga\10.2\pg4appc\log

    Refer to "Gateway Server Tracing" on page 8-12 in this guide for more information about tracing.

    - **If your gateway is using SNA**: Refer to Appendix A, "Gateway Initialization Parameters for SNA Protocol" in your *Oracle Procedural Gateway for APPC Installation and Configuration Guide* for more information about these parameters;

- **If your gateway is using TCP/IP**: Refer to Appendix B, "Gateway Initialization Parameters for TCP/IP Communications Protocol" in your *Oracle Procedural Gateway for APPC Installation and Configuration Guide* for more information about these parameters.

Rerun the client application and examine the trace (see the next step for details).

To disable the trace, reset

SET TRACE_LEVEL=0

7. Examine the trace output.

The TIP trace output can be saved in a spool file, such as:

```
spool tipname.trc
```

TIP trace output is written to a named DBMS_PIPE and can be retrieved under SQL*Plus by issuing the following command:

```
exec rtrace('tipname');
```
or it can be purged by issuing the following command:

```
exec ptrace('tipname');
```

> **Note:** `tipname` is case-sensitive and must be specified exactly as it is in the TIP.

Gateway server trace output is written to a log file in a default directory path specified by the SET LOG_DESTINATION gateway parameter in **%ORACLE_HOME%\pg4appc\admin\init*sid*.ora**. For example:

```
SET LOG_DESTINATION=C:\oracle\pga\10.2\pg4appc\log
```

Refer to "Gateway Server Tracing" on page 8-12 for more information.

The gateway server log file can be viewed be editing the file or by issuing other system commands that display file contents. The log file can also be copied and saved to document problem symptoms.

8.6 Problem Analysis with TIP Runtime Traces

TIPs should be generated by the PGAU GENERATE command with the PKGEX(DC) diagnostic option to include TIP data conversion trace logic in the TIP. TIP function call trace logic is always included in every TIP. This is runtime trace instrumentation and has some overhead when tracing is enabled, but negligible overhead when tracing is disabled. Refer to Section 2.6.9, "GENERATE" on page 2-17 in Chapter 2, "Procedural Gateway Administration Utility" for more information.

1. Regenerate TIPs with the PKGEX(DC, DR) options and recompile the TIP body file, **tipname.pkb**. Avoid recompiling the TIP specification.

2. Revise the application that calls the TIP initialization function (tipname_init) to pass the trace flags parameter with data conversion and function call tracing enabled. Refer to "Controlling TIP Runtime Data Conversion Tracing" on page 8-10.

   If the problem causes an exception to be raised in the TIP and the application contains an exception handler, the application exception handler should be Commented out to prevent it from handling the exception and preventing the exception point of origin from being reported. When the TIP exception is next raised, its source line number in the TIP is reported. Record this information.

3. Execute the application with diagnostic TIP initialization.

   If the TIP trace pipe inlet overflows due to the application calls causing the TIP to write trace messages in the TIP trace pipe inlet, you have one minute from the start of the overflow condition to begin Step 4 and empty the TIP trace pipe.

   Otherwise, exception "ORA-20703 PGA-TIP: pipe send error" is issued, ending the diagnostic session, possibly before any relevant trace information is generated.

4. Retrieve and record the TIP trace message stream.

   Use SQL*Plus to connect to the same Oracle user ID executing the application or the user ID under which the TIP is executed. This establishes a second session from which the trace pipe outlet can be read, preventing the TIP trace pipe from overflowing at the TIP trace pipe inlet.

   a. Issue the command:

   ```
   set serveroutput on size nnnnn
   ```

   b. Issue the command to record the trace output:

   ```
   spool tipname.trc
   ```

   c. Issue the command to retrieve the trace stream:

   ```
   exec rtrace('tipname');
   ```

   If the application is long-running, repeat this command as often as needed until all trace messages have been retrieved.

5. If any exceptions are raised, note their prefix, number, and full message text.

6. Analyze the TIP trace message stream. A normal trace is shown for the **pgadb2i** TIP in Appendix G, "Administration Utility Samples".

8.7 TIP Runtime Trace Controls

Runtime trace control is the second parameter specified on a TIP initialization call. It is a CHAR(8) datatype of the following form:

```
rc := yourtip_init(trannum,'wxyz0000');
```

Table 8–4 describes the value of positions one to four:

Table 8–4   Values of Positions 1 through 4 on Second Parameter of TIP Call

Item	Description
position 1 (w)	controls UTL_RAW warning. A value of 0 suppresses warnings; a value of 1 issues warnings.
position 2 (x)	controls the function entry/exit tracing. A value of 0 suppresses the function entry/exit tracing; a value of 1 enables the function entry/exit tracing.
position 3 (y)	controls data conversion tracing. A value of 0 suppresses data conversion tracing; a value of 1 enables data conversion tracing.
position 4 (z)	controls gateway exchange tracing. A value of 0 suppresses gateway exchange tracing; a value of 1 enables gateway exchange tracing.

Positions 5 through 8 are reserved and ignored.

8.7.1 Generating Runtime Data Conversion Trace and Warning Support

Use PGAU to regenerate the TIP and specify the GENERATE parameter DIAGNOSE(PKGEX(DC)). This includes runtime PL/SQL code in the TIP which tests for and displays warnings of correct, but possibly unexpected NUMBER_TO_RAW and RAW_TO_NUMBER conversions.

Refer to Section 2.6.9, "GENERATE" on page 2-17 in Chapter 2, "Procedural Gateway Administration Utility" for more information about this parameter.

Recompile the TIP body under SQL*Plus. Avoid recompiling the TIP specification.

8.7.2 Controlling TIP Runtime Conversion Warnings

After the TIP has been regenerated, the issuance of runtime warnings is under control of the application. By default, warnings are suppressed and are only issued when they are enabled.

Errors and exceptions are always issued if they occur.

To enable the issuance of warnings, an additional parameter must be supplied when calling the TIP initialization function. This parameter is a CHAR(8) datatype and each character position controls a particular TIP runtime diagnostic function.

To enable warnings in yourtip, the client application should call the TIP initialization function with the statement:

```
rc := yourtip_init(trannum,'10000000');
```

The following is input to the TIP trace pipe inlet at initialization time:

```
"UTL_PG warnings enabled"
```

8.7.3 Controlling TIP Runtime Function Entry/Exit Tracing

To enable function entry/exit tracing in `yourtip`, the client application should call the TIP initialization function with the statement:

```
rc := yourtip_init(trannum,'01000000');
```

The following is input to the TIP trace pipe inlet at initialization time:

```
'function entry/exit trace enabled'
'tipname_init entered'
'time date/time stamp'
```

8.7.4 Controlling TIP Runtime Data Conversion Tracing

To enable data conversion tracing in `yourtip`, the client application should call the TIP initialization function with the following statement:

```
rc := yourtip_init(trannum,'00100000');
```

The following is input to the TIP trace pipe inlet at initialization time:

```
'data conversion trace enabled'
```

8.7.5 Controlling TIP Runtime Gateway Exchange Tracing

To enable runtime gateway exchange tracing in `yourtip`, the client application should call the TIP initialization function with the following statement:

```
rc := yourtip_init(trannum,'00010000');
```

The following is input to the TIP trace pipe inlet at initialization time:

```
'gateway exchange trace enabled'
```

8.8 Suppressing TIP Warnings and Tracing

After debugging is finished, there are two ways to suppress the following:

- data conversion tracing
- conversion warnings
- function entry/exit tracing
- gateway exchange tracing

You can:

1. Call the TIP initialization function without passing any diagnostic control parameters:

   ```
   rc := yourtip_init(trannum);
   ```

2. Call the TIP initialization function passing a revised diagnostic control parameter which disables all tracing and warnings:

   ```
   rc := yourtip_init(trannum,'00000000');
   ```

   A third method, described in Method C, removes the logic for:

   - data conversion tracing
   - conversion warnings

3. Generate the TIP again without:

   ```
   PKGEX(DC)
   ```

Or you can recompile the previous version of the TIP body if it was saved.

Methods A and B allow you to use the same TIP without alteration, but without tracing or warnings. These methods are reversible without alteration or replacement of the TIP. Tracing and warnings can be redisplayed should a problem recur.

Method C also suppresses data conversion tracing and warnings and incurs reduced overhead by avoiding tests, but is not reversible without regenerating the TIP or recompiling an alternate version with data conversion tracing and warning diagnostics imbedded.

The logic for function entry/exit and gateway exchange tracing is included in every TIP and cannot be removed. It can be disabled by method A or B.

8.9 Gateway Server Tracing

The gateway contains extensive tracing logic in the gateway remote procedural calls (RPCs), and the APPC-specific code. Tracing is enabled through gateway initialization parameters or dynamic RPC calls to the gateway. The trace provides information about the execution of the gateway RPC functions and about the execution of the APPC interface. The trace file contains a text stream written in chronological sequence of events. The trace is designed to assist application programmers with the debugging of their OLTP transaction programs and Oracle applications that communicate with those transaction programs through the gateway.

A single trace file is created for an entire gateway session from the time the database link is opened until it is closed. The trace can be directed to a specific path/filename or to a path (directory) only. In the first case, the file is overwritten each time a new session begins for the gateway being traced. When the trace target is a directory, a separate file with a generated name (containing the operating system process ID) is written for each gateway session. The latter approach must be used whenever the gateway to be traced might be the target of new sessions after the desired trace is written but before it can be copied and saved. Conversely, in some situations you might choose to create a distinct gateway system identifier used solely for tracing, and direct its trace to a single specific filename. This avoids the problem of an ever-increasing set of trace files when, for example, repeated attempts are necessary to reproduce or debug a problem. A fixed filename should never be used if there is any chance that an unexpected gateway session could overlay a useful trace.

8.9.1 Defining the Gateway Trace Destination

This section describes how to define the destination of trace files to the gateway, and how to cause the gateway to create the trace files during initialization. Note that this does not enable any gateway tracing, it merely defines the destination of any trace output produced when the gateway tracing is enabled.

1. Choose a gateway system identifier to trace. Decide whether you will be tracing an existing gateway system identifier or a new one created specifically for tracing. If a new system identifier will be used, configure the new system identifier exactly the same as the old one by creating a new **init*sid*.ora** (a copy of the old), entries in **listener.ora** as necessary, and a new Oracle database link.

    Test the new system identifier to ensure it works before proceeding.

2. In **%ORACLE_HOME%\pg4appc\admin**, edit the **init*sid*.ora** file so it contains the following:

    ```
    SET TRACE_LEVEL=255
    SET LOG_DESTINATION=logdest
    ```

    where *logdest* is the directory path for the trace output. The logfile is usually in **%ORACLE_HOME%\pg4appc\log**. Refer to the earlier discussion about "Problem Analysis of Data Conversion and Truncation Errors" on page 8-6 for more information.

    > **Note:** Misspelled parameter names in **init*sid*.ora** are not detected. The parameter is ignored.

Once these two steps are completed, the gateway opens the specified trace file during initialization. Each session on this system identifier writes a trace file as specified by the SET LOG_DESTINATION parameter described in Step 2 above.

If a directory path was specified, each trace file has a name of the form:

```
sid_pid.log
```

where *sid* is the gateway sid and *pid* is the operating system process ID of the gateway server expressed in decimal.

8.9.2 Enabling the Gateway Trace

There are two ways to enable the gateway server tracing. The first is to set the tracing options in the gateway initialization file, **init*sid*.ora**. The second is to use the additional PGA remote procedural call (RPC) function, PGATCTL, to dynamically control the tracing from within the Oracle application. The first method causes tracing to be performed for all users of the gateway system identifier and is recommended only when the use of the gateway system identifier can be limited to users actually needing the trace. The second method is more flexible and allows the application programmer to selectively trace events on a single gateway session without affecting the operation of other users' gateway sessions.

Before the gateway server trace is enabled, perform the tasks listed in "Defining the Gateway Trace Destination" on page 8-12.

8.9.2.1 Enabling the Gateway Trace Using Initialization Parameters

Edit the **init*sid*.ora** file, and add the following line at the end of the file (or, if a SET TRACE_LEVEL parameter is already specified, modify it):

```
SET TRACE_LEVEL=trace
```

where `trace` is a numeric value from 1 to 255 indicating which traces are to be enabled. For further information on the use of this parameter, refer to "PGA Parameters" in Appendix A of the *Oracle Procedural Gateway for APPC Installation and Configuration Guide*.

Once this step is completed, tracing is enabled for the desired gateway system identifier.

8.9.2.2 Enabling the Gateway Trace Dynamically from PL/SQL

The following is only needed for user-written TIPs. PGAU-generated TIPs automatically include the following facilities. Refer to "Controlling TIP Runtime Gateway Exchange Tracing" on page 8-10 for more information.

Make the following changes to the PL/SQL application that calls the Transaction Interface Package(s) to execute remote transaction(s).

1. Add a call to PGATCTL before any calls to TIP initialization functions are made:

```
PGATCTL@dblink(convid,
               traceF,
               traceS);
```

Where Table 8–5 describes the parameters in PGATCTL:

Table 8–5   PGATCTL Parameters

Parameter	Description
dblink	is the name of the database link to the gateway
convid	**For a gateway using SNA**: Conversation identifier returned by the PGAINIT function to be used to identify the conversation.
	For a gateway using TCP/IP: Socket file descriptor returned by the PGAINIT function to be used to identify the conversation
traceF	is the trace control function to be performed.
traceS	specifies which traces are to be enabled, as described previously in the discussion of the SET TRACE_LEVEL initialization parameter.

This call sets the trace flags for all new conversations started after the call to the value specified by `traceS`.

2. Recompile the PL/SQL application to pick up the new trace call.

8.10 Sample Gateway Server Initialization and Trace Output Files

The following sections provide sample output files for a gateway using either the SNA communications protocol or TCP/IP support for IMS Connect.

8.10.1 Sample Trace Output File for Gateway Using SNA

The following is an example of the trace output produced by a gateway server using the SNA communications protocol, where:

- the TRACE_LEVEL parameter is set to 255

- the gateway sid is PGA

- the Oracle home directory is **C:\oracle\pga\10.2**

- the PGA_CAPABILITY parameter is set to SINGLE_SITE

- the gateway server was invoked by an Oracle Net database link from another system.

```
HGAINIT: entered.
      : LOG_DESTINATION => C:\oracle\pga\10.2\log\PGA_356.log
      : TRACE_LEVEL => 255
      ORACLE_HOME: C:\oracle\pga\10.2,  LEN <18>, MAX_LEN <256>.
      ORACLE_SID: PGA,  LEN <3>, MAX_LEN <256>.
      FDS_CLASS: PG4APPC_1001,  LEN <12>, MAX_LEN <30>.
      FDS_INSTANCE: PGA,  LEN <3>, MAX_LEN <256>.
      FDS_CLASS_VERSION: 3,  LEN <1>, MAX_LEN <30>.
      : PGA_CAPABILITY = SINGLE_SITE
HGAINIT: exiting, rc = 0
HGALGON: entered.
      : reco    = 0
      : namel   = 8
      : name    = PGAADMIN
      : passwdl = 8
      : passwd  = pgaadmin
      : gpau is at 153188
      : PGA_CAPABILITY = SINGLE_SITE
      : up->gpaucap = 1
      : up->gpauflg = 00
```

```
        : PGA_SECURITY_TYPE = NONE
SGTAINI: entered.
SGTAINI: exiting,  rc = 0
HGALGON: exiting, rc = 0
HGAUPLC: entered.
HGAUPLC: exiting,  rc = 0
HGAUPLD: entered.
HGAUPLD: exiting,  rc = 0
HGABEGN: entered, hgagt = ffbec0ec
HGABEGN: exiting, rc =  <0>
HGAPDSC: entered.
HGAPFND: entered.
HGAPFND: exit - procedure 'PGAXFER' was found
HGAPDSC: exiting, rc =  <0>
HGAPDSC: entered.
HGAPFND: entered.
HGAPFND: exit - procedure 'PGAINIT' was found
HGAPDSC: exiting, rc =  <0>
HGAPDSC: entered.
HGAPFND: entered.
HGAPFND: exit - procedure 'PGATCTL' was found
HGAPDSC: exiting, rc =  <0>
HGAPDSC: entered.
HGAPFND: entered.
HGAPFND: exit - procedure 'PGATERM' was found
HGAPDSC: exiting, rc =  <0>
HGACOMM: entered, keepinfo = 0
HGACOMM: exiting, rc = 0
HGABEGN: entered, hgagt = ffbec0ec
HGABEGN: exiting, rc =  <0>
HGAPEXE: entered.
HGAPFND: entered.
HGAPFND: exit - procedure 'PGAINIT' was found
GPACINI: entered.
GPACLOC: executed
GPACLOC: exit, rc = 0
PGAINIT: TP name      = 'IVTNO'
PGAINIT: LU name      = ''
PGAINIT: Mode name    = 'ORAPLU62'
PGAINIT: Profile name = 'IMSPGA'
PGAINIT: Sync level   = 0
PGAINIT: Userid       = PGAADMIN
PGAINIT: passwd       = pgaadmin
GPACAEC: entered
GPACAEC: exiting, rc = 0
    GPAC allocated at 188294
SGPAALL: entered.
    SGPA allocated at 188b30
    CMINIT issued: profile = IMSPGA
    CMINIT complete: rc = 0, errno = 11
    CMEPLN issued
    CMEPLN complete: rc = 0, errno = 11, luname = IMSPGA
    CMSTPN issued: tpname = IVTNO
    CMSTPN complete: rc = 0, errno = 11
    CMSMN issued: modename = ORAPLU62
    CMSMN complete: rc = 0, errno = 11
    XCSCST issued: security type = 0
    XCSCST complete: rc = 0, errno = 11
    CMSSL issued: synclevel = 0
    CMSSL complete: rc = 0, errno = 11
    CMSPTR issued: type = 1
    CMSPTR complete: rc = 0, errno = 11
    CMALLC issued
    CMALLC complete: rc = 0, errno = 11
SGPAALL: exiting, rc = 0
GPACAEC: entered
```

```
              GPACAEC: exiting, rc = 0
              PGAINIT: Conversation id - @ 18839c
                0000:  00000001 01000000 00000000        *............*............*
              GPACINI: exiting rc = <0>
              HGAPEXE: exiting, rc =  <0>
              HGAPEXE: entered.
              HGAPFND: entered.
              HGAPFND: exit - procedure 'PGAXFER' was found
              GPACXFR: entered
              GPACLOC: executed
              GPACLOC: exit, rc = 0
                  GPAU found at 153188
                  GPAC found at 188294
                  sbufsz = 49
                  rbufsz = 32763
                  slensz = 8
                  rlensz = 8
              PGAXFER: Conversation id - @ 187ddc
                0000:  00000001 01000000 00000000        *............*............*
              PGAXFER: Send buffer length = 49
              PGAXFER: Receive buffer length = 89
              PGAXFER: Send lengths array - @ 187d6c
                0000:  00000001 00000031                 *.......1*........*
              PGAXFER: Receive lengths array - @ 1ac528
                0000:  00000001 00000059                 *.......Y*........*
              PGAXFER: Send buffer - @ 187d9c
                0000:  40404040 C4C9E2D7 D3C1E840 9381A2A3 *@@@@.......@....*   DISPLAY last*
                0010:  F1404040 40404040 40404040 40404040 *.@@@@@@@@@@@@@@@*1          *
                0020:  40404040 40404040 40404040 40404040 *@@@@@@@@@@@@@@@@*          *
                0030:  40                                 *@* *
              GPACAEC: entered
              GPACAEC: exiting, rc = 0
                  sends = 1
                  receives = 1
                  slens[1] = 49
              SGPASND: entered, type = 2
              SGPARTS: entered, deal = 0
              SGPARTS: exiting, rc = 0, deal = 0.
                  CMSST issued: send type = 3
                  CMSST complete: rc = 0, errno = 11
              Sending data: - @ 187d9c
                0000:  40404040 C4C9E2D7 D3C1E840 9381A2A3 *@@@@.......@....*   DISPLAY last*
                0010:  F1404040 40404040 40404040 40404040 *.@@@@@@@@@@@@@@@*1          *
                0020:  40404040 40404040 40404040 40404040 *@@@@@@@@@@@@@@@@*          *
                0030:  40                                 *@* *
                  CMSEND issued: length = 49
                  CMSEND complete: rc = 0, errno = 11, rts = 0
              SGPASND: exiting, rc = 0
                  rlens[1] = 89
              SGPARCV: entered.
                  CMECS issued
                  CMECS complete: rc = 0, errno = 11, state = 4
                  CMRCV issued: length = -4276988
                  CMRCV complete: rc = 0, errno = 11, rts = 0, dr = 2, sr = 0, length = 89
              Received data: - @ 1a38e8
                0000:  C5D5E3D9 E840E6C1 E240C4C9 E2D7D3C1 *.....@...@......*ENTRY WAS DISPLA*
                0010:  E8C5C440 40404040 40404040 40404040 *...@@@@@@@@@@@@@*YED           *
                0020:  40404040 40404040 C4C9E2D7 D3C1E840 *@@@@@@@@.......@*      DISPLAY *
                0030:  D3C1E2E3 F1404040 4040C6C9 D9E2E3F1 *.....@@@@......*LAST1     FIRST1*
                0040:  40404040 F860F1F1 F160F1F1 F1F1C4F0 *@@@@.`...`......*   8-111-1111D0*
                0050:  F161D9F0 F1F0F0F0 F1       *.a.......*1/R010001*
                  conversation state is now RECEIVE
              SGPARCV: exiting, rc = 0
              PGAXFER: Receive buffer length = 89
              PGAXFER: Receive lengths array - @ 1ac528
                0000:  00000001 00000059                 *.......Y*........*
```

```
PGAXFER: Receive buffer - @ 1a38e8
  0000: C5D5E3D9 E840E6C1 E240C4C9 E2D7D3C1 *.....@...@......*ENTRY WAS DISPLA*
  0010: E8C5C440 40404040 40404040 40404040 *...@@@@@@@@@@@@@*YED             *
  0020: 40404040 40404040 C4C9E2D7 D3C1E840 *@@@@@@@@.......@*        DISPLAY *
  0030: D3C1E2E3 F1404040 4040C6C9 D9E2E3F1 *.....@@@@@......*LAST1     FIRST1*
  0040: 40404040 F860F1F1 F160F1F1 F1F1C4F0 *@@@@.`...`......*   8-111-1111D0*
  0050: F161D9F0 F1F0F0F0 F1                 *.a.......*1/R010001*
GPACAEC: entered
GPACAEC: exiting, rc = 0
GPACXFR: exiting rc = <0>
HGAPEXE: exiting, rc =  <0>
HGAPEXE: entered.
HGAPFND: entered.
HGAPFND: exit - procedure 'PGATERM' was found
GPACTRM: entered
GPACLOC: executed
GPACLOC: exit, rc = 0
    GPAU found at 153188
    GPAC found at 188294
PGATERM: Conversation id - @ ffbebf04
  0000:  00000001 01000000 00000000         *............*............*
PGATERM: Termination type = 0
GPACAEC: entered
GPACAEC: exiting, rc = 0
SGPADEA: entered, type = 0
SGPARTS: entered, deal = 1
    CMRTS issued
    CMRTS complete: rc = 0, errno = 11
    CMRCV issued: length = 0
    CMRCV complete: rc = 18, errno = 11, rts = 0, dr = 0, sr = 0, length = 0
    conversation state is now RECEIVE
SGPARTS: exiting, rc = 0, deal = 1.
SGPADEA: exiting, rc = 0
    cp  = 188294
    cpp = 0
    cpn = 0
    up->gpaucpt = 0
    freed gpac at 188294
GPACTRM: exiting.
HGAPEXE: exiting, rc =  <0>
HGACOMM: entered, keepinfo = 0
HGACOMM: exiting, rc = 0
HGABEGN: entered, hgagt = ffbec0ec
HGABEGN: exiting, rc =  <0>
HGAPDSC: entered.
HGAPFND: entered.
HGAPFND: exit - procedure 'PGAXFER' was found
HGAPDSC: exiting, rc =  <0>
HGADAFR: entered.
HGADAFR: exit <0>
HGAPDSC: entered.
HGAPFND: entered.
HGAPFND: exit - procedure 'PGAINIT' was found
HGAPDSC: exiting, rc =  <0>
HGADAFR: entered.
HGADAFR: exit <0>
HGAPDSC: entered.
HGAPFND: entered.
HGAPFND: exit - procedure 'PGATCTL' was found
HGAPDSC: exiting, rc =  <0>
HGADAFR: entered.
HGADAFR: exit <0>
HGAPDSC: entered.
HGAPFND: entered.
HGAPFND: exit - procedure 'PGATERM' was found
HGAPDSC: exiting, rc =  <0>
```

```
HGADAFR: entered.
HGADAFR: exit <0>
HGACOMM: entered, keepinfo = 0
HGACOMM: exiting, rc = 0
HGABEGN: entered, hgagt = ffbec0ec
HGABEGN: exiting, rc =  <0>
HGAPEXE: entered.
HGAPFND: entered.
HGAPFND: exit - procedure 'PGAINIT' was found
GPACINI: entered.
GPACLOC: executed
GPACLOC: exit, rc = 0
GPACAEC: entered
GPACAEC: exiting, rc = 0
GPACXFR: exiting rc = <0>
HGAPEXE: exiting, rc =  <0>
HGAPEXE: entered.
HGAPFND: entered.
HGAPFND: exit - procedure 'PGATERM' was found
GPACTRM: entered
GPACLOC: executed
GPACLOC: exit, rc = 0
    GPAU found at 153188
    GPAC found at 190a0c
PGATERM: Conversation id - @ ffbebf04
  0000:  00000002 01000001 00000000        *............*............*
PGATERM: Termination type = 0
GPACAEC: entered
GPACAEC: exiting, rc = 0
SGPADEA: entered, type = 0
SGPARTS: entered, deal = 1
    CMRTS issued
    CMRTS complete: rc = 0, errno = 11
    CMRCV issued: length = 0
    CMRCV complete: rc = 18, errno = 11, rts = 0, dr = 0, sr = 0, length = 0
    conversation state is now RECEIVE
SGPARTS: exiting, rc = 0, deal = 1.
SGPADEA: exiting, rc = 0
    cp  = 190a0c
    cpp = 0
    cpn = 0
    up->gpaucpt = 0
    freed gpac at 190a0c
GPACTRM: exiting.
HGAPEXE: exiting, rc =  <0>
HGACOMM: entered, keepinfo = 0
HGACOMM: exiting, rc = 0
HGADAFR: entered.
HGADAFR: exit <0>
HGADAFR: entered.
HGADAFR: exit <0>
HGADAFR: entered.
HGADAFR: exit <0>
HGADAFR: entered.
HGADAFR: exit <0>
HGALGOF: entered.
HGALGOF: exiting,  rc = 0
HGAEXIT: entered.
HGAEXIT: exiting, rc = 0
```

8.10.2 Sample Trace Output File for Gateway Using TCP/IP

The following is an example of the trace output produced by a gateway server using the TCP/IP communication protocol, where:

- the TRACE_LEVEL parameter is set to 255

- the gateway *sid* is PGA

- the Oracle home directory is **C:\oracle\pga\10.2.**

- the PGA_CAPABILITY parameter is set to SINGLE_SITE

- the gateway server was invoked by an Oracle Net database link from another system.

```
HGTINIT: entered.
pg4appc Version 10.2.0.1.0
      : LOG_DESTINATION => C:\oracle\pga\10.2\pg4appc\log\PGA_688.log
      : TRACE_LEVEL => 255
      ORACLE_HOME: C:\oracle\pga\10.2,  LEN <18>, MAX_LEN <256>.
      ORACLE_SID: PGA,  LEN <3>, MAX_LEN <256>.
      FDS_CLASS: PG4APPC_1001,  LEN <12>, MAX_LEN <30>.
      FDS_INSTANCE: PGA,  LEN <3>, MAX_LEN <256>.
      FDS_CLASS_VERSION: 3,  LEN <1>, MAX_LEN <30>.
       : HGT_CAPABILITY = SINGLE_SITE
HGTINIT: exiting, rc = 0
HGTLGON: entered.
      : reco    = 0
      : namel   = 8
      : name    = PGAADMIN
      : passwdl = 8
      : passwd  = pgaadmin
      : hgtlHgtUserP is at 153990
      : HGT_CAPABILITY = SINGLE_SITE
      : hgtlHgtUserP->hgtUCap = 1
      : hgtlHgtUserP->hgtTCap = SINGLE_SITE
      : HGT_SECURITY_TYPE = PROGRAM
      : hgtlHgtUserP->hgtUSec = 2
      : PGA_TCP_USER = pgaadmin
      : PGA_TCP_PASS = pgaadmin
      : PGA_TCP_DB = R901
HGTLGON: exiting, rc = 0
HGTUPLC: entered.
HGTUPLC: exiting,  rc = 0
HGTUPLD: entered.
HGTUPLD: exiting,  rc = 0.
HGTBEGN: entered, hgtgt = ffbec0ec
HGTBEGN: exiting, rc =  <0>
HGTPDSC: entered.
HGTPFND: entered.
HGTPFND: exit - procedure 'PGAINIT' was found.
HGTPDSC: exiting, rc =  <0>
HGTPDSC: entered.
HGTPFND: entered.
HGTPFND: exit - procedure 'PGAXFER' was found.
HGTPDSC: exiting, rc =  <0>
HGTPDSC: entered.
HGTPFND: entered.
HGTPFND: exit - procedure 'PGATERM' was found.
HGTPDSC: exiting, rc =  <0>
HGTCOMM: entered, keepinfo = 0
HGTCOMM: exiting, rc = 0
```

```
HGTBEGN: entered, hgtgt = ffbec0ec
HGTBEGN: exiting, rc =  <0>
HGTPEXE: entered.
HGTPFND: entered.
HGTPFND: exit - procedure 'PGAINIT' was found.
GPTINI: entered.
      TPName is FLIP
      LUName is
      LogModeName is
      SidePorfileName is IMSPGA
      Uid PGAADMIN.
      Pass PGAADMIN.
gptOciLgon: entered
    OCIEnvCreate, rc = 0
    OCIHandleAlloc -- ERROR,  rc = 0
    Allocation a server context handle, rc = 0
    Allocation a service context handle, rc = 0
    Allocation a user handle, rc = 0
    OCIHandleAlloc -- STMT  rc = 0
    Create a server context, rc = 0
    Set svr. attr. in the svc. sess. handle, rc = 0
    Set usr. attr. in the usr. sess. handle, rc = 0
    Set pswd. attr. in the usr. sess. handle, rc = 0
    OCISessionBegin, rc = 0
    set usr. sess. in the svc. context handle, rc 0
gptOciLgon Exit  -- RC = 0.
gptOciSelStmt: entered.
    OCIStmtPrepare
    The select statment is:
    SELECT HOSTNAME,PORTNUMBER,ANDRS,TIMER,SOCK,CLIENTID,
          COMMITMODE,IMSDESTID,LTERM,RACFGRPNAM
       FROM PGA_TCP_IMSC
       WHERE SIDEPROFILENAME ='IMSPGA'
    OCIStmtPrepare, rc = 0
    Defining output variable HostName.
    OCIDefineByPos for  HostName, rc = 0
    Defining output variable Port Number.
    OCIDefineByPos for  Port Number, rc = 0
    Defining output variable ANDRS.
    OCIDefineByPos for  ANDRS, rc = 0
    Defining output variable Timer.
    OCIDefineByPos for  Timer, rc = 0
    Defining output variable Socket
    OCIDefineByPos for  Socket, rc = 0
    Defining output variable Client ID.
    OCIDefineByPos for  Client ID, rc = 0
    Defining output variable Commit Mode.
    OCIDefineByPos for Commit Mode, rc = 0
    Defining output variable IMS Destination ID.
    OCIDefineByPos for  IMS Destination ID, rc = 0
    Defining output variable LTERM
    OCIDefineByPos for LTERM, rc = 0
    Defining output RACF Group Name.
    OCIDefineByPos for RACF Group Name, rc = 0
    OCIStmtExecute complete, rc = 0
    Host Name = 'MVS08.US.ORACLE.COM', LEN = 19
    Port Number = '9900', LEN = 4
    conversation protocol = ' ', LEN = 1
    IRM Timer = 'D', LEN = 1
    socket connection type = 'T', LEN = 1
```

```
    client ID = '        ', LEN = 8
    commit mode = '1', LEN = 1
    IMS destination ID = 'IMSE    ', LEN = 8
    LTERM = '        ', LEN = 8
    RACF group name = '$DEV    ', LEN = 8
gptOciSelStmt Exit  -- RC = 0.
gptOciLgof: entered.
    OCISessionEnd, rc = 0
    OCIServerDetach, rc = 0
    OCIHandleFree -- Error, rc = 0
    OCIHandleFree -- SERVER, rc = 0
    OCIHandleFree -- SVCCTX, rc = 0
    OCIHandleFree -- SESSION, rc = 0
     OCIHandleFree -- STMT, rc = 0
gptOciLgof Exit  -- RC = 0.
        Host is MVS08.US.ORACLE.COM
        Port is 9900
        CID->num = 10.
GPTIINI Exit  -- RC = 0.
HGTPEXE: exiting, rc =  <0>.
HGTPEXE: entered.
HGTPFND: entered.
HGTPFND: exit - procedure 'PGAXFER' was found.
GPTXFR: entered.
        CID->num = 10.
        Beginning of Send Series.
        The length for 'send total msg len' is 102.
        The RC for 'send total msg len' is 4.
        The RC for 'send (IRM_LEN) LL' is 2.
        The RC for 'send (IRM_RSV) ZZ'  is 2.
        The RC for length for 'send identifier' is 8.
        The RC for 'Reserved full word 0' is 4.
        The RC for '(IRM_F5)' is 1.
        The RC for '(IRM_TIMER)' is 1.
        The RC for '(IRM_SOCK)' is 1.
        The RC for '(IRM_ES)' is 1.
        The RC for 'Client ID' is 8.
        The RC for 'FLG 1' is 1.
        The RC for 'commit mode' is 1.
        The RC for 'sync level' is 1.
        The RC for 'Conversational Protocol' is 1.
        The RC for 'transaction code' is 8.
        The RC for 'datastore ID' is 8.
        The RC for 'LTERM' is 8.
        The RC for 'RACF User ID' is 8.
        The RC for 'RACF group name' is 8.
        The RC for 'RACF password' is 8.
        The RC for 'send LL'  is 2.
        The RC for 'send ZZ'  is 2.
        The RC for 'transaction code'  is 5.
        The RC for 'send segment(buffer)' is 5.
        The len of buffer is 5.
        The Actual Sent Buffer - @ 184d8c
  0000: 68656C6C 6F                        *hello*..%%?*
        Send final LL ZZ.
        Signal no more data to IMS Connect.
        The length for 'send LL'  is 2.
        The RC for 'send ZZ'  is 2.
        End of send.
        The RC for 'read total len' is 2.
```

```
               The RC for 'Read ZZ' is 2.
               The RC for 'Read received buffer' is 9.
       The Received Buffer - @ ffbdbd04
         0000:  4F4C4C45 48                        *OLLEH*|<<..*
       GPTXFR Exit  -- RC = 0.
       HGTPEXE: entered.
       HGTPFND: entered.
       HGTPFND: exit - procedure 'PGAXFER' was found.
       GPTXFR: entered.
               CID->num = 10.
               Beginning of Send Series.
               The length for 'send total msg len' is 147.
               The RC for 'send total msg len' is 4.
               The RC for 'send (IRM_LEN) LL' is 2.
               The RC for 'send (IRM_RSV) ZZ'  is 2.
               The RC for length for 'send identifier' is 8.
               The RC for 'Reserved full word 0' is 4.
               The RC for '(IRM_F5)' is 1.
               The RC for '(IRM_TIMER)' is 1.
               The RC for '(IRM_SOCK)' is 1.
               The RC for '(IRM_ES)' is 1.
               The RC for 'Client ID' is 8.
               The RC for 'FLG 1' is 1.
               The RC for 'commit mode' is 1.
               The RC for 'sync level' is 1.
               The RC for 'Conversational Protocol' is 1.
               The RC for 'transaction code' is 8.
               The RC for 'datastore ID' is 8.
               The RC for 'LTERM' is 8.
               The RC for 'RACF User ID' is 8.
               The RC for 'RACF group name' is 8.
               The RC for 'RACF password' is 8.
               The RC for 'send LL'  is 2.
               The RC for 'send ZZ'  is 2.
               The RC for 'transaction code'  is 6.
               The RC for 'send segment(buffer)' is 49.
               The len of buffer is 49.
               The Actual Sent Buffer - @ 1846e8
         0000:  20202020 44495350 4C415920 6C617374 *    DISPLAY last*.......&<...%/..*
         0010:  31202020 20202020 20202020 20202020 *1               *...............*
         0020:  20202020 20202020 20202020 20202020 *                *...............*
         0030:  20                                  * *.*
               Send final LL ZZ.
               Signal no more data to IMS Connect.
               The length for 'send LL'  is 2.
               The RC for 'send ZZ'  is 2.
               End of send.
               The RC for 'read total len' is 2.
               The RC for 'Read ZZ' is 2.
               The RC for 'Read received buffer' is 93.
       The Received Buffer - @ ffbdbd04
         0000:  454E5452 59205741 53204449 53504C41 *ENTRY WAS DISPLA*.+...........&<.*
         0010:  59454420 20202020 20202020 20202020 *YED             *...............*
         0020:  20202020 20202020 44495350 4C415920 *        DISPLAY *...........&<...*
         0030:  4C415354 31202020 20204649 52535431 *LAST1     FIRST1*<..............*
         0040:  20202020 382D3131 312D3131 31314430 *    8-111-1111D0*...............*
         0050:  312F5230 31303030 31                *1/R010001*.........*
       GPTXFR Exit  -- RC = 0.
       HGTPEXE: exiting, rc =  <0>.
       HGTPEXE: entered.
```

```
HGTPFND: entered.
HGTPFND: exit - procedure 'PGATERM' was found.
GPTTRM: entered.
        CID->num = 10.
        close socket returns: 0.
GPTTRM Exit  -- RC = 0.
HGTPEXE: exiting, rc =  <0>.
HGTCOMM: entered, keepinfo = 0
HGTCOMM: exiting, rc = 0
HGTDAFR: entered.
HGTDAFR: exit <0>.
HGTDAFR: entered.
HGTDAFR: exit <0>.
HGTDAFR: entered.
HGTDAFR: exit <0>.
HGTDAFR: entered.
HGTDAFR: exit <0>.
HGTLGOF: entered.
HGTLGOF: exiting,  rc = 0.
HGTEXIT: entered.
HGTEXIT: exiting, rc = 0.
```

8.10.3 Sample Output Log for pg4tcpmap Tool

The following is an example of the trace output produced by the pg4tcpmap tool,
where:

- the TRACE_LEVEL parameter is set to 255;

- ORACLE_SID is set to PGA;

- the ORACLE_HOME log directory is **%ORACLE_HOME%\pg4appc\log**, and

- the ORACLE_HOME directory is **C:\oracle\pga\10.2**

```
GPMGPRD Entered.
     LOG_DESTINATION  => C:\oracle\pga\10.2\pg4appc\log
     TRACE_LEVEL  => 255
     ORACLE_HOME  => C:\oracle\pga\10.2
     ORACLE_SID  => PGA
     PGA_TCP_USER  => pgaadmin
     PGA_TCP_USER Len  => 8
     PGA_TCP_PASS  => pgaadmin
     PGA_TCP_PASS Len  => 8
     PGA_TCP_DB  => R901
     PGA_TCP_DB  Len => 4
GPMGPRD Exit.
GPMEQ Entered.
     The value => i
GPMEQ Exit  -- RC = 0.
GPMEDETAILS Entered.
     Has chosen Side Profile.
gpmeTcpUniNameInput Entered.
     TCP_UNI_NAME => IMSPRF.
     TCP_UNI_NAME Len => 6.
gpmeTcpUniNameInput Exit -- RC = 0.
gpmeHostNameInput Entered.
     HOSTNAME => MVS08.US.MYCOMP.COM.
     HOSTNAME Len => 19.
gpmeHostNameInput Exit -- RC = 0.
gpmePortNumberInput Entered.
     PORT_NUMBER => 688.
     PORT_NUMBER Len => 3.
gpmePortNumberInput Exit -- RC = 0.
gpmeANDRS Entered.
     A/N/D/R/S =>  .
gpmeANDRSInput Exit -- RC = 0.
gpmeTimerInput Entered.
     Timer => D.
gpmeTimerInput Exit -- RC = 0.
gpmeSockInput Entered.
     Socket => T.
gpmeSockInput Exit -- RC = 0.
gpmeClientIDInput Entered.
     CLIENT ID =>          .
     CLIENT ID Len => 8.
gpmeClientIDInput Exit -- RC = 0.
gpmeCommModeInput Entered.
     Commit Mode => 1.
gpmeSockInput Exit -- RC = 0.
gpmeDatastoreInput Entered.
     IMS Destination ID => IMSA    .
     IMS Destination ID Len => 4.
gpmeDatastoreInput Exit -- RC = 0.
```

```
gpmeLTermInput Entered.
     IMS LTERM  =>         .
     IMS LTERM Len => 8.
gpmeLTermInput Exit -- RC = 0.
gpmRACFGrpInput Entered.
     RACF Group Name =>         .
     RACF Group Name Len => 8.
gpmRACFGrpInput Exit -- RC = 0.
GPMEDETAILS Exit -- RC = 0.
gpmlOci Entered.
gpmlOciLgon Entered.
     OCIInitialize, rc = 0
     OCIEnvInit, rc = 0
     OCIHandleAlloc -- STMT  rc = 0
     OCIHandleAlloc -- ERROR,  rc = 0
     logon complete, rc = 0
gpmlOciLgon Exit  -- RC = 0.
gpmlOciIns Entered.
     Host Name, or IP Address ==> 'MVS08.US.MYCOMP.COM', LEN = 19
     Port Number ==> '688', LEN = 3
     Conversational Protocol ==> ' ', LEN = 1
     IRM Timer ==> 'D', LEN = 1
     IRM Socket ==> 'T', LEN = 1
     Client ID ==> '         ', LEN = 8
     Commit Mode ==> '1', LEN = 1
     IMS Data Store ==> 'IMSA     ', LEN = 8
     IMS LTERM ==> '         ', LEN = 8
     RACF Group Name ==> '         ', LEN = 8
SideProfile or TCP Unique Name ==> IMSPRF, LEN = 6
The insert statement is:
INSERT INTO PGA_TCP_IMSC(SideProfileName,
HostName,                                      PortNumber,
ANDRS,                                         TIMER,
SOCK,                                          CLIENTID,
COMMITMODE,                                      IMSDESTID,
LTERM,                                         RACFGRPNAM)          VALUES
('IMSPRF','MVS08.US.MYCOMP.COM','688',' ','D','T',              '
','1','IMSA     ','         ','         ')
   OCIStmtPrepare, rc = 0
   OCIStmtExecute, rc = 0
gpmlOciIns Exit  -- RC = 0.
gpmlOciLgof Entered.
   logoff, rc = 0
    OCIHandleFree -- STMT, rc = 0
    OCIHandleFree -- Error, rc = 0
gpmlOciLgof Exit  -- RC = 0.
gpmlOci Exit  -- RC = 0.
gpmePrintData Entered.
    Requested to INSERT a row.
    Side Profile name is IMSPRF.
    remote host name is MVS08.US.MYCOMP.COM.
    IMS Connect port number is 688.
    LU name is .
    LogMode name is .
    TP name is .
    conversational protocol is  .
    Timer is D.
    socket connection type is T.
    client ID is         .
    commit mode is 1.
```

```
            Datastore name (IMS destination ID) is IMSA    .
            IMS LTERM override is          .
            RACF group name is          .
      gpmePrintData Exit -- RC = 0.
      Main Exit  -- RC = 0.
```

A

Procedural Gateway for APPC Data Dictionary

This appendix contains the following sections:

- "PG DD Environment Dictionary" on page A-2
- "PG DD Active Dictionary" on page A-7

The Procedural Gateway Data Dictionary (PG DD) is maintained in a conventional Oracle database. It is installed by a SQL*Plus installation script (**pgddcr8.sql** in the **%ORACLE_HOME%\pg4appc\admin** directory on the gateway system) and manipulated by PGAU statements and standard SQL statements.

The dictionary is divided into two sections:

- the environment dictionary
- the active dictionary

The environment dictionary is static and should not be changed. The contents of the environment dictionary support proper translation from the remote transaction's environment to the integrating server's environment, and is platform-specific. The active dictionary is updated at the user's location by the PGAU in response to definitions supplied by the user.

A.1 PG DD Environment Dictionary

The PGAU uses some dictionary tables strictly as input. These dictionary tables define environmental parameters for PGAU. Both table and values are installed by a SQL*Plus script at gateway installation time and are not to be modified by the installation.

The environment dictionary does not reference the active dictionary, but the active dictionary does reference environment dictionary entries.

A.1.1 Environment Dictionary Sequence Numbers

The environment dictionary requires unique identifying numbers in some columns to join environment dictionary entries together. Oracle sequence objects are therefore created by the Oracle Procedural Gateway for APPC to support this requirement.

Table A–1 presents the Oracle sequence objects and their descriptions.

Table A–1   Oracle Sequence Objects

Oracle Sequence Objects	Descriptions
pga.envrseq	Environment id tag
pga.compseq	Compiler id tag
pga.eattrseq	Environment Attribute id tag
pga.dtypeseq	Datatype id tag
pga.dtattseq	Datatype Attribute id tag

A.1.2 Environment Dictionary Tables

The environment dictionary tables contain constants that describe the following components of the operating environment:

- pga_maint

- pga_environments

- pga_env_attr

- pga_env_values

- pga_compilers

- pga_datatypes

- pga_datatype_attr

- pga_datatype_values

- pga_usage

- pga_modes

A.1.2.1 pga_maint

The pga_maint table stores the PG DD maintenance information, including version number and change history, as presented in Table A–2:

Table A–2   pga_maint

Column	Type	Contents
version	number(10,4)	PG DD version in format VVRRFF.rrff, where:
		VV - base version;
		RR - base release;
		FF - base fix;
		rr - port-specific release;
		ff - port-specific fix.
mntdate	date	Oracle date and time at which the PG DD was upgraded.
change	varchar2(256)	Description of the PG DD upgrade.

A.1.2.2 pga_environments

The pga_environments table stores the defined environment keywords, as presented in Table A–3:

Table A–3   pga_environments

Column	Type	Content
name	varchar2(16) not null	Environment. Primary key.
env#	number (9, 0) not null	Env id. Foreign key.

A.1.2.3 pga_env_attr

The pga_env_attr table stores the types of environmental attributes, as presented in Table A–4:

Table A–4   pga_env_attr

Column	Type	Content
name	varchar2 (16) not null	Attribute. Primary key.
attr#	number (9, 0) not null	Attribute id. Foreign key.
coltype	varchar2 (4) not null	Attr value type. Foreign key.

A.1.2.4 pga_env_values

The pga_env_values table stores the values for environments, as presented in Table A–5:

Table A–5   pga_env_values

Column	Type	Content
env#	number (9, 0) not null	Env id. Primary key.
attr#	number (9, 0) not null	Attribute id. Primary key.
numval	number (9, 0)	Numeric attribute value.
charval	varchar2 (64)	Character attribute value.
dateval	date	Date attribute value.

A.1.2.5 pga_compilers

The pga_compilers table stores the compiler environment names, as presented in Table A–6:

Table A–6   pga_compilers

Column	Type	Content
name	varchar2 (16) not null	Compiler name. Primary key.
plscomp	varchar2 (30)	PLS compiler name. Secondary key.
env#	number (9, 0) not null	Env id. Foreign key.
comp#	number (9, 0) not null	Compiler env id. Foreign key.
ddl_process	number (9, 0) not null	PGADDL processor number.

A.1.2.6 pga_datatypes

The pga_datatypes table stores the datatype keywords, as presented in Table A–7:

Table A–7   pga_datatypes

Column	Type	Content
comp#	number (9, 0) not null	Compiler env id. Primary key.
name	varchar2 (16) not null	Datatype keyword. Primary key.
dt#	number (9, 0) not null	Datatype_values. Foreign key.

A.1.2.7 pga_datatype_attr

The pga_datatype_attr table stores datatype attribute keywords, as presented in Table A–8:

Table A–8   pga_datatype_attr

Column	Type	Content
name	varchar2 (16) not null	Attribute keyword. Primary key.
attr#	number (9, 0) not null	Attribute id. Foreign key.
coltype	varchar2 (4) not null	Type of attr. Foreign key.

A.1.2.8 pga_datatype_values

The pga_datatype_values table stores the datatype attribute values, as presented in Table A–9:

Table A–9   pga_datatype_values

Column	Type	Content
comp#	number (9, 0) not null	Compiler env id. Primary key.
dt#	number (9, 0) not null	datatype_values. Foreign key.
attr#	number (9, 0) not null	Attribute id. Foreign key.
dag#	number (9, 0)	Datatype attr group no.
numval	number (9, 0)	Numeric attribute value.
charval	varchar2 (40)	Character attribute value.
dateval	date	Date attribute value.

A.1.2.9 pga_usage

The pga_usage table performs a referential integrity check of pga_data and pga_field column "usage" as presented in Table A–10:

Table A–10   pga_usage

Column	Type	Content
name	varchar2(6)	Value for the "usage" field of data dictionary tables. For example:
		'PASS'
		'SKIP
		'NULL'
		'ASIS'
		Primary key.
		Max length => 4-char string length.

A.1.2.10 pga_modes

The pga_modes table performs a referential integrity check of pga_call_parm column "mode", as presented in Table A–11:

Table A–11   pga_modes

Column	Type	Content
name	varchar2(6)	Name of valid parameter call modes. For example:
		"IN"
		"OUT"
		"IN OUT"
		Max length => 'IN OUT' string length.

A.2 PG DD Active Dictionary

The PG DD active data dictionary is created by **pgddcr8.sql** at installation, but maintained using PGAU. The active dictionary can refer to items (by ID number) in the environment dictionary.

A.2.1 Active Dictionary Versioning

The PG DD active dictionary tables contain the descriptions of transactions and data structures. There might be more than one version of a definition. Old versions are retained indefinitely.

In PGAU dictionary operations, a definition is referred to by its "name", which can be qualified by a specific version number. If omitted, the most recent version is assumed.

A.2.2 Active Dictionary Sequence Numbers

Because the active dictionary is constantly changing, the identifying numbers needed to join active dictionary entries together must also change. To support this requirement, PG DD installation creates the following Oracle sequence objects.

Table A–12 lists the Oracle sequence objects and their descriptions:

Table A–12   Active Dictionary Oracle Sequence Object Descriptions

Oracle Sequence Objects	Description
pga.transeq	Transaction id tag
pga.tranvers	Transaction Version id tag
pga.tattrseq	Transaction Attribute id tag
pga.callseq	APPC-Call id tag
pga.callvers	Call Version id tag
pga.parmseq	APPC-Call Parameter id tag
pga.dataseq	Data id tag
pga.fieldseq	Data subfield id tag
pga.datavers	Data Version id tag
pga.dattrseq	Data Attribute id tag

A.2.3 Active Dictionary Tables

Following is a list of active dictionary tables:

- pga_trans
- pga_trans_attr
- pga_trans_values
- pga_trans_calls
- pga_call
- pga_data
- pga_fields
- pga_data_attr
- pga_data_values

A.2.3.1 pga_trans

One row exists in the pga_trans table for each user transaction. The row is created by a PGAU DEFINE TRANSACTION statement and used by a PGAU GENERATE statement to create the PL/SQL package (TIP).

Table A–13 presents the column, type and content information for pga_trans:

Table A–13   pga_trans

Column	Type	Content
tname	varchar2(64)	Transaction name as defined by the customer.
		Primary key.
		Max length => APPC TPname string length.
version	number(9,0)	Version identification of this entry; it exists in the table because multiple archived or invalid entries might exist and be kept for possible future reactivation.
		Primary key.
		Set from an Oracle sequence object for transaction version inserted into the PG DD.
updtdate	date	Audit-trail date/time record last updated.
updtuser	varchar2(30)	Audit-trail user ID/program which last updated this record.
trans#	number(9,0)	PGA Transaction number, used for the define call, define data and define transaction statements.
		Foreign key.
		pga_trans_values(trans#), pga_trans_calls(trans#).
		Set from an Oracle sequence object for transaction inserted into the PG DD.

A.2.3.2 pga_trans_attr

The pga_trans_attr table relates a character string defining the transaction attributes supported by PGA to pga_trans_values entries through an attribute id number and type.

The pga_trans_attr table is also used for integrity checks of transaction attributes when new transactions are being defined.

There is an entry in the pga_trans_attr table for each transaction attribute name. All possible transaction attribute names supported by PGA on any defined transaction are specified. There is one row for each attribute, and no duplicates are allowed.

Table A–14 presents the column, type and content information for pga_trans_attr:

Table A–14   pga_trans_attr

Column	Type	Content
name	varchar2(16)	Character string name of attribute.
		Primary key.
		Contains:
		"ENVIRONMENT",
		"LUNAME",
		"TPNAME",
		"LOGMODE",
		"SIDEPROFILE",
		"SYNCLEVEL",
		"NLS_LANGUAGE",
		"REMOTE_MBCS"
		"LOCAL_MBCS"
attr#	number(9,0)	Attribute id assigned.
		Foreign key.
		pga_data_values(attr#).
		Set from an Oracle sequence object for each supported transaction attribute inserted into the PG DD.
coltype	varchar2(4)	Type of Oracle column from which attribute value is retrieved from pga_tran_values. For example:
		'NUM ' => pga_tran_values(numval)
		'CHAR' => pga_tran_values(charval)
		'DATE' => pga_tran_values(dateval)
required	char(1)	If not null, required keyword for DEFINE TRANSACTION; if null, optional.

A.2.3.3 pga_trans_values

The pga_trans_values table describes the values of transaction attributes.

A row exists to specify the value of each attribute of each transaction defined in the data dictionary.

The column, type and content information for pga_trans_values is presented in Table A–15:

Table A–15   pga_trans_values

Column	Type	Content
trans#	number(9,0)	Transaction id from pga_trans(trans#).
		Primary key.
		Set from an Oracle sequence object for transaction inserted into the PG DD.
attr#	number(9,0)	Attribute id from pga_trans_attr(attr#),
		Primary key.
		Set from an Oracle sequence object for each supported transaction attribute inserted into the PG DD.
numval	number(9,0)	Attribute's numeric value, for example for a given transaction's SYNCLEVEL attribute 0.
charval	varchar2(64)	Attribute's character value; for example, a given transaction's TPNAME attribute.
dateval	date	Attribute's date value. Probably always null; included for completeness.

A.2.3.4 pga_trans_calls

The pga_trans_calls table relates all calls available with any single transaction to each specific call definition through a call ID number.

An entry exists in the pga_trans_calls table for each PL/SQL call referenced in a transaction definition through the CALL(*cname*,...) operand. One row per transaction call; no duplicates.

The column, type and content information for pga_trans_calls is presented in Table A–16:

Table A–16   pga_trans_calls

Column	Type	Content
trans#	number(9,0)	Transaction id number from pga_trans(trans#).
		Primary key.
		Set from an Oracle sequence object for transaction inserted into the PG DD.
seq#	number(9,0)	Sequence number of this call.
		Primary key.
call#	number(9,0)	Call id number in pga_call(call#).
		Foreign key.
		Copied from pga_call.call# for the referenced call when this transaction definition was inserted or updated.

A.2.3.5 pga_call

The pga_call table relates all calls that are available for all defined transactions, to a unique call id number and PL/SQL remote procedural call (RPC) name. One entry exists in this table for each PL/SQL call (defined in a DEFINE CALL statement).

One row per call, duplicates are possible when multiple transactions make identical calls. The plsrpc specification must be unique within the Oracle server which makes the calls, and rows are uniquely distinguished by call#.

The column, type and content information for pga_call are presented in Table A–17:

Table A–17   pga_call

Column	Type	Content
cname	varchar2(48)	Call name for PGAU reference; Primary key. Max length => COBOL name string length
plsrpc	varchar2(30)	RPC call name for reference in PL/SQL (public procedure to be generated). Max length => PL/SQL RPC name length
updtdate	date	Audit trail date/time of record's last update.
updtuser	varchar2(30)	Audit trail user id/program which last updated this record.
version	number(9,0)	Version identification of this entry, because multiple archived or invalid entries might exist and be kept for possible future reactivation. Primary key. Set from an Oracle sequence object for call version inserted into PG DD.
call#	number(9,0)	Call id number. Foreign key. pga_trans_calls(call#), pga_call_parm(call#). Set from an Oracle sequence object for each call inserted into the PG DD.

A.2.3.6 pga_call_parm

The pga_call_parm table relates all parameters of any single transaction call to the data definitions describing each parameter.

One entry exists in the pga_call_parm table for each parameter on a call in the PARMS() operand of the PGAU DEFINE CALL statement.   One row per parameter, duplicates allowed when multiple calls (in the pga_call table) refer to the same parameters.

Table A–18 presents the column, type and content information for pga_call_parm:

Table A–18   pga_call_parm

Column	Type	Content
call#	number(9,0)	Call number for the referencing call from pga_calls.
		Primary key.
		Set from an Oracle sequence object for each call inserted into the PG DD.
parm#	number(9,0)	Position in the PARMS() argument of DEFINE CALL operation (1,2,3...).
		Primary key.
cmode	varchar2(6)	Call mode of this parameter; one of the values in pga_data_modes. For example:
		'IN', 'OUT', 'IN OUT'
		Max length => 'IN OUT' string length
data#	number(9,0)	Data definition # in pga_data(data#) of this item.
		Foreign key.
		pga_data(data#),pga_data_values(data#).
		Copied from pga_data.data# for the data item when this call/parm definition was inserted or updated.

A.2.3.7 pga_data

The pga_data table defines each data item used as a parameter in a call and relates the remote host data name to its PL/SQL variables and any component subfields or clauses within each data item (if the data item is an aggregate, such as a record). Each data item might have attributes related to it through its corresponding field definition. Even atomic data items have a single row in the pga_field table.

One row exists in the pga_data table for each data item defined by a PGAU DEFINE DATA or REDEFINE DATA statement.

Table A–19 presents the column, type and content information for pga_data:

Table A–19   pga_data

Column	Type	Content
comp#	number(9,0)	Compiler id number.;
		Foreign key.
		(pga_compiler(comp#).
		Set from pga_compiler(comp#) based on the language parameter specified on the DEFINE DATA statement when the data definition is inserted.
compopts	varchar2(100)	Compiler options from the COMPOPTS keyword on the DEFINE DATA statement.
dname	varchar2(255)	Name from the DEFINE statement;
		Primary key.
		Max length => COBOL name length
plsdvar	varchar(30)	PL/SQL variable name of data item for reference in PL/SQL.
		Max length => PL/SQL variable length
version	number(9,0)	Version number of this entry. Set from an Oracle sequence object for data version inserted into the PGADD.
updtdate	date	Audit-trail date/time this control record last updated.
updtuser	varchar2(30)	Audit-trail user id/program which last updated this record.
usage	varchar2(6)	Default usage of this data item: PASS, SKIP, NULL, ASIS.
		Used primarily by PGAU REPORT.
		Max length => 4-char string length
data#	number(9,0)	Data definition number.
		Foreign key.
		(pga_call_parm(data#), (pga_field(data#)
		Set from an Oracle sequence object.

A.2.3.8 pga_fields

The pga_fields table defines each field within a data item and relates the remote host data field to its PL/SQL variables or nested records. Each field item might have attributes related to it (by field#) in the pga_data_attr and pga_data_values tables.

One row exists in the pga_fields table for each atomic item, field, clause, or nested record defined by a PGAU DEFINE DATA statement. Several rows would exist (related by a single data# and incrementing fld#) to define an aggregate data item, one row per field or group.

Table A–20 presents the column, type and content information for pga_fields:

Table A–20   pga_fields

Column	Type	Content
data#	number(9,0)	Data definition number.
		Primary key.
		(pga_data(data#), pga_call_parm(data#).
		Set from an Oracle sequence object.
fname	varchar2(255)	Extracted or derived name of a field if dname defines aggregate data.
		Max length => COBOL name length
plsfvar	varchar2(30)	PL/SQL variable name of subfield in aggregate data for reference in PL/SQL. Max length => PL/SQL variable length
updtdate	date	Audit-trail date/time this control record last updated.
updtuser	varchar2(30)	Audit-trail user id/program which last updated this record.
fld#	number(9,0)	Clause or field within data definition id no.
		Foreign key.
		pga_data_values(fld#).
		Set from an Oracle sequence object.
pos#	number(9,0)	Relative position number of each field defined within an aggregate data item (for example, 1, 2 3, and so on) or NULL if data is atomic.
usage	varchar2(6)	Usage of this data field:
		'PASS', 'SKIP', 'NULL', 'ASIS'.
		Max length => 4-char string length
mask	varchar2(30)	Datatype or Mask value. For example:
		'S9(4)'
		'X(24)'
		'VARCHAR2(24)'
		'BINARY_INTEGER(16)'
		NULL
		When NULL, item defined is assumed to be a COBOL group or PL/SQL nested record.
		Max length => arbitrarily chosen

Table A–20   (Cont.) pga_fields

Column	Type	Content
maskopts	varchar2(100)	Datatype or Mask options value. For example:
		'USAGE COMP-4'
		'DISPLAY'
		NULL
		Max length => arbitrarily chosen

A.2.3.9 pga_data_attr

The pga_data_attr table defines all possible data attribute names allowed by PGA and relates each attribute name to a number and type, by which the value of this attribute for a specific data item can be selected from pga_data_values.

The pga_data_attr table is also used for integrity checks of data attributes when new data items are defined.

There is one entry in the pga_data_attr table for every possible attribute name to which any PGA supported data item might relate.

Table A–21 presents the column, type and content information for pga_data_attr:

Table A–21   pga_data_attr

Column	Type	Content
name	varchar2(16)	Character string name of attribute.
		Primary key.
		Contains:

		`"LEVEL"`
		 `"RENAMEMF" (renames member first)`
		 `"RENAMEML" (renames member last)`
		 `"REMAPSMF" (redefines member first)`
		 `"REMAPSML" (redefines member last)`
		 `"REMAPSWM" (redefines when member)`
		 `"REMAPSWC" (redefines when char value)`
		 `"REMAPSWN" (redefines when num value)`
		 `"REPGRPFF" (occurs n)`
		 `"REPGRPVF" (odo first n)`
		 `"REPGRPVL" (odo last n)`
		 `"REPGRPVM" (odo depending member)`
		 `"REPGRPKA" (either Key Asc name)`
		 `"REPGRPKD" (either Key Desc name)`
		 `"REPGRPIX" (either index name)`
		 `"PLSTYPE"`
		 `"JUST"     (justified char data)`
		 `"SYNC"     (aligned aggregate data)`
		 `"LOCAL_LANGUAGE"`
		 `"REMOTE_LANGUAGE"`
		 `"LENGTH"   (LENGTH IS variable)`

		`Max length => attr name string lengths`
attr#	number(9,0)	Attribute id assigned.
		Foreign key.
		pga_data_values(attr#). Set from an Oracle sequence object for each supported data attribute inserted into the PG DD.

Table A–21   (Cont.) pga_data_attr

Column	Type	Content
coltype	varchar2(4)	Type of Oracle column from which attribute value is retrieved from pga_data_values. For example:
		'NUM ' => pga_data_values(numval)
		'CHAR'=> pga_data_values(charval)
		'DATE' => pga_data_values(dateval)
required	char(1)	If not null, required keyword.

A.2.3.10 pga_data_values

A row exists in the pga_data_values table for each attribute of each data item defined by each data definition.

Table A–22 presents the column, type and content information for pga_data_values:

Table A–22   pga_data_values

Column	Type	Content
fld#	number(9,0)	Data Field Definition number from pga_data(fld#). Primary key.
attr#	number(9,0)	Attribute id from pga_data_attr(attr#).
		Primary key.
numval	number(9,0)	Attribute's numeric value. For example:
| | | ```
number for "LEVEL"
number for "REMAPSWN" (redefines)
number for "REPGRPFF" (occurs n)
number for "REPGRPVF" (odo first n)
number for "REPGRPVL" (odo last n)
``` |
| | | If a non-numeric attribute, this item is NULL. |
| charval | varchar2(40) | Attribute's character value. |
| | | ```
fname for "RENAMEMF (renames first)
fname for "RENAMEML" (renames last)
fname for "REMAPSMF" (redefines first)
fname for "REMAPSML" (redefines last)
fname for "REMAPSWM" (redefines when)
fname for "REPGRPVM" (odo member)
string for "REMAPSWC" (redefines)
string for "REPGRPKA" (occurs key)
string for "REPGRPKD" (occurs key)
string for "REPGRPIX" (occurs index)
string for "PLSTYPE" (PL/SQL data type)
string for "JUST"
string for "SYNC"
string for "REMOTE_LANGUAGE"
fname for "LENGTH"
``` |
		If a non-character attribute, this item is NULL.
		Max length => NLS_charset string length
dateval	date	Attribute's date value. Always null, included for completeness.

Table A–22   (Cont.) pga_data_values

Column	Type	Content
qual	number (9,0)	Qualified name number.
		Foreign key.

B

Gateway RPC Interface

To execute a remote transaction program using the Oracle Procedural Gateway for APPC you must execute a PL/SQL program to call the gateway functions, using a remote procedural call (RPC). The gateway functions handle the initiation, data exchange and termination for the gateway conversation with the remote transaction program.

The Oracle Procedural Gateway for APPC includes a tool, PGAU, to generate the PL/SQL packages (TIPs) automatically, based on definitions you provide in the form of COBOL record layouts and PGDL (Procedural Gateway Definition Language).

This appendix contains the following section:

- "Calling Gateway Functions to Execute Transaction Programs" on page B-2

B.1 Calling Gateway Functions to Execute Transaction Programs

The gateway functions are all executed through remote procedural calls (RPC). The functions are called from PL/SQL code as follows:

```
function@dblink(parm1,parm2,...,parmn);
```

Where Table B–1 describes the parameters in this syntax:

Table B–1   Gateway Functions

Item	Description
function	is the name of the function being called.
dblink	is the name of a predefined database link to the gateway server on the Windows system.
parm1, parm2,parmn	are the function-specific parameters described later in this appendix.

Calling a function in PL/SQL code with the `@dblink` notation following the function name is a remote procedural call.

B.1.1 PGAINIT and PGAINIT_SEC

PGAINIT and PGAINIT_SEC are remote procedural calls that initiate an APPC conversation with a specified transaction program. The difference between the two is that PGAINIT_SEC includes the added capability of being able to set the gateway conversation security user ID and password to values other than the current Oracle user ID and password. Upon successful completion of either function, the conversation is ready to send data to the remote transaction program.

Table B–2 presents the PGAINIT and PGAINIT_SEC parameters that are common in both procedures. It lists the type, datatype and description of each parameter:

Table B–2   Common PGAINIT and PGAINIT_SEC Parameters

Parameters	Type	Datatypes	Descriptions
CONVID	OUT	RAW(12)	**For a gateway using SNA:** Conversation identifier returned by the PGAINIT function to be used to identify the conversation to the PGAXFER and PGATERM functions. After PGAINIT is called, this variable must never be modified, or results will be unpredictable.
			For a gateway using TCP/IP: Socket file descriptor returned by the PGAINIT function to be used to identify the conversation to the PGAXFER and PGATERM functions. After PGAINIT is called, this variable must never be modified, or results will be unpredictable.

Table B–2   (Cont.) Common PGAINIT and PGAINIT_SEC Parameters

Parameters	Type	Datatypes	Descriptions
TPNAME	IN	VARCHAR2(64)	Transaction program name of the remote transaction program with which a conversation is to be established. For most OLTPs, the name must be the transaction name as defined to the OLTP. This name can be from 1 to 64 characters in length.
			Note: For TCP/IP support, the maximum size is 8 characters. For more information, refer to Appendix B of the *Oracle Procedural Gateway for APPC Installation and Configuration Guide.*
LUNAME	IN	VARCHAR2(17)	**For a gateway using SNA:** the LU name of the OLTP under which the remote transaction program executes. This parameter is the fully-qualified LU name or alias and can be from 1 to 17 characters in length.
			For a gateway using TCP/IP: this parameter is not applicable.
MODENAME	IN	VARCHAR2(8)	**For a gateway using SNA**: Logmode entry name of the logmode table entry on the remote host, which defines the session characteristics for the APPC conversation. This name can be from 1 to 8 characters in length.
			For a gateway using TCP/IP: this parameter is not applicable.
PROFNAME	IN	VARCHAR2(8)	Profile name of the SNA Side Information profile which defines the conversation. This name can be from 1 to 8 characters in length.
			For a gateway using TCP/IP: this name represents a group of IMS transactions similar of similar TCP/IP and IMS Connect attributes.
SYNCLEVEL	IN	CHAR(1)	Sync level for this conversation. This value must be either '0' or '1'.
			Sync level 0 indicates that the remote transaction program has no synchronization capabilities.
			Sync level 1 indicates that the remote transaction program is capable of responding to CONFIRM requests and is used to ensure data integrity when the remote transaction program is making updates to a database on the remote host.

Table B–3 lists the PGAINIT_SEC parameters which are specific to the procedure:

Table B–3   PGAINIT_SEC Parameters Specific to the Procedure

Parameter	Type	Datatype	Description
USERID	IN	VARCHAR2(8)	Conversation security user ID to be passed to the target OLTP. The value must be from 1 to 8 characters in length.
PASSWORD	IN	VARCHAR2(8)	Conversation security password to be passed to the target OLTP. The value must be from 1 to 8 characters in length.

For Gateways Using the SNA Protocol:

There is an interrelationship between PROFNAME and LUNAME/TPNAME/MODENAME. If PROFNAME is set to blanks or a null value, the LUNAME, TPNAME, and MODENAME parameters are all required to be non-blank values. If they are not all set to non-blank values, an exception is generated. However, if PROFNAME is set to a valid Side Information Profile name, the LUNAME, TPNAME, and MODENAME parameters can be null or blank, because the Side Information profile specifies all the information necessary to establish the conversation. In this case, any non-blank, non-null values specified for LUNAME, TPNAME, or MODENAME override values set in the Side Information profile.PROFNAME must be set and cannot be blank or null.

For Gateways Using the TCP/IP protocol:

PROFNAME and TPNAME must be set and cannot be blank or null.

B.1.2 PGAXFER

PGAXFER is called to transfer data to and from a remote transaction program on the gateway conversation initialized by PGAINIT. The function sends and/or receives data items based on the calling parameters.

Table B–4 lists the types, datatypes and descriptions of PGAXFER parameters:

Table B–4   PGAXFER Parameters

Parameter	Type	Datatype	Description
CONVID	IN	RAW(12)	**For a gateway using SNA:** Conversation identifier returned by the PGAINIT function to be used to identify the conversation.
			For a gateway using TCP/IP: Socket file descriptor returned by the PGAINIT function to be used to identify the conversation.
SENDBUF	IN	RAW(32763)	Buffer containing all the data items to be sent to the remote transaction program. The data items are sent as is, with no changes. Data items must appear in the buffer in the exact order in which the remote transaction program expects to receive them. The total size of all the data items cannot exceed the maximum size for a single gateway send, which is 32,763 bytes for a mapped gateway conversation.
SENDBUFL	IN	BINARY_INTEGER	Total length of the data items contained in SENDBUF. The range is 0-32,763 bytes. A value of '0' is used when there are no data items to send.
SENDLNS	IN	RAW(1024)	Buffer containing an array of up to 256 4-byte integer values. The first integer value specifies the number of data items contained in the send buffer (SENDBUF). Following that data item count is a series of integer values specifying the lengths of the data items. There must be an exact match between the data item count and the number of data item length values. Up to 255 data items can be described by this array. The sum of all the data item lengths cannot exceed the total length in SENDBUFL.
RECVBUF	OUT	RAW(32763)	Buffer to contain all the data items received from the remote transaction program. The data items are stored in this buffer in the exact order in which the remote transaction program sends them. The total size of all the data items cannot exceed the maximum size of 32,763 bytes.
RECVBUFL	IN	BINARY_INTEGER	Total length of the receive buffer. The range is 0-32,763 bytes. A value of '0' is used when there are no data items to receive.

Table B–4   (Cont.) PGAXFER Parameters

Parameter	Type	Datatype	Description
RECVLNS	INOUT	RAW(1024)	Buffer containing an array of up to 256 4-byte integer values. The first integer value specifies the number of data items to be received into the receive buffer (RECVBUF). Following the data item count is a series of integer values specifying the maximum lengths of the data items to be received. On output, these values are replaced with the actual lengths of the data items received. There must be an exact match between the data item count and the number of data item length values. Up to 255 data items can be described by this array. The sum of all the data item lengths cannot exceed the total length of the receive buffer (RECVBUFL).

When PGAXFER is called, either or both of SENDBUFL and RECVBUFL must be nonzero; in other words, at least one data item must be sent to or received from the remote transaction program. If PGAXFER is called with no data items to send or receive, it generates an exception.

> **Note:**   On each PGAXFER call, all send processing occurs first, followed by all receive processing. If a transaction operates in a manner that requires multiple sets of send and receives, then PGAXFER can be called more than once to accommodate the transaction. If more than 32,763 bytes of data are to be sent or received, multiple calls to PGAXFER must be made.

B.1.3 PGATERM

PGATERM is called to terminate an the gateway conversation that was initiated by a previous call to PGAINIT. Upon successful completion of this function, the conversation is deallocated and all storage associated with it is freed.

Table B–5 presents the types, datatypes and descriptions of PGATERM parameters:

Table B–5   PGATERM Parameters

Parameter	Type	Datatype	Description
CONVID	IN	RAW(12)	**For a gateway using SNA:** Conversation identifier returned by the PGAINIT function to be used to identify the conversation.
			For a gateway using TCP/IP: Socket file descriptor returned by the PGAINIT function to be used to identify the conversation.
TERMTYPE	IN	CHAR(1)	Type of termination to be performed.'0' indicates normal completion and '1' indicates abnormal termination, which is only requested if there is an error.

B.1.4 PGATCTL

PGATCTL is called by the TRACE_LEVEL parameter at
%ORACLE_HOME%\pg4appc\admin\init*sid*.ora file. Using PGATCTL, the trace
level can be changed dynamically from within a PL/SQL stored procedure. This
facility is useful when debugging a new PL/SQL application.

Table B–6 presents the types, datatypes and descriptions of parameters in PGATCTL:

Table B–6   PGATCTL Parameters

Parameter	Type	Datatype	Description
CONVID	IN	RAW(12)	**For a gateway using SNA**: Conversation identifier returned by the PGAINIT function to be used to identify the conversation.
			For a gateway using TCP/IP: Socket file descriptor returned by the PGAINIT function to be used to identify the conversation.
TRFUNC	IN	CHAR(1)	Trace control function to be performed. The valid values are:
			'S' - set trace flags to the exact value specified by the TRFLAGS parameter.
			'E' - enable the trace flags specified by the TRFLAGS parameter, without changing any other flags.
			'D' - disable the trace flags specified by the TRFLAGS parameter, without changing any other flags.
TRFLAGS	IN	BINARY_ INTEGER	Trace flags.
			Turn on TRACE_LEVEL. Refer to Appendix A "Gateway Initialization Parameter File Using SNA Protocol" in the *Oracle Procedural Gateway for APPC Installation and Configuration Guide* for more information if your protocol is SNA.
			Refer to Appendix B of the *Oracle Procedural Gateway for APPC Installation and Configuration Guide* if your protocol is TCP/IP.

B.1.5 PGATRAC

This function is called to write a line of user data into the PGA trace file. Using PGATRAC, the flow within a PL/SQL procedure can be traced, along with the events traced, based on the TRACE_LEVEL at **%ORACLE_HOME%\pg4appc\admin\init*sid*.ora**. This is a useful debugging tool when developing a new PL/SQL application.

Table B–7 presents the type, datatype and description of the PGATRAC parameter:

Table B–7   PGATRAC Parameter

Parameter	Type	Datatype	Description
TRDATA	IN	VARCHAR2(120)	Line of user data to be written into the gateway trace file. The contents must be printable characters.

C

The UTL_PG and UTL_RAW Interface

The Oracle Procedural Gateway for APPC requires the use of the RAW datatype to transfer data to and from PL/SQL without any alteration by Oracle Net. This is necessary because only the PL/SQL applications have information about the format of the data being sent to and received from the remote transaction programs. Oracle Net only has information about the systems where the PL/SQL application and the gateway server are running. If Oracle Net is allowed to perform translation on the data flowing between PL/SQL and the gateway, the data can end up in the wrong format.

This appendix contains the following sections:

- "UTL_RAW Functions" on page C-2
- "UTL_PG Functions" on page C-18
- "NUMBER_TO_RAW and RAW_TO_NUMBER Argument Values" on page C-32

C.1 UTL_RAW Functions

The UTL_RAW package is an extension to PL/SQL that provides a full complement of RAW data manipulation functions. Using these functions, data sent to remote transaction programs can be converted into the correct format by the PL/SQL application, and data received from remote transaction programs can be converted back into Oracle formats.

All of the functions listed in this section are called in the standard PL/SQL manner, which is `package_name.function_name(arguments)`. In the case of the UTL_RAW routines, this is `UTL_RAW.function_name(arguments)`.

For each function listed in the following sections, the function name, arguments and their datatypes, and the return value datatype are provided. Unless otherwise specified, the parameters are IN, not OUT, parameters.

C.1.1 BIT_AND

BIT_AND performs a bitwise logical AND operation of the values in `r1` and `r2` and returns the resulting value.

If `r1` and `r2` have different lengths, then the AND operation is terminated after the last byte of the shorter of the two RAW values. The unprocessed portion of the longer RAW value is appended to the partial result to produce the final result returned. The result length equals the longer of the two input RAW values.

Syntax

```
function BIT_AND (r1 IN RAW, r2 IN RAW) RETURN RAW;
```

Where Table C–1 describes the parameters in this function:

Table C–1   BIT_AND Function Parameters

Parameter	Description
r1	is a RAW value to be combined with r2.
r2	is a RAW value to be combined with r1.

Defaults

None.

Return Value

A RAW value which is the bitwise logical AND of `r1` and `r2`. Or a null value if either `r1` or `r2` is null.

Error and Warning Messages

If you receive an ORA-xxxx error or warning message, refer to the *Oracle Database Error Messages* guide for an explanation and information about how to handle it.

C.1.2 BIT_COMPLEMENT

BIT_COMPLEMENT performs a bitwise logical COMPLEMENT operation of the value r and returns the resulting RAW value. The length of the result equals the length of the input RAW value r.

Syntax

```
function BIT_COMPLEMENT (r IN RAW) RETURN RAW;
```

where:

"r" is the RAW value on which to perform the COMPLEMENT operation.

Defaults

None

Return Value

A RAW value which is the bitwise logical COMPLEMENT of r. Or a null value if the input value r is null.

Error and Warning Messages

If you receive an ORA-xxxx error or warning message, refer to the *Oracle Database Error Messages* guide for an explanation and information about how to handle it.

C.1.3 BIT_OR

BIT_OR performs a bitwise logical OR operation of the values in `r1` and `r2` and returns the resulting value.

If `r1` and `r2` have different lengths, then the OR operation is terminated after the last byte of the shorter of the two RAW values. The unprocessed portion of the longer RAW value is appended to the partial result to produce the final result returned. The resulting length equals the longer of the two input values, `r1` and `r2`.

Syntax

```
function BIT_OR (r1 IN RAW, r2 IN RAW) RETURN RAW;
```

Where Table C–2 describes the parameters in the function:

Table C–2   BIT_OR Function Parameters

Parameter	Description
r1	is a RAW value to OR with `r2`.
r2	is a RAW value to OR with `r1`.

Defaults

None

Return Value

A RAW value which is the bitwise logical OR of `r1` and `r2`. Or a null value if both `r1` and `r2` is null.

Error and Warning Messages

If you receive an ORA-xxxx error or warning message, refer to the *Oracle Database Error Messages* guide for an explanation and information about how to handle it.

C.1.4 BIT_XOR

BIT_XOR performs a bitwise logical EXCLUSIVE OR operation of the values in `r1` and `r2` and returns the resulting value.

If `r1` and `r2` have different lengths, then the EXCLUSIVE OR operation is terminated after the last byte of the shorter of the two RAW values. The unprocessed portion of the longer RAW value is appended to the partial result to produce the final result returned. The result length equals the longer of the two input RAW values.

Syntax

```
function BIT_XOR (r1 IN RAW, r2 IN RAW) RETURN RAW
```

Where Table C–3 describes the parameters in this function:

Table C–3   BIT_XOR Function Parameters

Parameter	Description
r1	is the RAW value to XOR with `r2`.
r2	is the RAW value to XOR with `r1`.

Defaults

None

Return Value

A RAW value which is the bitwise logical XOR of `r1` and `r2`. Or a null value if `r1` and `r2` have identical values.

Error and Warning Messages

If you receive an ORA-xxxx error or warning message, refer to the *Oracle Database Error Messages* guide for an explanation and information about how to handle it.

C.1.5 CAST_TO_RAW

CAST_TO_RAW converts a VARCHAR2 `c` into a RAW with the same number of bytes.

The input is treated as if it is composed of single 8-bit bytes, not characters. Multibyte character boundaries are ignored. The data is not modified in any way, it is just changed to a RAW datatype.

Syntax

```
function CAST_TO_RAW (c IN VARCHAR2) RETURN RAW;
```

where: "`c`" is a VARCHAR2 value to be changed to a RAW value.

Defaults

None

Return Value

A RAW value having the same data and byte length as the input VARCHAR2 value. Or a null value if `c` is null.

Error and Warning Messages

If you receive an ORA-xxxx error or warning message, refer to the *Oracle Database Error Messages* guide for an explanation and information about how to handle it.

C.1.6 CAST_TO_VARCHAR2

CAST_TO_VARCHAR2 converts a RAW `r` into a VARCHAR2 with the same number of data bytes.

The result is treated as if it is composed of single 8-bit bytes, not characters. Multibyte character boundaries are ignored. The data is not modified in any way, it is just changed to a VARCHAR2 datatype.

Syntax

```
function CAST_TO_VARCHAR2 (r IN RAW) RETURN VARCHAR2;
```

where:

"`r`" is a RAW value to be changed to a VARCHAR value.

Defaults

None

Return Value

A VARCHAR2 value having the same data as the RAW input value. Or a null value if `r` is null.

Error and Warning Messages

If you receive an ORA-xxxx error or warning message, refer to the *Oracle Database Error Messages* guide for an explanation and information about how to handle it.

C.1.7 COMPARE

COMPARE compares RAW `r1` with RAW `r2`.

If they are identical, COMPARE returns zero. Otherwise, COMPARE returns the position of the first byte that does not match. If the input values are of different length, the shorter RAW value is padded on the right with the byte specified by `pad`.

Syntax

```
function COMPARE (r1 IN RAW, r2 IN RAW,
 pad RAW DEFAULT NULL) RETURN NUMBER;
```

Where Table C–4 describes the parameters in this function:

Table C–4   COMPARE Function Parameters

Parameter	Description
r1	is the first RAW value to be compared. This may be null and/or have a length of 0.
r2	is the second RAW value to be compared. This might be null and/or have a length of 0.
pad	is a 1 byte value used to pad the shorter RAW value.

Defaults

where "`pad`" is optional and defaults to x'00'.

Return Value

A value of 0 if the strings are null or identical. Or the position, numbered from 1, of the first mismatched byte.

Error and Warning Messages

If you receive an ORA-xxxx error or warning message, refer to the *Oracle Database Error Messages* guide for an explanation and information about how to handle it.

C.1.8 CONCAT

CONCAT concatenates a set of up to 12 RAW values (`r1 - r12`) into a single RAW and returns it.

Input RAW values are appended together in the resulting RAW, left to right, in the order they appear in the parameter list. Input values need not be contiguous. Null input RAW values are skipped and the concatenation continues with the next non-null input RAW value. If the sum of the lengths of the input RAWs exceeds the maximum allowable length for a RAW (32767), an error is returned.

Syntax

```
function CONCAT (r1 IN RAW DEFAULT NULL,
 r2 RAW DEFAULT NULL,
 r3 RAW DEFAULT NULL,
 r4 RAW DEFAULT NULL,
 r5 RAW DEFAULT NULL,
 r6 RAW DEFAULT NULL,
 r7 RAW DEFAULT NULL,
 r8 RAW DEFAULT NULL,
 r9 RAW DEFAULT NULL,
 r10 RAW DEFAULT NULL,
 r11 RAW DEFAULT NULL,
 r12 RAW DEFAULT NULL) RETURN RAW;
```

where:

"`r1 - r12`" are the RAW items to be concatenated.

Defaults

None

Return Value

A RAW value with the concatenated items.

Error and Warning Messages

If you receive an ORA-xxxx error or warning message, refer to the *Oracle Database Error Messages* guide for an explanation and information about how to handle it.

C.1.9 CONVERT

CONVERT converts RAW `r` from character set `from_charset` to character set `to_charset` and returns the resulting RAW value.

Both `from_charset` and `to_charset` must specify supported character sets defined to the Oracle server.

Syntax

```
function CONVERT (r IN RAW,
 to_charset IN VARCHAR2,
 from_charset IN VARCHAR2) RETURN RAW;
```

Where Table C–5 describes the parameters in this function:

Table C–5   CONVERT Function Parameters

Parameter	Description
r	is the RAW byte-string to be converted.
to_charset	is the NLS character set to convert `r` to.
from_charset	is the NLS character set that `r` is currently using.

Defaults

None

Return Value

A RAW string of bytes converted according to the specified character set.

Error and Warning Messages

If you receive an ORA-xxxx error or warning message, refer to the *Oracle Database Error Messages* guide for an explanation and information about how to handle it.

C.1.10 COPIES

COPIES returns n copies of RAW r concatenated together.

Syntax

```
function COPIES (r IN RAW,
 n IN NUMBER) RETURN RAW;
```

Where Table C–6 describes the parameters in this function:

Table C–6   COPIES Function Parameters

Parameter	Description
r	is the RAW value to be copied.
n	is the number of times to copy the RAW value. This must be a positive value.

Defaults

None

Return Value

The RAW value copied n times.

Error and Warning Messages

If you receive an ORA-xxxx error or warning message, refer to the *Oracle Database Error Messages* guide for an explanation and information about how to handle it.

C.1.11 LENGTH

LENGTH returns the length in bytes of RAW r.

Syntax

```
function LENGTH (r IN RAW) RETURN NUMBER;
```

where:

"r" is the byte stream to be measured.

Defaults

None

Return Value

The current length of the RAW input value.

Error and Warning Messages

If you receive an ORA-xxxx error or warning message, refer to the *Oracle Database Error Messages* guide for an explanation and information about how to handle it.

C.1.12 OVERLAY

OVERLAY replaces the specified portion of RAW `target` with RAW `overlay`, beginning at byte position `pos` of `target`, and proceeding for `len` bytes.

If `overlay` has fewer than `len` bytes, then `overlay` is padded to `len` bytes using the byte specified by `pad`. If `overlay` has more than `len` bytes, then the extra bytes in `overlay` are ignored. If `len` bytes beginning at position `pos` of `target` exceeds the length of `target`, `target` is extended to contain the entire length of `overlay`. If `pos` exceeds the length of `target`, `target` is padded with `pad` bytes to position `pos` and then `target` is further extended with `overlay` bytes.

Syntax

```
function OVERLAY (overlay IN RAW,
 target IN RAW,
 pos IN BINARY_INTEGER DEFAULT 1,
 len IN BINARY_INTEGER DEFAULT NULL,
 pad IN RAW DEFAULT NULL) RETURN RAW;
```

Where Table C–7 describes the parameters in this function:

Table C–7   OVERLAY Function Parameters

Parameter	Description
overlay	is a byte-string used to overlay the target. Bytes are always selected from the overlay RAW beginning with the leftmost byte.
target	is the byte-string to be overlayed.
pos	is the position within the target RAW, numbered from 1, at which to begin overlaying. This value must be greater than zero. This parameter is optional.
len	is the number of bytes to overlay. This must be greater than or equal to zero. This parameter is optional.
pad	is a single byte value used to pad when `len` exceeds overlay length or `pos` exceeds target length. This parameter is optional.

Defaults

Table C–8 describes the OVERLAY function defaults:

Table C–8   OVERLAY Function Defaults

Parameters	Description
pos	defaults to 1.
len	defaults to length of overlay.
pad	defaults to x'00'.

Return Value

The target byte-string overlayed as specified.

Error and Warning Messages

If you receive an ORA-xxxx error or warning message, refer to the *Oracle Database Error Messages* guide for an explanation and information about how to handle it.

C.1.13 REVERSE

Reverse the byte sequence in RAW r from end-to-end. For example, x'0102F3' would be reversed into x'F30201' and 'xyz' would be reversed to 'zyx'. The result length is the same as the input RAW length.

Syntax

```
function REVERSE (r IN RAW) RETURN RAW;
```

where "r" is the RAW value to reverse.

Defaults

None

Return Value

A RAW value containing the reverse of the input RAW value.

Error and Warning Messages

If you receive an ORA-xxxx error or warning message, refer to the *Oracle Database Error Messages* guide for an explanation and information about how to handle it.

C.1.14 SUBSTR

SUBSTR extracts a portion of RAW `r`, starting at byte position `pos` and including `len` bytes.

If `pos` is positive, SUBSTR counts from the beginning of `r` to find the first byte. If `pos` is negative, SUBSTR counts backwards from the end of `r`. If `len` is not specified, SUBSTR returns all bytes to the end of `r`.

Syntax

```
function SUBSTR (r IN RAW,
                 pos IN BINARY_INTEGER,
                 len BINARY_INTEGER DEFAULT NULL) RETURN RAW;
```

Where Table C–9 describes the parameters in this function:

Table C–9   SUBSTR Function Parameters

Parameters	Description
r	is the RAW byte-string from which a portion is to be extracted.
pos	is the byte position from which to start extraction. This value cannot be zero. If this value is negative, SUBSTR counts backwards from the end of `r`.
len	is the number of bytes from `pos` to extract from `r`. This value must be greater than zero. This parameter is optional.

Defaults

Defaults to the length of position `pos` to the end of `r`.

Return Value

The portion of `r` beginning at `pos` for `len` bytes. Or a null value if `r` is null.

Error and Warning Messages

If you receive an ORA-xxxx error or warning message, refer to the *Oracle Database Error Messages* guide for an explanation and information about how to handle it.

C.1.15 TRANSLATE

TRANSLATE returns the RAW `r` after changing the bytes in `from_set` according to bytes in `to_set`.

Successive bytes in `r` are looked up in `from_set` and if found, the byte at the same offset in `to_set` is copied to the result or omitted from the result if the offset exceeds the length of `to_set`. Bytes that appear in `r` but not in `from_set` are copied to the result. Only the first (leftmost) occurrence of a byte in `from_set` is used and subsequent duplicate occurrences are ignored.

If `from_set` contains more bytes than `to_set`, the extra bytes at the end of `from_set` have no corresponding bytes in `to_set`. Any bytes in `r` matching such uncorresponded `from_set` bytes are omitted from the resulting RAW value.

TRANSLATE differs from TRANSLITERATE in the following ways:

- translation RAWs have no defaults
- `r` bytes undefined in the `to_set` translation RAW are omitted
- resulting RAW value can be shorter than the input RAW value

Syntax

```
function TRANSLATE (r IN RAW,
 from_set IN RAW,
 to_set IN RAW) RETURN RAW;
```

Table C–10 describes the parameters in this function:

Table C–10   TRANSLATE Function Parameters

Parameter	Description
`r`	is the RAW source byte-string to be changed.
`from_set`	is the RAW byte-codes to be matched, if present in `r`.
`to_set`	is the RAW byte-codes to which corresponding `from_set` bytes are changed.

Defaults

None

Return Value

A RAW translated byte-string.

Error and Warning Messages

If you receive an ORA-xxxx error or warning message, refer to the *Oracle Database Error Messages* guide for an explanation and information about how to handle it.

C.1.16 TRANSLITERATE

TRANSLITERATE returns the RAW `r` after replacing all occurrences of any bytes in `from_set` with the corresponding bytes in `to_set`.

Successive bytes in `r` are looked up in `from_set` and, if not found, are copied unaltered to the resulting RAW value. If found, they are replaced in the resulting RAW value by either the byte at the same offset in `to_set`, or the `pad` byte when the offset exceeds the `to_set` length. Bytes found in `r` but not found in `from_set` are copied to the result. Only the first (leftmost) occurrence of a byte in `from_set` is used; subsequent duplicate occurrences are ignored. The result of TRANSLITERATE is always the same length as RAW `r`.

If `to_set` is shorter than `from_set`, then the `pad` byte is placed in the resulting RAW value when a selected `from_set` byte has no corresponding byte in `to_set`.

TRANSLITERATE differs from TRANSLATE in the following ways:

- `r` bytes undefined in `to_set` are padded

- the resulting RAW value is always the same length as the input RAW value

Syntax

```
function TRANSLITERATE (r IN RAW,
 to_set IN RAW DEFAULT NULL,
 from_set IN RAW DEFAULT NULL,
 pad IN RAW DEFAULT NULL) RETURN RAW;
```

Where Table C–11 describes the parameters in this function:

Table C–11   TRANSLITERATE Syntax

Item	Description
r	is the RAW source byte-string to be changed.
to_set	is the RAW byte-codes to which corresponding `from_set` bytes are changed This value can be of any valid RAW length.
from_set	is the RAW byte-codes to be matched, if present in `r`. This value can be of any valid RAW length.
pad	is a 1 byte value used when `to_set` is shorter than `from_set`.

Defaults

Table C–12 describes the TRANSLITERATE function defaults:

Table C–12   Transliterate Function Defaults

Parameter	Description
to_set	defaults to a null value, and effectively extended with `pad` to the length of `from_set` as necessary.
from_set	defaults to x'00 through x'ff'.
pad	defaults to x'00'.

Return Value

A RAW transliterated byte-string.

Error and Warning Messages

If you receive an ORA-xxxx error or warning message, refer to the *Oracle Database Error Messages* guide for an explanation and information about how to handle it.

C.1.17 XRANGE

XRANGE 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`, the succession of result bytes begin with `start_byte`, wrap from x'ff' to x'00', and end at `end_byte`.

If specified, `start_byte` and `end_byte` must be `single-byte` RAW values.

Syntax

```
function XRANGE (start_byte IN RAW DEFAULT NULL,
 end_byte IN RAW DEFAULT NULL) RETURN RAW;
```

Where Table C–13 describes the parameters in this function:

Table C–13   XRANGE Function Parameters

Parameter	Description
start_byte	is the 1-byte beginning byte-code value of the resulting sequence.
end_byte	is the 1-byte ending byte-code value of the resulting sequence.

Defaults

Where Table C–14 describes the XRANGE function defaults:

Table C–14   XRANGE Function Defaults

Parameters	Description
start_byte	defaults to x'00'.
end_byte	defaults to x'ff'.

Return Value

A RAW value containing a succession of 1-byte hexadecimal encodings.

Error and Warning Messages

If you receive an ORA-xxxx error or warning message, refer to the *Oracle Database Error Messages* guide for an explanation and information about how to handle it.

C.2 UTL_PG Functions

The UTL_PG package is an extension to PL/SQL that provides a full set of functions for converting COBOL number formats into Oracle numbers and Oracle numbers into COBOL number formats.

UTL_PG conversion format RAWs are not portable in this release. Additionally, generation of conversion format RAWs on one system and transfer to another system is not supported.

The functions listed in this section are called in the standard PL/SQL manner:

```
package_name.function_name(arguments)
```

Specifically for UTL_PG routines, this is:

```
UTL_PG.function_name(arguments)
```

For each function listed below, the function name, arguments and their datatypes, and the return value datatype are provided. Unless otherwise specified, the parameters are IN, not OUT, parameters.

C.2.1 Common Parameters

The following UTL_PG functions share several similar parameters among themselves:

- RAW_TO_NUMBER

- MAKE_NUMBER_TO_RAW_FORMAT

- MAKE_RAW_TO_NUMBER_FORMAT

- NUMBER_TO_RAW

These similar parameters are described in detail in Table C–15 and then referenced only by name in subsequent tables listing the parameters for each UTL_PG function in this Appendix.

C.2.1.1 Common Input Parameters

Table C–15 describes the input parameters that are common to all of the UTL_PG functions:

Table C–15   Input Parameters Common to UTL_PG Function

Parameter	Description
mask	is the compiler datatype mask. This is the datatype to be converted, specified in the source language of the named compiler (compname). This implies the internal format of the data as encoded according to the compiler and host platform.
maskopts	is the compiler datatype mask options or NULL. These are additional options associated with the mask, as allowed or required, and are specified in the source language of compname. These can further qualify the type of conversion as necessary.
envrnmnt	is the compiler environment clause or NULL. These are additional options associated with the environment in which the remote data resides, as allowed or required, and is specified in the source language of compname. This parameter typically supplies aspects of data conversion dictated by customer standards, such as decimal point or currency symbols if applicable.

Table C–15   (Cont.) Input Parameters Common to UTL_PG Function

Parameter	Description
compname	is the compiler name. The only supported value is IBMVSCOBOLII.
compopts	is the compiler options or NULL.
nlslang	is the zoned decimal code page specified in Oracle NLS format, language_territory.charset. This defaults to AMERICAN_AMERICA.WE8EBCDIC37C.
wind	is the warning indicator. A Boolean indicator which controls whether conversion warning messages are to be returned in the wmsgblk OUT parameter.
wmsgbsiz	is the warning message block declared size in bytes. It is a BINARY_INTEGER set to the byte length of wmsgblk. The warning message block must be at least 512 and not more than 8192 bytes in length. When declaring wmsgblk, plan on approximately 512 bytes for each warning returned, depending on the nature of the requested conversion.

C.2.1.2 Common Output Parameter

Table C–16 describes the output parameter that is common to the UTL_PG functions:

Table C–16   Output Parameters Common to UTL_PG Functions

Parameter	Description
wmsgblk	is the warning message block. It is a RAW value which can contain multiple warnings in both full message and substituted parameter formats, if wind is TRUE. This parameter should be passed to the WMSGCNT function to test if warnings were issued and to WMSG to extract any warning that are present.
	If wind is TRUE and no warnings are issued or if wind is FALSE, the length of wmsgblk is 0. This parameter does not need to be reset before each use. The warning message is documented in the *Oracle Database 10g Error Messages* manual. This parameter must be allocated and passed as a parameter in all cases, regardless of how wind is specified.

C.2.2 RAW_TO_NUMBER

RAW_TO_NUMBER converts a RAW byte-string `r` from the remote host internal format specified by `mask`, `maskopts`, `envrnmnt`, `compname`, `compopts`, and `nlslang` into an Oracle number.

Warnings are issued, if enabled, when the conversion specified conflicts with the conversion implied by the data or when conflicting format specifications are supplied.

For detailed information about the `mask`, `maskopts`, `envrnmnt`, `compname`, and `compopts` arguments, refer to "NUMBER_TO_RAW and RAW_TO_NUMBER Argument Values" on page C-32.

Syntax

```
function RAW_TO_NUMBER (r IN RAW,
 mask IN VARCHAR2,
 maskopts IN VARCHAR2,
 envrnmnt IN VARCHAR2,
 compname IN VARCHAR2,
 compopts IN VARCHAR2,
 nlslang IN VARCHAR2,
 wind IN BOOLEAN,
 wmsgbsiz IN BINARY_INTEGER,
 wmsgblk OUT RAW) RETURN NUMBER;
```

Where Table C–17 describes the parameters in this function:

Table C–17   RAW_TO_NUMBER Function Parameters

Parameter	Description
r	is the remote host data to be converted.
mask	is the compiler datatype mask.
maskopts	are the compiler datatype mask options or NULL.
envrnmnt	is the compiler environment clause or NULL.
compname	is the compiler name.
compopts	are the compiler options or NULL.
nlslang	is the zoned decimal code page in Oracle NLS format.
wind	is a warning indicator.
wmsgbsiz	is the warning message block size in bytes.
wmsgblk	is the warning message block. This is an OUT parameter.

Defaults and Optional Parameters

Table C–18 describes the default and optional parameters of the RAW_TO_NUMBER function:

Table C–18   Optional and Default Parameters of the RAW_TO_NUMBER Function

Parameters	Description
maskopts	null allowed, no default value
envrnmnt	null allowed, no default value
compopts	null allowed, no default value

Return Value

An Oracle number corresponding in value to `r`.

Error and Warning Messages

If you receive an ORA-xxxx error or warning message, refer to the *Oracle Database Error Messages* guide for an explanation and information about how to handle it.

C.2.3 NUMBER_TO_RAW

NUMBER_TO_RAW converts an Oracle number `n` of declared precision and scale into a RAW byte-string in the remote host internal format specified by `mask`, `maskopts`, `envrnmnt`, `compname`, `compopts`, and `nlslang`.

Warnings are issued, if enabled, when the conversion specified conflicts with the conversion implied by the data or when conflicting format specifications are supplied.

For detailed information about the `mask`, `maskopts`, `envrnmnt`, `compname`, and `compopts` arguments, refer to"NUMBER_TO_RAW and RAW_TO_NUMBER Argument Values" on page C-32.

Syntax

```
function NUMBER_TO_RAW (n IN NUMBER,
 mask IN VARCHAR2,
 maskopts IN VARCHAR2,
 envrnmnt IN VARCHAR2,
 compname IN VARCHAR2,
 compopts IN VARCHAR2,
 nlslang IN VARCHAR2,
 wind IN BOOLEAN,
 wmsgbsiz IN BINARY_INTEGER,
 wmsgblk OUT RAW) RETURN RAW;
```

Where Table C–19 describes the parameters in this function:

Table C–19   NUMBER_TO_RAW Function Parameters

Parameter	Description
n	is the Oracle number to be converted.
mask	is the compiler datatype mask.
maskopts	are the compiler datatype mask options or NULL.
envrnmnt	is the compiler environment clause or NULL.
compname	is the compiler name.
compopts	are the compiler options or NULL.
nlslang	is the zoned decimal code page in Oracle NLS format.
wind	is a warning indicator .
wmsgbsiz	is the warning message block size in bytes.
wmsgblk	is the warning message block. This is an OUT parameter.

Defaults and Optional Parameters

Table C–20 describes the defaults and optional parameters for the NUMBER_TO_
RAW function:

Table C–20   Defaults and Optional Parameters for NUMBER_TO_RAW Function

Parameter	Description
maskopts	null allowed, no default value
envrnmnt	null allowed, no default value
compopts	null allowed, no default value

Return Value

A RAW value corresponding in value to n.

Error and Warning Messages

If you receive an ORA-xxxx error or warning message, refer to the *Oracle Database
Error Messages* guide for an explanation and information about how to handle it.

C.2.4 MAKE_RAW_TO_NUMBER_FORMAT

MAKE_RAW_TO_NUMBER_FORMAT makes a RAW_TO_NUMBER format conversion specification used to convert a RAW byte-string from the remote host internal format specified by `mask`, `maskopts`, `envrnmnt`, `compname`, `compopts`, and `nlslang` into an Oracle number of comparable precision and scale.

Warnings are issued, if enabled, when the conversion specified conflicts with the conversion implied by the data or when conflicting format specifications are supplied.

This function returns a RAW value containing the conversion format which can be passed to UTL_PG.RAW_TO_NUMBER_FORMAT.

For detailed information about the `mask`, `maskopts`, `envrnmnt`, `compname`, and `compopts` arguments, refer to "NUMBER_TO_RAW and RAW_TO_NUMBER Argument Values" on page C-32.

Syntax

```
function MAKE_RAW_TO_NUMBER_FORMAT (mask IN VARCHAR2,
 maskopts IN VARCHAR2,
 envrnmnt IN VARCHAR2,
 compname IN VARCHAR2,
 compopts IN VARCHAR2,
 nlslang IN VARCHAR2,
 wind IN BOOLEAN,
 wmsgbsiz IN BINARY_INTEGER,
 wmsgblk OUT RAW) RETURN RAW;
```

Where Table C–21 describes the parameters in this function:

Table C–21   MAKE_RAW_TO_NUMBER_FORMAT Function Parameters

Parameter	Description
`mask`	is the compiler datatype mask.
`maskopts`	are the compiler datatype mask options or NULL.
`envrnmnt`	is the compiler environment clause or NULL.
`compname`	is the compiler name.
`compopts`	are the compiler options or NULL.
`nlslang`	is the zoned decimal code page in Oracle NLS format.
`wind`	is a warning indicator.
`wmsgbsiz`	is the warning message block size in bytes.
`wmsgblk`	is the warning message block. This is an OUT parameter.

Defaults and Optional Parameters

Table C–22 describes the defaults and optional parameters of the
MAKE_RAW_TO_NUMBER_FORMAT function:

Table C–22   Default and Optional MAKE_RAW_TO_NUMBER_FORMAT Parameters

Parameter	Description
maskopts	null allowed, no default value
envrnmnt	null allowed, no default value
compopts	null allowed, no default value

Return Value

A RAW(2048) format conversion specification for RAW_TO_NUMBER.

Error and Warning Messages

If you receive an ORA-xxxx error or warning message, refer to the *Oracle Database
Error Messages* guide for an explanation and information about how to handle it.

C.2.5 MAKE_NUMBER_TO_RAW_FORMAT

MAKE_NUMBER_TO_RAW_FORMAT makes a NUMBER_TO_RAW format conversion specification used to convert an Oracle number of declared precision and scale to a RAW byte-string in the remote host internal format specified by `mask`, `maskopts`, `envrnmnt`, `compname`, `compopts`, and `nlslang`.

Warnings are issued, if enabled, when the conversion specified conflicts with the conversion implied by the data or when conflicting format specifications are supplied.

This function returns a RAW value containing the conversion format which can be passed to UTL_PG.NUMBER_TO_RAW_FORMAT. The implementation length of the result format RAW is 2048 bytes.

For detailed information about the `mask`, `maskopts`, `envrnmnt`, `compname`, and `compopts` arguments, refer to "NUMBER_TO_RAW and RAW_TO_NUMBER Argument Values" on page C-32.

Syntax

```
function MAKE_NUMBER_TO_RAW_FORMAT (mask IN VARCHAR2,
 maskopts IN VARCHAR2,
 envrnmnt IN VARCHAR2,
 compname IN VARCHAR2,
 compopts IN VARCHAR2,
 nlslang IN VARCHAR2,
 wind IN BOOLEAN,
 wmsgbsiz IN BINARY_INTEGER,
 wmsgblk OUT RAW) RETURN RAW;
```

Where Table C–23 describes the parameters in this function:

Table C–23   MAKE_NUMBER_TO_RAW_FORMAT Function Parameters

Parameter	Description
mask	is the compiler datatype mask.
maskopts	are the compiler datatype mask options or NULL.
envrnmnt	is the compiler environment clause or NULL.
compname	is the compiler name.
compopts	are the compiler options or NULL.
nlslang	is the zoned decimal code page in Oracle NLS format.
wind	is a warning indicator.
wmsgbsiz	is the warning message block size in bytes.
wmsgblk	is the warning message block. This is an OUT parameter.

Defaults and Optional Parameters

Table C–24 describes the defaults and optional parameters for the MAKE_NUMBER_TO_RAW_FORMAT function:

Table C–24   Optional, Default Parameters: MAKE_NUMBER_TO_RAW_FORMAT

Parameter	Description
maskopts	null allowed, no default value
envrnmnt	null allowed, no default value
compopts	null allowed, no default value

Return Value

A RAW(2048) format conversion specification for NUMBER_TO_RAW.

Error and Warning Messages

If you receive an ORA-xxxx error or warning message, refer to the *Oracle Database Error Messages* guide for an explanation and information about how to handle it.

C.2.6 RAW_TO_NUMBER_FORMAT

RAW_TO_NUMBER_FORMAT converts, according to the RAW_TO_NUMBER conversion format `r2nfmt`, a RAW byte-string `rawval` in the remote host internal format into an Oracle number.

Syntax

```
function RAW_TO_NUMBER_FORMAT (rawval IN RAW,
 r2nfmt IN RAW) RETURN NUMBER;
```

where Table C–25 describes the parameters in this function:

Table C–25   RAW_TO_NUMBER_FORMAT Function Parameters

Parameter	Description
`rawval`	is the remote host data to be converted.
`r2nfmt`	is a RAW(2048) format specification returned from MAKE_RAW_TO_NUMBER_FORMAT.

Defaults

None

Return Value

An Oracle number corresponding in value to `r`.

Error and Warning Messages

If you receive an ORA-xxxx error or warning message, refer to the *Oracle Database Error Messages* guide for an explanation and information about how to handle it.

C.2.7 NUMBER_TO_RAW_FORMAT

NUMBER_TO_RAW_FORMAT converts, according to the NUMBER_TO_RAW conversion format `n2rfmt`, an Oracle number `numval` of declared precision and scale into a RAW byte-string in the remote host internal format.

Syntax

```
function NUMBER_TO_RAW_FORMAT (numval IN NUMBER,
 n2rfmt IN RAW) RETURN RAW;
```

Where Table C–26 describes the parameters in this function:

Table C–26   NUMBER_TO_RAW_FORMAT Function Parameters

Parameters	Description
numval	is the Oracle number to be converted.
n2rfmt	is a RAW(2048) format specification returned from MAKE_NUMBER_TO_RAW_FORMAT.

Defaults

None

Return Value

A RAW value corresponding in value to `n`.

Error and Warning Messages

If you receive an ORA-xxxx error or warning message, refer to the *Oracle Database Error Messages* guide for an explanation and information about how to handle it.

C.2.8 WMSGCNT

WMSGCNT tests a `wmsgblk` to determine how many warnings, if any, are present.

Syntax

```
function WMSGCNT (wmsgblk IN RAW) RETURN BINARY_INTEGER;
```

Where Table C–27 describes the parameter in this function.

Table C–27   WMSGCNT Function Parameter

Parameter	Description
wmsgblk	is the warning message block returned from one of the following functions: ■ MAKE_NUMBER_TO_RAW_FORMAT ■ MAKE_RAW_TO_NUMBER_FORMAT ■ NUMBER_TO_RAW ■ RAW_TO_NUMBER

Defaults

None

Return Value

A BINARY_INTEGER value equal to the count of warnings present in the RAW `wmsgblk`.

Table C–28 lists possible returned values:

Table C–28   WMSGCNT Return Values

	Description
>0	indicates a count of warnings present in `wmsgblk`.
0	indicates that no warnings are present in `wmsgblk`.

Error and Warning Messages

If you receive an ORA-xxxx error or warning message, refer to the *Oracle Database Error Messages* guide for an explanation and information about how to handle it.

C.2.9 WMSG

WMSG extracts a warning message specified by `wmsgitem` from `wmsgblk`.

Syntax

```
function WMSG (wmsgblk IN RAW,
 wmsgitem IN BINARY_INTEGER,
 wmsgno OUT BINARY_INTEGER,
 wmsgtext OUT VARCHAR2,
 wmsgfill OUT VARCHAR2) RETURN BINARY_INTEGER;
```

Where Table C–29 describes the parameters in this function:

Table C–29   WMSG Function Parameters

Parameter	Description
`wmsgblk`	is a RAW warning message block returned from one of the following functions: ■ MAKE_NUMBER_TO_RAW_FORMAT ■ MAKE_RAW_TO_NUMBER_FORMAT ■ NUMBER_TO_RAW ■ RAW_TO_NUMBER
`wmsgitem`	is a BINARY_INTEGER value specifying which warning message to extract, numbered from 0 for the first warning through n minus 1 for the nth warning.
`wmsgno`	is an OUT parameter containing the BINARY_INTEGER (hexadecimal) value of the warning number. This value, after conversion to decimal, is documented in the *Oracle Database 10g Error Messages* manual.
`wmsgtext`	is a VARCHAR2 OUT parameter value containing the fully-formatted warning message in ORA-xxxxx format, where xxxxx is the decimal warning number documented in the *Oracle Database 10g Error Messages* manual.
`wmsgfill`	is a VARCHAR2 OUT parameter value containing the list of warning message parameters to be substituted into a warning message in the following format: warnparm1;;warnparm2;;...;;warnparmn where each warning parameter is delimited by a double semicolon.

Defaults

None

Return Value

A BINARY_INTEGER value containing a status return code.

A return code of "0" indicates that `wmsgno`, `wmsgtext`, and `wmsgfill` are assigned and valid.

Error and Warning Messages

If you receive an ORA-xxxx error or warning message, refer to the *Oracle Database Error Messages* guide for an explanation and information about how to handle it.

Table C–30 describes the error messages you could receive:

Table C–30   WMSG Function Errors

Error	Description
-1	indicating the warning specified by `wmsgitem` was not found in `wmsgblk`.
-2	indicating an invalid message block.
-3	indicating `wmsgblk` is too small to contain the warning associated with `wmsgitem`. A partial or no warning message might be present for this particular `wmsgitem`.
-4	indicating there are too many substituted warning parameters.

C.3 NUMBER_TO_RAW and RAW_TO_NUMBER Argument Values

This table lists the valid values for the format arguments for NUMBER_TO_RAW and RAW_TO_NUMBER and related functions. Following are examples of some valid COBOL picture masks. Any valid COBOL picture mask may be used. Refer to the appropriate IBM COBOL programming guides for an explanation of COBOL picture masks.

mask: COBOL picture mask

```
  PIC  9(n)        where 1 <= n    <= 18
  PIC S9(n)        where 1 <= n    <= 18
  PIC  9(n)V9(s)   where 1 <= n+s <= 18
  PIC S9(n)V9(s)   where 1 <= n+s <= 18
  PIC S9999999V99
  PIC V99999
  PIC SV9(5)
  PIC  999.00
  PIC  99/99/99
  PIC  ZZZ.99
  PIC  PPP99
  PIC +999.99
  PIC  999.99+
  PIC -999.99
  PIC  999.99-
  PIC $$$$$,$$$.99
  PIC $9999.99DB
  PIC $9999.99CR
```

maskopts: COBOL picture mask options

```
  COMP
  USAGE IS COMP
  USAGE IS COMPUTATIONAL
  COMP-3
  USAGE IS COMP-3
  USAGE IS COMPUTATIONAL-3
  COMP-4
  USAGE IS COMP-4
  USAGE IS COMPUTATIONAL-4
  DISPLAY
  USAGE IS DISPLAY
  SIGN IS LEADING
  SIGN IS LEADING SEPARATE
  SIGN IS LEADING SEPARATE CHARACTER
  SIGN IS TRAILING
  SIGN IS TRAILING SEPARATE
  SIGN IS TRAILING SEPARATE CHARACTER
```

envrnmnt: COBOL environment clause

```
  CURRENCY SIGN IS x where x is a valid currency sign character
  DECIMAL-POINT IS COMMA
```

compname: COBOL compiler name

```
  IBMVSCOBOLII
```

compopts: COBOL compiler options

(no values are supported at this time)

D

Datatype Conversions

You must convert datatypes and data formats properly when you are using the PGAU tool to generate TIPs and when you are developing a custom TIP using PL/SQL and the UTL_RAW and UTL_PG functions.

Read this appendix to learn about datatype conversion as it relates to TIPs.

This appendix contains the following sections:

- Length Checking on page D-2
- Conversion on page D-3

D.1 Length Checking

PGAU-generated TIPs perform length checking at the end of every parameter sent and received.

Table D–1 provides a list of length parameters generated by PGAU:

Table D–1   Length Parameters

Parameter	Description
expected length	Is computed by PGAU when the TIP is generated.
convert length	Is summed by the TIP from each converted field.
send length	Is the transmitted send data length and is also equal to the actual length for send parameters.
receive length	Is the transmitted receive data length.

An exception is raised when the convert length of a sent parameter does not equal its expected length. This occurs if too many or too few send field conversions are performed.

An exception is raised when the convert length of a received parameter does not equal its received length. These length exceptions result when too few or too many conversions are performed.

A warning is issued when the expected length of a received parameter does not equal its convert or received length and data conversion tracing is enabled. This occurs when a maximum length record is expected, but a shorter record is transmitted and correctly converted.

D.1.1 Parameters Over 32K in Length

PGAU generates TIPs that support transmission of individual data parameters which exceed 32K bytes.

PGAU includes this support automatically when PGAU GENERATE processing detects the maximum length of a data parameter exceeding 32K.

This support is driven by the data definitions placed in the PG DD and cannot be selected by the user. To include the support, the data definition must actually or possible exceed 32K. To remove the support, you must decrease the parameter length to less than 32K, REDEFINE the data, and GENERATE the TIP again.

This support tests for field positions crossing the 32K buffer boundaries before and after conversion of those fields which lie across such boundaries. In the case of repeating groups, This can be many fields, for repeating groups, or few fields in the case of simple linear records.

Each test and the corresponding buffer management logic adds overhead.

> **Caution:**   The target of a REDEFINE clause cannot reside in a previously processed buffer. Run-time TIP processing of the fields containing such REDEFINE clauses get unpredictable results.

D.2 Conversion

The PG DD and TIPs generated by PGAU support COBOL, specified as IBMVSCOBOLII when defining data.

D.2.1 USAGE(PASS)

When USAGE(PASS) has been specified on the PGAU DEFINE DATA statement, the following datatype and format conversions are supported:

- PIC X

- PIC G

PIC X Datatype Conversions

PGAU TIPs convert the COBOL X datatype to a PL/SQL CHAR datatype of the same character length. NLS character set translation is also performed.

Note: COBOL lacks a datatype specifically designated for variable length data. It is represented in COBOL as a subgroup containing a PIC 9 length field followed by a PIC X character field. For example:

10 NAME.

  15 LENGTH PIC S9(4).

  15 LETTERS PIC X(30).

Given this context, it cannot be guaranteed that all instances of an S9(4) field followed by an X field are always variable length data. Rather than PGAU TIPs converting the above COBOL group NAME to a VARCHAR, the TIPs instead construct a nested PL/SQL record as follows:

```
TYPE NAME_typ is RECORD (
  LENGTH NUMBER(4,0),
  LETTERS CHAR(30));

TYPE ... is RECORD(
  ...
  NAME NAME_typ,
  ...
```

It is the client application's responsibility (based upon specific knowledge of the remote host data) to extract NAME.LENGTH characters from NAME.LETTERS and assign the result to a PL/SQL VARCHAR, if a VARCHAR is desired.

Character set conversion is performed for single byte encoded:

- remote host character data, using either:

  - DEFINE TRANSACTION NLS_LANGUAGE character set for an entire transaction, or

  - REDEFINE DATA REMOTE_LANGUAGE character set for a single field, if specified.

- local Oracle character data, using either:

  - LANGUAGE character set of integrating server for an entire transaction, or

  - REDEFINE DATA LOCAL_LANGUAGE character set for a single field, if specified.

PIC G Datatype Conversions

PGAU generated TIPs convert the COBOL G datatype to a PL/SQL VARCHAR2 datatype of the same length, allowing 2 bytes for every character position.

Character set conversion is performed for double-byte and multi-byte encoded:

- remote host character data, using either:
  - DEFINE TRANSACTION REMOTE_MBCS character set for an entire transaction, or
  - REDEFINE DATA REMOTE_LANGUAGE character set for a single field, if specified.
- local Oracle character data, using either:
  - DEFINE TRANSACTION LOCAL_MBCS character set for an entire transaction, or
  - REDEFINE DATA LOCAL_LANGUAGE character set for a single field, if specified.

Alphanumeric and DBCS Editing Field Positions

Table D–2 illustrates how PGAU interprets COBOL symbols in datatype conversions, by providing the definitions for the symbols.

Table D–2   COBOL Symbol Definitions

COBOL Symbols	Oracle Definition of COBOL Symbols - Data Content
'B'	blank (1 byte SBCS or 2 bytes DBCS depending on USAGE)
'0'	zero (1 byte SBCS)
'/'	forward slash (1 byte SBCS)
'G'	double byte

Edited positions in COBOL statement data received from the remote host are converted by PGAU along with the entire field and passed to the client application in the corresponding PL/SQL VARCHAR2 output variable.

When editing symbols are present, they are interpreted to mean the remote host field contains the COBOL data content and length indicated. The editing positions are included in the length of the data field, but conversion of all field positions is processed by PGAU as a single string and no special scanning or translation is done for edited byte positions.

Edited positions in COBOL statement data sent to the remote host are converted by PGAU along with the entire PL/SQL VARCHAR2 input variable passed from the client application.

Table D–3 provides an example of how PGAU converts COBOL datatypes:

Table D–3   COBOL-PGAU Conversion

COBOL Datatype	Description of Conversion by PGAU
PIC XXXBBXX	Is an alphanumeric field 7 bytes in length and would be converted in a single UTL_RAW.CONVERT call. No testing or translation is done on the contents of the byte positions indicated by 'B'. While COBOL language rules indicate that these positions contain "blank" in the character set specified for the remote host, what data is actually present is the user's responsibility.
PIC GGBGGG	Is a DBCS field 12 bytes in length and would be converted in a single UTL_RAW.CONVERT call. No testing or translation is done on the contents of the byte positions indicated by 'B'. While COBOL language rules indicate that these positions contain "blank" in the character set specified for the remote host, what data is actually present is the user's responsibility.
PIC 9	PGAU TIPs convert the COBOL 9 datatype to a PL/SQL NUMBER datatype of the same precision and scale. NLS character set translation is also performed on signs, currency symbols, and spaces.
	The following are supported:
	■ COMPUTATIONAL (binary)
	■ COMPUTATIONAL-3 (packed decimal)
	■ COMPUTATIONAL-4 (binary)
	■ DISPLAY (zoned decimal)
	For DISPLAY datatypes, the following sign specifications are supported:
	■ SEPARATE [CHARACTER]
	■ LEADING
	■ TRAILING
	Refer to "NUMBER_TO_RAW and RAW_TO_NUMBER Argument Values" in Appendix C, "The UTL_PG and UTL_RAW Interface" for more information about numeric datatype conversions.
	COMPUTATIONAL-1 and COMPUTATIONAL-2 (floating point) datatypes are not supported.
FILLER	COBOL FILLER fields are recognized by PGAU by the spelling of the element name FILLER. PGAU does not generate any data conversion for such elements, but does require their space be properly allocated to preserve offsets within the records exchanged with the remote host transaction.
	If a RENAMES or REDEFINES definition covers a FILLER element, PGAU generates data conversion statements for the same area when it is referenced as a component of the RENAMES or REDEFINES variable. Such data conversion reflects only the format of the RENAMES or REDEFINES definition and not the bounds of the FILLER definition.

Format Conversion

Table D–4 describes format conversion:

Table D–4   Format Conversion Descriptions

Item	Description
JUSTIFIED | JUSTIFIED RIGHT	This causes remote host transaction data to be converted as a PL/SQL CHAR datatype according to character datatype, as discussed in "PIC X Datatype Conversions" on page D-3, for both IN and OUT parameters.
	IN parameter data passed from the application is stripped of its rightmost blanks and left padded as required. Then it is sent to the remote host.
	OUT parameter data is aligned as it is received from the remote host and padded with blanks as required on the left. Then it is passed to the application.
JUSTIFIED LEFT	This causes warnings to be issued during TIP generation. No alignment is performed. This is treated as documentation.
	The remote host transaction data is converted as a PL/SQL CHAR datatype according to character datatype, as discussed in "PIC X Datatype Conversions" on page D-3, for both IN and OUT parameters.
LENGTH IS field-2	This is an Oracle extension to the data definition as stored in the PG DD. This extension exists only in the PGAU context and is not valid COBOL syntax.
	The purpose of this extension is to provide a means for variable-length character data to be processed more efficiently by the TIP conversion logic. This is an alternative to defining a variable-length PIC X field as PIC X(1) OCCURS DEPENDING ON field-2, where field-2 is the length of the field. With this extension, the same field could be defined as PIC X(5000) LENGTH IS field-2, where field -2 is the length of the field. The TIP is able to pick up the length and do the character set conversion on the field with a single UTL_RAW.CONVERT call instead of using a loop to do the conversion one character at a time.
	Note that the use of this construct does not affect the COBOL program. The PIC X (or PIC G) field is still fixed-length as far as COBOL is concerned, so the position of the data does not change, nor does the amount of data that is transferred between the gateway and the OLTP. However, if the field is the last field in a COBOL definition, then the COBOL program could be modified to send only the number of bytes required to satisfy the length set in the field-2 field referenced by the LENGTH IS clause.
	The LENGTH IS clause can be specified only for PIC X and PIC G fields, and the picture mask for those fields cannot contain editing characters.

Table D–4   (Cont.) Format Conversion Descriptions

Item	Description
OCCURS n TIMES	This causes conversion of exactly 'n' instances of a set of PL/SQL variables to or from a repeating group area within the remote host record, the size of which area equals the group length times 'n' repetitions. PGAU generated TIPs employ PL/SQL RECORDs of TABLEs to implement an array-like subscript on fields within a repeating group. PL/SQL supports a single dimension TABLE, and consequently PGAU supports only a single level of an OCCURS group. Nested OCCURS groups are not supported. The conversion and formatting performed are dictated by the COBOL datatype of each subfield defined within the repeating group, as documented in "PIC X Datatype Conversions" on page D-3 and "Format Conversion" on page D-6.
OCCURS m TO n TIMES DEPENDING ON field-2	This causes conversion of at least 'm' and not over 'n' instances of a set of PL/SQL variables to or from a repeating group area within the remote host record, the size of which area equals the group length times the repetition count contained in the named field. PGAU generated TIPs employ PL/SQL RECORDs of TABLEs to implement an array-like subscript on fields within a repeating group. PL/SQL supports a single dimension TABLE, and consequently PGAU supports only a single level of an OCCURS DEPENDING ON group. Nested OCCURS DEPENDING ON groups are not supported. The conversion and formatting performed are dictated by the COBOL datatype of each subfield defined within the repeating group, as documented in "PIC X Datatype Conversions" on page D-3 and "Format Conversion" on page D-6. Range conversion: PGAU-generated TIPs use a 'FOR ... LOOP' algorithm with a range of 1 to whatever TIMES upper limit was specified. When the TIP has been generated with the DIAGNOSE(PKGEX(DC)) option, the PL/SQL FOR statement which iterates an OCCURS DEPENDING ON repeating group is preceded by an IF test to ensure at TIP runtime that the DEPENDING ON field contains a number which lies within the specified range for which the lower limit need not be 1. An exception is raised if this test fails.
RENAMES item-2 THRU item-3	A single PL/SQL variable declaration corresponds to a RENAMES definition. If all the subfields covered by a RENAMES definition are PIC X, then the PL/SQL variable is a VARCHAR2. Otherwise any non-PIC X subfield causes the PL/SQL variable datatype to be RAW. Lengths of renamed fields do not contribute to the overall parameter data length because the original fields dictate the lengths.

Table D–4   (Cont.) Format Conversion Descriptions

Item	Description
REDEFINES item-2 WHEN item-3=value	The 'WHEN item-3=value' is an Oracle extension to the data definition as stored in the PG DD. This extension exists only in the PGA context and is not valid COBOL syntax.
	The purpose of this extension is to provide a means for the gateway administrator or application developer to specify the criteria by which the redefinition is to be applied. For example, a record type field is often present in a record and different record formats apply depending on which record type is being processed. The specification of which type value applies to which redefinition is typically buried in the transaction programming logic, not in the data definition. To specify which conversion to perform on redefined formats in the TIP, the WHEN criteria was added to PGA data definitions.
	PGAU generates PL/SQL nested record declarations which correspond in name and datatype to the subordinate elements covered by the REDEFINES definition. The standard PGAU datatype determination described in "PIC X Datatype Conversions" on page D-3.
	LEVEL 01 REDEFINE is ignored:
	This permits remote host copybooks to include definitions which REDEFINE other transaction working storage buffers without having to define such buffers in the TIP or alter the copybook used as input for the definition.
SYNCHRONIZED | SYNCHRONIZED RIGHT	This causes the numeric field to be aligned on boundaries as dictated by the remote host environment, compiler language, and datatype.
	Numeric conversion is performed on the aligned data fields according to numeric datatype, as discussed in "PIC X Datatype Conversions" on page D-3, for both IN and OUT parameters.
SYNCHRONIZED LEFT	This causes warnings to be issued during TIP generation and no realignment is performed. This is treated as documentation.
	Numeric conversion is performed on the aligned data fields according to numeric datatype, as discussed in "PIC X Datatype Conversions" on page D-3, for both IN and OUT parameters.

D.2.2 USAGE(ASIS)

When USAGE(ASIS) is specified on the PGAU DEFINE DATA statement, no conversion is performed. Consequently, each such field is simply copied to a PL/SQL RAW of the same byte length. No conversion, translation, or reformatting is done.

D.2.3 USAGE(SKIP)

When USAGE(SKIP) is specified on the PGAU DEFINE DATA statement, no data exchange is performed. The data is skipped as if it did not exist. Consequently, such fields are not selected from the PG DD, not reflected in the TIP logic, and presumed absent from the data streams exchanged with the remote host. The purpose of "SKIP" is to have definitions in the PG DD, but not active, perhaps because a remote host has either removed the field or has yet to include the field. SKIP allows an existing data definition to be used even though some fields do not exist at the remote host.

D.2.4 PL/SQL Naming Algorithms

Delimiters

COBOL special characters in record, group, and element names are translated when PGAU DEFINE inserts definitions into the PG DD, and by PGAU GENERATE when definitions are selected from the PG DD. Special characters are translated as follows:

- hyphen is translated to underscore (_)

- period is deleted

Qualified Compound Names

PL/SQL variable names are fully qualified and composed from:

- PL/SQL record name as the leftmost qualifier corresponding to level 01 or 77 COBOL record name.

- PL/SQL nested record names corresponding to COBOL group names.

- PL/SQL nested fields corresponding to COBOL elements of datatype:

  - CHAR or NUMBER corresponding to non-repeating COBOL elements.

  - TABLE corresponding to COBOL elements which fall within an OCCURS or OCCURS DEPENDING ON group (COBOL repeating fields correspond to PL/SQL nested RECORDs of TABLE's).

Note that when referencing PL/SQL variables from calling applications, the TIP package name must be prefixed as the leftmost qualifier. Thus the fully qualified reference to the PL/SQL variable which corresponds to:

- SKILL is:

```
tipname.EMPREC_Typ.SKILL(SKILL_Key)
```

- HOME_ADDRESS ZIP is:

```
tipname.EMPREC_Typ.HOME_ADDRESS.ZIP.FIRST_FIVE
tipname.EMPREC_Typ.HOME_ADDRESS.ZIP.LAST_FOUR
```

Truncated and Non-Unique Names

PGAU truncates field names and corresponding PL/SQL variable names when the name exceeds:

- 26 bytes for fields within an aggregate record or group

  This is due to the need to suffix each field or PL/SQL variable name with:

  - "_Typ" for group names

  - "_Tbl" for element names with a repeating group

  or

- 30 bytes due to the PL/SQL limitation of 30 bytes for any name

  The rightmost four characters are truncated. This imposes the restriction that names be unique to 26 characters.

Duplicate Names

COBOL allows repetitive definition of the same group or element names within a record, and the context of the higher level groups serves to uniquely qualify names. However, because PGAU-generated TIPs declare PL/SQL record variables which reference nested PL/SQL records for subordinate groups and fields, such nested PL/SQL record types can have duplicate names.

Given the following COBOL definition, note that ZIP is uniquely qualified in COBOL, but the corresponding PL/SQL declaration would have a duplicate nested record type for ZIP.

```
01  EMPREC.
    05 HIREDATE            PIC X(8).
    05 BIRTHDATE           PIC X(8).
    05 SKILL               PIC X(12) OCCURS 4.
    05 EMPNO               PIC 9(4).
    05 EMPNAME.
       10 FIRST-NAME       PIC X(10).
       10 LAST-NAME        PIC X(15).
    05 HOME-ADDRESS.
       10 STREET           PIC X(20).
       10 CITY             PIC X(15).
       10 STATE            PIC XX.
       10 ZIP.
          15 FIRST-FIVE    PIC X(5).
          15 LAST-FOUR     PIC X(4).
    05 DEPT                PIC X(45).
    05 OFFICE-ADDRESS.
       10 STREET           PIC X(20).
       10 CITY             PIC X(15).
       10 STATE            PIC XX.
       10 ZIP.
          15 FIRST-FIVE    PIC X(5).
          15 LAST-FOUR     PIC X(4).
    05 JOBTITLE            PIC X(20).
```

PGAU avoids declaring duplicate nested record types, and generates the following PL/SQL:

```
SKILL_Key BINARY_INTEGER;
TYPE SKILL_Tbl is TABLE of CHAR(12)
        INDEX by BINARY_INTEGER;
    TYPE EMPNAME_Typ is RECORD (
        FIRST_NAME          CHAR(10),
        LAST_NAME           CHAR(15));
    TYPE ZIP_Typ is RECORD (
        FIRST_FIVE          CHAR(5),
        LAST_FOUR           CHAR(4));
    TYPE HOME_ADDRESS_Typ is RECORD (
        STREET              CHAR(20),
        CITY                CHAR(15),
        STATE               CHAR(2),
        ZIP                 ZIP_Typ);
    TYPE OFFICE_ADDRESS_Typ is RECORD (
        STREET              CHAR(20),
        CITY                CHAR(15),
        STATE               CHAR(2),
        ZIP                 ZIP_Typ);

    TYPE EMPREC_Typ is RECORD (
```

```
        HIREDATE                CHAR(8),
        BIRTHDATE               CHAR(8),
        SKILL                   SKILL_Tbl,
        EMPNO                   NUMBER(4,0),
        EMPNAME                 EMPNAME_Typ,
        HOME_ADDRESS            HOME_ADDRESS_Typ,
        DEPT                    CHAR(45),
        OFFICE_ADDRESS          OFFICE_ADDRESS_Typ,
        JOBTITLE                CHAR(20));
```

However, in the case where multiple nested groups have the same name but have different subfields (see ZIP following):

```
05 HOME-ADDRESS.
      10 STREET              PIC X(20).
      10 CITY                PIC X(15).
      10 STATE               PIC XX.
      10 ZIP.
          15 LEFTMOST-FOUR   PIC X(4).
          15 RIGHMOST-FIVE   PIC X(5).
05 DEPT                      PIC X(45).
05 OFFICE-ADDRESS.
   10 STREET                 PIC X(20).
   10 CITY                   PIC X(15).
   10 STATE                  PIC XX.
   10 ZIP.
      15 FIRST-FIVE          PIC X(5).
      15 LAST-FOUR           PIC X(4).
05 JOBTITLE                  PIC X(20).
```

PGAU alters the name of the PL/SQL nested record type for each declaration in which the subfields differ in name, datatype, or options. Note the "02" appended to the second declaration (ZIP_Typ02), and its reference in OFFICE_ADDRESS.

```
TYPE EMPNAME_Typ is RECORD (
    FIRST_NAME              CHAR(10),
    LAST_NAME               CHAR(15));
TYPE ZIP_Typ is RECORD (
    LEFTMOST_FOUR           CHAR(4),
    RIGHTMOST_FIVE          CHAR(5));
TYPE HOME_ADDRESS_Typ is RECORD (
    STREET                  CHAR(20),
    CITY                    CHAR(15),
    STATE                   CHAR(2),
    ZIP                      ZIP_Typ);
TYPE ZIP_Typ02 is RECORD (
    FIRST_FIVE              CHAR(5),
    LAST_FOUR                CHAR(4));
TYPE OFFICE_ADDRESS_Typ is RECORD (
    STREET                  CHAR(20),
    CITY                    CHAR(15),
    STATE                   CHAR(2),
    ZIP                     ZIP_Typ02);
TYPE EMPREC_Typ is RECORD (
    HIREDATE                CHAR(8),
    BIRTHDATE               CHAR(8),
    SKILL                   SKILL_Tbl,
    EMPNO                   NUMBER(4,0),
    EMPNAME                 EMPNAME_Typ,
    HOME_ADDRESS            HOME_ADDRESS_Typ,
    DEPT                    CHAR(45),
```

```
OFFICE_ADDRESS          OFFICE_ADDRESS_Typ,
JOBTITLE                 CHAR(20));
```

And the fully qualified reference to the PL/SQL variable which corresponds to:

- HOME_ADDRESS.ZIP is:

```
tipname.EMPREC_Typ.HOME_ADDRESS.ZIP.LEFTMOST_FOUR
tipname.EMPREC_Typ.HOME_ADDRESS.ZIP.RIGHTMOST_FIVE
```

- OFFICE_ADDRESS.ZIP is:

```
tipname.EMPREC_Typ.OFFICE_ADDRESS.ZIP.FIRST_FIVE
tipname.EMPREC_Typ.OFFICE_ADDRESS.ZIP.LAST_FOUR
```

Note that the nested record type name ZIP_Typ02 is not used in the reference, but is implicit within PL/SQL's association of the nested records.

E

National Language Support

This appendix documents the National Language Support (NLS) information for the gateway. More information about using NLS is in the *Oracle Database 10g Server Application Developer's Guide.*

This appendix includes the following sections:

- "Overview" on page E-2
- "Languages Supported for Messages" on page E-2
- "Languages Supported for Data Conversion" on page E-3

E.1 Overview

National Language Support is a technology that enables Oracle applications to interact with users in their native language, using their conventions for displaying data.

The Oracle NLS architecture is data-driven, enabling support for specific languages and character encoding schemes to be added without requiring any changes in source code.

E.2 Languages Supported for Messages

The Oracle Procedural Gateway for APPC is capable of issuing its messages in languages other than English.

Table E–1 shows the currently supported languages in the left column, and the right column presents each language's corresponding language, territory, and recommended character set specifications for use in the language setting. Other character sets can be used, but care must be taken to ensure that the character set used contains the full set of characters required by the specified language.

Table E–1   Languages Supported for Messages Issued by the Gateway

Language	Setting for language_territory.charset
American English	AMERICAN_AMERICA.US7ASCII
Brazilian Portuguese	BRAZILIAN PORTUGUESE_BRAZIL.WE8ISO8859P1
Czech	CZECH_CZECH REPUBLIC.EE8ISO8859P2
Danish	DANISH_DENMARK.WE8ISO8859P1
Dutch	DUTCH_THE NETHERLANDS.WE8ISO8859P1
Egyptian Abrabic	ARABIC_EGYPT.AR8ISO8859P6
French	FRENCH_FRANCE.WE8ISO8859P1
German	GERMAN_GERMANY.WE8ISO8859P1
Greek	GREEK_GREECE.EL8ISO8859P7
Hungarian	HUNGARIAN_HUNGARY.EE8ISO8859P2
Israeli Hebrew	HEBREW_ISRAEL.IW8ISO8859P8
Italian	ITALIAN_ITALY.WE8ISO8859P1
Japanese	JAPANESE_JAPAN.JA16SJIS
Korean	KOREAN_KOREA.KO16KSC5601
Norwegian	NORWEGIAN_NORWAY.WE8ISO8859P1
Polish	POLISH_POLAND.EE8ISO8859P2
Portuguese	PORTUGUESE_PORTUGAL.WE8ISO8859P1
Romanian	ROMANIAN_ROMANIA.EE8ISO8859P2
Russian	RUSSIAN_CIS.CL8ISO8859P5
Simplified Chinese	SIMPLIFIED CHINESE_CHINA.ZHS16GBK
Slovak	SLOVAK_SLOVAKIA.EE8ISO8859P2
Spanish	SPANISH_SPAIN.WE8ISO8859P1
Spainish Catalan	CATALAN_SPAIN.WE8ISO8859P1
Swedish	SWEDISH_SWEDEN.WE8ISO8859P1

Table E–1 (Cont.) Languages Supported for Messages Issued by the Gateway

Language	Setting for language_territory.charset
Thai	THAI_THAILAND.TH8TISASCII
Traditional Chinese	TRADITIONAL CHINESE_TAIWAN.ZHT16BIG5
Turkish	TURKISH_TURKEY.WE8ISO8859P9

The language used by the Oracle Procedural Gateway for APPC server is specified by the LANGUAGE parameter in the **init***sid***.ora** file for the gateway. The syntax for the LANGUAGE parameter is:

```
LANGUAGE=language[_territory.charset]
```

where `language`, `territory`, and `charset` are valid values from the preceding table. If no LANGUAGE parameter is specified in the **init***sid***.ora** file, the default language used is American English.

The language used by the Procedural Gateway Administration Utility, PGAU, is specified by the NLS_LANG environment variable, in the same syntax as described for the LANGUAGE parameter for the server. If the NLS_LANG environment variable is not set, then the default language used is American English.

E.3 Languages Supported for Data Conversion

The Oracle Procedural Gateway for APPC does no data conversion in the gateway itself. Instead, all conversion of data flowing between the integrating server and the gateway is performed by the Oracle Integrating Server. This is accomplished through calls within the TIP to the UTL_RAW.CONVERT function, which converts data from the local character set at the integrating server to the character set of the OLTP system.

The character set of the remote OLTP system is specified for:

- single-byte encoded data, using either:

  - PGAU DEFINE TRANSACTION NLS_LANGUAGE character set for an entire transaction

  - PGAU REDEFINE DATA REMOTE_LANGUAGE character set for a single field, if specified.

- double-byte and multi-byte encoded data, using either:

  - PGAU DEFINE TRANSACTION REMOTE_MBCS character set for an entire transaction, or

  - PGAU REDEFINE DATA REMOTE_LANGUAGE character set for a single field, if specified.

This information is generated into the TIP automatically by PGAU.

> **Attention:** It is extremely important to ensure that the character set or codepage of the OLTP data be specified for PGAU using these parameters:
>
> - DEFINE TRANSACTION . . . NLS_LANGUAGE
> - DEFINE TRANSACTION . . .REMOTE_MBCS
> - REDEFINE DATA   . . . REMOTE_LANGUAGE

Languages and character sets supported by the UTL_RAW.CONVERT function are the same as those supported by the Oracle Integrating Server. Refer to the platform-specific documentation for your Oracle Integrating Server for information on which languages and character sets are supported. If your integrating server is on an ASCII platform, then the platform-specific documentation might not list any of the EBCDIC character sets. This does not necessarily mean they are not supported. You should contact Oracle Support Services in this case.

F

Tip Internals

PGAU generates complete and operational TIPs for most circumstances. TIP internals information is provided to assist you in diagnosing problems with PGAU-generated TIPs, and in writing custom TIPs, if you choose to do so.

❏ **If your gateway is using the SNA communication protocol:**

This appendix refers to a sample called **pgadb2i**. The source for this TIP is in file **pgadb2i.sql** in the **%ORACLE_HOME%\pg4appc\demo\CICS** directory.

❏ **If your gateway is using the TCP/IP communication protocol:**

This appendix refers to a sample called **pgaims**. The source for this TIP is in file **pgtflipd.sql** in the **%ORACLE_HOME%\pg4appc\demo\IMS** directory.

This appendix contains the following sections:

- "Background Reading" on page F-2
- "PL/SQL Package and TIP File Separation" on page F-2

F.1 Background Reading

Several topics are important to understanding TIP operation and development; following is a list of concepts that are key to TIP operation and suggested sources to which you can refer for more information.

- For information about PL/SQL Packages, refer to the *PL/SQL User's Guide and Reference*.

- For information about PGA Application Concepts, refer to the following chapters in this guide:

- **If your communication protocol is SNA:** refer to Chapter 4, "Client Application Development (SNA Only)";

- **If your communication protocol is TCP/IP:** refer to Chapter 7, "Client Application Development (TCP/IP Only)".

- For information about PGA RPC Interface, refer to Appendix B, "Gateway RPC Interface".

- For information about PGA UTL_PG/UTL_RAW Interface, refer to Appendix C, "The UTL_PG and UTL_RAW Interface".

F.2 PL/SQL Package and TIP File Separation

PGAU GENERATE writes each output TIP into a standard PL/SQL package specification file and body file. This separation is beneficial and important. Refer to the *Oracle Database 10g Application Developer's Guide* and the *PL/SQL User's Guide and Reference* for more information. Also refer to "GENERATE" on page 2-17 in Chapter 2, "Procedural Gateway Administration Utility" for more information about building the PL/SQL package.

TIPs are PL/SQL packages. Any time a package specification is recompiled, all objects which depend on that package are invalidated and implicitly recompiled as they are referenced, even if the specification did not change.

Objects which depend on a TIP specification include client applications that call the TIP to interact with remote host transactions.

It might be important to change the TIP body for the following reasons:

- Oracle ships maintenance which affects the TIP body, or

- Oracle ships maintenance for the UTL_RAW or UTL_PG conversion functions upon which the TIP body relies.

  Refer to Appendix C, "The UTL_PG and UTL_RAW Interface" for more detailed information about these functions.

- if the remote host network or program location parameters have changed. Refer to "DEFINE TRANSACTION" in Chapter 2, "Procedural Gateway Administration Utility" for more information

Provided that the TIP specification does not need to change or be recompiled, the TIP body can be regenerated and recompiled to pick up changes without causing invalidation and implicit recompilation of client applications that call the TIP.

It is for this reason that PGAU now separates output TIPs into specification and body files. Refer to "GENERATE" on page 2-17 in Chapter 2, "Procedural Gateway Administration Utility" for a discussion of file identification.

F.2.1 Independent TIP Body Changes

Independent TIP body changes are internal and require no change to the TIP specification. Examples of such changes include: a change in UTL_RAW or UTL_PG conversions, inclusion of diagnostics, or a change to network transaction parameters.

In these cases, when PGAU is used to regenerate the TIP, the new TIP specification file can be saved or discarded, but should not be recompiled. The new TIP body should be recompiled under SQL*Plus. Provided that the TIP body change is independent, the new body compilation completes without errors and the former TIP specification remains valid.

F.2.1.1 Determine if a Specification Has Remained Valid

To determine if a specification has remained valid, issue the following statements from SQL*Plus, depending upon your communication protocol:

- **If your gateway is using the SNA communication protocol,** issue the following:

```
SQL>  column ddl_date format A22 heading 'LAST_DDL'
SQL> select object_name,
  2    object_type,
  3    to_char(last_ddl_time,'MON-DD-YY HH:MM:SS') ddl_date,
  4    status
  5    from all_objects where owner = 'PGAADMIN'
  6    order by object_type, object_name;


OBJECT_NAME  OBJECT_TYPE   LAST_DDL              STATUS
-----------  -----------   --------------------  ---------
PGADB2I      PACKAGE       NOV-24-1999 09:09:13  VALID
PGADB2I      PACKAGE BODY  NOV-24-1999 09:11:44  VALID
DB2IDRIV     PROCEDURE     DEC-30-1999 12:12:14  VALID
DB2IDRVM     PROCEDURE     DEC-30-1999 12:12:53  VALID
DB2IFORM     PROCEDURE     DEC-14-1999 11:12:24  VALID
```

The LAST_DDL column is the date and time at which the last DDL change against the object was done. It shows that the order of compilation was:

```
PGADB2I PACKAGE (the specification)
DB2IDRVM PROCEDURE (1st client application depending on PGADB2I)
DB2IFORM PROCEDURE (2nd client application depending on PGADB2I)
DB2IDRIV PROCEDURE (3rd client application depending on PGADB2I)
PGADB2I PACKAGE BODY (a recompilation of the body)
```

Note that the recompilation of the body does not invalidate its dependent object, the specification, or the client application indirectly.

- **If your gateway is using the TCP/IP communication protocol,** issue the following fro SQL*Plus:

```
SQL>  column ddl_date format A22 heading 'LAST_DDL'
SQL> select object_name,
  2   object_type,
  3   to_char(last_ddl_time,'MON-DD-YY HH:MM:SS') ddl_date,
  4   status
  5   from all_objects where owner = 'PGAADMIN'
  6   order by object_type, object_name;


OBJECT_NAME   OBJECT_TYPE    LAST_DDL              STATUS
-----------   -----------    --------------------  ---------
PGTFLIP       PACKAGE        APR-24-03 03:04:58    VALID
PGTFLIP       PACKAGE BODY   APR-24-03 03:04:02    VALID
PGTFLIPD      PROCEDURE      APR-24-03 03:04:09    VALID
```

The LAST_DDL column is the date and time at which the last DDL change against the object was done. It shows that the order of compilation was:

```
PGTFLIP PACKAGE (the specification)
PGTFLIPD PROCEDURE (client application depending on PGADB2I)
PGTFLIP PACKAGE BODY (a recompilation of the body)
```

Note that the recompilation of the body does not invalidate its dependent object, the specification, or the client application indirectly.

F.2.2 Dependent TIP Body or Specification Changes

You can also change the data structures or call exchange sequences of the remote host transaction. However, this kind of change is exposed to dependent client applications because the public datatypes or functions in the TIP specification will also change and necessitate recompilation, which in turn causes the Oracle server to recompile such dependent client applications.

- **If your gateway is using the SNA communication protocol**, issue the following:

```
SQL> column ddl_date format A22 heading 'LAST_DDL'
SQL> select object_name,
  2   object_type,
  3   to_char(LAST_DDL_TIME,'MON-DD-YY HH:MM:SS') ddl_date,
  4   status
  5   from all_objects where owner = 'PGAADMIN'
  6   order by object_type, object_name;


OBJECT_NAME   OBJECT_TYPE    LAST_DDL              STATUS
-----------   -----------    --------------------  ---------
PGADB2I       PACKAGE        NOV-24-1999 09:09:13  VALID
PGADB2I       PACKAGE BODY   NOV-24-1999 09:11:44  INVALID
DB2IDRIV      PROCEDURE      DEC-30-1999 12:12:14  INVALID
DB2IDRVM      PROCEDURE      DEC-30-1999 12:12:53  INVALID
DB2IFORM      PROCEDURE      DEC-14-1999 11:12:24  INVALID
```

- **If your gateway is using the TCP/IP communication protocol,** issue the following:

```
SQL> column ddl_date format A22 heading 'LAST_DDL'
SQL> select object_name,
  2    object_type,
  3    to_char(LAST_DDL_TIME,'MON-DD-YY HH:MM:SS') ddl_date,
  4    status
  5    from all_objects where owner = 'PGAADMIN'
  6    order by object_type, object_name;

OBJECT_NAME    OBJECT_TYPE    LAST_DDL               STATUS
----------     -----------    --------------------   ---------
PGTFLIP        PACKAGE        APR-24-03 03:04:58     VALID
PGTFLIP        PACKAGE BODY   APR-24-03 05:03:52     INVALID
PGTFLIP        PROCEDURE      APR-24-03 05:04:29     INVALID
```

F.2.2.1 Recompile the TIP Body

Note that the recompilation of the specification has invalidated its dependent objects, the three client applications in addition to the package body. To complete these changes, the body must be recompiled to bring it into compliance with the specification and then the three client applications could be compiled manually, or the Oracle server compiles them automatically as they are referenced.

If the client applications are recompiled by the Oracle server as they are referenced, there is a one-time delay during recompilation.

Recompilation errors in the client application, if any, are due to:

- customer changes in the client application source

- an altered PG DD definition for the TIP if the TIP has been regenerated

- the wrong version being generated from multiple transaction entry versions saved in the PG DD if the TIP has been regenerated

F.2.3 Inadvertent Alteration of TIP Specification

If you make a mistake when you generate a tip (for example, if you alter a PG DD transaction definition, or if you've inadvertently specified the wrong version during regeneration), then the recompiled body will not match the stored specification; as a result, the Oracle Integrating Server would invalidate the specification and any dependent client applications.

You may have to regenerate and recompile the TIP and its dependent client applications to restore correct operation.

Refer to "Listing Dependency Management Information," in the *Oracle Database 10g Server Application Developer's Guide* for more information.

G

Administration Utility Samples

Use the following sample input statements and report output for the Procedural Gateway Administration Utility to guide you in designing your own PGAU statements.

This appendix contains these sample PGAU statements:

G.1 Sample PGAU DEFINE DATA Statements

```
DEFINE DATA EMPNO
       PLSDNAME (EMPNO)
       USAGE (PASS)
       LANGUAGE (IBMVSCOBOLII)
       (
       01 EMP-NO PIC X(6).
       );

DEFINE DATA EMPREC
       PLSDNAME (DCLEMP)
       USAGE (PASS)
       LANGUAGE (IBMVSCOBOLII)
       INFILE("emp.cob");
```

where the file **emp.cob** contains the following:

```
01 DCLEMP.
   10 EMPNO            PIC X(6).
   10 FIRSTNME.
     49 FIRSTNME-LEN    PIC S9(4) USAGE COMP.
     49 FIRSTNME-TEXT   PIC X(12).
   10 MIDINIT          PIC X(1).
   10 LASTNAME.
     49 LASTNAME-LEN    PIC S9(4) USAGE COMP.
     49 LASTNAME-TEXT   PIC X(15).
   10 WORKDEPT         PIC X(3).
   10 PHONENO          PIC X(4).
   10 HIREDATE         PIC X(10).
   10 JOB              PIC X(8).
   10 EDLEVEL          PIC S9(4) USAGE COMP.
   10 SEX              PIC X(1)
   10 BIRTHDATE        PIC X(10).
   10 SALARY           PIC S9999999V99 USAGE COMP-3.
   10 BONUS            PIC S9999999V99 USAGE COMP-3.
   10 COMM             PIC S9999999V99 USAGE COMP-3.

DEFINE DATA DB2INFO
       PLSDNAME (DB2)
       USAGE (PASS)
       LANGUAGE (IBMVSCOBOLII)
       INFILE("db2.cob");
```

where the file **db2.cob** contains the following:

```
01  DB2.
    05 SQLCODE         PIC S9(9) COMP-4.
    05 SQLERRM.
      49 SQLERRML      PIC S9(4) COMP-4.
      49 SQLERRT       PIC X(70).
    05 DSNERRM.
      49 DSNERRML      PIC S9(4) COMP-4.
      49 DSNERRMT      PIC X(240) OCCURS 8 TIMES
                                      INDEXED BY ERROR-INDEX
```

G.2 Sample PGAU DEFINE CALL Statements

```
DEFINE CALL DB2IMAIN
       PKGCALL (PGADB2I_MAIN)
       PARMS ( (EMPNO      IN ),
               (EMPREC     OUT) );
DEFINE CALL DB2IDIAG
       PKGCALL (PGADB2I_DIAG)
       PARMS ( (DB2INFO    OUT) );
```

G.3 Sample PGAU DEFINE TRANSACTION Statement

```
DEFINE TRANSACTION DB2I
   CALL (  DB2IMAIN,
           DB2IDIAG  )
   SIDEPROFILE(CICSPROD)
   TPNAME(DB2I)
   LOGMODE(ORAPLU62)
   SYNCLEVEL(0)
   NLS_LANGUAGE("AMERICAN_AMERICA.WE8EBCDIC37C");
```

G.4 Sample PGAU GENERATE Statement

```
GENERATE DB2I
   PKGNAME(PGADB2I)
   OUTFILE("pgadb2i");
```

A user's high-level application now uses this TIP by referencing these PL/SQL datatypes passed and returned.

Table G–1 provides a description of the TIP user transaction datatypes in package name PGADB2I:

Table G–1   TIP User Transaction Datatypes Used in Package Name PGADB2I

Datatype	Description
PGADB2I.EMPNO	is a PL/SQL variable corresponding to COBOL EMPNO.
PGADB2I.DCLEMP	Which is a PL/SQL RECORD corresponding to COBOL DCLEMP.
PGADB2I.DB2	Which is a PL/SQL RECORD corresponding to COBOL DB2INFO.

and the application calls:

```
PGADB2I.PGADB2I_INIT(trannum);
PGADB2I.PGADB2I_MAIN( trannum, empno, emprec );
PGADB2I.PGADB2I_DIAG( trannum, db2 );
PGADB2I.PGADB2I_TERM(trannum, termtype);
```

G.5 Sample Implicit Versioning Definitions

The examples are sample definitions of DATA, CALL, and TRANSACTION entries with implicit versioning.

This example creates a new DATA version of 'EMPREC' because 'EMPREC' DATA was defined previously:

```
DEFINE DATA EMPREC
       PLSDNAME (NEWEMP)
       USAGE (PASS)
       LANGUAGE (IBMVSCOBOLII)
       INFILE("emp2.cob");
```

where the file **emp2.cob** contains the following:

```
01 NEWEMP.
    10 EMPNO                  PIC X(6).
    10 FIRSTNME.
      49 FIRSTNME-LEN         PIC S9(4) USAGE COMP.
      49 FIRSTNME-TEXT        PIC X(12).
    10 MIDINIT                PIC X(1).
    10 LASTNAME.
      49 LASTNAME-LEN         PIC S9(4) USAGE COMP.
      49 LASTNAME-TEXT        PIC X(15).
    10 WORKDEPT               PIC X(3).
    10 PHONENO                PIC X(3).
    10 HIREDATE               PIC X(10).
    10 JOB                    PIC X(8).
    10 EDLEVEL                PIC S9(4) USAGE COMP.
    10 SEX                    PIC X(1).
    10 BIRTHDATE              PIC X(10).
    10 SALARY                 PIC S9999999V99 USAGE COMP-3.
    10 BONUS                  PIC S9999999V99 USAGE COMP-3.
    10 COMM                   PIC S9999999V99 USAGE COMP-3.
    10 YTD.
      15 SAL                  PIC S9(9)V99 USAGE COMP-3.
      15 BON                  PIC S9(9)V99 USAGE COMP-3.
      15 COM                  PIC S9(9)V99 USAGE COMP-3.
```

To determine which DATA version number was assigned, this SQL query can be issued:

```
SELECT MAX(pd.version)
       FROM pga_data pd
       WHERE pd.dname = 'EMPREC';
```

To determine additional information related to the updated version of 'EMPREC' this query can be used:

```
SELECT *
       FROM pga_data pd
       WHERE pd.dname = 'EMPREC';
```

This example creates a new CALL version of 'DB2IMAIN' because the 'DB2IMAIN' CALL was defined previously:

```
DEFINE CALL DB2IMAIN
       PKGCALL (PGADB2I_MAIN)
       PARMS ( (EMPNO     IN                  ),
               (EMPREC    OUT   VERSION(ddddd) )  );
```

where `ddddd` is the version number of the EMPREC DATA definition queried after the previous DEFINE DATA updated EMPREC.

To determine which call version number was assigned, this SQL query can be issued:

```
SELECT MAX(pc.version)
       FROM pga_call pc
       WHERE pc.cname = 'DB2IMAIN';
```

To determine additional information related to the updated version of 'DB2IMAIN' this query can be used:

```
 SELECT *
        FROM pga_call pc
        WHERE pc.cname = 'DB2IMAIN';
```

The DEFINE TRANSACTION example creates a new TRANSACTION version of 'DB2I' because the 'DB2I' TRANSACTION was defined previously. The essential difference of the new version of the DB2I transaction is that the first call uses a new PL/SQL record format "NEWEMP" (which corresponds to the COBOL NEWEMP format) to query the employee data.

> **Caution:** Record format changes like that discussed above must be synchronized with the requirements of the remote transaction program. Changes to the PGA TIP alone result in errors. A new remote transaction program with the corequisite changes could be running on a separate CICS system and started through the change from "CICSPROD" to "CICSTEST" in the SIDEPROFILE parameter below.

```
DEFINE TRANSACTION DB2I
       CALL (  DB2IMAIN   VERSION (ccccc),
               DB2IDIAG )
       SIDEPROFILE(CICSTEST)
       TPNAME(DB2I)
       LOGMODE(ORAPLU62)
       SYNCLEVEL(0)
       NLS_LANGUAGE("AMERICAN_AMERICA.WE8EBCDIC37C");
```

where `ccccc` is the version number of the DB2IMAIN CALL definition queried after the previous DEFINE CALL updated DB2IMAIN.

There are two versions of the DB2I transaction definition in the PGA DD. The original uses the old "DCLEMP" record format and starts transaction "DB2I" on the production CICS system. The latest uses the "NEWEMP" record format and starts transaction "DB2I" on the test CICS system.

To determine which transaction version number was assigned, this SQL query can be issued:

```
SELECT MAX(pt.version)
  FROM pga_trans pt
  WHERE pt.tname = 'DB2I';
```

To determine additional information related to the updated version of 'DB2I' this query can be used:

```
SELECT *
  FROM pga_trans pt
  WHERE pt.tname = 'DB2I';
```

This example generates a new package using the previously defined new versions of the TRANSACTION, CALL, and DATA definitions:

```
GENERATE DB2I
VERSION(ttttt)
PKGNAME(NEWDB2I)
OUTFILE("pgadb2i");
```

where ttttt is the version number of the DB2I TRANSACTION definition queried after the previous DEFINE TRANSACTION updated DB2I.

Note that the previous PL/SQL package files **pgadb2i.pkh** and **pgadb2i.pkb** are overwritten. To keep the new package separate, change the output file specification. For example:

```
GENERATE DB2I
VERSION(ttttt)
PKGNAME(NEWDB2I)
OUTFILE("newdb2i");
```

A user's high-level application now uses this TIP by referencing the PL/SQL datatypes passed and returned.

Table G–2 provides a description of the TIP user transaction datatypes in package name NEWDB2I:

Table G–2   TIP User Transaction Datatypes for Package Name NEWDB2I

Datatype	Description
NEWDB2I.EMPNO	Is a PL/SQL variable corresponding to COBOL EMPNO.
NEWDB2I.NEWEMP	Is a PL/SQL RECORD corresponding to COBOL NEWEMP.
NEWDB2I.DB2	Is a PL/SQL RECORD corresponding to COBOL DB2.

and the application calls:

```
NEWDB2I.PGADB2I_INIT(trannum);
NEWDB2I.PGADB2I_MAIN( trannum, empno, newemp );
NEWDB2I.PGADB2I_DIAG( trannum, db2 );
NEWDB2I.PGADB2I_TERM(trannum, termtype);
```

G.6 Sample PGAU REDEFINE DATA Statements

Single-field redefinition in which EDLEVEL USAGE becomes COMP-3:

```
REDEFINE DATA EMPREC
        PLSDNAME(DCLEMP)
        LANGUAGE(IBMVSCOBOLII)
        FIELD(EDLEVEL)
        PLSFNAME(PLSRECTYPE)
        (
        10 EDLEVEL  PIC S9(4) USAGE IS COMP-3.
        );
```

By default, this redefines the latest version of EMPREC which implicitly affects the latest call and transaction definitions which refer to it.

Sample multi-field redefinition in which the employee's first and last name fields are expanded and the employee's middle initial is removed.

```
REDEFINE DATA EMPREC
        VERSION(1)
        PLSDNAME(DCLEMP)
        LANGUAGE(IBMVSCOBOLII)
        INFILE("emp1.cob");
```

 where the file **emp1.cob** contains the following:

```
01 DCLEMP.
  10 EMPNO                 PIC X(6).
  10 FIRSTNME.
      49 FIRSTNME-LEN      PIC S9(4) USAGE COMP.
      49 FIRSTNME-TEXT     PIC X(15).
  10 LASTNAME.
      49 LASTNAME-LEN      PIC S9(4) USAGE COMP.
      49 LASTNAME-TEXT     PIC X(20).
  10 WORKDEPT             PIC X(3).
  10 PHONENO              PIC X(4).
  10 HIREDATE             PIC X(10).
  10 JOB                  PIC X(8).
  10 EDLEVEL              PIC S9(4) USAGE COMP.
  10 SEX                  PIC X(1).
  10 BIRTHDATE            PIC X(10).
  10 SALARY               PIC S9999999V99 USAGE COMP-3.
  10 BONUS                PIC S9999999V99 USAGE COMP-3.
  10 COMM                 PIC S9999999V99 USAGE COMP-3.
```

The assumption is that version 1 of the data definition for 'EMPREC' is to be redefined. This causes a redefinition of the first 'EMPREC' sample data definition without changing the version number. Thus, existing call and transaction definitions which referenced version 1 of 'EMPREC' automatically reflect the changed 'EMPREC'. This change becomes effective when a TIP is next generated for a transaction that references the call which referenced version 1 of 'EMPREC'.

This implicitly affects both versions of the transaction because both refer to EMPREC in the second call to update the employee data.

G.7 Sample PGAU UNDEFINE Statements

These samples illustrate the deletion of a specific version of a definition which has multiple versions, followed by deletion of all versions of a specific named definition.

Deletion of DATA Definitions:

```
UNDEFINE DATA EMPREC VERSION (ddddd);
UNDEFINE DATA EMPREC VERSION (ALL);
UNDEFINE CALL DB2IMAIN VERSION (ccccc);
UNDEFINE CALL DB2IMAIN VERS (all);
UNDEFINE TRANSACTION DB2I vers (ttttt);
UNDEFINE TRANSACTION DB2I vers (all);
```

Note that the previous UNDEFINE statements leave the DATA definition for EMPNO and the CALL definition for DB2IDIAG in the PGA DD.

G.8 Sample PGAU REPORT Output

```
PGAU> report transaction db2i;

   /* Transaction DB2I version 298 created by PGAADMIN on FEB 14, 1995 17:38.02 */
      DEFINE TRANSACTION DB2I
                         CALL (DB2IMAIN VERSION(672),
                               DB2IDIAG VERSION(673))
                         SIDEPROFILE(CICSPGA)
                         TPNAME(DB2I)
                         LOGMODE(ORAPLU62)
                         SYNCLEVEL(0)
                         NLS_LANGUAGE(american_america.we8ebcdic37c);


 PGAU> report transaction db2i with calls;

   /* Call DB2IMAIN version 672 created by PGAADMIN on FEB 14, 1995 17:38.01 */
    DEFINE CALL DB2IMAIN PKGCALL(PGADB2I_MAIN)
              PARMS(EMPNO IN VERSION(638),
                    EMPREC OUT VERSION(639));

   /* Call DB2IDIAG version 673 created by PGAADMIN on FEB 14, 1995 17:38.02 */
    DEFINE CALL DB2IDIAG PKGCALL(PGADB2I_DIAG)
              PARMS(DB2INFO OUT VERSION(640));

   /* Transaction DB2I version 298 created by PGAADMIN on FEB 14, 1995 17:38.02 */
      DEFINE TRANSACTION DB2I
                         CALL (DB2IMAIN VERSION(672),
                               DB2IDIAG VERSION(673))
                         SIDEPROFILE(CICSPGA)
                         TPNAME(DB2I)
                         LOGMODE(ORAPLU62)
                   SYNCLEVEL(0)
                   NLS_LANGUAGE(american_america.we8ebcdic37c);


 PGAU> report transaction db2i with data;

   /* Data Definition EMPNO version 638 created/updated by PGAADMIN */
   /* on FEB 14, 1995 17:02.58                  */
    DEFINE DATA EMPNO LANGUAGE(IBMVSCOBOLII) USAGE(PASS)
(
         01 EMPNO PICTURE IS X(6).
);


   /* Data Definition EMPREC version 639 created/updated by PGAADMIN */
   /* on FEB 14,1995 17:02.59                 */
    DEFINE DATA EMPREC PLSDNAME(DCLEMP) LANGUAGE(IBMVSCOBOLII) USAGE(PASS)
    (
         01 DCLEMP.
          10 EMPNO PICTURE IS X(6).
          10 FIRSTNME.
       * Field FIRSTNME-LEN has PL/SQL name FIRSTNME_LEN
            49 FIRSTNME-LEN USAGE IS COMP PICTURE IS S9(4).
       * Field FIRSTNME-TEXT has PL/SQL name FIRSTNME_TEXT
            49 FIRSTNME-TEXT PICTURE IS X(12).
          10 MIDINIT PICTURE IS X(1).
          10 LASTNAME.
```

```
        * Field LASTNAME-LEN has PL/SQL name LASTNAME_LEN
             49 LASTNAME-LEN USAGE IS COMP PICTURE IS S9(4).
        * Field LASTNAME-TEXT has PL/SQL name LASTNAME_TEXT
             49 LASTNAME-TEXT PICTURE IS X(15).
          10 WORKDEPT PICTURE IS X(3).
          10 PHONENO PICTURE IS X(4).
          10 HIREDATE PICTURE IS X(10).
          10 JOB PICTURE IS X(8).
          10 EDLEVEL USAGE IS COMP PICTURE IS S9(4).
          10 SEX PICTURE IS X(1).
          10 BIRTHDATE PICTURE IS X(10).
          10 SALARY USAGE IS COMP-3 PICTURE IS S9999999V99.
          10 BONUS USAGE IS COMP-3 PICTURE IS S9999999V99.
          10 COMM USAGE IS COMP-3 PICTURE IS S9999999V99.
);

   /* Call DB2IMAIN version 672 created by PGAADMIN on FEB 14, 1995 17:38.01*/
    DEFINE CALL DB2IMAIN PKGCALL(PGADB2I_MAIN)
              PARMS(EMPNO IN VERSION(638),
                    EMPREC OUT VERSION(639));

   /*Data Definition DB2INFO version 640 created/updated by PGAADMIN */
   /* on FEB 14,1995 17:02.00*/
    DEFINE DATA DB2INFO PLSDNAME(DB2) LANGUAGE(IBMVSCOBOLII) USAGE(PASS)
(
        01 DB2.
           05 SQLCODE USAGE IS COMP-4 PICTURE IS S9(9).
           05 SQLERRM.
             49 SQLERRML USAGE IS COMP-4 PICTURE IS S9(4).
             49 SQLERRMT PICTURE IS X(70).
           05 DSNERRM.
             49 DSNERRML USAGE IS COMP-4 PICTURE IS S9(4).
             49 DSNERRMT PICTURE IS X(240) OCCURS 8 TIMES.
);

   /* Call DB2IDIAG version 673 created by PGAADMIN on FEB 14, 1995 17:38.02*/
    DEFINE CALL DB2IDIAG PKGCALL(PGADB2I_DIAG)
              PARMS(DB2INFO OUT VERSION(640));

   /* Transaction DB2I version 298 created by PGAADMIN on  FEB 14, 1995 17:38.02*/
     DEFINE TRANSACTION DB2I
                     CALL (DB2IMAIN VERSION(672),
                           DB2IDIAG VERSION(673))
                     SIDEPROFILE(CICSPGA)
                     TPNAME(DB2I)
                     LOGMODE(ORAPLU62)
                     SYNCLEVEL(0)
                     NLS_LANGUAGE(american_america.we8ebcdic37c);

PGAU> report transaction db2i with data with debug;

   /* Data Definition EMPNO version 638 created/updated by PGAADMIN */
   /* on FEB 14, 1995 17:02.58        */
    DEFINE DATA EMPNO /* d#=565 */ LANGUAGE(IBMVSCOBOLII) USAGE(PASS)
    (
        * f#=4005
        01 EMPNO PICTURE IS X(6).
    );
```

```
    /* Data Definition EMPREC version 639 created/updated by PGAADMIN */
    /* on FEB 14,1995 17:02.59         */
     DEFINE DATA EMPREC /* d#=566 */ PLSDNAME(DCLEMP) LANGUAGE(IBMVSCOBOLII) USAGE(PASS)
     (
          * f#=4006
            01 DCLEMP.
          * f#=4007
            10 EMPNO PICTURE IS X(6).
          * f#=4008
            10 FIRSTNME.
          * f#=4009
          * Field FIRSTNME-LEN has PL/SQL name FIRSTNME_LEN
              49 FIRSTNME-LEN USAGE IS COMP PICTURE IS S9(4).
          * f#=4010
          * Field FIRSTNME-TEXT has PL/SQL name FIRSTNME_TEXT
              49 FIRSTNME-TEXT PICTURE IS X(12).
          * f#=4011
            10 MIDINIT PICTURE IS X(1).
          * f#=4012
            10 LASTNAME.
          * f#=4013
          * Field LASTNAME-LEN has PL/SQL name LASTNAME_LEN
              49 LASTNAME-LEN USAGE IS COMP PICTURE IS S9(4).
          * f#=4014
          * Field LASTNAME-TEXT has PL/SQL name LASTNAME_TEXT
              49 LASTNAME-TEXT PICTURE IS X(15).
          * f#=4015
            10 WORKDEPT PICTURE IS X(3).
          * f#=4016
            10 PHONENO PICTURE IS X(4).
          * f#=4017
            10 HIREDATE PICTURE IS X(10).
          * f#=4018
            10 JOB PICTURE IS X(8).
          * f#=4019
            10 EDLEVEL USAGE IS COMP PICTURE IS S9(4).
          * f#=4020
            10 SEX PICTURE IS X(1).
          * f#=4021
            10 BIRTHDATE PICTURE IS X(10).
          * f#=4022
            10 SALARY USAGE IS COMP-3 PICTURE IS S9999999V99.
          * f#=4023
            10 BONUS USAGE IS COMP-3 PICTURE IS S9999999V99.
          * f#=4024
            10 COMM USAGE IS COMP-3 PICTURE IS S9999999V99.
);

    /* Call DB2IMAIN version 672 created by PGAADMIN on FEB 14, 1995 17:38.01*/
     DEFINE CALL DB2IMAIN PKGCALL(PGADB2I_MAIN) /* c#=672 */
             PARMS(EMPNO IN VERSION(638) /* d#=565 */,
                   EMPREC OUT VERSION(639) /* d#=566 */);

    /*Data Definition DB2INFO version 640 created/updated by PGAADMIN */
    /* on FEB 14,1995 17:02.00*/
     DEFINE DATA DB2INFO /* d#=567 */ PLSDNAME(DB2) LANGUAGE(IBMVSCOBOLII) USAGE(PASS)
     (
          * f#=4025
           01 DB2.
          * f#=4026
```

```
            05 SQLCODE USAGE IS COMP-4 PICTURE IS S9(9).
      * f#=4027
         05 SQLERRM.
      * f#=4028
           49 SQLERRML USAGE IS COMP-4 PICTURE IS S9(4).
      * f#=4029
           49 SQLERRMT PICTURE IS X(70).
      * f#=4030
         05 DSNERRM.
      * f#=4031
           49 DSNERRML USAGE IS COMP-4 PICTURE IS S9(4).
      * f#=4032
           49 DSNERRMT PICTURE IS X(240) OCCURS 8 TIMES.
   );

 /* Call DB2IDIAG version 673 created by PGAADMIN on FEB 14, 1995 17:38.02*/
  DEFINE CALL DB2IDIAG PKGCALL(PGADB2I_DIAG) /* c#=673 */
             PARMS(DB2INFO OUT VERSION(640) /* d#=567 */);

 /* Transaction DB2I version 298 created by PGAADMIN on  FEB 14, 1995 17:38.02*/
   DEFINE TRANSACTION DB2I /* t#=298 */
                     CALL (DB2IMAIN VERSION(672) /* c#=672 */,
                           DB2IDIAG VERSION(673) /* c#=673 */)
                     SIDEPROFILE(CICSPGA) /* a#=2 */
                     TPNAME(DB2I) /* a#=3 */
                     LOGMODE(ORAPLU62) /* a#=4 */
                     SYNCLEVEL(0) /* a#=6 */
                     NLS_LANGUAGE(american_america.we8ebcdic37c) /* a#=7 */ ;

PGAU>

PGAU>  spool off
```

G.9 Sample TIP Content Documentation

```
PG4APPC TIP pgadb2i contents:


TIP generation date: 15-FEB-03 10:14:17
TIP generation user: PGAADMIN
PG DD version:
PGAU  version:        10.2.0.1.0


/*----------------------------------------------------------------*/
/*  TIP user-transaction definition                               */
/*----------------------------------------------------------------*/
   remote host transaction program name: DB2I
   remote host transaction sync level:   0
   remote host application LU name:
   remote host application mode entry:   ORAPLU62
   gateway CPI-C Side Profile name:      CICSPGA

   remote host environment: IBM370
   remote host NLS name:    AMERICAN_AMERICA.WE8EBCDIC37C


/*----------------------------------------------------------------*/
/*  TIP user-transaction-data type declarations                   */
/*----------------------------------------------------------------*/


   PL/SQL Variable name       PL/SQL Variable Type


   TYPE FIRSTNME_Typ is RECORD (
       FIRSTNME_LEN          NUMBER(4,0),
       FIRSTNME_TEXT         CHAR(12));

   TYPE LASTNAME_Typ is RECORD (
       LASTNAME_LEN          NUMBER(4,0),
       LASTNAME_TEXT         CHAR(15));

   TYPE DCLEMP_Typ is RECORD (
       EMPNO                 CHAR(6),
       FIRSTNME              FIRSTNME_Typ,
       MIDINIT               CHAR(1),
       LASTNAME              LASTNAME_Typ,
       WORKDEPT              CHAR(3),
       PHONENO               CHAR(4),
       HIREDATE              CHAR(10),
       JOB                   CHAR(8),
       EDLEVEL               NUMBER(4,0),
       SEX                   CHAR(1),
       BIRTHDATE             CHAR(10),
       SALARY                NUMBER(9,2),
       BONUS                 NUMBER(9,2),
       COMM                  NUMBER(9,2));

   TYPE SQLERRM_Typ is RECORD (
       SQLERRML              NUMBER(4,0),
       SQLERRMT              CHAR(70));

   DSNERRMT_Key BINARY_INTEGER;

   TYPE DSNERRMT_Tbl is TABLE of CHAR(240)
        INDEX by BINARY_INTEGER;
```

```
   TYPE DSNERRM_Typ is RECORD (
         DSNERRML                NUMBER(4,0),
         DSNERRMT                DSNERRMT_Tbl);

   TYPE DB2_Typ is RECORD (
         SQLCODE                 NUMBER(9,0),
         SQLERRM                 SQLERRM_Typ,
         DSNERRM                 DSNERRM_Typ);

/*------------------------------------------------------------------*/
/*  TIP default functions                                           */
/*------------------------------------------------------------------*/

   TYPE override_Typ IS RECORD (
             tranname        VARCHAR2(2000), /* Transaction Program */
             transync        BINARY_INTEGER, /* RESERVED            */
             trannls         VARCHAR2(50),   /* RESERVED            */
             oltpname        VARCHAR2(2000), /* Logical Unit        */
             oltpmode        VARCHAR2(2000), /* LOG Mode Entry      */
             netaddr         VARCHAR2(2000));/* Side Profile        */

   FUNCTION pgadb2i_init(             /* init standard  */
             tranuse  IN OUT BINARY_INTEGER)
             RETURN INTEGER;

   FUNCTION pgadb2i_init(             /* init override  */
             tranuse  IN OUT BINARY_INTEGER,
             override IN     override_Typ)
             RETURN INTEGER;

   FUNCTION pgadb2i_init(             /* init diagnostic */
             tranuse  IN OUT BINARY_INTEGER,
             tipdiag  IN     CHAR)
             RETURN INTEGER;

   FUNCTION pgadb2i_init(             /* init over-diag  */
             tranuse  IN OUT BINARY_INTEGER,
             override IN     override_Typ,
             tipdiag  IN     CHAR)
             RETURN INTEGER;

   FUNCTION pgadb2i_term(             /* terminate       */
             tranuse  IN     BINARY_INTEGER,
             termtype IN     BINARY_INTEGER)
             RETURN INTEGER;

/*------------------------------------------------------------------*/
/*  TIP user-transaction-call function definitions                  */
/*------------------------------------------------------------------*/
   FUNCTION PGADB2I_MAIN(
             tranuse     IN     BINARY_INTEGER,
             EMPNO           IN     CHAR,
             DCLEMP          OUT    DCLEMP_Typ)
             RETURN INTEGER;

   FUNCTION PGADB2I_DIAG(
             tranuse     IN     BINARY_INTEGER,
             DB2             OUT    DB2_Typ)
             RETURN INTEGER;
```

```
/*-------------------------------------------------------------------*/
/*  TIP field variables                                              */
/*-------------------------------------------------------------------*/

   PL/SQL name                                    PL/SQL type

   EMPNO                                          CHAR(6)

   DCLEMP                                         DCLEMP_Typ
   DCLEMP.EMPNO                                   CHAR(6)
   DCLEMP.FIRSTNME                                FIRSTNME_Typ
   DCLEMP.FIRSTNME.FIRSTNME_LEN                   NUMBER(4,0)
   DCLEMP.FIRSTNME.FIRSTNME_TEXT                  CHAR(12)
   DCLEMP.MIDINIT                                 CHAR(1)
   DCLEMP.LASTNAME                                LASTNAME_Typ
   DCLEMP.LASTNAME.LASTNAME_LEN                   NUMBER(4,0)
   DCLEMP.LASTNAME.LASTNAME_TEXT                  CHAR(15)
   DCLEMP.WORKDEPT                                CHAR(3)
   DCLEMP.PHONENO                                 CHAR(4)
   DCLEMP.HIREDATE                                CHAR(10)
   DCLEMP.JOB                                     CHAR(8)
   DCLEMP.EDLEVEL                                 NUMBER(4,0)
   DCLEMP.SEX                                     CHAR(1)
   DCLEMP.BIRTHDATE                               CHAR(10)
   DCLEMP.SALARY                                  NUMBER(9,2)
   DCLEMP.BONUS                                   NUMBER(9,2)
   DCLEMP.COMM                                    NUMBER(9,2)

   DB2                                            DB2_Typ
   DB2.SQLCODE                                    NUMBER(9,0)
   DB2.SQLERRM                                    SQLERRM_Typ
   DB2.SQLERRM.SQLERRML                           NUMBER(4,0)
   DB2.SQLERRM.SQLERRMT                           CHAR(70)
   DB2.DSNERRM                                    DSNERRM_Typ
   DB2.DSNERRM.DSNERRML                           NUMBER(4,0)
   DB2.DSNERRM.DSNERRMT(DSNERRMT_Key)             CHAR(240)
```

G.10 Sample TIP Trace Output

```
SQL> set serveroutput on size 20000
SQL> exec db2idriv('000340','11110000');
Name
JASON R GOUNOT
Sex Birthdate  Edlevel
M   05/17/1926    16
Empno  Dept   Job     Phone Hiredate    Salary      Bonus   Commission
000340 E21  FIELDREP  5698 05/05/1947   23840.00     500.00    1907.00


PL/SQL procedure successfully completed.

SQL> exec rtrace('PGADB2I');
UTL_PG warnings enabled
function entry/exit trace enabled
data conversion trace enabled
gateway exchange trace enabled
enter PGADB2I_init diagnostic
time 22-MAR-1995 13:54:47
from PGAADMIN
exit PGADB2I_init diagnostic
enter PGADB2I_MAIN
enter updsndlt
sendcnt, parmlenm => 1, 6
sendllst(sendcnt) => 0
sendllst(sendcnt) => 6
exit updsndlt
EMPNO CHAR(6)
at pos, off, len => 1, 1, 6
enter updrcvlt
recvcnt, parmlenm => 1, 91
recvllst(recvcnt) => 0
recvllst(recvcnt) => 91
exit updrcvlt
enter dataxfer
tranuse => 2
enter bldsxfrl
sendmax => 1
sendbufl => 6
sxfrllst => 0000000100000006
exit bldsxfrl
recvmax => 1
enter bldrxfrl
recvbufl => 91
rxfrllst => 000000010000005B
exit bldrxfrl
enter xfersr
tranuse  => 2
trancid  => 200E8E08200F2E1800000000
sendbufl => 6
sxfrllst => 0000000100000006000000000
recvbufl => 91
rxfrllst => 000000010000005B00000000
recvbufl => 91
rxfrllst => 000000010000005B00000000
exit xfersr
exit dataxfer
enter getrcvla
recvcnt => 1
```

```
recvalst(recvcnt) => 0
recvalst(recvcnt) => 91
numplen => 91
exit getrcvla
DCLEMP.EMPNO CHAR(6)
at pos, off, len => 1, 1, 6
DCLEMP.FIRSTNME.FIRSTNME_LEN NUMBER(4,0)
at pos, off, len => 7, 7, 2
DCLEMP.FIRSTNME.FIRSTNME_TEXT CHAR(12)
at pos, off, len => 9, 9, 12
DCLEMP.MIDINIT CHAR(1)
at pos, off, len => 21, 21, 1
DCLEMP.LASTNAME.LASTNAME_LEN NUMBER(4,0)
at pos, off, len => 22, 22, 2
DCLEMP.LASTNAME.LASTNAME_TEXT CHAR(15)
at pos, off, len => 24, 24, 15
DCLEMP.WORKDEPT CHAR(3)
at pos, off, len => 39, 39, 3
DCLEMP.PHONENO CHAR(4)
at pos, off, len => 42, 42, 4
DCLEMP.HIREDATE CHAR(10)
at pos, off, len => 46, 46, 10
DCLEMP.JOB CHAR(8)
at pos, off, len => 56, 56, 8
DCLEMP.EDLEVEL NUMBER(4,0)
at pos, off, len => 64, 64, 2
DCLEMP.SEX CHAR(1)
at pos, off, len => 66, 66, 1
DCLEMP.BIRTHDATE CHAR(10)
at pos, off, len => 67, 67, 10
DCLEMP.SALARY NUMBER(9,2)
at pos, off, len => 77, 77, 5
DCLEMP.BONUS NUMBER(9,2)
at pos, off, len => 82, 82, 5
DCLEMP.COMM NUMBER(9,2)
at pos, off, len => 87, 87, 5
exit PGADB2I_MAIN
enter PGADB2I_term
exit PGADB2I_term
time 22-MAR-1995 13:54:49
from PGAADMIN
no more PGADB2I trace messages

PL/SQL procedure successfully completed.

SQL> spool off
```

G.11 Sample TIP Output

```
SQL> exec db2idriv('000340');
Name
JASON R GOUNOT
Sex Birthdate  Edlevel
M   05/17/1926    16
Empno  Dept   Job     Phone  Hiredate    Salary      Bonus    Commission
000340 E21  FIELDREP  5698 05/05/1947   23840.00     500.00    1907.00


PL/SQL procedure successfully completed.


SQL> spool off
```

Index