

## **Oracle® Multimedia**

User's Guide

11g Release 1 (11.1)

**B28415-01**

July 2007

Oracle Multimedia is a feature that enables Oracle Database to store, manage, and retrieve images, audio, video, DICOM format medical images and other objects, or other heterogeneous media data in an integrated fashion with other enterprise information. Oracle Multimedia extends Oracle Database reliability, availability, and data management to multimedia content in traditional, Internet, electronic commerce, medical, financial, and other media-rich applications.

Oracle Multimedia User's Guide, 11g Release 1 (11.1)

B28415-01

Copyright © 1999, 2007, Oracle. All rights reserved.

Primary Author: Sue Pelski

Contributors: Robert Abbott, Fengting Chen, Bill Gettys, Dongbai Guo, Dong Lin, Susan Mavris, Valarie Moore, Prajna Parida, James Steiner, Yingmei Sun, Simon Watt, Manjari Yalavarthy, Jie Zhang

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software—Restricted Rights (June 1987). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

---

---

# Contents

<b>Preface</b> .....	xi
Audience .....	xi
Documentation Accessibility .....	xi
Related Documents .....	xii
Conventions .....	xiii
<b>What's New</b> .....	xv
Changes Since Release 10.2 .....	xv
New Features for Release 11.1 .....	xvi
<b>1 Introduction to Oracle Multimedia</b>	
1.1 Object Relational Technology .....	1-1
1.2 Oracle Multimedia Capabilities .....	1-2
1.3 Audio Concepts .....	1-4
1.3.1 Digitized Audio .....	1-4
1.3.2 Audio Components .....	1-4
1.4 ORDDoc or Heterogeneous Media Data Concepts .....	1-4
1.4.1 Digitized Heterogeneous Media Data .....	1-4
1.4.2 Heterogeneous Media Data Components .....	1-5
1.5 Image Concepts .....	1-5
1.5.1 Digitized Images .....	1-5
1.5.2 Image Components .....	1-5
1.5.3 Metadata in Images .....	1-6
1.5.4 Medical Imaging .....	1-6
1.5.5 Metadata Extraction .....	1-7
1.5.6 Image Processing .....	1-7
1.5.7 SQL/MM Still Image Standard Support .....	1-7
1.6 Video Concepts .....	1-8
1.6.1 Digitized Video .....	1-8
1.6.2 Video Components .....	1-8
1.7 Multimedia Storage and Querying .....	1-9
1.7.1 Storing Multimedia Data .....	1-9
1.7.2 Querying Multimedia Data .....	1-10
1.8 Loading Multimedia Data .....	1-10
1.9 Accessing Multimedia Data .....	1-10

1.9.1	Oracle Multimedia Java Classes .....	1-11
1.9.2	Streaming Content from Oracle Database .....	1-11
1.9.3	Support for Web Technologies .....	1-12
1.9.4	Oracle Multimedia Custom DataSource and DataSink Classes for JMF 2.0/2.1.....	1-13
1.9.5	Oracle Multimedia Support for Java Advanced Imaging (JAI) .....	1-13
1.10	Oracle Multimedia Architecture.....	1-14
1.11	Extending Oracle Multimedia.....	1-15

## 2 Application Development

2.1	Developing PL/SQL Client Applications Using the PL/SQL API.....	2-4
2.1.1	Setting Up Your Environment for PL/SQL .....	2-4
2.1.2	Media Query in PL/SQL .....	2-5
2.1.3	Media Download in PL/SQL.....	2-5
2.1.4	Media Upload in PL/SQL .....	2-6
2.1.5	Handling Oracle Multimedia Exceptions in PL/SQL .....	2-7
2.1.5.1	Handling the Setting of Properties for Unknown Image Formats .....	2-7
2.1.5.2	Handling Image Processing for Unknown Image Formats.....	2-8
2.2	Developing PL/SQL Web Applications .....	2-8
2.2.1	Using the PL/SQL Gateway and PL/SQL Web Toolkit.....	2-9
2.3	Developing Java Client Applications Using JDBC.....	2-12
2.3.1	Setting Up Your Environment for Java.....	2-12
2.3.2	Media Retrieval in Java.....	2-14
2.3.3	Media Upload in Java.....	2-15
2.3.4	Handling Oracle Multimedia Exceptions in Java .....	2-17
2.3.4.1	Handling the Setting of Properties for Unknown Image Formats .....	2-18
2.3.4.2	Handling Image Processing for Unknown Image Formats.....	2-18
2.4	Developing Java-Based Web Applications.....	2-18
2.4.1	Media Retrieval in Java-Based Web Applications .....	2-19
2.4.1.1	Media URL.....	2-20
2.4.1.2	Media Delivery Component .....	2-20
2.4.2	Media Upload in Java-Based Web Applications.....	2-21

## 3 Oracle Multimedia Photo Album Sample Applications

3.1	Oracle Multimedia PL/SQL Photo Album Sample Application .....	3-2
3.1.1	Running the PL/SQL Photo Album Application .....	3-4
3.1.2	Description of the PL/SQL Photo Album Application.....	3-5
3.1.2.1	Browsing the Photo Album.....	3-7
3.1.2.2	Adding Images to the Photo Album.....	3-10
3.1.2.3	Searching for Images by Keyword or Phrase .....	3-15
3.1.2.4	Viewing Full-Size Images.....	3-16
3.1.2.5	Examining Image Metadata .....	3-18
3.1.2.6	Writing New XMP Metadata to Images .....	3-19
3.1.2.7	Searching for Images That Contain Specific Metadata Attributes .....	3-22
3.2	Oracle Multimedia Java Servlet Photo Album Sample Application .....	3-25
3.2.1	Running the Java Servlet Photo Album Application.....	3-26
3.2.2	Description of the Java Servlet Photo Album Application.....	3-26
3.3	Oracle Multimedia JSP Photo Album Sample Application .....	3-34

3.3.1	Running the JSP Photo Album Application.....	3-35
3.3.2	Description of the JSP Photo Album Application.....	3-35
3.4	Oracle Multimedia ASP/VBScript Photo Album Sample Application .....	3-42
3.4.1	Running the ASP/VBScript Photo Album Application.....	3-43
3.4.2	Description of the ASP/VBScript Photo Album Application.....	3-44

## 4 Oracle Multimedia Code Wizard Sample Application

4.1	Using the Code Wizard Sample Application.....	4-1
4.1.1	Creating a New DAD or Choosing an Existing DAD .....	4-2
4.1.2	Authorizing a DAD .....	4-3
4.1.3	Creating and Testing Media Upload and Retrieval Procedures.....	4-5
4.1.4	Creating a Media Upload Procedure .....	4-6
4.1.5	Creating a Media Retrieval Procedure .....	4-11
4.1.6	Using the PL/SQL Gateway Document Table .....	4-15
4.1.7	How Time Zone Information Is Used to Support Browser Caching .....	4-15
4.2	Sample Session Using Images .....	4-16
4.3	Sample Session Using Multiple Object Columns .....	4-25
4.4	Known Restrictions of the Oracle Multimedia Code Wizard.....	4-35

## 5 Oracle Multimedia IMExample Sample Application

5.1	Overview of the IMExample Application .....	5-1
5.2	Description of the IMExample Application .....	5-2
5.2.1	IMProductDialog Class.....	5-4
5.2.2	IMImagePanel Class.....	5-6
5.2.3	IMGetMetadataDialog Class.....	5-10
5.2.4	IMPutMetadataDialog Class .....	5-11
5.2.5	IMVideoPanel Class .....	5-12
5.2.6	IMAudioPanel Class.....	5-14
5.2.7	IMDocPanel Class.....	5-17
5.2.8	IMLoadFile Class .....	5-19
5.2.9	IMUtil Class .....	5-23

## 6 Working with Metadata in Images

6.1	Metadata Concepts .....	6-1
6.2	Oracle Multimedia Image Metadata Concepts.....	6-1
6.3	Image File Formats .....	6-2
6.4	Image Metadata Formats .....	6-2
6.4.1	EXIF.....	6-2
6.4.2	IPTC-IIM.....	6-2
6.4.3	XMP .....	6-2
6.5	Representing Metadata Outside Images .....	6-3
6.6	Oracle Multimedia Image Metadata Examples .....	6-3
6.6.1	Creating a Table for Metadata Storage .....	6-3
6.6.2	Extracting Image Metadata .....	6-4
6.6.3	Embedding Image Metadata.....	6-5
6.7	Metadata References.....	6-7

6.8	Extracting and Mapping DICOM Metadata Attributes in Images .....	6-7
6.9	DICOM Image File Format .....	6-7
6.10	Oracle Multimedia DICOM Image Metadata Example.....	6-8
6.10.1	Creating a Table for DICOM Metadata Storage.....	6-8
6.10.2	Extracting DICOM Image Metadata .....	6-8
6.11	DICOM References .....	6-9

## 7 Extending Oracle Multimedia

7.1	Supporting Other External Sources.....	7-1
7.1.1	Packages or PL/SQL Plug-ins.....	7-1
7.1.1.1	ORDPLUGINS.ORDX_FILE_SOURCE Package .....	7-2
7.1.1.2	ORDPLUGINS.ORDX_HTTP_SOURCE Package .....	7-3
7.1.1.3	Extending Oracle Multimedia to Support a New Data Source.....	7-5
7.2	Supporting Other Media Data Formats.....	7-8
7.2.1	Supporting Other ORDAudio Data Formats.....	7-8
7.2.1.1	ORDPLUGINS.ORDX_DEFAULT_AUDIO Package.....	7-8
7.2.1.2	Extending Oracle Multimedia to Support a New Audio Data Format .....	7-9
7.2.2	Supporting Other ORDDoc Data Formats .....	7-11
7.2.2.1	ORDPLUGINS.ORDX_DEFAULT_DOC Package .....	7-11
7.2.2.2	Extending Oracle Multimedia to Support a New ORDDoc Data Format .....	7-11
7.2.3	Supporting Other Video Data Formats .....	7-12
7.2.3.1	ORDPLUGINS.ORDX_DEFAULT_VIDEO Package .....	7-12
7.2.3.2	Extending Oracle Multimedia to Support a New Video Data Format .....	7-13
7.2.4	Supporting Other Image Data Formats.....	7-15
7.3	Extending Oracle Multimedia with a New Type .....	7-15
7.4	Supporting Media Data Processing.....	7-16
7.4.1	Supporting Audio Data Processing .....	7-16
7.4.2	Supporting Video Data Processing .....	7-16

## 8 Tuning Tips for DBAs

8.1	Understanding the Performance Profile of Oracle Multimedia Operations .....	8-1
8.2	Choosing LOB Storage Parameters for Oracle Multimedia Objects.....	8-3
8.2.1	SecureFile LOBs and BasicFile LOBs .....	8-3
8.2.2	TABLESPACE .....	8-3
8.2.3	CACHE, NOCACHE, and CACHE READS .....	8-3
8.2.4	LOGGING and NOLOGGING .....	8-3
8.2.5	CHUNK.....	8-4
8.2.6	Example of Setting LOB Storage Options .....	8-4
8.3	Setting Database Initialization Parameters .....	8-5

## 9 Oracle Multimedia Examples

9.1	Audio Data Examples.....	9-1
9.1.1	Using Audio Types with Object Views .....	9-1
9.1.2	Scripts for Populating an ORDAudio Object with BLOB Data.....	9-3
9.1.2.1	Create an Audio Data Load Directory.....	9-3
9.1.2.2	Create and Populate the soundtable Table .....	9-4

9.1.2.3	Create and Populate the audio_table Table.....	9-6
9.1.2.4	Load the Audio Data.....	9-7
9.1.2.5	Copy the BLOB Data to the ORDAudio Object .....	9-8
9.1.2.6	Show the Properties of the Loaded Audio Data .....	9-9
9.1.2.7	Automate the ORDAudio Examples .....	9-10
9.1.2.8	Clean Up the ORDAudio Examples .....	9-11
9.2	Media Data Examples.....	9-12
9.2.1	Scripts for Populating an ORDDoc Object from a File Data Source .....	9-12
9.2.1.1	Create a Media Data Load Directory .....	9-12
9.2.1.2	Create and Populate the doc_table Table.....	9-13
9.2.1.3	Load the Media Data.....	9-14
9.2.1.4	Read the Media Data from the BLOB .....	9-15
9.2.1.5	Show the Properties of the Loaded Media Data .....	9-16
9.2.1.6	Automate the ORDDoc Examples.....	9-17
9.2.1.7	Clean Up the ORDDoc Examples.....	9-18
9.3	Image Data Examples.....	9-18
9.3.1	Scripts for Populating an ORDImage Object from a File Data Source.....	9-18
9.3.1.1	Create an Image Data Load Directory.....	9-19
9.3.1.2	Create and Populate the image_table Table .....	9-20
9.3.1.3	Load the Image Data .....	9-21
9.3.1.4	Read the Image Data from the BLOB .....	9-21
9.3.1.5	Show the Properties of the Loaded Image Data .....	9-23
9.3.1.6	Automate the ORDImage Examples.....	9-24
9.3.1.7	Clean Up the ORDImage Examples.....	9-24
9.3.2	Script for Loading an Image Table from an HTTP Data Source.....	9-25
9.3.3	Addressing Globalization Support Issues.....	9-26
9.4	Video Data Examples .....	9-26

## **A Oracle Multimedia Sample Applications**

A.1	Oracle Multimedia Image OCI C Sample Application.....	A-1
A.2	Oracle Multimedia Java Sample Applications.....	A-3
A.3	Oracle Multimedia PL/SQL Sample Applications .....	A-3
A.4	Oracle Multimedia ASP/VBScript Sample Application .....	A-4
A.5	Other Oracle Multimedia Sample Applications.....	A-4

## **B Managing Oracle Multimedia Installations**

B.1	Installing Oracle Multimedia .....	B-1
B.1.1	Installation Decisions .....	B-2
B.1.2	Preinstallation Tasks .....	B-2
B.1.3	Installation Steps.....	B-3
B.2	Upgrading an Installed Version of Oracle Multimedia.....	B-3
B.3	Verifying an Installed Version of Oracle Multimedia .....	B-3
B.4	Downgrading an Installed Version of Oracle Multimedia .....	B-4
B.5	Removing Oracle Multimedia.....	B-4

## **Index**

## List of Examples

2-1	Image Query (Height, Width, and MimeType Attributes).....	2-5
2-2	Audio Query (MimeType Attribute).....	2-5
2-3	Video Query (MimeType Attribute) .....	2-5
2-4	URL Format Required for Invoking mod_plsql in a Web Browser.....	2-11
2-5	URL Format Required to Invoke mod_plsql in a Web Browser for the Photo Album Application.....	2-11
3-1	Procedure view_album .....	3-8
3-2	Procedure print_album .....	3-9
3-3	Procedure print_image_link.....	3-9
3-4	Procedure deliver_media.....	3-10
3-5	Procedure print_upload_form .....	3-11
3-6	Procedure insert_new_photo .....	3-12
3-7	Procedure view_entry .....	3-17
3-8	Procedure view_metadata .....	3-19
3-9	Procedure print_metadata .....	3-19
3-10	Procedure write_metadata .....	3-21
3-11	Procedure search_metadata .....	3-24
7-1	Show the Package Body for Extending Support to a New Data Source.....	7-5
7-2	Show the Package Body for Extending Support to a New Audio Data Format.....	7-10
7-3	Show the Package Body for Extending Support to a New ORDDoc Data Format.....	7-12
7-4	Show the Package Body for Extending Support to a New Video Data Format.....	7-14
7-5	Extend Oracle Multimedia ORDImage with a New Object Type.....	7-15
9-1	Define a Relational Table Containing No ORDAudio Object.....	9-2
9-2	Define an Object View Containing an ORDAudio Object and Relational Columns .....	9-2
9-3	create_mediadir.sql.....	9-4
9-4	create_soundtable.sql .....	9-5
9-5	create_audtable.sql .....	9-6
9-6	import_aud.sql .....	9-7
9-7	copy_audblob.sql .....	9-8
9-8	showprop_aud.sql .....	9-9
9-9	setup_audsample.sql .....	9-10
9-10	cleanup_audsample.sql.....	9-11
9-11	create_mediadir.sql.....	9-13
9-12	create_doctable.sql.....	9-14
9-13	import_doc.sql.....	9-14
9-14	read_doc.sql .....	9-15
9-15	showprop_doc.sql.....	9-16
9-16	setup_docsample.sql.....	9-17
9-17	cleanup_docsample.sql .....	9-18
9-18	create_mediadir.sql.....	9-19
9-19	create_imgtable.sql .....	9-20
9-20	import_img.sql .....	9-21
9-21	read_image.sql.....	9-22
9-22	showprop_img.sql .....	9-23
9-23	setup_imgsample.sql .....	9-24
9-24	cleanup_imgsample.sql.....	9-24
9-25	import_imghttp.sql Script.....	9-25
9-26	Address a Globalization Support Issue .....	9-26



## List of Figures

1-1	Oracle Multimedia Architecture.....	1-15
2-1	Components of the PL/SQL Development Environment .....	2-10
3-1	View album Page with Five Uploaded Images .....	3-7
3-2	Completed Upload photo Page .....	3-11
3-3	Search album Page Showing Results .....	3-16
3-4	View entry Page with a Full-Size Image.....	3-17
3-5	View metadata Page with Metadata for an Uploaded Image.....	3-18
3-6	Completed Write XMP metadata Page with XMP Metadata for an Uploaded Image..	3-20
3-7	Completed Search metadata Page for an Uploaded Image.....	3-23
4-1	Main Menu for the interMedia Code Wizard for the PL/SQL Gateway .....	4-4
4-2	Authorizing the SCOTTCW DAD .....	4-4
4-3	List of Authorized DADs.....	4-5
4-4	Using the SCOTTCW DAD .....	4-6
4-5	Create a Media Upload Procedure .....	4-7
4-6	Step 1: Select Database Table and Procedure Type.....	4-7
4-7	Step 2: Select PL/SQL Gateway Document Upload Table (Part 1) .....	4-8
4-8	Step 2: Select PL/SQL Gateway Document Upload Table (Part 2) .....	4-8
4-9	Step 3: Select Data Access and Media Column(s) .....	4-9
4-10	Step 4: Select Additional Columns and Procedure Name .....	4-9
4-11	Step 5: Review Selected Options.....	4-10
4-12	Compile Procedure and Review Generated Source.....	4-10
4-13	Template Upload Form .....	4-11
4-14	Template Upload Procedure -- Media Uploaded Successfully Message.....	4-11
4-15	Create a Media Retrieval Procedure .....	4-12
4-16	Step 1: Select Database Table and Procedure Type.....	4-12
4-17	Step 2: Select Media Column and Key Column.....	4-13
4-18	Step 3: Select Procedure Name and Parameter Name.....	4-13
4-19	Step 4: Review Selected Options.....	4-14
4-20	Compile Procedure and Review Generated Source.....	4-14
4-21	Displayed Image 1981 .....	4-15

## List of Tables

3-1	PL/SQL Photo Album Sample Application Overview .....	3-5
7-1	Methods Supported in the ORDPLUGINS.ORDX_FILE_SOURCE Package .....	7-3
7-2	Methods Supported in the ORDPLUGINS.ORDX_HTTP_SOURCE Package .....	7-4
7-3	Methods Supported in the ORDPLUGINS.ORDX_DEFAULT_AUDIO Package .....	7-9
7-4	Method Supported in the ORDPLUGINS.ORDX_DEFAULT_DOC Package .....	7-11
7-5	Methods Supported in the ORDPLUGINS.ORDX_DEFAULT_VIDEO Package .....	7-13
8-1	Performance Profile For All Multimedia Types .....	8-2
8-2	Performance Profile For ORDImage Methods.....	8-2
8-3	Performance Profile For ORDDicom Methods.....	8-2
8-4	Performance Profile For ORDAudio and ORDVideo Methods.....	8-3
9-1	Audio Scripts .....	9-3
9-2	Media Scripts .....	9-12
9-3	Image Scripts .....	9-19

---

---

# Preface

This guide describes how to use Oracle Multimedia, which ships with Oracle Database.

For information about Oracle Database and the features and options that are available to you, see *Oracle Database New Features Guide*.

## Audience

This guide is for application developers and database administrators who are interested in storing, retrieving, and manipulating audio, image, video, and heterogeneous media data in a database, including developers of audio, heterogeneous media data, image, and video specialization options.

The sample code in this guide will not necessarily match the code shipped with the Oracle installation. If you want to run examples that are shipped with the Oracle installation on your system, use the files provided with the installation. Do not attempt to compile and run the code in this guide.

## 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

---

---

**Note:** For information added after the release of this guide, refer to the online `README.txt` file under your `<ORACLE_HOME>` directory. Depending on your operating system, this file may be in:

`<ORACLE_HOME>/ord/im/admin/README.txt`

See your operating system-specific installation guide for more information.

---

---

For more information about using Oracle Multimedia in a development environment, see the following documents in the Oracle Database software documentation set:

- *Oracle Multimedia Reference*
- *Oracle Multimedia DICOM Developer's Guide*
- *Oracle Call Interface Programmer's Guide*
- *Oracle Database Advanced Application Developer's Guide*
- *Oracle Database SecureFiles and Large Objects Developer's Guide*
- *Oracle Database Concepts*
- *Oracle Database PL/SQL Language Reference*
- *Oracle Database Java Developer's Guide*

For more information about using JDBC, see *Oracle Database JDBC Developer's Guide and Reference*.

For more information about using XML, see *Oracle XML DB Developer's Guide*.

For reference information on Oracle Multimedia Java classes in Javadoc format, see the following Oracle API documentation (also known as Javadoc) in the Oracle Database Online Documentation Library:

- *Oracle Multimedia Java API Reference*
- *Oracle Multimedia Servlets and JSP Java API Reference*
- *Oracle Multimedia DICOM Java API Reference*

For information about using Multimedia Tag Library, see *Oracle Application Server 10g Multimedia Tag Library for JSP User's Guide and Reference* in the Oracle Application Server Online Documentation Library.

For more information about Java, including information about Java Advanced Imaging (JAI), see the API documentation provided by Sun Microsystems.

Many of the examples in this book use the sample schemas, which are installed by default when you install Oracle. See *Oracle Database Sample Schemas* for information about how these schemas were created and how you can use them yourself.

## Conventions

In this guide, Oracle *interMedia* (now known as Oracle Multimedia) was sometimes referred to as *interMedia*.

In examples, an implied carriage return occurs at the end of each line, unless otherwise noted. You must press the Return key at the end of a line of input.

Also in examples, vertical ellipsis points indicate that information not directly related to the example has been omitted.

In statements or commands, horizontal ellipsis points indicate that parts of the statement or command not directly related to the example have been omitted.

Also in statements or commands, angle brackets enclose user-supplied names and brackets enclose optional clauses from which you can choose one or none.

Although Boolean is a proper noun, it is presented as boolean in this guide when its use in Java code requires case-sensitivity.

The following text conventions are also used in this guide:

<b>Convention</b>	<b>Meaning</b>
<b>boldface</b>	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.



---

---

# What's New

This document summarizes the changes to this guide since the previous release as well as the new features introduced in the current release.

## Changes Since Release 10.2

The following changes have been made to this guide since Oracle Database 10g Release 2 (10.2):

- In Oracle Database 11g Release 1 (11.1), the name Oracle *interMedia* has been changed to Oracle Multimedia. The feature remains the same, only the name has changed. References to Oracle *interMedia* will be replaced with Oracle Multimedia, however some references to Oracle *interMedia* or *interMedia* may still appear in graphical user interfaces, code examples, and related documents in the Documentation Library for Oracle Database 11g Release 1 (11.1).
- Title page - Added DICOM to the list of supported media in the summary description.
- Chapter 1 - Added a new subsection about medical imaging under Section 1.5 "Image Concepts" that mentions the major DICOM features. Added cross-references to the new DICOM documentation. Replaced references to 4GB BLOB limits with new limits (8 TB to 128 TB, depending on the block size). Included a cross-reference to *Oracle Database SecureFiles and Large Objects Developer's Guide*. Removed Section 1.5.7 (content-based retrieval). Removed information about image matching from Section 1.5.8. (This feature is deprecated in the current release.)
- Chapter 2 - Updated information about Oracle Java libraries.
- Chapter 3 - Removed the documentation for the Code Wizard sample application from this chapter. Updated examples to use SecureFile LOBs.
- Chapter 4 - Created a new chapter that includes the documentation for the Code Wizard sample application, which had been in Chapter 3. Updated examples to use SecureFile LOBs. Updated examples to follow the new security requirement for case-sensitive passwords.
- Chapter 6 - Slightly reorganized this chapter to downplay information about DICOM in this manual. Added cross-references to the new DICOM documentation. Updated examples to use SecureFile LOBs.
- Chapter 7 - Removed the content-based retrieval concepts chapter from the book (This feature is deprecated in the current release).

- Chapter 8 - Replaced references to 4GB BLOB limits with new limits (8 TB to 128 TB, depending on the block size). Included a cross-reference to *Oracle Database SecureFiles and Large Objects Developer's Guide*. Revised chapter to include information about improvements in performance and scalability using SecureFile LOBs.
- Chapter 9 - Updated descriptions for table creation examples to remove references to data files that will no longer ship with the kits. Updated examples to use SecureFile LOBs.
- Appendix B - Included minor updates to install procedure. Added new subsection about removing Oracle Multimedia.
- Glossary - Added a new glossary of Oracle Multimedia-specific terminology, highlighting it in a central location within the Oracle Multimedia documentation set.
- Index - Added new index entries and improved existing entries.
- Ongoing minor improvements and bug fixes

## New Features for Release 11.1

Oracle Database 11g Release 1 (11.1) includes several new features for Oracle Multimedia DICOM and improvements in the area of performance and scalability.

### **Oracle Multimedia DICOM**

Oracle Multimedia DICOM was introduced in release 10.2. Release 11.1 adds the following Oracle Multimedia DICOM features:

- Creation of DICOM objects
- DICOM image processing
- DICOM object conformance validation
- DICOM metadata extraction
- DICOM format support
- Making DICOM objects anonymous
- ORDDicom object type
- Run-time, updatable DICOM data model

See *Oracle Multimedia DICOM Developer's Guide* and *Oracle Multimedia DICOM Java API Reference* for information about these features.

### **Oracle Multimedia Performance and Scalability**

Release 11.1 also includes performance and scalability improvements. This release provides improvements for applications that require increased throughput in certain image processing operations as well as the ability to manage very large media objects.

In this release, the size limit of media data that can be stored and retrieved within database storage structures (BLOB) in an Oracle Multimedia object type in the database is extended to the BLOB size limit, which is between 8 terabytes (TB) and 128 terabytes depending on the block size.

In addition to storing and retrieving large images, Oracle Multimedia can also extract image attributes including height, width, and compressionFormat for images that contain up to two billion pixels, or with a resolution of up to 46000x46000. For images



that support metadata extraction (IPTC, EXIF and XMP), Oracle Multimedia can extract and manage embedded metadata for any size image that it can store.

Oracle Multimedia provides image processing functions that change image content. For example, you can scale and crop an image or convert it to a different file format. Processing an image requires interpreting the pixel values of the image, an operation that often impacts system performance and memory. As a result, Oracle Multimedia may be unable to successfully process images that it can successfully store. The maximum image size that Oracle Multimedia can process depends on the image format and the system platform. See the Oracle Multimedia `README.txt` file for guidelines on image processing limits for each supported format.

Performance and scalability improvements have been made for the most popular image processing operation, generating a thumbnail image. Significant performance improvements have been made for generating these images from TIFF and JPEG sources, as well as from DICOM sources with JPEG encoding. In addition, improvements in scale-down operations enable the fast generation of thumbnail images from very large source images (JPEG, TIFF, and DICOM with JPEG or RAW encoding.)

In this release, Oracle Multimedia also supports the next generation of LOBs, which is called Oracle SecureFiles. SecureFiles introduces a completely reengineered large object (LOB) to dramatically improve performance and significantly strengthen the native content management capabilities of Oracle Database. Oracle Multimedia object types, methods, and packages have been tested to ensure correct operation with SecureFiles. Oracle recommends using SecureFiles to store media content within Oracle Multimedia object types to reap the performance benefits of this new BLOB implementation.



---

---

# Introduction to Oracle Multimedia

Oracle Multimedia (formerly Oracle *interMedia*) is a feature that enables Oracle Database to store, manage, and retrieve images, audio, video, or other heterogeneous media data in an integrated fashion with other enterprise information. Oracle Multimedia extends Oracle Database reliability, availability, and data management to multimedia content in traditional, Internet, electronic commerce, and media-rich applications. Oracle Multimedia does not control media capture or output devices; this function is left to application software.

Oracle Multimedia manages multimedia content by providing the following:

- Storage and retrieval (see [Section 1.7.1](#), [Section 1.8](#), and [Section 1.9](#))
- Media and application metadata management (see [Section 1.3](#), [Section 1.4](#), [Section 1.5](#), and [Section 1.6](#))
- Support for popular formats (see the audio, image, and video data format appendixes in *Oracle Multimedia Reference*)
- Access through traditional and Web interfaces (see [Section 1.9](#))
- Querying using associated relational data (see [Section 1.7](#))
- Querying using extracted metadata (see [Section 1.5.5](#))
- Querying using media content with optional specialized indexing (see [Section 1.7.2](#))

Oracle Multimedia provides media content services to Oracle JDeveloper, Oracle Content Management SDK, Oracle Application Server Portal, and Oracle partners. This guide describes the management and integration of audio, image, and video, or other heterogeneous media data with other Oracle tools and software, as well as with third-party tools and software.

## 1.1 Object Relational Technology

Oracle Database is an object relational database management system. This means that in addition to its traditional role in the safe and efficient management of relational data, it provides support for the definition of object types, including the data associated with objects and the operations (methods) that can be performed on them. Object relational technology includes integral support for BLOBs to provide the basis for adding complex objects, such as digitized audio, image, and video, to databases.

Within Oracle Multimedia, audio data characteristics have an object relational type known as **ORDAudio**, heterogeneous data characteristics have an object relational type known as **ORDDoc**, image data characteristics have an object relational type known as **ORDImage**, and video data characteristics have an object relational type

known as **ORDVideo**. All four types store data source information in an object relational type known as **ORDSource**.

See the following references for extensive information about using BLOBs and BFILEs:

- *Oracle Database SecureFiles and Large Objects Developer's Guide*
- *Oracle Database Concepts* -- see the chapter on Object Types.

See *Oracle Multimedia Reference* for more information about the multimedia object types and methods, and for more information about the **ORDSource** object type and methods.

## 1.2 Oracle Multimedia Capabilities

The capabilities of Oracle Multimedia include the storage, retrieval, management, and manipulation of multimedia data managed by Oracle Database.

Multimedia applications have common and unique requirements. Oracle Multimedia object types support common application requirements and can be extended to address application-specific requirements. With Oracle Multimedia, multimedia data can be managed as easily as standard attribute data.

Oracle Multimedia is accessible to applications through both relational and object interfaces. Database applications written in Java, C++, or traditional third-generation languages (3GLs) can interact with Oracle Multimedia through modern class library interfaces, or PL/SQL and Oracle Call Interface (OCI).

Oracle Multimedia supports storage of the popular file formats, including desktop publishing images, and streaming audio and video formats in databases. Oracle Multimedia provides the means to add audio, image, and video, or other heterogeneous media columns or objects to existing tables, and insert and retrieve multimedia data. This enables database designers to extend existing databases with multimedia data, or to build new end-user multimedia database applications. Oracle Multimedia developers can use the basic functions provided here to build specialized multimedia applications.

Oracle Multimedia uses object types, similar to Java or C++ classes, to describe multimedia data. These object types are called **ORDAudio**, **ORDDoc**, **ORDImage**, and **ORDVideo**. An instance of these object types consists of attributes, including **metadata** and the **media data**, and **methods**. Media data is the actual audio, image, or video, or other heterogeneous media data. Metadata is information about the data, such as object length, compression type, or format. Methods are procedures that can be performed on objects, such as `getContent()` and `setProperties()`.

The Oracle Multimedia objects have a common media data storage model. The media data component of these objects can be stored in the database, in a BLOB under transaction control. The media data can also be stored outside the database, without transaction control. In this case, a pointer is stored in the database under transaction control, and the media data is stored in:

- File-based large object (BFILE)
- An HTTP server-based URL
- A user-defined source on a specialized media data server, or other server

Media data stored outside the database can provide a convenient mechanism for managing large, existing or new, media repositories that reside as flat files on erasable or read-only media. This data can be imported into BLOBs at any time for transaction control. [Section 1.8](#) describes several ways of loading multimedia data into a database.

Media metadata is stored in the database under Oracle Multimedia control. Whether media data is stored within or outside the database, Oracle Multimedia manages metadata for all the media types and may automatically extract it for audio, image, and video. This metadata includes the following attributes:

- Storage information about audio, image, and video, or other heterogeneous media data, including the source type, location, and source name, and whether the data is stored locally (in the database) or externally
- Update time stamp information for audio, image, and video, or other heterogeneous media data
- Audio and video data description
- Audio, image, and video, or other heterogeneous media data format
- MIME type of the audio, image, and video, or other heterogeneous media data
- Audio characteristics: encoding type, number of channels, sampling rate, sample size, compression type, and play time (duration)
- Image characteristics: height and width, image content length, image content format, and image compression format
- Video characteristics: frame width and height, frame resolution, frame rate, play time (duration), number of frames, compression type, number of colors, and bit rate
- Extracted metadata in XML, such as the director or producer of a movie

In addition to metadata extraction methods, a minimal set of image manipulation methods is provided. For images, this includes performing format conversion, page selection, and quantize operations, and compression, scaling, cropping, copying, flipping, mirroring, rotating, and adjusting the gamma (brightness) of images.

Oracle Multimedia is extensible. It supports a base set of popular audio, image, and video data formats for multimedia processing that also can be extended, for example, to support additional formats, new digital compression and decompression schemes (**codecs**), data sources, and even specialized data processing algorithms for audio and video data. See [Chapter 7](#) for more information about extending Oracle Multimedia.

Oracle Multimedia is a building block for various multimedia applications rather than being an end-user application. It consists of object types along with related methods for managing and processing multimedia data. Some example applications for Oracle Multimedia are:

- Repositories for digital check images
- Electronic health records, including DICOM medical images
- Call centers (for example, 911 and product call centers)
- Physical asset inventories
- Distance learning and online learning
- Real estate marketing
- Stock photography archives (for example, digital art galleries and professional photographers)
- Document imaging archives
- Financial news service customer information
- Web publishing

- Audio and video Web stores

## 1.3 Audio Concepts

This section contains information about digitized audio concepts and using the ORDAudio object type to build audio applications or specialized ORDAudio objects.

### 1.3.1 Digitized Audio

ORDAudio integrates the storage, retrieval, and management of digitized audio data in a database.

Audio may be produced by an audio recorder, an audio source such as a microphone, digitized audio, other specialized audio recording devices, or even by program algorithms. Audio recording devices take an analog or continuous signal, such as the sound picked up by a microphone or sound recorded on magnetic media, and convert it into digital values with specific audio characteristics such as format, encoding type, number of channels, sampling rate, sample size, compression type, and audio duration.

### 1.3.2 Audio Components

Digitized audio consists of the audio data (digitized bits) and attributes that describe and characterize the audio data. Audio applications sometimes associate application-specific information, such as the description of the audio clip, date recorded, author or artist, and so forth, with audio data by storing descriptive text in an attribute or column in the database table.

The audio data can have different formats, encoding types, compression types, numbers of channels, sampling rates, sample sizes, and playing times (duration) depending upon how the audio data was digitally recorded. ORDAudio can store and retrieve audio data of any supported data format. ORDAudio can automatically extract metadata from audio data of a variety of popular audio formats. ORDAudio can also extract application attributes and store them in the comments field of the object in XML form. See *Oracle Multimedia Reference* for a list of supported data formats from which ORDAudio can extract and store attributes and other audio features. ORDAudio is extensible and can be made to recognize and support additional audio formats.

The size of digitized audio (number of bytes) tends to be large compared to traditional computer objects, such as numbers and text. Therefore, several encoding schemes are used that squeeze audio data into fewer bytes, thus putting a smaller load on storage devices and networks.

## 1.4 ORDDoc or Heterogeneous Media Data Concepts

This section contains information about heterogeneous media data concepts and using the ORDDoc object type to build applications or specialized ORDDoc objects.

### 1.4.1 Digitized Heterogeneous Media Data

ORDDoc integrates the storage, retrieval, and management of heterogeneous media data in a database.

The ORDDoc type can store any heterogeneous media data including audio, image, and video data in a database column. Instead of having separate columns for audio,

image, text, and video objects, you can use one column of ORDDoc objects to represent all types of multimedia.

## 1.4.2 Heterogeneous Media Data Components

Heterogeneous media data components consist of the data (digitized bits) and attributes that describe and characterize the heterogeneous media data.

Heterogeneous media data can have different formats, depending upon the application generating the media data. Oracle Multimedia can store and retrieve media data of any supported data format. The ORDDoc type can be used in applications that require you to store different types of heterogeneous media data (such as audio, image, video, and any other type of media data) in the same column so you can build a common metadata index on all the different types of media data. Using this index, you can search across all the different types of heterogeneous media data. Note that you cannot use this same search technique if the different types of heterogeneous media data are stored in different types of objects, in different columns of relational tables.

ORDDoc can automatically extract metadata from data of a variety of popular audio, image, and video data formats. ORDDoc can also extract application attributes and store them in the comments attribute of the object in XML form. See *Oracle Multimedia Reference* for a list of supported data formats from which Oracle Multimedia can extract and store attributes. ORDDoc is extensible and can be made to recognize and support other heterogeneous media data formats.

## 1.5 Image Concepts

This section contains information about digitized image concepts and using the ORDImage object type to build image applications or specialized ORDImage objects.

### 1.5.1 Digitized Images

ORDImage integrates the storage, retrieval, and management of digitized images in a database.

ORDImage supports two-dimensional, static, digitized raster images stored as binary representations of real-world objects or scenes. Images may be produced by a document or photograph scanner, a video source such as a digital camera or VCR connected to a video digitizer or frame grabber, other specialized image capture devices, or even by program algorithms. Capture devices take an analog or continuous signal such as the light that falls onto the film in a camera, and convert it into digital values on a two-dimensional grid of data points known as pixels. Devices involved in the capture and display of images are under application control.

### 1.5.2 Image Components

Digitized images consist of the image data (digitized bits) and attributes that describe and characterize the image data. Image applications sometimes associate application-specific information, such as the name of the person pictured in a photograph, description of the image, date photographed, photographer, and so on, with image data by storing this descriptive text in an attribute or column in the database table.

The image data (pixels) can have varying depths (bits per pixel) depending on how the image was captured, and can be organized in various ways. The organization of the image data is known as the data format. ORDImage can store and retrieve image data

of any data format. ORDIImage can process and automatically extract properties of images of a variety of popular data formats. See *Oracle Multimedia Reference* for a list of supported data formats for which ORDIImage can process and extract metadata. In addition, certain foreign images (formats not natively supported by ORDIImage) have limited support for image processing. See *Oracle Multimedia Reference* for more information.

The storage space required for digitized images can be large compared to traditional attribute data such as numbers and text. Many compression schemes are available to squeeze an image into fewer bytes, thus reducing storage device and network load. Lossless compression schemes squeeze an image so that when it is decompressed, the resulting image is bit-for-bit identical with the original. Lossy compression schemes do not result in an identical image when decompressed, but rather, one in which the changes may be imperceptible to the human eye. As compared with **lossless compression schemes**, **lossy compression schemes** generally provide higher compression.

The **image interchange format** describes a well-defined organization and use of image attributes, data, and often compression schemes, allowing different applications to create, exchange, and use images. Interchange formats are often stored as disk files. They may also be exchanged in a sequential fashion over a network and be referred to as **protocols**. There are many application subdomains within the digitized imaging world and many applications that create or utilize digitized images within these. ORDIImage supports storage and retrieval of all image data formats, and processing and attribute extraction of many image data formats (see *Oracle Multimedia Reference*).

### 1.5.3 Metadata in Images

Oracle Database 10g, Release 2 added an image metadata feature to Oracle Multimedia. The metadata feature enhanced the behavior of the Oracle Multimedia ORDIImage object type by adding the ability to read (or extract) and write (or embed) application metadata in images. In addition, this feature adopted a standard way to represent metadata when it is separate from an image file. Metadata can be stored in a database, indexed, searched, and made available to applications using the standard mechanisms of Oracle Database.

See [Chapter 6](#) for more information about the metadata feature.

### 1.5.4 Medical Imaging

Oracle Database 10g, Release 2 added the Digital Imaging and Communications in Medicine (DICOM) feature to Oracle Multimedia. The Oracle Multimedia DICOM feature enhanced the behavior of the Oracle Multimedia ORDIImage object type by allowing Oracle Multimedia to recognize standalone DICOM objects and extract a subset of embedded DICOM attributes relating to patient, study, and series.

See [Chapter 6](#) for more information about the Oracle Multimedia DICOM feature introduced in Oracle Database 10g, Release 2.

Oracle Database 11g, Release 1 adds the following medical imaging format support for Oracle Multimedia DICOM:

- A DICOM object creation capability
- DICOM image processing enhancements
- A DICOM object conformance validation enhancement
- DICOM metadata extraction enhancements



- The ability to make DICOM objects anonymous
- The new ORDDicom object type
- A run-time, updatable DICOM data model

In addition to the services and support provided for images, DICOM (medical images), audio, and video data, Oracle Multimedia manages DICOM content by providing the following:

- Storage and retrieval of medical imaging data in the database to synchronize the DICOM data with the associated business data
- Full object and relational interfaces to Oracle Multimedia DICOM services
- Extraction of DICOM metadata into user-specifiable XML documents
- Querying using associated relational data and extracted metadata
- Image processing, such as thumbnail generation
- Creation of new DICOM objects
- Conformance validation based on a set of user-specified conformance rules
- Making DICOM objects anonymous based on user-defined rules that specify the set of attributes to be made anonymous and how to make those attributes anonymous
- The ability to update run-time behaviors, such as the version of the DICOM standard supported, without installing a new release of Oracle Database

See *Oracle Multimedia DICOM Developer's Guide* for more information about the Oracle Multimedia DICOM feature enhancements introduced in Oracle Database 11g, Release 1.

### 1.5.5 Metadata Extraction

Oracle Multimedia provides the ability to extract content and format metadata from media sources (audio and video files), and collects and organizes this metadata as an XML formatted CLOB. Once metadata has been extracted and stored, you can index the metadata for powerful full text and thematic media searches using Oracle Text. Thus, the database can be queried to locate the media data based on the metadata extracted from the media. See the `setProperties()` method in *Oracle Multimedia Reference* for more information.

### 1.5.6 Image Processing

Oracle Multimedia supports image processing, such as image format transcoding, image cutting, image scaling, and generating thumbnail images. In addition, specifically when the destination image file format is RAW Pixel (RPIX) format or Microsoft Windows Bitmap (BMPF) image format, Oracle Multimedia supports a variety of operators for changing the format characteristics. See *Oracle Multimedia Reference* for more information.

### 1.5.7 SQL/MM Still Image Standard Support

Oracle Multimedia also provides support for the first edition of the ISO/IEC 13249-5:2001 SQL MM Part5:StillImage standard (commonly referred to as the SQL/MM Still Image standard), which includes these object relational types for image characteristics: `SI_StillImage`, `SI_AverageColor`, `SI_Color`, `SI_ColorHistogram`, `SI_FeatureList`, `SI_PositionalColor`, and `SI_Texture`.

The following ORDImage features are not specified by the SQL/MM Still Image Standard, and therefore are not available for StillImage objects:

- Storing image data outside the database
- Image processing operations (such as scaling up, compressing, and so on) that are specific to ORDImage
- Java client API

See *Oracle Multimedia Reference* for more information about the SQL/MM Still Image Standard object types.

## 1.6 Video Concepts

This section contains information about digitized video concepts and using ORDVideo to build video applications or specialized ORDVideo objects.

### 1.6.1 Digitized Video

ORDVideo integrates the storage, retrieval, and management of digitized video data in a database.

Video may be produced by a video recorder, a video camera, digitized animation video, other specialized video recording devices, or even by program algorithms. Some video recording devices take an analog or continuous signal, such as the video picked up by a video camera or video recorded on magnetic media, and convert it into digital values with specific video characteristics such as format, encoding type, frame rate, frame size (width and height), frame resolution, video length, compression type, number of colors, and bit rate.

### 1.6.2 Video Components

Digitized video consists of the video data (digitized bits) and the attributes that describe and characterize the video data. Video applications sometimes associate application-specific information, such as the description of the video training tape, date recorded, instructor's name, producer's name, and so forth, within the video data.

The video data can have different formats, compression types, frame rates, frame sizes, frame resolutions, playing times, compression types, number of colors, and bit rates depending upon how the video data was digitally recorded. ORDVideo can store and retrieve video data of any supported data format. ORDVideo can:

- Automatically extract metadata from video data of a variety of popular video formats
- Extract application attributes and store them in the comments attribute of the object in XML form

See *Oracle Multimedia Reference* for a list of supported data formats from which Oracle Multimedia can extract and store attributes and other video features

- Be made to recognize and support additional video formats (because it is extensible)

The size of digitized video (number of bytes) tends to be large compared to traditional computer objects, such as numbers and text. Therefore, several encoding schemes are used that squeeze video data into fewer bytes, thus putting a smaller load on storage devices and networks.

## 1.7 Multimedia Storage and Querying

Media can be stored in Oracle Multimedia object types, or directly in BLOBs or BFILEs. You will realize the most benefit by storing media in Oracle Multimedia object types. However, many of the features of Oracle Multimedia are available to media stored in BLOBs and BFILEs using the relational interface.

The Oracle Multimedia relational interface lets developers use static methods of Oracle Multimedia object types with existing and new media stored in BLOBs and BFILEs. Specifically, developers can move media data between the local file system and the database; parse and extract the properties of the media data; and store these properties in an XMLType or an XML formatted CLOB, and optionally, in individual relational columns. Developers are not required to make changes to their existing application schema or to instantiate Oracle Multimedia object types to take advantage of this relational interface. Oracle Multimedia static methods can also be used to perform image processing operations such as cut, scale, compress, and convert format. See *Oracle Multimedia Reference* for more information.

The ORDAudio, ORDDoc, ORDImage, and ORDVideo object types all contain an attribute of type ORDSrc and methods for multimedia data source manipulation.

---

---

**Note:** ORDSrc methods should not be called directly. Instead, invoke the wrapper method of the media object corresponding to the ORDSrc method. This information is presented for users who want to write their own user-defined sources.

---

---

### 1.7.1 Storing Multimedia Data

Oracle Multimedia can store multimedia data as an internal source within the database, under transactional control as a BLOB. It can also externally reference digitized multimedia data stored as an external source in an operating system-specific file in a local file system, as a URL on an HTTP server, or as a user-defined source on other servers, such as media servers. Although these external storage mechanisms are particularly convenient for integrating existing sets of multimedia data with a database, the multimedia data will not be under transactional control if it is not stored in the database.

BLOBs are stored in the database tablespaces in a way that optimizes space and provides efficient access. Large BLOBs may not be stored inline (BLOBs under 4 kilobytes can be stored inline) with other row data. Depending on the size of the BLOB, a locator is stored in the row and the actual BLOB (up to 8 terabytes to 128 terabytes, depending on the block size) is stored in other tablespaces. The locator can be considered a pointer to the actual location of the BLOB value. When you select a BLOB, you are selecting the locator instead of the value, although this is done transparently. An advantage of this design is that multiple BLOB locators can exist in a single row. For example, you might want to store a short video clip of a training tape, an audio recording containing a brief description of its contents, a syllabus of the course, a picture of the instructor, and a set of maps and directions to each training center all in the same row.

Because BFILEs are not under the transactional control of the database, users could change the external source without updating the database, thus causing an inconsistency with the BFILE locator. See *Oracle Database SecureFiles and Large Objects Developer's Guide* and *Oracle Call Interface Programmer's Guide* for detailed information about using BLOBs and BFILEs.

Oracle Multimedia ORDAudio, ORDDoc, ORDImage, and ORDVideo object types provide wrapper methods to do the following:

- Set the source of the data as local or external
- Modify the time an object was last updated
- Set information about the external source type, location, and name of the data
- Transfer data into or out of the database
- Obtain information about the local data content such as its length, location, or its handle to the BLOB, put the content into a temporary BLOB, or delete it
- Access source data by opening it, reading it, writing to it, trimming it, and closing it

### 1.7.2 Querying Multimedia Data

Once stored within a database, multimedia data can be queried and retrieved by using the various alphanumeric columns or object attributes of the table to find a row that contains the desired data. For example, you can select a video clip from the Training table where the course name is 'Oracle Database Concepts'.

Multimedia data can be queried by extracted metadata, by other relational table columns, and by content, such as image content-based retrieval with optional specialized indexing.

## 1.8 Loading Multimedia Data

Multimedia data can be managed best by Oracle Database. Your multimedia data should be loaded into the database to take advantage of its reliability, scalability, availability, and data management capabilities. To bulk load multimedia data into the database, you can use:

- SQL\*Loader

SQL\*Loader is an Oracle utility that lets you load data, and in this case, multimedia data (LOB data), from external multimedia files into a table of a database containing Oracle Multimedia object type columns.

- PL/SQL

A procedural extension to SQL, PL/SQL is an advanced fourth-generation programming language (4GL) of Oracle. You can write PL/SQL procedures to load multimedia data from BLOB, file system, and URL media data sources into Oracle Multimedia object type columns.

An advantage of using SQL\*Loader is that it is easy to create and test the control file that controls your data loading operation. See *Oracle Database Utilities* for more information.

An advantage of using PL/SQL scripts to load your data is that you can call methods as you load data to generate thumbnail images, or extract properties. See *PL/SQL User's Guide and Reference* for more information.

## 1.9 Accessing Multimedia Data

Applications access and manipulate multimedia data using SQL, PL/SQL, OCI, or Java through the object relational types OrdAudio, OrdDoc, OrdImage, and OrdVideo.

The following subsections describe ways in which applications, Oracle development tools, and third-party development tools can access multimedia data stored in the database using Oracle Multimedia object types.

### 1.9.1 Oracle Multimedia Java Classes

Oracle Multimedia Java classes enable Java applications on any tier (client, application server, or database) to manipulate and modify audio, image, and video data, or heterogeneous media data stored in a database. Oracle Multimedia Java classes make it possible for Java database connectivity (JDBC) result sets to include both traditional relational data and Oracle Multimedia media objects. This support enables applications to easily select and operate on a result set that contains sets of Oracle Multimedia columns plus other relational data. These classes also enable access to object attributes and invocation of object methods.

See *Oracle Multimedia Java API Reference* and *Oracle Multimedia Servlets and JSP Java API Reference* for more information about Oracle Multimedia Java classes.

### 1.9.2 Streaming Content from Oracle Database

You can stream audio and video content stored in Oracle Database using an Oracle Multimedia plug-in that supports a third-party streaming server, and deliver this content for play on a client that uses the browser-supported streaming player.

#### **Oracle Multimedia Plug-in for RealNetworks Streaming Servers**

Oracle Multimedia Plug-in for RealNetworks Streaming Server is a data source plug-in that enables RealNetworks Streaming Server to stream media data directly from Oracle Database to a media player client. The plug-in is installed with RealNetworks Streaming Server and configured and managed using the administration tool of the streaming server. The plug-in is format neutral; any format supported by the streaming server can be sourced by Oracle Database.

See *Oracle Multimedia Plug-in for RealNetworks Streaming Servers Readme* for more information about this plug-in. Oracle Multimedia Plug-in for RealNetworks Streaming Server can be downloaded from the Oracle Multimedia Software section of the Oracle Technology Network Web site at

<http://www.oracle.com/technology/products/multimedia/>

For more information about streaming servers from RealNetworks, see the RealNetworks Web site at

<http://www.realnetworks.com/>

#### **Oracle Multimedia Plug-in for Microsoft Windows Media Services**

Oracle Multimedia Plug-in for Microsoft Windows Media Services allows Microsoft Windows Media servers to stream multimedia content to a client directly from Oracle Database. This package also includes a Plug-in Property Page that can be accessed from the Windows Media Services Administrative interfaces. The Plug-in Property Page allows users to inspect, define, and edit the Plug-in mount points that map to media content in Oracle Database. The Plug-in mount points are used to configure the source URL of a server publishing point, from which a Microsoft Windows Media Player client requests media content stored in Oracle Database.

See *Oracle Multimedia Plug-in for Microsoft Windows Media Services Readme* for more information about this plug-in. Oracle Multimedia Plug-in for Microsoft Windows Media Services can be downloaded from the Oracle Multimedia Software section of the Oracle Technology Network Web site at

<http://www.oracle.com/technology/products/multimedia/>

### 1.9.3 Support for Web Technologies

Using Oracle Multimedia support for Web technologies, you can easily integrate multimedia data into Web and Java applications. You can also store, retrieve, and manage rich media content in a database.

#### **Oracle Multimedia Servlets and JSP Java API**

Oracle Multimedia Servlets and JSP Java API facilitates the upload and retrieval of multimedia data stored in a database using the Oracle Multimedia `OrdAudio`, `OrdDoc`, `OrdImage`, and `OrdVideo` object types. Oracle Multimedia Servlets and JSP Java API uses Oracle Multimedia Java API to access data stored in the Oracle Multimedia object types. However, Oracle Multimedia Servlets and JSP Java API can also be used to handle upload and retrieval of data using BLOBs directly.

The `OrdHttpResponseHandler` class facilitates the retrieval of multimedia data from a database and its delivery to a browser or other HTTP client from [Java servlets](#). The `OrdHttpJspResponseHandler` class provides the same features for [JavaServer Pages \(JSP\)](#).

---

---

**Note:** JSP engines are not required to support access to the servlet binary output stream. Therefore, not all JSP engines support the delivery of multimedia data using the `OrdHttpJspResponseHandler` class. See *Oracle Multimedia Servlets and JSP Java API Reference* for more information.

---

---

Form-based file uploading using HTML forms encodes form data and uploaded files in Post requests using the multipart/form-data format. The `OrdHttpUploadFormData` class facilitates the processing of such requests by parsing the Post data and making the contents of regular form fields and the contents of uploaded files readily accessible to a Java servlet or JSP. The handling of uploaded files is facilitated by the `OrdHttpUploadFile` class, which provides an easy-to-use API that applications call to load audio, image, and video data, or heterogeneous media data into a database.

See *Oracle Multimedia Servlets and JSP Java API Reference* for more information about Oracle Multimedia Servlets and JSP Java API.

#### **Integration with Oracle Application Server Portal**

Oracle Application Server Portal is used to create useful and appealing enterprise portals. A key feature of the Oracle Application Server Portal framework are [portlets](#), which provide a convenient way to access any type of data including rich content such as images, audio, and video. Oracle Application Server Portal has components that give the developer a declarative way to create objects that capture, act upon, and display data from an Oracle table or view. These Oracle Application Server Portal components can be connected together to create Web applications that can be applied directly to enterprise databases. And, as Oracle Multimedia objects are stored in Oracle tables, they can be included in the types of data available to Oracle Application Server Portal components.

Two Oracle Application Server Portal components are predefined: Forms and Reports. Oracle Application Server Portal contains wizards to help easily create a form to interact with the data in one or more database tables or views. The Forms component builds an appealing Web interface that lets users interact with data -- they can add,

query, update, and delete information stored in the database. Rich content can be both uploaded and downloaded between the database and the portal framework by building a form on tables containing Oracle Multimedia objects.

In addition to a form component, Oracle Application Server Portal offers a report component. The Reports component is used to display dynamic data in a columnar report format through a Web interface. Rich media content stored in tables can be downloaded, and again, wizards facilitate the creation of reports.

For more information about the use of Oracle Application Server Portal and Oracle Multimedia, see *Oracle Application Server Portal Developer's Guide* in the Oracle Application Server Online Documentation Library.

### **Integration with Oracle Application Development Framework Business Components**

For rapid development of media-rich Web applications, Oracle offers developers a Java integrated development environment (IDE), Oracle JDeveloper, that maximizes developer productivity. Oracle JDeveloper enables developers to build multitier, component-based Internet applications in Java that use Oracle Multimedia features to create visually attractive applications. Oracle Application Development Framework Business Components (ADF Business Components) is the component of JDeveloper that provides a set of intelligent software building blocks to manage common facilities. An Oracle Multimedia/ADF Business Components integration package includes media-specific domain classes and a set of utilities. The domain classes are wrappers of the classes of Oracle Multimedia Java API, and inherit all the underlying multimedia retrieval, upload, and manipulation methods. The domain classes support the ADF Business Components APIs and provide built-in integrated multimedia capabilities, while the utility classes support the retrieval, rendering, and uploading of multimedia content. Together, they provide a fully featured, integrated application development environment that enables a developer to create a wide variety of media-rich applications.

For more information, see the Oracle Multimedia/ADF Business Components Interactive Demonstration in the Oracle Multimedia Training section on the Oracle Technology Network Web site at

<http://www.oracle.com/technology/products/multimedia/>

### **1.9.4 Oracle Multimedia Custom DataSource and DataSink Classes for JMF 2.0/2.1**

Oracle Multimedia Custom DataSource and DataSink classes are an extension to the current Java Media Framework (JMF) version 2.0/2.1 developed by Sun Microsystems. This software allows a JMF application to upload and retrieve time-based media data stored in a database using Oracle Multimedia OrdAudio and OrdVideo objects.

For more information about these classes, see *Oracle Multimedia Custom DataSource and DataSink Classes Readme*. These classes can be downloaded from the Oracle Multimedia Software section of the Oracle Technology Network Web site at

<http://www.oracle.com/technology/products/multimedia/>

For more information about JMF, see the Sun Microsystems Web site at

<http://java.sun.com/>

### **1.9.5 Oracle Multimedia Support for Java Advanced Imaging (JAI)**

Oracle Multimedia Java API describes three types of stream objects, which provide interfaces to BLOB and BFILE data, that can be used by Java Advanced Imaging (JAI).

These Java classes allow a JAI application to read and write image data stored in a database using Oracle Multimedia `OrdImage` objects, or in BLOBs or BFILEs.

For more information about the Java classes for JAI stream objects provided by Oracle Multimedia, see *Oracle Multimedia Java API Reference*.

For more information about JAI, see the Sun Microsystems Web site at

<http://java.sun.com/>

## 1.10 Oracle Multimedia Architecture

Oracle Multimedia is a single, integrated feature that extends the database by storing, managing, and retrieving image, audio, and video data, and by supporting Web technologies for multimedia data.

The Oracle Multimedia architecture defines the framework (see [Figure 1-1](#)) through which media-rich content as well as traditional data are supported in the database. This content and data can then be securely shared across multiple applications written with popular languages and tools, easily managed and administered by relational database management and administration technologies, and offered on a scalable database that supports thousands of users.

[Figure 1-1](#) shows the Oracle Multimedia architecture from a three-tier perspective: database tier -- Oracle Database; application server tier -- Oracle Application Server; and client tier -- thin and thick clients.

In the first tier, through the use of Oracle Multimedia, Oracle Database holds rich content in tables along with traditional data. Through a database-embedded JVM, a server-side media parser is supported as well as an image processor. The media parser has object-oriented and relational interfaces, supports format and application metadata parsing, and can be extended to support additional formats. The image processor includes JAI and provides image processing for operations such as producing thumbnail-size images, converting image formats, and image indexing and matching.

Using Oracle Multimedia methods, import and export operations between the database and operating system files (external file storage) are possible. Oracle Multimedia also supports special delivery types of servers, such as streaming content from a database. Using the Oracle Multimedia Plug-ins for RealNetworks or Windows Media Services, the Helix Universal Server or Windows Media Streaming Server can stream multimedia data to a client directly out of the database using Real-Time Streaming Protocol (RTSP). In addition, third-party media processors such as speech recognition engines can run external to the database to process media stored in the database and return results to the database.

In the second or middle tier, Oracle Application Server provides access to Oracle Multimedia through Oracle Multimedia Java classes, which enable Java applications on any tier (client, application server, or database) to access, manipulate, and modify audio, image, and video data stored in a database.

In addition, Oracle Multimedia Servlets and JSP Java API facilitates the upload and retrieval of multimedia data stored in a database using the Oracle Multimedia `OrdAudio`, `OrdDoc`, `OrdImage`, and `OrdVideo` object types. Oracle Multimedia Servlets and JSP Java API can access data stored in the Oracle Multimedia objects or BLOBs or BFILEs directly.

Developers can also use Oracle JDeveloper and Oracle Multimedia to build media-rich Java applications quickly and easily using the Oracle Multimedia/ADF Business Components integration package. Oracle Multimedia rich content can also be easily

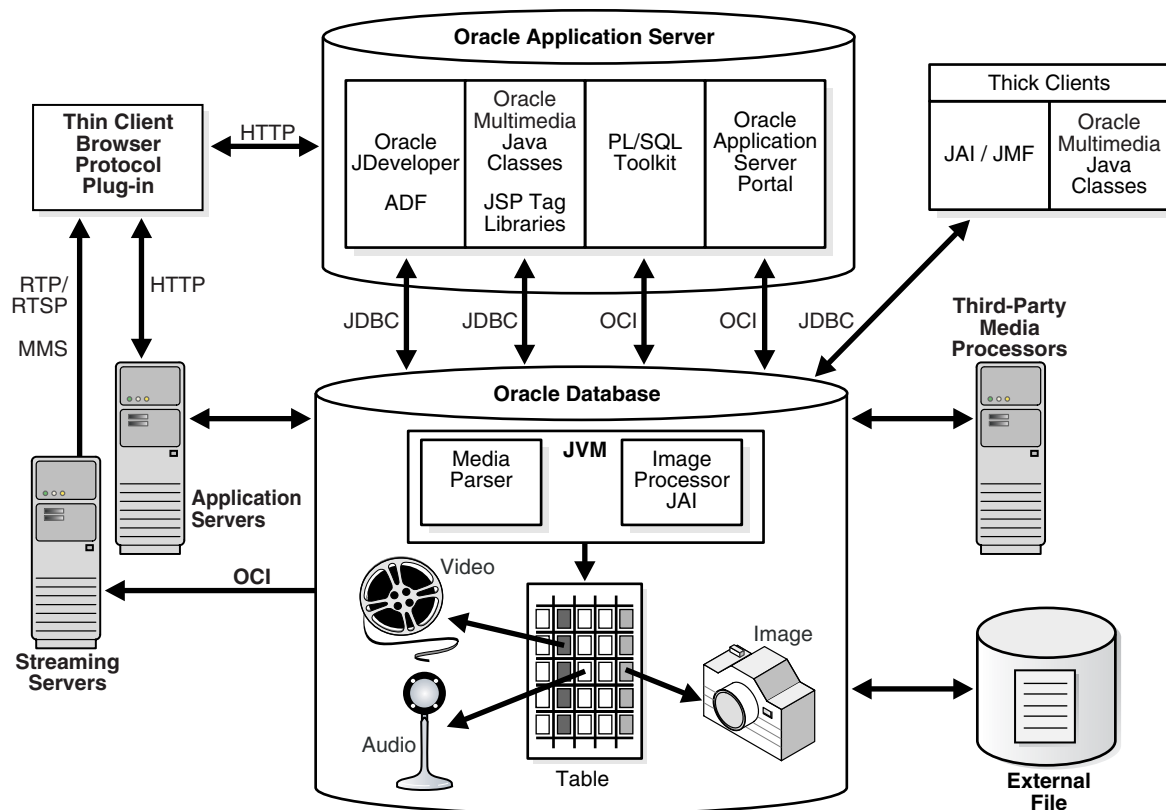


and transparently incorporated into Oracle Portal forms and reports, which can then be published as portlets.

SQL developers familiar with the database can develop Web applications that use Oracle Application Server exclusively, and Oracle Database using the PL/SQL development environment. The steps include using the PL/SQL Gateway (mod\_plsql) feature of the Oracle HTTP Server and the PL/SQL Web Toolkit. Web application developers can write PL/SQL servlets and PL/SQL server pages (PSP) that invoke PL/SQL procedures stored in the database through an Oracle Net connection and OCI.

In the third or client tier, the ability to perform local processing is supported through Oracle Multimedia Java classes, JAI, and JMF. JAI and JMF provide a set of APIs for media processing on the client, and Oracle Multimedia Java classes supply direct access to all media types from the client.

**Figure 1-1 Oracle Multimedia Architecture**



Oracle Multimedia features available only on Oracle Technology Network (OTN) include the following:

- Oracle Multimedia Plug-in for RealNetworks Streaming Servers, see [Section 1.9.2](#)
- Oracle Multimedia Plug-in for Microsoft Windows Media Services, see [Section 1.9.2](#)
- Oracle Multimedia Custom DataSource and DataSink classes JMF 2.0/2.1 (requires JMF 2.0 or higher), see [Section 1.9.4](#)

## 1.11 Extending Oracle Multimedia

Oracle Multimedia can be extended to support:

- Other external sources of media data not currently supported (other than BLOB, BFILE, or URL)
- Other media data formats not currently supported

---

---

**Note:** Oracle Multimedia can store any format. However, it can only extract metadata and process (image only) media data for formats that are supported or known to Oracle Multimedia. See the audio, image, and video data format appendixes in *Oracle Multimedia Reference* for lists of supported formats.

---

---

- Audio and video data processing

For more information about extending Oracle Multimedia, see [Chapter 7](#).

---

---

## Application Development

You can develop traditional client/server or two-tier applications, or you can develop multitier applications. Either method can then deploy Web applications to run on an application server tier, be tightly integrated with Oracle Database, and allow users access to the application from their desktop through a Web browser.

Using a complete development framework supported by class library interfaces, you can create production quality Oracle Multimedia applications for use in a production environment where users can interact with the application through either the standalone client interface or a Web browser. For Web applications, which are based on standards such as TCP/IP, HTTP, HTML, XML, and XHTML, this is all facilitated by rapid developments in the underlying technology. As key software components become more tightly integrated, developers' tasks to design, create, and manage Web applications become faster, easier, and simpler to implement.

Using either the object type interface or the relational interface, Oracle Multimedia provides Internet support for Oracle Application Server and Oracle Database and authoring tools so you can quickly develop Web-based applications to upload to the database, retrieve from it, and manipulate multimedia data for delivery to Web browsers.

Oracle Multimedia supports application development by:

- Providing class libraries that allow access (insert, update, and retrieve) and manipulation (process) of multimedia data stored in the database. Class libraries provide access to multimedia data stored in the database in the following ways:
  - Using Oracle Multimedia Java API. Using the Java database connectivity (JDBC) interface, you can quickly develop applications for use on any tier (client, application server, or database) to manipulate and modify audio, image, and video data, or heterogeneous media data stored in a database. Oracle Multimedia Java API makes it possible for JDBC result sets to include both traditional relational data and Oracle Multimedia columns of object type media data, to easily select and operate on the result set, to access object attributes, and to invoke object methods. See *Oracle Multimedia Java API Reference* for more information. See [Section 1.9.1](#) for more general information, and [Section 2.3](#) for information about how to develop media-rich Java client applications using these Java classes.
  - Using Oracle Multimedia Servlets and JSP Java API. These additional Java classes support Web technologies. See *Oracle Multimedia Servlets and JSP Java API Reference* for more information. See [Section 1.9.3](#) for more general information, and [Section 2.4](#) for information about how to develop media-rich Java-based Web applications using these Java classes. See [Section 3.2](#) for an example of a Java servlet application, and [Section 3.3](#) for an example of a JavaServer Pages (JSP) application.

- 
- Using the Oracle Multimedia/Oracle Application Development Framework Business Components (ADF Business Components) integration package, which includes the Oracle Multimedia domain classes and a set of utilities. These classes are for use with Oracle JDeveloper, the Java integrated development environment (IDE) tool that supports the application framework (ADF Business Components) that enables you to build multitier, component-based Internet applications. See [Section 1.9.3](#) and *Oracle JDeveloper 10g* help for more information.
  - Using Oracle Multimedia Custom DataSource and DataSink classes. These classes are an extension to JMF version 2.0/2.1 that allows a JMF application to upload and retrieve time-based media data stored in a database using Oracle Multimedia OrdAudio and OrdVideo object types. See [Section 1.9.4](#) for more information.
  - Using Java Advanced Imaging (JAI) classes. Oracle Multimedia Java API describes three types of stream objects, which provide interfaces to BLOB and BFILE data, that can be used by JAI. These classes allow a JAI application to read and write image data stored in a database using Oracle Multimedia OrdImage objects, or in BLOBs or BFILES. See [Section 1.9.5](#) for more information.
  - Using Oracle Application Server Multimedia Tag Library for JSP. This tag library is an extension of Oracle Multimedia Servlets and JSP Java API. It provides JSP tags that simplify retrieving and uploading media data from and to Oracle Database in multimedia JSP Web applications. See *Oracle Application Server 10g Multimedia Tag Library for JSP User's Guide and Reference* in the Oracle Application Server Online Documentation Library for more information.
  - Using C++ and traditional 3GLs through modern class library interfaces.
  - Using the PL/SQL Gateway (mod\_plsql) feature of the Oracle HTTP Server and the PL/SQL Web Toolkit features of Oracle Application Server and Oracle Database to listen for browser requests, to execute stored PL/SQL procedures in the database using Oracle Net and Oracle Call Interface (OCI), and to generate an HTML page containing data and code for the response returned to the Web browser for display. As a Web application developer, you can write PL/SQL servlets and PL/SQL server pages (PSP) that invoke PL/SQL procedures stored in the database through an Oracle Net connection and OCI. See [Section 2.2](#) for more information.
  - Integrating Oracle development tools with tightly integrated components to enable you to quickly and easily develop applications that provide access to (insert, update, and retrieve) and manipulation (process) of multimedia data stored in the database for delivery to Web browsers and client applications. These development tools include:
    - Oracle Application Server Portal -- a simple browser-based environment for building and deploying enterprise information portlets (EIPs). An enterprise portal provides access to **portlets**, which are summarized versions of applications and Web content situated in defined regions of the Web page. Oracle Application Server Portal portlets execute PL/SQL stored procedures residing in the database, which in turn generate an HTTP response in the form of a generated HTML page. Oracle Application Server Portal contains two predefined components: Forms and Reports, which both support rich media content being uploaded or downloaded between the database and the portal framework form or report. See [Section 1.9.3](#) for more information.

- 
- Oracle JDeveloper -- written 100% in Java, is the IDE tool that supports the application framework (Oracle Application Development Framework Business Components). An Oracle Multimedia/ADF Business Components integration package includes the Oracle Multimedia domain classes and a set of utilities. The domain classes are wrappers of Oracle Multimedia Java API and inherit all the underlying multimedia retrieval, upload, and manipulation methods. The domain classes support the ADF Business Components APIs and provide built-in integrated multimedia capabilities, while the utility classes support the retrieval, rendering, and uploading of multimedia content. See [Section 1.9.3](#) for more information.
  - Oracle Designer -- a tool used to manage software configuration management for controlling the evolution of an application from identification of components, through initiation, evaluation, authorization, development, and implementation. Oracle Designer can generate C++ classes that enable applications running on the client, on Oracle Application Server, or on Oracle Database to call Oracle Multimedia methods.
  - Oracle Content Management SDK -- lets you create custom file system applications using XML and Java that use the features and capabilities of the database, and a variety of Web-based interfaces, such as Java servlets and JSP pages, or executing SQL or calling stored PL/SQL procedures for execution in the transaction context of the database.
  - Integration with third-party streaming media servers to allow dynamic and direct delivery of multimedia data stored in the database to a media player client. These third-party streaming servers include:
    - Oracle Multimedia Plug-in for RealNetworks Server. This component is a data source plug-in that enables a RealNetworks server to stream media data directly from Oracle Database to a media player client. The plug-in is installed with RealNetworks Server and configured and managed using the administration tool of the streaming server. The plug-in is format neutral; any format supported by the streaming server can be sourced by Oracle Database. See [Section 1.9.2](#) for more information.
    - Oracle Multimedia Plug-in for Microsoft Windows Media Services. This tool allows Microsoft Windows Media servers to stream multimedia content to a client directly from Oracle Database. This plug-in is installed on Windows 2003 Server and configured with Windows Media Services. See [Section 1.9.2](#) for more information.

[Section 2.2](#) describes how to use PL/SQL Gateway and PL/SQL Web Toolkit to develop PL/SQL Web applications. [Section 2.3](#) describes how to use Java and JDBC to develop media-rich Java client applications based on Oracle Multimedia Java classes. [Section 2.4](#) describes how to develop Java-based Web applications based on Oracle Multimedia Java API and Oracle Multimedia Servlets and JSP Java API.

[Section 3.1](#) in Chapter 3 describes a sample Web application, Oracle Multimedia Photo Album, which is implemented using the following different technologies: PL/SQL, Java servlets and JavaServer Pages (JSP), and Microsoft Active Server Pages (ASP)/Visual Basic (VB). This sample application demonstrates how to apply the steps described in [Section 2.2](#) and [Section 2.4](#) in a real Web application to upload and retrieve media data stored in a database.

[Chapter 4](#) describes an Oracle Multimedia Code Wizard application that lets you create PL/SQL stored procedures for the PL/SQL Gateway to upload and retrieve media data stored in a database using Oracle Multimedia object types.

[Chapter 5](#) describes a sample Java application, `IMExample`, which is implemented using Java, JDBC, and Oracle Multimedia Java classes. This sample application demonstrates how to apply the steps described in [Section 2.3](#) in a real Java application to upload and retrieve media data stored in a database.

## 2.1 Developing PL/SQL Client Applications Using the PL/SQL API

PL/SQL is a completely portable, high-performance transaction processing language that combines the data manipulation power of SQL with the data processing power of procedural languages.

This section briefly describes how to manipulate Oracle Multimedia database objects with the PL/SQL Application Programming Interface (API). The following Oracle Multimedia object types are available for storing media in the database:

- ORDAudio
- ORDDoc
- ORDImage
- ORDVideo

[Section 2.1.1](#) describes how to set up the environment to use Oracle Multimedia with PL/SQL. [Section 2.1.2](#) describes how to use standard SQL queries with Oracle Multimedia objects. [Section 2.1.3](#) describes how to retrieve media data from Oracle Multimedia objects in a PL/SQL application. [Section 2.1.4](#) describes how to upload media data into Oracle Multimedia database objects in a PL/SQL application. [Section 2.1.5](#) describes how to handle exceptions in a PL/SQL application.

All the examples in this section use the sample schemas, which are installed by default when you install Oracle. Refer to *Oracle Database Sample Schemas* for information about how these schemas were created and how you can use them yourself.

See *Oracle Multimedia Reference* for details about the Oracle Multimedia object types and available methods in the PL/SQL API.

### 2.1.1 Setting Up Your Environment for PL/SQL

To access files with PL/SQL, you must create a directory object in the database that points to a directory that is accessible by the database server. For example, the following command creates the `MEDIA_DIR` directory in the sample schema:

```
CREATE DIRECTORY MEDIA_DIR AS
    'c:\oracle\product\10.2.0\db_1\demo\schema\product_media';
```

To retrieve media data from the database to a file, you must grant the write permission on the specified directory to the appropriate user. For example:

```
GRANT WRITE ON DIRECTORY MEDIA_DIR TO SCOTT;
```

To upload media data from a file to the database, you must grant the read permission on the specified directory to the appropriate user. For example:

```
GRANT READ ON DIRECTORY MEDIA_DIR TO SCOTT;
```

---



---

**WARNING:** Oracle-supplied Oracle Multimedia data types are installed under the database users ORDSYS, ORDPLUGINS, and SI\_INFORMTN\_SCHEMA. These users are created during database installation, and they are subject to change in future releases.

The functions, packages, types, and views supplied by Oracle Multimedia are reserved by Oracle, and they are subject to change in future releases. The ORDSYS and SI\_INFORMTN\_SCHEMA users and their contents must not be modified by the customer. Do not connect to these users or add your own tables, functions, packages, types, or views to these users. In addition, the packages supplied by Oracle Multimedia in the ORDPLUGINS user must not be modified by the customer. Doing so could cause internal errors and security violations in the database management system.

---



---

## 2.1.2 Media Query in PL/SQL

You can include media attributes (for example: height, width, and MIME type) in standard SQL queries by using accessor methods (for example: getHeight, getWidth, and getMimeType). [Example 2-1](#), [Example 2-2](#), and [Example 2-3](#) show how to use these accessor methods to query one or more object attributes for image, audio, and video objects, respectively.

### **Example 2-1 Image Query (Height, Width, and MimeType Attributes)**

```
SELECT t.product_id          id,
       t.product_photo.getHeight() height,
       t.product_photo.getWidth() width,
       t.product_photo.getMimeType() mimetype
FROM pm.online_media t;
```

### **Example 2-2 Audio Query (MimeType Attribute)**

```
SELECT t.product_id          id,
       t.product_audio.getMimeType() mimetype
FROM pm.online_media t;
```

### **Example 2-3 Video Query (MimeType Attribute)**

```
SELECT t.product_id          id,
       t.product_video.getMimeType() mimetype
FROM pm.online_media t;
```

## 2.1.3 Media Download in PL/SQL

To download media from the database into a file on the file system, call the export method of the Oracle Multimedia object. The following code example exports the image in the row with product\_id 3117 to a file named 3117.jpg in the directory MEDIA\_DIR. This code example highlights in bold the PL/SQL statements where this export operation takes place.

```
DECLARE
  img ORImage;
  ctx RAW(64) := NULL;
BEGIN
  SELECT product_photo
  INTO img
  FROM pm.online_media
```

```
        WHERE product_id = 3117;
        img.export(ctx, 'FILE', 'MEDIA_DIR', '3117.jpg');
    END;
    /
```

## 2.1.4 Media Upload in PL/SQL

Media upload means importing media data from the file system into the database tablespaces. The following series of steps is typical:

1. Insert a new row into the table, creating new objects by using the `init` method of the Oracle Multimedia object type.
2. Call the `import` method of the Oracle Multimedia object to bring the data from the file system into the database.
3. Call the `setProperties` method of the Oracle Multimedia object to determine and populate the attributes of the object.
4. Update the table so that the Oracle Multimedia object in the table contains the attribute values extracted in the previous step.

The PL/SQL code that implements these steps for inserting a new row in the `PM.ONLINE_MEDIA` table is shown in the following example:

```
DECLARE
    img ORDIImage;
    aud ORDAudio;
    vid ORDVideo;
    ctx RAW(64) := NULL;
BEGIN
    -- Insert a new row into the pm.online_media table.
    DELETE FROM pm.online_media WHERE product_id = 3003;
    INSERT INTO pm.online_media
        (product_id,
         product_photo,
         product_audio,
         product_video)
    VALUES (3003,
            ORDIImage.init('FILE', 'MEDIA_DIR', 'laptop.jpg'),
            ORDAudio.init('FILE', 'MEDIA_DIR', 'laptop.mpa'),
            ORDVideo.init('FILE', 'MEDIA_DIR', 'laptop.rm'))
    RETURNING product_photo, product_audio, product_video
    INTO img, aud, vid;

    -- Bring the media into the database and populate the attributes.
    img.import(ctx);
    -- ORDIImage.import also calls ORDIImage.setProperties.

    aud.import(ctx);
    aud.setProperties(ctx);

    vid.import(ctx);
    vid.setProperties(ctx);

    -- Update the table with the properties we have extracted.
    UPDATE pm.online_media
    SET     product_photo = img,
           product_audio = aud,
           product_video = vid
    WHERE  product_id = 3003;
```



```

    COMMIT;
END;
/

```

## 2.1.5 Handling Oracle Multimedia Exceptions in PL/SQL

Possible errors that can occur during run time should always be handled in your application. This practice enables the program to continue its operation even when it encounters a run-time error. This practice also enables users to know what went wrong during program operation. Proper error handling practices ensure that, whenever possible, you will always be able to recover from an error while running an application. In addition, proper error handling provides you with the information you need so that you will always know what went wrong.

This section demonstrates proper error handling practices through the use of code examples. These examples show how to handle some common Oracle Multimedia errors as well as other types of errors in PL/SQL programs. These examples are extracted from the PL/SQL sample applications described in [Chapter 3](#) and [Chapter 4](#). See *Oracle Multimedia Reference* for more examples.

When handling exceptions, PL/SQL uses exception blocks. For example, in PL/SQL, the exception may appear as:

```

BEGIN
<some program logic>
EXCEPTION
    WHEN OTHERS THEN
        <some exception logic>
END;

```

When you design, code, and debug your application, you are aware of the places in your program where processing might stop due to a failure to anticipate an error. Those are the places in your program where you must add exception handling blocks to handle the potential errors. For more information about handling PL/SQL exceptions, see *Oracle Database PL/SQL Language Reference*.

The following examples describe exception handling in the Oracle Multimedia PL/SQL Web Toolkit Photo Album sample application.

### 2.1.5.1 Handling the Setting of Properties for Unknown Image Formats

If your program tries to set the properties of an uploaded image (it reads the image data to get the values of the object attributes so it can store them in the appropriate attribute fields) and the image format is not recognized, then the `setProperties()` method will fail. To catch this exception and work around this potential problem, the application uses the following exception block:

```

BEGIN
    new_image.setProperties();
EXCEPTION
    WHEN OTHERS THEN
        new_image.contentLength := upload_size;
        new_image.mimeType := upload_mime_type;
END;

```

In this example, this exception handler sets the MIME type and length of the image based on the values from the upload table described at the beginning of the `insert_new_photo` procedure. The browser sets a MIME type header when the file is uploaded. The application reads this header to set the `ORDImage` field.

### 2.1.5.2 Handling Image Processing for Unknown Image Formats

If your program tries to process an image in cases when the image format is unknown, then the `processCopy()` method will always fail. To work around this potential problem, the application uses the following exception block:

```
BEGIN
    new_image.processCopy( 'maxScale=50,50', new_thumb);
EXCEPTION
    WHEN OTHERS THEN
        new_thumb.deleteContent();
        new_thumb.contentLength := 0;
END;
```

In this example from the Oracle Multimedia PL/SQL Web Toolkit Photo Album application, when the image format is unknown and a thumbnail image cannot be created, this exception handler deletes the content of the thumbnail image and sets its length to zero.

## 2.2 Developing PL/SQL Web Applications

SQL developers who are familiar with the database can develop Web applications that exclusively use Oracle Application Server and Oracle Database using the PL/SQL development environment. With the PL/SQL development environment, developers can come quickly up to speed to develop PL/SQL-based Web applications.

Developing Web applications using PL/SQL consists of developing one or more PL/SQL packages consisting of sets of stored procedures that interact with Web browsers through HTTP. Stored procedures can be executed in several ways:

- From a hypertext link that calls a stored procedure when it is selected
- By clicking **Submit** on an HTML form to denote the completion of a task such as filling out a form supplied on the HTML page
- By passing parameters to a stored procedure based on user choices from a list

Information in the stored procedure, such as tagged HTML text, is displayed in the Web browser as a Web page. These dynamic Web pages are generated by the database and are based on the database contents and the input parameters passed in to the stored procedure. Using PL/SQL stored procedures is especially efficient and powerful for generating dynamic Web page content.

There are two ways of generating HTML output from PL/SQL:

- Using function calls to generate each HTML tag for output using the PL/SQL Web Toolkit package that is part of Oracle Application Server and Oracle Database and whose `owa` packages are loaded into a common schema so that all users can access it
- Embedding PL/SQL code in Web pages (PL/SQL server pages)

Use Oracle Multimedia when media data such as images, audio, video, or combinations of all three are to be uploaded into and retrieved from database tables using the Oracle Multimedia object types and their respective sets of methods.

Media upload procedures first perform a SQL `INSERT` operation to insert a row of data in the media table, which also initializes instances of the respective Oracle Multimedia object columns with an empty `BLOB`. Next, a SQL `SELECT FOR UPDATE` operation selects the object columns for update. Finally, a SQL `UPDATE` operation updates the media objects in their respective columns. Oracle Multimedia methods are called to do the following:

- Initialize the object columns with an empty BLOB.
- Set attributes to indicate media data is stored internally in a BLOB.
- Get values of the object attributes and store them in the object attributes.
- When exceptions occur, determine the length of the BLOB content and its MIME type.

Media retrieval operations involve the following tasks:

- Retrieving the object from the database into a local object
- Checking the cache validity of the object based on its updated time versus that of the HTTP header time
- Determining where the media object is located: in the database, in a BFILE, or at a URL location; then, getting the media, and downloading it for display on an HTML page

Oracle Multimedia methods are called to get the time that the media object was last updated, to determine if the media is stored locally in the database, in a BFILE, or at a URL location, to get the MIME type of the media object, and finally to retrieve the media data.

## 2.2.1 Using the PL/SQL Gateway and PL/SQL Web Toolkit

Oracle Application Server and Oracle Database install Oracle HTTP Server powered by the Apache HTTP server that contains the PL/SQL Gateway to communicate directly with a client Web browser.

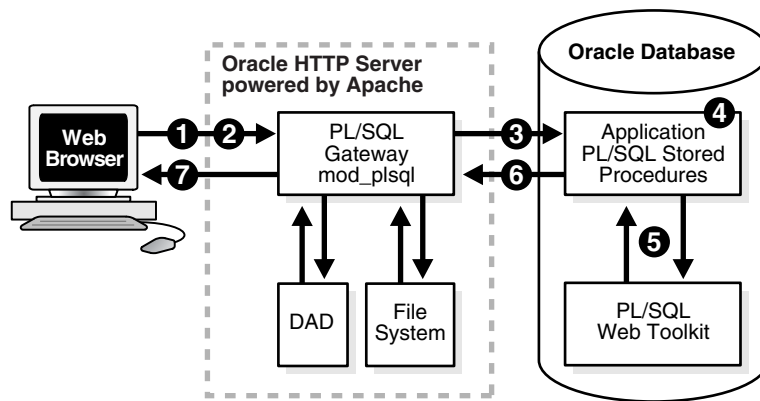
Oracle HTTP Server serves mainly the static HTML files, images, and so forth, that a Web application uses, and is usually located in the file system where Oracle HTTP Server is installed. Oracle HTTP Server contains modules or plug-ins that extend its functions. One of these modules supplied by Oracle is the `mod_plsql` module, also known as the PL/SQL Gateway. The PL/SQL Gateway serves data dynamically from the database to Web browsers by calling PL/SQL stored procedures. The PL/SQL Gateway receives requests from a Web browser in the form of PL/SQL servlets or PL/SQL server pages that are mapped to PL/SQL stored procedure calls. PL/SQL stored procedures retrieve data from the database and generate an HTTP response containing the data and code from the PL/SQL Web Toolkit to display the generated Web page in a Web browser. The PL/SQL Web Toolkit contains a set of packages called `http`, `htf`, and `owa` packages that can be used in the stored procedures to get information about the request, construct HTML tags, and return header information to the client Web browser.

[Figure 2–1](#) shows these main components of the PL/SQL development environment, Oracle HTTP Server (a component of Oracle Application Server and Oracle Database), the Web browser, and the database. The following information describes how a client Web browser request is turned into a Web page response from the execution of the PL/SQL procedure:

1. A client Web browser sends a PL/SQL server page or servlet request to Oracle HTTP Server.
2. Oracle HTTP Server routes the request to the PL/SQL Gateway (`mod_plsql`).
3. The PL/SQL Gateway forwards the request to the database using configuration information stored in the database access descriptor (DAD) and connects to the database.

4. The PL/SQL Gateway prepares the call parameters and invokes the PL/SQL package and the PL/SQL stored procedure in the application.
5. The PL/SQL procedure generates an HTML page using data from the database and special packages in the PL/SQL Web Toolkit accessed from the database. The PL/SQL Web Toolkit contains a set of packages called `http`, `htf`, and `owa` packages that are used in the stored procedures to get information about the request, construct HTML tags, and return header information back to the client Web browser as the response returned to the PL/SQL Gateway.
6. The PL/SQL Gateway sends the response to Oracle HTTP Server.
7. Oracle HTTP Server sends the response to the client Web browser for display as a formatted Web page.

**Figure 2–1 Components of the PL/SQL Development Environment**



Usually, the returned formatted Web page has one or more additional links, and each link, when selected, sends another request to the database through the PL/SQL Gateway to execute one or more stored procedures. The generated response displays data on the client Web page usually with additional links, which, when selected, execute more stored procedures that return the generated response for display as yet another formatted Web page, and so forth. This is how the PL/SQL application in the PL/SQL development environment is designed to work.

Web application developers who use the PL/SQL development environment, create a PL/SQL package specification and body that describe procedures and functions that comprise the application. The package specification defines the procedures and functions used by the application, and the package body is the implementation of each procedure and function. All packages are compiled and stored in the database to perform specific operations for accessing data in the database and formatting HTML output for Web page presentation. To invoke these stored PL/SQL procedures, Web application developers use the request/response PL/SQL servlets and PL/SQL server pages (PSP) to allow Web browser clients to send requests and get back responses using HTTP.

Oracle HTTP Server maps a URL entered in a browser to a specific PL/SQL procedure stored in the database. It does this by storing specific configuration information by means of a DAD for each stored procedure. Thus, each DAD contains the database connection information that is needed by the Web server to translate the URL entered into a database connection in order to call the stored procedure.

Oracle HTTP Server listens for a request, routes the request to the PL/SQL Gateway, which forwards it to the database. Configuration information values stored in a DAD determine the database alias to use, the connection string to use for remote access, the

procedure to use for uploading or downloading documents, and the user name and password information to allow access to the database. From the Web browser, the user specifies the URL that invokes the PL/SQL Gateway. The URL has a defined format specifying all required and optional parameters needed, including the location of the DAD and the name of the PL/SQL stored procedure to run, as shown in [Example 2-4](#).

**Example 2-4 URL Format Required for Invoking `mod_plsql` in a Web Browser**

```
protocol://hostname[:port number]/DAD-name/[!][schema name.][package
name.]procedure_name[?query_string]
```

For a detailed description of each parameter and options available, see *Oracle HTTP Server `mod_plsql` User's Guide*. However, to use the photo album application for Oracle Multimedia and the PL/SQL Web Toolkit described in [Section 3.1](#), the URL can be simplified to the format shown in [Example 2-5](#).

**Example 2-5 URL Format Required to Invoke `mod_plsql` in a Web Browser for the Photo Album Application**

```
protocol://<hostname>[:<port-number>]/DAD-name/]procedure_name
```

When the URL is entered in the Web browser, it includes the protocol (HTTP or HTTPS), the name of the hosting Web server, and the port number to which it is listening to handle requests. Next, the specified virtual path includes `/pls/<DAD-name>` to indicate that the Web server is configured to invoke `mod_plsql`, and the location of the DAD on the Web server.

In [Example 2-4](#), the last five parameters include the exclamation point (!) character, schema name, package name, procedure name, and query string. From the syntax, the exclamation point, schema name, package name, and query string parameters are optional; only the procedure name is required.

The exclamation point indicates that flexible parameter passing is being used. The schema name, if omitted, is resolved based on the user name. The package name, if omitted, means the procedure is standalone. The query string parameters are for the stored procedure and follow a special format. Of these five parameters, the procedure name must be specified in both the DAD and the URL. The other four parameters are specified in either the DAD or the URL, or not at all, depending on the application.

The URL displays the home page for the specified DAD. When the URL is entered in the address field of the Web browser page, it invokes either the specified DAD location only, or the specified DAD location along with the procedure name, or the specified DAD location along with the `schema.package.procedure` name. The response is returned as an HTML page. The HTML page contains the requested data and any other specified code for display in the client's Web browser. The Code Wizard described in [Chapter 4](#) illustrates how this works. For example, to invoke the Code Wizard administration URL, enter the following URL shown in that chapter:

```
http://<hostname>:<port-number>/pls/ordcwadmin
```

The virtual path includes `pls` to indicate that the Web server is configured to invoke `mod_plsql`, followed by the name of the DAD used for the Code Wizard administrator, `ordcwadmin`.

When the HTML page is displayed, it resolves to the following URL for the Code Wizard administrator:

```
http://<hostname>:<port-number>/pls/ordcwadmin/ORDCWPKG.menu
```

ORDCWPKG.menu represents the package.procedure name, which is specified as the default home page in the ordcadmin DAD.

When the PL/SQL Gateway is invoked, it uses the stateless model and does not allow a transaction to span across multiple HTTP requests. In this stateless model, applications typically can create a session to maintain state by using one of the following techniques: HTTP cookies, a hidden HTML field as an HTML form element of the HTML Form package, or storage of vital information in database tables for query. For more information, see *Oracle Database Advanced Application Developer's Guide*.

## 2.3 Developing Java Client Applications Using JDBC

Developers who are familiar with Java and Java database connectivity (JDBC) can write media-rich Java applications using Oracle Multimedia Java API. The classes in Oracle Multimedia Java API are the Java proxy classes for Oracle Multimedia database objects. These Java classes provide access to Oracle Multimedia database objects through JDBC in a Java application.

The Java classes in Oracle Multimedia Java API are included in the `oracle.ord.im.*` package. These Java classes are named similarly to the Oracle Multimedia database objects, and in compliance with the standard Java naming convention:

- OrdAudio
- OrdDoc
- OrdImage
- OrdVideo

[Section 2.3.1](#) describes how to set up the environment to use Oracle Multimedia Java API. [Section 2.3.2](#) describes how to retrieve media data from Oracle Multimedia objects in a Java application. [Section 2.3.3](#) describes how to upload media data into Oracle Multimedia database objects in a Java application. [Section 2.3.4](#) describes how to handle exceptions in a Java application.

All the examples in this section use the sample schemas, which are installed by default when you install Oracle. Refer to *Oracle Database Sample Schemas* for information about how these schemas were created and how you can use them yourself.

See *Oracle Multimedia Java API Reference* for details about the available methods in these classes.

### 2.3.1 Setting Up Your Environment for Java

Before you can begin using Oracle Multimedia Java API, you must set up your environment to compile and run Java programs. Follow these steps:

1. Specify the environment variable CLASSPATH, and ensure that it includes the appropriate Oracle Java libraries for the Oracle Multimedia and other features you intend to use.

For each Java library, the following table lists the name of the Java library, the Oracle Multimedia or other features that require that library, details about the JDK version that supports the library, the platform, and the path name under the `<ORACLE_HOME>` directory where you can obtain the library JAR file.

Name of Oracle Java Library	Related Feature	JDK Version, Platform, and Location
Oracle JDBC library	All Oracle Multimedia features	JDK 5 or later, on Linux and UNIX: <ORACLE_HOME>/jdbc/lib/ojdbc5.jar  JDK 5 or later, on Windows: <ORACLE_HOME>\jdbc\lib\ojdbc5.jar
Oracle Multimedia Java classes library	All Oracle Multimedia features	JDK 5 or later, on Linux and UNIX: <ORACLE_HOME>/ord/jlib/ordim.jar  JDK 5 or later, on Windows: <ORACLE_HOME>\ord\jlib\ordim.jar
Oracle Multimedia DICOM Java classes library	DICOM feature	JDK 5 or later, on Linux and UNIX: <ORACLE_HOME>/ord/jlib/orddicom.jar  JDK 5 or later, on Windows: <ORACLE_HOME>\ord\jlib\orddicom.jar
Oracle XDB Java classes library	DICOM feature Oracle Multimedia metadata extraction	JDK 5 or later, on Linux and UNIX: <ORACLE_HOME>/rdbms/jlib/xdb.jar  JDK 5 or later, on Windows: <ORACLE_HOME>\rdbms\jlib\xdb.jar
Oracle Multimedia Java classes for servlets and JSP library	Java servlets and JSP applications	JDK 5 or later, on Linux and UNIX: <ORACLE_HOME>/ord/jlib/ordhttp.jar  JDK 5 or later, on Windows: <ORACLE_HOME>\ord\jlib\ordhttp.jar
NLS Character Set Conversion library (Optional) <ORACLE_HOME>/jlib/orai18n.jar	NLS character set conversion required <sup>1</sup>	JDK 5 or later, on Linux and UNIX: <ORACLE_HOME>/jlib/orai18n.jar  JDK 5 or later, on Windows: <ORACLE_HOME>\jlib\orai18n.jar

<sup>1</sup> If NLS character set conversion is required between the client application and the database, you must include the `orai18n.jar` file in the CLASSPATH variable. If NLS character set conversion is required, but the appropriate library is not specified, character-based attributes of Oracle Multimedia object types may be returned as hex-encoded strings. See *Oracle Database JDBC Developer's Guide and Reference* for more information about NLS character set conversion.

---

**Note:** If you are using the JDBC OCI driver, you must specify the location of the JDBC OCI shared library in one of the following variables: LD\_LIBRARY\_PATH (for Linux or UNIX) or PATH (for Windows).

Depending on your platform, store the JDBC OCI shared library at one of the following locations under the <ORACLE\_HOME> directory:

<ORACLE\_HOME>/lib (for libocijdbc11.so on Linux and UNIX)  
<ORACLE\_HOME>/bin (for ocijdbc11.dll on Windows)

Because this library path is shared, it may have been specified previously to enable the use of other client applications such as SQL\*Plus.

---

2. Add one or more of the following import statements to the Java program:

Along with the standard JDBC classes included in the `java.sql` package, you must also import the Oracle JDBC extension class `oracle.jdbc.OracleResultSet`, as follows:

```
import oracle.jdbc.OracleResultSet;
```

Based on the type of media to be handled in the Java application, you might also need to add one or more of following import statements:

```
import oracle.ord.im.OrdAudio;
import oracle.ord.im.OrdDoc;
import oracle.ord.im.OrdImage;
import oracle.ord.im.OrdVideo;
```

## 2.3.2 Media Retrieval in Java

Oracle Multimedia objects can be retrieved into Java applications as Java proxy objects to the Oracle Multimedia database objects with the same names: `OrdAudio`, `OrdDoc`, `OrdImage`, and `OrdVideo`. After the JDBC connection is established, follow these steps to retrieve Oracle Multimedia Java objects:

1. Create the JDBC statement to select Oracle Multimedia objects from the database:

```
String query = "select product_photo, product_audio, "+
    " product_video, product_testimonials from" +
    " pm.online_media where product_id=3117";
PreparedStatement pstmt = conn.prepareStatement(query);
```

2. Execute the query and obtain the result set:

```
OracleResultSet rset = (OracleResultSet)pstmt.executeQuery();
```

3. Retrieve the Oracle Multimedia Java object from the result set:

```
if ( rset.next() )
{
    OrdImage imgProxy = (OrdImage)rset.getORADData(
        "product_photo", OrdImage.getORADDataFactory());
    OrdAudio audProxy = (OrdAudio)rset.getORADData(
        "product_audio", OrdAudio.getORADDataFactory());
    OrdVideo vidProxy = (OrdVideo)rset.getORADData(
        "product_video", OrdVideo.getORADDataFactory());
    OrdDoc docProxy = (OrdDoc)rset.getORADData(
        "product_testimonials",
        OrdDoc.getORADDataFactory());
}
```

---

---

**Note:** In Oracle Multimedia release 8i and release 9i, the `getCustomDatum` method is used to retrieve the Oracle Multimedia Java objects. In Oracle JDBC release 10g, the `getCustomDatum` method was deprecated and replaced by the `getORADData` method.

In your Java program environment, be sure to use the same version for both the Oracle Multimedia Java Client library (`ordim.jar`) and the Oracle JDBC library.

---

---



4. Retrieve the media attributes. Media attributes can be retrieved directly from Oracle Multimedia Java objects. For example:

```
int height = imgProxy.getHeight();
int width = imgProxy.getWidth();
String audFormat = audProxy.getFormat();
String vidMimetype = vidProxy.getMimeType();
```

### 2.3.3 Media Upload in Java

Follow these steps to upload media data into Oracle Multimedia database objects in a Java application:

1. Enter the following statement to enable the JDBC connection object to set the autocommit flag to `false`:

```
conn.setAutoCommit(false);
```

2. Retrieve Oracle Multimedia Java objects from the database for updating. You can load media data into existing Oracle Multimedia objects in a table or into nonexisting Oracle Multimedia objects by creating a new row in a table.

The following example includes a query you can use to load media data into existing Oracle Multimedia objects in a table.

```
//"for update" is required in the query string
//since we will update the row later.
String query1 = "select product_photo," +
" product_audio, product_video," +
" product_testimonials from" +
" pm.online_media where product_id=3106" +
" for update";

PreparedStatement pstmt = conn.prepareStatement(query1);

OracleResultSet rset = (OracleResultSet)pstmt.executeQuery();

if ( rset.next() )
{

    OrdImage imgProxy = (OrdImage)rset.getORADData(
        "product_photo", OrdImage.getORADDataFactory());
    OrdAudio audProxy = (OrdAudio)rset.getORADData(
        "product_audio", OrdAudio.getORADDataFactory());
    OrdVideo vidProxy = (OrdVideo)rset.getORADData(
        "product_video", OrdVideo.getORADDataFactory());
    OrdDoc docProxy = (OrdDoc)rset.getORADData(
        "product_testimonials",
        OrdDoc.getORADDataFactory());
}

rset.close();
pstmt.close();
```

The following example includes a query you can use to load media data into nonexisting Oracle Multimedia objects by creating a new row.

---

**Note:** This code segment assumes that there is no row with `product_id=3106` in the `pm.online_media` table.

---

```

String query2 =
    "begin insert into pm.online_media " +
    " (product_id, product_photo, product_audio," +
    " product_video, product_testimonials) values" +
    " (3106, ordimage.init()," +
    " ordaudio.init(), ordvideo.init()," +
    " orddoc.init()) returning product_photo," +
    " product_audio, product_video," +
    " product_testimonials into ?, ?, ?, ?;end;";

OracleCallableStatement cstmt =
    (OracleCallableStatement) conn.prepareCall(query2);
cstmt.registerOutParameter(1, OrdImage._SQL_TYPECODE,
    OrdImage._SQL_NAME);
cstmt.registerOutParameter(2, OrdAudio._SQL_TYPECODE,
    OrdAudio._SQL_NAME);
cstmt.registerOutParameter(3, OrdVideo._SQL_TYPECODE,
    OrdVideo._SQL_NAME);
cstmt.registerOutParameter(4, OrdDoc._SQL_TYPECODE,
    OrdDoc._SQL_NAME);

cstmt.execute();

OrdImage imgProxy = (OrdImage)cstmt.getORADData(1,
    OrdImage.getORADDataFactory());
OrdAudio audProxy = (OrdAudio)cstmt.getORADData(2,
    OrdAudio.getORADDataFactory());
OrdVideo vidProxy = (OrdVideo)cstmt.getORADData(3,
    OrdVideo.getORADDataFactory());
OrdDoc docProxy = (OrdDoc)cstmt.getORADData(4,
    OrdDoc.getORADDataFactory());

cstmt.close();

```

3. Load the media data from a file to the Oracle Multimedia Java objects by calling the `loadDataFromFile` method:

```

String imageFileName = "laptop.jpg";
String audioFileName = "laptop.mpa";
String videoFileName = "laptop.rm";
String docFileName = "laptop.jpg";
imgProxy.loadDataFromFile(imageFileName);
audProxy.loadDataFromFile(audioFileName);
vidProxy.loadDataFromFile(videoFileName);
docProxy.loadDataFromFile(docFileName);

```

4. Set the properties of the Oracle Multimedia objects by populating the Java object fields with media attributes (optional):

```

imgProxy.setProperties();
audProxy.setProperties(new byte[1][64]);
vidProxy.setProperties(new byte[1][64]);
docProxy.setProperties(new byte[1][64], true);

```

---

**Note:** The `setProperties` method will try to recognize the format of the media and populate the objects field with media information such as image height, image width, format, MIME type, and so on. If the media format is not recognized, the `java.sql.SQLException` error will be thrown.

---

5. Update the database table with Oracle Multimedia Java objects that have data already loaded:

```
String query3 = "update pm.online_media set" +
  " product_photo=?, product_audio=?," +
  " product_video=?, product_testimonials=?" +
  " where product_id=3106";
OraclePreparedStatement pstmt =
(OraclePreparedStatement) conn.prepareStatement(query3);
pstmt.setORADData(1, imgProxy);
pstmt.setORADData(2, audProxy);
pstmt.setORADData(3, vidProxy);
pstmt.setORADData(4, docProxy);

pstmt.execute();
pstmt.close();
```

6. Commit the transaction:

```
conn.commit();
```

### 2.3.4 Handling Oracle Multimedia Exceptions in Java

Possible errors that can occur during run time should always be handled in your application. This practice enables the program to continue its operation even when it encounters a run-time error. This practice also enables users to know what went wrong during program operation. Proper error handling practices ensure that, whenever possible, you will always be able to recover from an error while running an application. In addition, proper error handling provides you with the information you need so that you will always know what went wrong.

This section demonstrates proper error handling practices through the use of code examples. These examples show how to handle some common Oracle Multimedia errors as well as other types of errors in Java programs. These examples are extracted from the Java sample applications described in [Chapter 3](#) and [Chapter 5](#). See Oracle Multimedia Reference for more examples.

When handling exceptions, Java uses the try/catch block. For example, in Java, the exception may appear as:

```
try {
    //<some program logic>
}
catch (exceptionName a) {
    //Exception logic
}
finally {
    //Execute logic if try block is executed even if an exception is caught
}
```

When you design, code, and debug your application, you are aware of the places in your program where processing might stop due to a failure to anticipate an error. Those are the places in your program where you must add exception handling blocks to handle the potential errors. For more information about handling Java exceptions, see *Oracle Database Java Developer's Guide* and *Oracle Database JDBC Developer's Guide and Reference*.

The following examples describe exception handling using the try/catch block. These examples are in either the Oracle Multimedia IMExample application or are in the

Oracle Multimedia Java Servlet Photo Album application, the Oracle Multimedia JavaServer Pages Photo Album application, or both of these applications.

### 2.3.4.1 Handling the Setting of Properties for Unknown Image Formats

The `IMUtil` class of the Oracle Multimedia Java sample application called `IMExample`, contains utility methods for common image functions. One of these methods is the `setProperties()` method. The static method takes an `OrdImage` object as an input parameter and calls the `setProperties()` method on the object.

```
static boolean setProperties(OrdImage img)
{
    try
    {
        img.setProperties();
        return true;
    }
    catch (SQLException e)
    {
        return false;
    }
}
```

If an exception is thrown, the `setProperties()` method returns `false` to indicate failure; otherwise it returns `true`.

See [Chapter 5](#) for a full description of the `IMExample` application and for more information about using the `setProperties()` method in a Java application.

### 2.3.4.2 Handling Image Processing for Unknown Image Formats

In the `insertNewPhoto()` method in both the `PhotoAlbumServlet` class of the Oracle Multimedia Java Servlet Photo Album application and in the `PhotoAlbumBean` class of the Oracle Multimedia JavaServer Pages Photo Album application, a new photograph is inserted into the photo album, creating a thumbnail image at the same time. If the application tries to process an image in cases when the image format is unknown, then when the application calls the `processCopy()` method, the application will always fail. To work around this potential problem, the application uses the following `try` block and `catch` block to catch any SQL exceptions:

```
try
{
    image.processCopy( "maxScale=50,50", thumb );
}
catch ( SQLException e )
{
    thumb.deleteContent();
    thumb.setContentLength( 0 );
}
```

In this example, when the image format is unknown and a thumbnail image cannot be created, the application catches the SQL exception and calls the `deleteContent()` method to delete the content of the thumbnail image, and then calls the `setContentLength()` method to set its length to zero.

## 2.4 Developing Java-Based Web Applications

On the Java platform, a Web application is a dynamic extension of a Web server. A Java-based Web application is composed of Java servlets, JSP pages, or both. Java

Servlets are Java classes that dynamically process HTTP requests and construct HTTP responses. JSP pages are text-based documents that execute as servlets but that allow a more natural approach to creating static content.

Oracle Multimedia Servlets and JSP Java API is based on Oracle Multimedia Java API. The classes in Oracle Multimedia Servlets and JSP Java API facilitate the retrieval and uploading of media data from and to Oracle Database in a Java-based Web application.

The Java classes in Oracle Multimedia Servlets and JSP Java API are included in the `oracle.ord.im.*` package. The classes are as follows:

- `OrdHttpResponseHandler`
- `OrdHttpJspResponseHandler`
- `OrdHttpUploadFormData`
- `OrdHttpUploadFile`
- `OrdMultipartFilter`
- `OrdMultipartWrapper`

The `OrdHttpResponseHandler` class facilitates the retrieval of the media data from Oracle Database and its delivery to an HTTP client from a Java servlet. The `OrdHttpJspResponseHandler` class provides the same features for JSP pages. The `OrdHttpUploadFormData`, `OrdHttpUploadFile`, `OrdMultipartFilter`, and `OrdMultipartWrapper` classes facilitate the uploading of media data from a Web client to Oracle Database.

[Section 2.4.1](#) describes how to retrieve media data from Oracle Multimedia objects in a Java-based Web application. [Section 2.4.2](#) describes how to upload media data into database Oracle Multimedia objects in a Java-based Web application.

Before you can begin using Oracle Multimedia Servlets and JSP Java API, you must set up your environment with the appropriate Java libraries, as described in Step 1, [Section 2.3.1](#). In addition to the items in that list, you must include the Oracle Multimedia Java Web library `<ORACLE_HOME>/ord/jlib/ordhttp.jar` in your CLASSPATH environment variable.

See *Oracle Multimedia Servlets and JSP Java API Reference* for details about the available methods in these classes.

## 2.4.1 Media Retrieval in Java-Based Web Applications

In general, displaying a Web page that contains images in a Web browser requires two HTTP round trips.

In the first trip, the Web browser makes an HTTP request to the URL of the Web page that contains the images. The Web server responds with the Web page text content and the URLs for the media content. The URL is the `src` attribute of the `<img>` tag in the Web page.

In the second trip, the Web browser makes another HTTP request to the URL in the `<img>` tag to get the image binary data, and then displays the image in the browser.

In a Java-based Web application, sending media data from the database to an HTTP client (Web browser) requires the proper media URL (generated in the first HTTP response); and the proper media delivery component (a servlet or JSP for the second HTTP response).

### 2.4.1.1 Media URL

When media data is stored as static files on the Web server, the media URL is the relative or absolute path to the media files on the file system. When media data is stored in a database, the media URL is generally composed of a media delivery component (a servlet or JSP) and the parameters for the media delivery component. The media delivery component is the target for the second HTTP request to retrieve the media data. The parameters for the media delivery component are used by the media delivery component to query and locate the media data in the database. For example:

```

```

where `OrdGetMedia.jsp` in the media URL `"OrdGetMedia.jsp?id=1"` is the media delivery component, and `id=1` is the parameter to the media delivery component.

### 2.4.1.2 Media Delivery Component

Because media data is stored in the database as Oracle Multimedia objects, the media delivery component must dynamically retrieve the media data as Java objects (see [Section 2.3.2](#)), based on certain query conditions. Then, you can use either the `OrdHttpResponseHandler` or the `OrdHttpJspResponsehandler` class in Oracle Multimedia Servlets and JSP Java API to deliver the data to the HTTP client (Web browser).

The following example demonstrates the use of a Java servlet as the media delivery component, and highlights in bold the SQL statements and significant areas in the code where this operation takes place.

```
import oracle.ord.im.OrdHttpResponseHandler;

protected void doGet(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, java.io.IOException
{
    // obtain oracle.ord.im.OrdImage object
    // imgProxy follows the Section "Media Retrieval in Java"
    . . .

    // deliver the image data to the browser
OrdHttpResponseHandler handler =
new OrdHttpResponseHandler( request, response);
handler.sendImage(imgProxy);

    . . .
}
```

The following example demonstrates the use of a JSP page as the media delivery component, and highlights in bold the SQL statements and significant areas in the code where this operation takes place.

```
<%@ page
import="oracle.ord.im.OrdHttpJspResponseHandler"
%>

<jsp:useBean id = "handler" scope="page"
    class = "oracle.ord.im.OrdHttpJspResponseHandler"
/>

<%
```

```

// obtain oracle.ord.im.OrdImage object
// imgProxy follows the Section "Media Retrieval in Java"
. . .

// deliver the image data to the browser
handler.setPageContext( pageContext );
handler.sendImage(imgProxy);
return;
%>

```

## 2.4.2 Media Upload in Java-Based Web Applications

The HTML form enables you to input and upload data from a Web browser to a Web server for processing. The following HTML code segment is an example of the HTML form that uploads a file. This code example highlights in bold the SQL statements and areas in the code where this operation takes place.

```

<form action="uploadAction.jsp" method="post"
enctype="multipart/form-data">
id: <input type="text" name="id"/>
description: <input type="text" name="description"/>
Photo: <input type="file" name="photo"/>
</form>

```

Referring to the preceding code example, setting the value of the `enctype` attribute in the `<form>` tag to `"multipart/form-data"` specifies multipart/form-data format encoding of the uploaded data. The value of the `action` attribute in the `<form>` tag represents the name of the JSP that will handle the uploaded data.

To handle the uploaded data in a JSP or servlet, follow these steps:

1. Decode the uploaded file. Because this file is encoded in multipart/form-data format, the data must be decoded before further processing can proceed. You can use the `OrdHttpUploadFormData` class to decode the encoded HTTP request data and obtain the uploaded file as an instance of the `OrdHttpUploadFile` object. You can use this class explicitly or implicitly to get the decoded uploaded file.

The following example demonstrates how to use the `OrdHttpUploadFormData` class *explicitly* to get the uploaded file, and highlights in bold the SQL statements and significant areas in the code where this operation takes place. Use this method within the servlet or JSP that handles the upload HTTP request.

```

//
// Import OrdHttpUploadFormData and OrdHttpUploadFile class:
// In a servlet:
// import oracle.ord.im.OrdHttpUploadFormData;
// import oracle.ord.im.OrdHttpUploadFile;
// In a JSP:
// <%@ page import="oracle.ord.im.OrdHttpUploadFormData" %>
// <%@ page import="oracle.ord.im.OrdHttpUploadFile" %>
//

//
// Following code snippets should be within <% %> if in a JSP.
//

// Create an OrdHttpUploadFormData object and use it to parse
// the multipart/form-data message.
//
OrdHttpUploadFormData formData = new OrdHttpUploadFormData(
request );

```

```
formData.parseFormData();

//
// Get the description, location, and photo.
//
String id = formData.getParameter( "id" );
String description = formData.getParameter( "description" );
OrdHttpUploadFile photo = formData.getFileParameter( "photo" );

//
// Process the uploaded file.
//
...

//
// Release the resources.
//
formData.release();
```

To avoid instantiating and releasing the `OrdHttpUploadFormData` class explicitly in each JSP or servlet that handles the uploaded data, you can use the `OrdHttpUploadFormData` class implicitly by configuring the Web application with the `OrdMultipartFilter` class. Using the `OrdMultipartFilter` class ensures that any HTTP request that is encoded in multipart/form-data format will be decoded and passed along to the JSP or servlet that further processes the request.

The following substeps and accompanying examples describe how to use the `OrdHttpUploadFormData` class *implicitly* to get the uploaded file. These code examples highlight in bold the SQL statements and significant areas in the code where this operation takes place.

- a. Configure the filter by adding the following code to the `web.xml` file in your Web application:

```
<filter>
  <filter-name>OrdMultipartFilter</filter-name>
  <filter-class>
oracle.ord.im.OrdMultipartFilter
  </filter-class>
</filter>
<filter-mapping>
  <filter-name>OrdMultipartFilter</filter-name>
  <servlet-name>*.jsp</servlet-name>
</filter-mapping>
```

- b. Obtain the form data and the uploaded file in the JSP or servlet after the filter is configured:

```
//
// Get the id, description, and photo.
//
String id = request.getParameter( "id" );
String description = request.getParameter( "description" );
oracle.ord.im.OrdHttpUploadFile photoFile =
request.getFileParameter( "photo" );
```

where `request` is the `HttpServletRequest` object passed to the JSP or servlet.

2. Save the uploaded file to the database. After the `OrdHttpUploadFile` object is obtained by explicitly or implicitly using the `OrdHttpUploadFormData` class, the



uploaded file is ready to be loaded into an Oracle Multimedia object in the database, using the following statement:

```
photoFile.loadImage(imgProxy);
```

where `photoFile` is the `OrdHttpUploadFile` object and `imgProxy` is an `OrdImage` object obtained in Step 1 in [Section 2.3.3](#).

The `loadImage` method implicitly calls the `setProperties` method to populate the object fields.

After the data is loaded into the Oracle Multimedia Java object, you can update the corresponding Oracle Multimedia object in the database table by following Steps 4 and 5 in [Section 2.3.3](#).



---

---

## Oracle Multimedia Photo Album Sample Applications

This chapter describes several types of media upload and retrieval Web applications using Oracle Multimedia object types. The following Oracle Multimedia Photo Album sample Web applications are described in this chapter:

- [Section 3.1](#) describes the PL/SQL Photo Album sample application that uses the PL/SQL Gateway and PL/SQL Web Toolkit for Oracle Application Server and Oracle Database.
- [Section 3.2](#) describes the Java Servlet Photo Album sample application that uses Oracle Multimedia Servlets and JSP Java API.
- [Section 3.3](#) describes the JSP Photo Album sample application that uses Oracle Multimedia Servlets and JSP Java API.
- [Section 3.4](#) describes the Active Server Pages (ASP)/Visual Basic (VB) Photo Album sample application for the Microsoft Internet Information Server (IIS) Web Server.

---

---

**Note:** The Photo Album sample applications described in [Section 3.2](#), [Section 3.3](#), and [Section 3.4](#) differ from the PL/SQL Photo Album sample application described in [Section 3.1](#).

---

---

This chapter assumes the following:

- You are familiar with:
  - Developing PL/SQL applications using the PL/SQL Gateway and PL/SQL Web Toolkit
  - Developing Java-based Web applications using JDBC, creating Java source code, compiling it into byte code (`.class`) files, and deploying class files into respective servlet containers required by Oracle HTTP Server for Oracle Application Server and Oracle Database
  - Developing ASP/VB scripts for the Microsoft IIS Web Server
- You have already installed and configured the following sample applications:
  - Oracle Multimedia PL/SQL Web Toolkit Photo Album application
  - Oracle Multimedia Java Servlet Photo Album application
  - Oracle Multimedia JSP Photo Album application
  - Oracle Multimedia ASP/VBScript Photo Album application

See the `README.txt` file for each respective sample application for installation and configuration information.

See [Chapter 4](#) for a description of the Oracle Multimedia Code Wizard sample application, a media upload and retrieval Web application for the PL/SQL Gateway.

See [Chapter 5](#) for a description of the Oracle Multimedia IMExample sample application. This sample application lets you retrieve, save, play, and delete multimedia data from the Oracle Database sample schemas using Oracle Multimedia Java classes and Oracle Multimedia object types.

## 3.1 Oracle Multimedia PL/SQL Photo Album Sample Application

The Oracle Multimedia PL/SQL Web Toolkit Photo Album sample application demonstrates how to perform the following operations:

- Use the Oracle Multimedia image object type to upload, retrieve, and process media data stored in Oracle Database.
- Combine the image metadata methods of Oracle Multimedia with the XML document management capabilities of Oracle XML DB and the full-text indexing and search features of Oracle Text to create a solution that can extract, store, and search metadata that is embedded in binary image files.
- Collect new metadata from a user, format the metadata into an XML document, and store the document in the binary image.

When installed, this photo album application creates a number of schema objects that are important to the following discussion. These objects include the `photos` table, which is defined by the following CREATE TABLE statement:

```
CREATE TABLE photos( id          NUMBER PRIMARY KEY,
                    description  VARCHAR2(40) NOT NULL,
                    metaORDImage XMLTYPE,
                    metaEXIF     XMLTYPE,
                    metaIPTC     XMLTYPE,
                    metaXMP      XMLTYPE,
                    image        ORDSYS.ORDIMAGE,
                    thumb        ORDSYS.ORDIMAGE )

--
-- store full-size images and thumbnail images as SecureFile LOBs
--
LOB(image.source.localdata) STORE AS SECUREFILE
LOB(thumb.source.localdata) STORE AS SECUREFILE;
--
-- and bind the XMLType columns to the interMedia metadata schemas
XMLType COLUMN metaORDImage
  XMLSCHEMA "http://xmlns.oracle.com/ord/meta/ordimage"
  ELEMENT "ordImageAttributes"
XMLType COLUMN metaEXIF
  XMLSCHEMA "http://xmlns.oracle.com/ord/meta/exif"
  ELEMENT "exifMetadata"
XMLType COLUMN metaIPTC
  XMLSCHEMA "http://xmlns.oracle.com/ord/meta/iptc"
  ELEMENT "iptcMetadata"
XMLType COLUMN metaXMP
  XMLSCHEMA "http://xmlns.oracle.com/ord/meta/xmp"
  ELEMENT "xmpMetadata";
```

The data types for the `image` and `thumb` columns are defined as Oracle Multimedia image object types. These columns are used to store the full-size images and the

generated thumbnail images, respectively. The LOB storage clauses indicate that storage for the full-size images is to be allocated in 32-kilobyte chunks, thus enabling the fastest reading and writing of the image data. Similarly, storage for the thumbnail images is allocated in 16-kilobyte chunks, enabling fast access and efficient storage. In addition, using a smaller chunk size reduces the allocation of empty space.

The table also defines four columns of type `XMLType` to store XML documents that contain four different kinds of image metadata. Each column is bound to a specific Oracle Multimedia metadata schema. Each metadata schema defines precisely the data model of the metadata document. These schemas are registered with Oracle XML DB when the database is created. The column definitions specify that the database uses structured storage to manage the XML metadata documents. Some advantages of using structured storage to manage XML include optimized memory management, reduced storage requirements, B-tree indexing, and in-place updates. For more information about XML DB, see *Oracle XML DB Developer's Guide*.

When installed, this photo album application also creates other schema objects. These schema objects include two types of indexes that accelerate metadata searches: a `CONTEXT` text index and an `XMLIndex` index.

The `CONTEXT` type is a text index over all columns that contain descriptive information about the image. These columns include `PHOTOS.DESCRPTION`, which is a `VARCHAR2` data type, and the following four `XMLType` columns: `PHOTOS.META IPTC`, `PHOTOS.META EXIF`, `PHOTOS.META XMP`, and `PHOTOS.META ORDIMAGE`. The `CONTEXT` text index is used to accelerate metadata searches by implementing the photo album search feature that allows users to search for photographs by keyword or phrase.

The `CONTEXT` text index is created by the following statements. (This example assumes that this photo album application has been installed in the `SCOTT` schema.)

```
-- Create preference PA_CTXIDX.
ctx_ddl.create_preference('SCOTT.PA_CTXIDX', 'MULTI_COLUMN_DATASTORE');

-- Create a multicolumn datastore.
ctxcols := 'description, ' ||
           'SCOTT.photo_album.getClob(META IPTC), ' ||
           'SCOTT.photo_album.getClob(META EXIF), ' ||
           'SCOTT.photo_album.getClob(META XMP), ' ||
           'SCOTT.photo_album.getClob(META ORDIMAGE)';
ctx_ddl.set_attribute( ctxpref, 'COLUMNS', ctxcols );

-- Create the CONTEXT text index.
create index pa_ctx_idx on photos(description)
indextype is ctxsys.context
parameters ( 'DATASTORE SCOTT.PA_CTXIDX' );
```

For more information about creating and using text indexing, see *Oracle Text Application Developer's Guide*.

The `XMLIndex` index is used to accelerate metadata searches by allowing users to search only certain types of image metadata as well as limit the search to specific portions of an XML document. For example, the following statements create three indexes of type `XMLIndex` to speed up `existsNode()` queries on columns of type `XMLType`:

```
create index pa_path_iptc_idx on photos( metaIptc )
indextype is XDB.XMLIndex;
```

```
create index pa_path_exif_idx on photos( metaExif )
  indextype is XDB.XMLIndex;

create index pa_path_xmp_idx on photos( metaXMP )
  indextype is XDB.XMLIndex;
```

For more information about creating and using XMLIndex indexes, see *Oracle XML DB Developer's Guide*.

During the installation, as prescribed by the PL/SQL Gateway, a document upload table is defined by the following CREATE TABLE statement:

```
CREATE TABLE PHOTOS_UPLOAD( name          VARCHAR2(256) UNIQUE NOT NULL,
                             mime_type     VARCHAR2(128),
                             doc_size      NUMBER,
                             dad_charset   VARCHAR2(128),
                             last_updated  DATE,
                             content_type  VARCHAR2(128),
                             blob_content  BLOB )
--
-- store BLOBs as SecureFile LOBs
--
LOB(blob_content) STORE AS SECUREFILE;
```

Each image uploaded using the PL/SQL Gateway is stored in the PHOTOS\_UPLOAD table. An upload procedure (insert\_new\_photo) automatically moves the uploaded image from the specified PHOTOS\_UPLOAD table to the photo album applications table called photos.

The sample application files and README.txt file are located at:

<ORACLE\_HOME>/ord/http/demo/plsqlwtk (on Linux and UNIX)

<ORACLE\_HOME>\ord\http\demo\plsqlwtk (on Windows)

The following subsections describe how to run the PL/SQL Photo Album application. See the README.txt file for additional requirements and instructions on installing and using this sample application.

### 3.1.1 Running the PL/SQL Photo Album Application

After you have completed the setup tasks and have built this photo album application, including creating a database access descriptor (DAD) entry as described in the README.txt file, you can run this photo album application by entering the following URL in the address field of your Web browser:

```
<protocol><hostname:port-number>/photoalbum
```

The <protocol> field is http:// and the <hostname:port-number> field is the host name and port number of the system where your HTTP server is running.

When first invoked, this photo album application displays any images that are currently stored in the album. By default, the photo album is empty when first installed. To upload a new photograph, select **Upload photo**. Enter a description of the photograph and the name of the image file, or browse to its directory location. Then, click **Upload photo**.

The contents of the photo album are displayed, along with a picture of the new photograph. Click the thumbnail image to view the full-size version of the photograph. When this photo album application displays the text **view image** instead

of its thumbnail image, the image format that was uploaded was not recognized by Oracle Multimedia. Click **view image** to display the full-size image.

You can now begin to load your photo album application with your favorite photographs.

### 3.1.2 Description of the PL/SQL Photo Album Application

The user interface for the PL/SQL Photo Album application consists of a set of Web pages. You can use these Web pages to perform the tasks shown in [Table 3-1](#). The tasks and the Web pages are introduced in this section and described in further detail in the following sections.

**Table 3-1 PL/SQL Photo Album Sample Application Overview**

User Task	Web Page	PL/SQL Procedures
Browsing the photo album <a href="#">Section 3.1.2.1</a>	<b>View album</b> <a href="#">Figure 3-1</a>	<code>view_album</code> <a href="#">Example 3-1</a>
		<code>print_album</code> <a href="#">Example 3-2</a>
		<code>print_image_link</code> <a href="#">Example 3-3</a>
		<code>deliver_media</code> <a href="#">Example 3-4</a>
Adding images to the photo album <a href="#">Section 3.1.2.2</a>	<b>Upload photo</b> <a href="#">Figure 3-2</a>	<code>view_upload_form</code>
		<code>print_upload_form</code> <a href="#">Example 3-5</a>
		<code>insert_new_photo</code> <a href="#">Example 3-6</a>
Searching for images by keyword or phrase <a href="#">Section 3.1.2.3</a>	<b>Search album</b> <a href="#">Figure 3-3</a>	<code>view_album</code> <a href="#">Example 3-1</a>
		<code>print_album</code> <a href="#">Example 3-2</a>
Viewing full-size images <a href="#">Section 3.1.2.4</a>	<b>View entry</b> <a href="#">Figure 3-4</a>	<code>view_entry</code> <a href="#">Example 3-7</a>
		<code>print_image_link</code> <a href="#">Example 3-3</a>
		<code>deliver_media</code> <a href="#">Example 3-4</a>
Examining image metadata <a href="#">Section 3.1.2.5</a>	<b>View metadata</b> <a href="#">Figure 3-5</a>	<code>view_metadata</code> <a href="#">Example 3-8</a>
		<code>print_metadata</code> <a href="#">Example 3-9</a>
Writing new XMP metadata to images <a href="#">Section 3.1.2.6</a>	<b>Write XMP metadata</b> <a href="#">Figure 3-6</a>	<code>write_metadata</code> <a href="#">Example 3-10</a>
Searching for images that contain specific metadata attributes <a href="#">Section 3.1.2.7</a>	<b>Search metadata</b> <a href="#">Figure 3-7</a>	<code>search_metadata</code> <a href="#">Example 3-11</a>

You can explore this photo album application using the navigation bar near the top of each Web task page. The leftmost entry of the navigation bar displays the name of the current Web page. On the right, there are links to other Web pages you can access from the current page. Each Web task page contains a link to the **View album** page, which is the home page for the application.

This photo album application is implemented as a set of PL/SQL procedures and functions organized in a single PL/SQL package. These procedures combine several database features to create the application. Oracle Multimedia is used to store and process image data. It is also used to extract metadata from images and embed new metadata into images. The XMLType feature is used to store and process the XML metadata documents. Oracle Text indexes are used to accelerate two kinds of metadata searches. Finally, the PL/SQL Web Toolkit is used to create HTML pages and deliver media content.

For detailed information about these database features, see *Oracle XML DB Developer's Guide*, *Oracle Text Application Developer's Guide*, and *Oracle Application Server 10g PL/SQL Web Toolkit Reference* in the Oracle Application Server Online Documentation Library.

**Browsing the Photo Album Using the View album Page** The **View album** page displays thumbnail-size versions of all the images in the photo album as well as a description link positioned under each thumbnail image. When you select a thumbnail image, the full-size image is displayed. When you select the description link for an image, all the metadata for that image is displayed. The **View album** page is the home page for the application.

**Adding Images to the Photo Album Using the Upload photo Page** The **Upload photo** page displays a simple form to collect a description for a new image and the directory path to the location of the image on the local computer. When you click the **Upload photo** button, the browser sends the image to the Web server and the image is stored in the database.

**Searching for Images by Keyword or Phrase Using the Search album Page** The **Search album** page displays a search album form to collect keywords or phrases to initiate full text searches through all image metadata. The application queries the database for all images with metadata that contains the specified keywords or phrases. The search results are displayed as a set of thumbnail images. The search album form is also available from the **View album** page.

**Viewing Full-Size Images Using the View entry Page** The **View entry** page displays the full-size image of a specified photograph, including any description text that was entered for that image when it was uploaded.

**Examining Image Metadata Using the View metadata Page** The **View metadata** page displays all the metadata that was extracted from the image when it was uploaded. Up to four types of metadata can be displayed.

**Writing New XMP Metadata to Images Using the Write XMP metadata Page** The **Write XMP metadata** page displays a form to collect input for five metadata attributes. These attributes are formatted into an XML document that is embedded within the binary image. The new XMP metadata overwrites any existing XMP metadata.

**Searching for Images That Contain Specific Metadata Attributes Using the Search metadata Page** The **Search metadata** page collects input for advanced metadata searches. You can specify the type of metadata to be searched. Optionally, you can also



limit the search to a specific XML tag within the specified document. The search results are displayed as a set of thumbnail images.

### 3.1.2.1 Browsing the Photo Album

The home page for this photo album application, **View album**, displays the contents of the photo album as thumbnail images in four-column format. Each thumbnail image is also a link to the **View entry** page. When you click a thumbnail image link, the application displays the full-size image on a View entry page. Included under each thumbnail image on the **View album** page is the image description that was entered when the image was uploaded to the album. The description is also a link to the **View metadata** page where all the metadata for this photograph can be examined.

Near the top of the **View album** page, there is a text entry field (in the shape of a rectangular box) that accepts user input for a full text search through all the photo album metadata. The **Search** button to the right of the text field initiates the search. The search results are displayed on the **Search album** page, which is discussed in [Section 3.1.2.3](#).

At the top of the **View album** page, there is a navigation bar, which includes links to other photo album pages. From the **View album** page, you can navigate to the **Search metadata** page or the **Upload photo** page. These pages are described in [Section 3.1.2.7](#) and [Section 3.1.2.2](#), respectively.

[Figure 3-1](#) shows the **View album** page for an album that contains five images.

**Figure 3-1 View album Page with Five Uploaded Images**



The PL/SQL procedures `view_album`, `print_album`, `print_image_link`, and `deliver_media` are the primary application components that implement the **View album** page. The `view_album` procedure is a public procedure that takes a single optional argument. By default, the argument has a `NULL` value. Or, it can have the value of the string entered in the text entry field on the **Search album** page. When the search argument is `NULL`, the `SELECT` statement retrieves the `id`, `description`, and `thumb` columns for all entries in the `photos` table. When the search string is not `NULL`, the `SELECT` statement uses the `CONTAINS` operator to restrict the result set to only images with metadata that matches the search string. (Section 3.1 describes how the application creates a multicolumn text index over the four XMLType columns `PHOTOS.METAIPTC`, `PHOTOS.METAEXIF`, `PHOTOS.METAXMP`, and `PHOTOS.METAORDIMAGE` as well as the `PHOTOS.DESCRPTION` column.)

Example 3–1 contains some relevant lines of code in the `view_album` procedure.

**Example 3–1 Procedure `view_album`**

```
--
-- no search criteria so fetch all entries
--
IF search IS NULL THEN
  OPEN album_cur FOR
    SELECT id, description, thumb
    FROM photos
    ORDER BY id;
  print_album( album_cur, 'The photo album is empty.' );
  CLOSE album_cur;
ELSE
  -- use the full-text index to select entries matching the search criteria
  --
  OPEN album_cur FOR
    SELECT id, description, thumb
    FROM photos
    WHERE CONTAINS( description, trim(search) ) > 0
    ORDER BY id;
  print_album( album_cur, 'No photos were found.' );
  CLOSE album_cur;
END IF;
```

The `SELECT` statement is bound to the cursor variable `album_cur` and passed to the procedure `print_album`, which creates the HTML output.

The `print_album` procedure uses the `HTP` and `HTF` packages from the PL/SQL Web Toolkit to create the HTML tags that format the output into a four-column table. Each cell in the table contains two links or anchor tags. The first link is to the **View entry** page, which displays the full-size version of the image. This anchor is implemented by `PHOTO_ALBUM.VIEW_ENTRY`, and passes `entry_id` as a query string input argument. If the thumbnail image has a nonzero length, then procedure `print_image_link` is called to create an HTML `<img>` tag that is the content (the thumbnail image) of the anchor link. The string `thumb` and the `entry_id` are passed to procedure `print_image_link`, along with the image description, and the height and width of the thumbnail image. These values are used to create the `<img>` tag.

If an image is in a format that Oracle Multimedia does not support, the application will not be able to create a thumbnail version of the image. In this case, the content of the anchor link is the text **view image**.

Example 3–2 contains some relevant lines of code in the `print_album` procedure.

**Example 3–2 Procedure `print_album`**

```

-- escape the description text
sc_description := htf.escape_sc( entry.description );

--
-- Display the thumb-nail image as an anchor tag which can be used
-- to display the full-size image. If the image format isn't
-- supported by interMedia, then a thumb-nail wouldn't have been
-- produced when the image was uploaded, so use the text '[view
-- image]' instead of the thumb-nail.
--
http.print( '<td headers="c' || colIdx || '" align="center" >
            <a href="PHOTO_ALBUM.VIEW_ENTRY?entry_id=' ||
            entry.id || '">' );
IF entry.thumb.contentLength > 0
THEN
    print_image_link( 'thumb', entry.id, sc_description,
                    entry.thumb.height, entry.thumb.width );
ELSE
    http.prn( '[view image]' );
END IF;
http.print( '</a>' );

-- Create link to the metadata
http.prn('<br>');
http.anchor( curl=>'PHOTO_ALBUM.VIEW_METADATA?entry_id=' || entry.id,
            ctext=>sc_description );
http.prn('</td>');

```

The procedure `print_image_link` uses the height and width arguments to populate the height and width attributes of the `<img>` tag. The description argument is used to create text for the alt attribute. If the description argument is empty, a default string is constructed. Finally, the `src` attribute is set to the URL `PHOTO_ALBUM.DELIVER_MEDIA` with two query string arguments, `media` and `entry_id`. The `media` argument controls whether the thumbnail or full-size version of the image is delivered. The `entry_id` argument identifies the image to be delivered.

[Example 3–3](#) contains some relevant lines of code in the `print_image_link` procedure.

**Example 3–3 Procedure `print_image_link`**

```

-- add height and width to tag if non zero
IF height > 0 AND width > 0 THEN
    attributes := attributes || ' height=' || height || ' width=' || width;
END IF;

-- create an alt text if none given
IF alt IS NULL THEN
    IF type = 'thumb' THEN
        alt2 := 'thumb-nail image ';
    ELSE
        alt2 := 'full-size image ';
    END IF;
    alt2 := alt2 || 'for album entry ' || entry_id;
ELSE
    alt2 := alt;
END IF;

```

```

http.img( curl=>'PHOTO_ALBUM.DELIVER_MEDIA?media=' || type ||
          ampersand || 'entry_id=' || entry_id,
          calt=>alt2, cattributes=>attributes );

```

The procedure `deliver_media` fetches the image content from the database. The `If-Modified-Since` HTTP request header is compared to the last modification time of the image. If the image has not been modified, a response is sent that the browser can display the image from its cache. Otherwise, the image MIME type and last modified time are sent to the Web server, along with the image content.

[Example 3-4](#) contains some relevant lines of code in the `deliver_media` procedure.

#### **Example 3-4 Procedure `deliver_media`**

```

--
-- Fetch the thumb-nail or full-size image from the database.
--
IF media = 'thumb'
THEN
    SELECT thumb INTO local_image FROM photos WHERE id = entry_id;
ELSE
    SELECT image INTO local_image FROM photos WHERE id = entry_id;
END IF;

--
-- Check update time if browser sent If-Modified-Since header
--
IF ordplsgwyutil.cache_is_valid( local_image.getUpdateTime() )
THEN
    owa_util.status_line( ordplsgwyutil.http_status_not_modified );
    RETURN;
END IF;

--
-- Set the MIME type and deliver the image to the browser.
--
owa_util.mime_header( local_image.mimeType, FALSE );
ordplsgwyutil.set_last_modified( local_image.getUpdateTime() );
owa_util.http_header_close();

IF owa_util.get_cgi_env( 'REQUEST_METHOD' ) <> 'HEAD' THEN
    wpg_docload.download_file( local_image.source.localData );
END IF;

```

### **3.1.2.2 Adding Images to the Photo Album**

The **Upload photo** page is used to add new images to the photo album. The page displays a form with two text entry fields. In the **Description:** field, you can optionally enter a word or short phrase that describes the image. In the **File name:** field, enter the name of the image file or click **Browse...** to locate the image file to be uploaded. The **Upload photo** button under the **File name:** field starts the upload operation. When the image is successfully uploaded, the **View album** page appears. From that page, you can display the contents of the photo album, as described in [Section 3.1.2.1](#).

At the top of the **Upload photo** page, there is a navigation bar, which includes links to other photo album pages. From the **Upload photo** page, you can return to the **View album** page or select the **Search metadata** page. These pages are described in [Section 3.1.2.1](#) and [Section 3.1.2.7](#), respectively.

Figure 3–2 shows an **Upload photo** page with all the entry fields completed.

**Figure 3–2 Completed Upload photo Page**



The PL/SQL procedures `view_upload_form`, `print_upload_form`, and `insert_new_photo` are the primary application components that implement the **Upload photo** page. Together, `view_upload_form` and `print_upload_form` create the HTML page that is displayed. The page contains a form tag, a portion of which is shown in [Example 3–5](#). The target of the form is `PHOTO_ALBUM.INSERT_NEW_PHOTO`.

[Example 3–5](#) contains some relevant lines of code in the `print_upload_form` procedure.

**Example 3–5 Procedure `print_upload_form`**

```
<form action="PHOTO_ALBUM.INSERT_NEW_PHOTO"
method="post"
enctype="multipart/form-data">
database.
```

Procedure `insert_new_photo` receives the form, processes the inputs, and stores the new image in the database.

First, the `insert_new_photo` procedure checks that a file name was entered into the upload form. The image size, MIME type, and BLOB locator for the image content are selected from the document upload table, and the size is checked to ensure that the image is not of zero length. If the `description` field is blank, a description is created using the file name.

Next, the `ORDSYS.ORDIMAGE.INIT()` function is called to initialize the `thumb` and `image` `ORDImage` object type columns with an empty BLOB for the new row to be stored in the `photos` table. A SQL `SELECT FOR UPDATE` statement fetches the newly initialized thumbnail image and full-size image object type columns for updating. A `DBMS_LOB.COPY` operation loads the image from the upload table into the `image` `ORDImage` object type column.

The `ORDImage` object method `setProperties()` reads the image and sets the image object attributes. Because some browsers cannot display some image formats inline, in this sample application, BMP formatted images are converted to a JPEG image format (for images with more than 8 bits of color), or a GIFF image format (for images with less than 9 bits of color) by calling the `get_preferred_format` function. A `processCopy()` operation is performed on the full-size image to create the thumbnail image.

The `ORDImage` object `getMetadata()` method is called to extract all supported types of image metadata. The root element of each XML document in the return vector is examined to discover the metadata type so that the documents can be stored in the correct columns.

Then, a SQL `UPDATE` statement stores the full-size image, the thumbnail image, and the image metadata documents in the database. Procedure `sync_indexes` is called to force an update of the text indexes. Finally, the form data input is deleted from the document upload table. A success message is returned to the browser, and the browser is redirected to the **View album** page.

[Example 3-6](#) contains some relevant lines of code in the `insert_new_photo` procedure.

#### **Example 3-6 Procedure `insert_new_photo`**

```
--
-- Make sure a file name has been provided. If not, display an error
-- message, then re-display the form.
--
IF new_photo IS NULL OR LENGTH( new_photo ) = 0
THEN
    print_page_header;
    print_error( 'Please supply a file name.' );
    print_upload_form;
    print_page_trailer( TRUE );
    return;
END IF;

--
-- Get the length, MIME type and the BLOB of the new photo from the
-- upload table.
--
SELECT doc_size,
       mime_type,
       blob_content
INTO   upload_size,
       upload_mime_type,
       upload_blob
FROM   photos_upload
WHERE  name = new_photo;

--
-- Make sure we have a valid file.
--
```

```

IF upload_size = 0
THEN
    print_page_header;
    print_heading( 'Error message' );
    http.print( '<hr size="-1"><p>Please supply a valid image file.</p>' );
    print_upload_form;
    print_page_trailer( TRUE );
    return;
END IF;

--
-- If the description is blank, then use the file name.
--
IF c_description IS NULL
THEN
    c_description := new_photo;
    pos := INSTR( c_description, '/', -1 );
    IF pos > 0
    THEN
        c_description := SUBSTR( c_description, pos + 1 );
    END IF;
    c_description := SUBSTR( 'Image from file: ' ||
        c_description || '.', 1, 40 );
END IF;
--
-- Insert a new row into the table, returning the newly allocated sequence
-- number.
INSERT INTO photos ( id, description, metaExif, metaIPTC, metaXMP,
    image, thumb )
VALUES ( photos_sequence.nextval, c_description, NULL, NULL, NULL,
    ORDSYS.ORDIMAGE.INIT(), ORDSYS.ORDIMAGE.INIT() )

RETURN id
INTO new_id;

--
-- Fetch the newly initialized full-size and thumb-nail image objects.
--
SELECT image,
    thumb
INTO new_image,
    new_thumb
FROM photos
WHERE id = new_id
FOR UPDATE;

--
-- Load the photo from the upload table into the image object.
--
DBMS_LOB.COPY( new_image.source.localData, upload_blob, upload_size );
new_image.setLocal();
--
-- Set the properties. If the image format is not recognized, then
-- the exception handler will set the MIME type and length from the
-- upload table.
--
BEGIN
    new_image.setProperties();
EXCEPTION
    WHEN OTHERS THEN
        new_image.contentLength := upload_size;

```

```
        new_image.mimeType := upload_mime_type;
END;

--
-- Some image formats are supported by interMedia but may not be able
-- to be displayed in-line by a browser. The BMP format is one example.
-- Convert the image to a GIF or JPEG based on number of colors in the
-- image.
--
IF new_image.contentFormat IS NOT NULL AND
( new_image.mimeType = 'image/bmp' OR
  new_image.mimeType = 'image/x-bmp' )
THEN
  BEGIN
    new_image.process(
      'fileFormat=' ||
      get_preferred_format( new_image.contentFormat ) );
  EXCEPTION
    WHEN OTHERS THEN
      NULL;
  END;
END IF;

--
-- Try to copy the full-size image and process it to create the thumb-nail.
-- This may not be possible if the image format is not recognized.
--
BEGIN
  new_image.processCopy( thumb_scale, new_thumb );
EXCEPTION
  WHEN OTHERS THEN
    new_thumb.deleteContent();
    new_thumb.contentLength := 0;
END;

--
-- fetch the metadata and sort the results
--
BEGIN
  metav := new_image.getMetadata( 'ALL' );
  FOR i IN 1..metav.count() LOOP
    meta_root := metav(i).getRootElement();
    CASE meta_root
      WHEN 'ordImageAttributes' THEN xmlORD := metav(i);
      WHEN 'xmpMetadata' THEN xmlXMP := metav(i);
      WHEN 'iptcMetadata' THEN xmlIPTC := metav(i);
      WHEN 'exifMetadata' THEN xmlEXIF := metav(i);
      ELSE NULL;
    END CASE;
  END LOOP;
EXCEPTION
  WHEN OTHERS THEN
    NULL;
END;

--
-- Update the full-size and thumb-nail images in the database.
-- Update metadata columns
--
UPDATE photos
SET image = new_image,
```



```

        thumb = new_thumb,
        metaORDImage = xmlORD,
        metaEXIF = xmlEXIF,
        metaIPTC = xmlIPTC,
        metaXMP = xmlXMP
WHERE id = new_id;

--      -- update the text indexes
--      sync_indexes;

--
-- Delete the row from the upload table.
--
DELETE FROM photos_upload WHERE name = new_photo;
COMMIT;

--
-- Redirect browser to display full album.
--      print_page_header(
--          '<meta http-equiv="refresh" content="2;url=PHOTO_ALBUM.VIEW_ALBUM">' );
print_heading( 'Photo successfully uploaded into photo album' );

```

### 3.1.2.3 Searching for Images by Keyword or Phrase

You can use the **View album** and **Search album** pages to perform a keyword or phrase search of the metadata stored in the photo album. On either of these pages, enter the keyword or phrase in the **Full text search:** text entry field and click **Search**. This photo album application uses the `CONTEXT` text index to locate images that have metadata containing the text you entered. If the search is successful, the thumbnail versions of the matching images are displayed in a four-column table. Select the thumbnail image to view the full-size version, or select the description link below the thumbnail image to view the metadata for the image. If the search fails, the message "No photos were found" is displayed.

At the top of the **Search album** page, there is a navigation bar, which includes links to other photo album pages. From the **Search album** page, you can return to the **View album** page or select the **Search metadata** or **Upload photo** pages. These pages are described in [Section 3.1.2.1](#), [Section 3.1.2.7](#), and [Section 3.1.2.2](#), respectively.

[Figure 3–3](#) shows a **Search album** page that contains the results of a successful search operation.

**Figure 3–3 Search album Page Showing Results**

Full text searching of the photo album is implemented by the `view_album` and `print_album` procedures. See [Section 3.1.2.1](#) for a discussion of these procedures.

#### 3.1.2.4 Viewing Full-Size Images

When you select a thumbnail image, the application directs you to the **View entry** page. This page displays the description of the image and the full-size version of the image.

At the top of the **View entry** page, there is a navigation bar, which includes links to other photo album pages. From the **View entry** page, you can return to the **View album** page, or select any of the **View metadata**, **Write metadata**, **Search metadata**, or **Upload photo** pages. These pages are described in [Section 3.1.2.1](#), [Section 3.1.2.5](#), [Section 3.1.2.6](#), [Section 3.1.2.7](#), and [Section 3.1.2.2](#), respectively.

[Figure 3–4](#) shows a **View entry** page that contains the description and the full-size version of an image.

**Figure 3–4 View entry Page with a Full-Size Image**

The PL/SQL procedures `view_entry`, `print_image_link`, and `deliver_media` are the primary application components that implement the **View entry** page. The procedure `view_entry` takes a single parameter, `entry_id`, which uniquely locates the image in the `photos` table. The description and image object are fetched from the `photos` table. The procedure `print_image_link` creates the HTML `<img>` tag, and then calls procedure `deliver_media` to fetch the image content. See [Section 3.1.2.1](#) for more information about the `print_image_link` and `deliver_media` procedures.

[Example 3–7](#) contains some relevant lines of code in the `view_entry` procedure.

**Example 3–7 Procedure `view_entry`**

```
--
-- Fetch the row.
--
BEGIN
  SELECT htf.escape_sc(description), image
  INTO sc_description, photo
  FROM photos
  WHERE id = entry_id;
  EXCEPTION
  WHEN no_data_found THEN
    print_error( 'Image <b>' || htf.escape_sc(entry_id) ||
                '</b> was not found.</p>' );
    print_page_trailer( TRUE );
  return;
```

```
END;
```

```
print_image_link( 'image', entry_id, sc_description,
photo.height, photo.width );
```

### 3.1.2.5 Examining Image Metadata

You can use the **View metadata** page to examine all the metadata for a specific image. Typically, you access this page from the **View album** page by selecting the description link below a thumbnail image. You can also access this page by selecting the **View metadata** link from the navigation bar. The **View metadata** page displays the thumbnail version of the image. To the right of the thumbnail image, there is a list of the metadata documents for this image. Each entry in the list is a link that takes you to the metadata document on the **View metadata** page.

At the top of the **View metadata** page, there is a navigation bar, which includes links to other photo album pages. From the **View metadata** page, you can return to the **View album** page, or select any of the **View entry**, **Write metadata**, **Search metadata**, or **Upload photo** pages. These pages are described in [Section 3.1.2.1](#), [Section 3.1.2.4](#), [Section 3.1.2.6](#), [Section 3.1.2.7](#), and [Section 3.1.2.2](#), respectively.

[Figure 3–5](#) shows a **View metadata** page that contains two types of metadata (XMP and ORDIMAGE) for an image.

**Figure 3–5 View metadata Page with Metadata for an Uploaded Image**



The PL/SQL procedures `view_metadata` and `print_metadata` are the primary application components that implement the **View metadata** page. The procedure

`view_metadata` is passed the argument `entry_id`, which uniquely identifies the image in the `photos` table. A `SELECT` statement retrieves all the `XMLType` metadata columns for the specified entry. If the metadata column is not `NULL`, procedure `print_metadata` is called to display the XML document inside an HTML `<pre>` tag.

[Example 3-8](#) contains some relevant lines of code in the `view_metadata` procedure.

#### **Example 3-8 Procedure `view_metadata`**

```
--
-- Fetch the row.
--
SELECT metaOrdImage, metaEXIF, metaIPTC, metaXMP
INTO   metaO, metaE, metaI, metaX
FROM   photos
WHERE  id = entry_id;

-- display the EXIF metadata
IF metaE IS NOT NULL THEN
  http.print( '<span class="bigBlue" id="exifMetadata">EXIF</span>' );
  http.print( '<br><pre>' );
  print_metadata( metaE );      http.print( '</pre>' );
END IF;
```

The `print_metadata` procedure accepts an `XMLType` document as an argument. It uses the `getClobVal()` method to access the document as a `CLOB`. The content of the `CLOB` is read in a loop and formatted in the HTML page using the `http.prints` procedure. The `http.prints` procedure escapes the `'<'` and `'>'` characters so that they are rendered properly by the Web browser.

[Example 3-9](#) contains some relevant lines of code in the `print_metadata` procedure.

#### **Example 3-9 Procedure `print_metadata`**

```
metaClob := meta.getClobVal();
len := dbms_lob.getLength( metaClob );
IF bufSize > len THEN
  bufSize := len;
END IF;
WHILE len > 0 LOOP
  dbms_lob.read( metaClob, bufSize, pos, buf );
  http.prints( buf );
  pos := pos + bufSize;
  len := len - bufSize;
END LOOP;
```

### **3.1.2.6 Writing New XMP Metadata to Images**

You can use the **Write XMP metadata** page to write new or replace existing XMP metadata in an image. Oracle Multimedia provides support for writing XMP metadata only. You can access the **Write XMP metadata** page by selecting the **Write metadata** link in the navigation bar from either the **View entry** page or the **View metadata** page.

The **Write XMP metadata** page displays the thumbnail version of the image to be modified. The page also displays an input form to collect metadata attributes in these five text entry fields:

- **Title:** Specify a title for the photograph.
- **Creator:** Enter the name of the person who took the photograph. This field is optional.

- **Date:** Enter the date the photograph was taken. This field is optional.
- **Description:** Enter a description, such as the subject of the photograph. This field is optional.
- **Copyright:** Enter the month and year when the photograph was taken. This field is optional.

Click **Write it!** to send the form to the application and embed the metadata in XMP format in the image.

At the top of the **Write XMP metadata** page, there is a navigation bar, which includes links to other photo album pages. From the **Write XMP metadata** page, you can return to the **View album** page, or select any of the **View entry**, **View metadata**, **Search metadata**, or **Upload photo** pages. These pages are described in [Section 3.1.2.1](#), [Section 3.1.2.4](#), [Section 3.1.2.5](#), [Section 3.1.2.7](#), and [Section 3.1.2.2](#), respectively.

Figure 3–6 shows a **Write XMP metadata** page with completed entries for an image.

**Figure 3–6 Completed Write XMP metadata Page with XMP Metadata for an Uploaded Image**



The PL/SQL procedure `write_metadata` receives the form input fields from the browser. The procedure creates an XML document (as a string buffer) that is valid to the Oracle Multimedia XMP schema `http://xmlns.oracle.com/ord/meta/xmp`. The string buffer is used to create an XMLType object.

A `SELECT FOR UPDATE` statement retrieves the image to be modified. The Oracle Multimedia method `putMetadata()` is called to embed the XML document into the

image. The modified image is stored back to the photos table. Finally, procedure `sync_indexes` is called to update the text indexes.

[Example 3-10](#) contains some relevant lines of code in the `write_metadata` procedure.

**Example 3-10 Procedure `write_metadata`**

```
-- Create the XMP packet it must be schema valid
-- to "http://xmlns.oracle.com/ord/meta/xmp"
-- and contain an <RDF> element. This example uses
-- the Dublin Core schema as implemented by Adobe XMP
buf := '<xmpMetadata xmlns="http://xmlns.oracle.com/ord/meta/xmp"
      xsi:schemaLocation="http://xmlns.oracle.com/ord/meta/xmp
      http://xmlns.oracle.com/ord/meta/xmp"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
      <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
      <rdf:Description about="" xmlns:dc="http://purl.org/dc/elements/1.1/">
      <dc:title>' || htf.escape_sc(title) || '</dc:title>';

IF c_creator IS NOT NULL THEN
  buf := buf || '<dc:creator>' || htf.escape_sc(c_creator)
          || '</dc:creator>';
END IF;
IF c_date IS NOT NULL THEN
  buf := buf || '<dc:date>' || htf.escape_sc(c_date)
          || '</dc:date>';
END IF;
IF c_description IS NOT NULL THEN
  buf := buf || '<dc:description>' || htf.escape_sc(c_description)
          || '</dc:description>';
END IF;
IF c_copyright IS NOT NULL THEN
  buf := buf || '<dc:copyright>' || htf.escape_sc(c_copyright)
          || '</dc:copyright>';
END IF;
buf := buf || '
      </rdf:Description>
      </rdf:RDF>
      </xmpMetadata>';

xmp := XMLType.createXML(buf, 'http://xmlns.oracle.com/ord/meta/xmp');

-- -- select image for update
-- description is selected to force update of CTX index
--
SELECT image, description
INTO img, des
FROM photos
WHERE id = entry_id
FOR UPDATE;

--
-- write the metadata
--
img.putMetadata( xmp, 'XMP' );

--
-- save updated image and new metadata to table
-- description updated to force update of CTX index
--
```

```

UPDATE photos
SET image = img,
    metaXMP = xmp,
    description = des
WHERE id = entry_id;

-- update the text indexes
sync_indexes;

```

The input data shown in [Example 3–10](#) would result in the storage of the following metadata in the image:

```

<xmpMetadata xmlns="http://xmlns.oracle.com/ord/meta/xmp"
  xsi:schemaLocation="http://xmlns.oracle.com/ord/meta/xmp
    http://xmlns.oracle.com/ord/meta/xmp"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <rdf:Description about="" xmlns:dc="http://purl.org/dc/elements/1.1/">
      <dc:title>Story time</dc:title>
      <dc:creator>father</dc:creator>
      <dc:date>July 4, 2001</dc:date>
      <dc:description>family reading</dc:description>
      <dc:copyright>mother</dc:copyright>
    </rdf:Description>
  </rdf:RDF>
</xmpMetadata>

```

### 3.1.2.7 Searching for Images That Contain Specific Metadata Attributes

You can use the **Search metadata** page to search a specific metadata type as well as to limit your search to a specific tag within a metadata document. You can access the **Search metadata** page by selecting the **Search metadata** link in the navigation bar of any photo album application Web page.

The **Search metadata** page displays a form with four fields to define how the search is to be performed. Use the pull-down menu in the **Search in metadata:** field to select the type of metadata (**EXIF**, **IPTC**, or **XMP**) to be searched. When this field is changed, the fields **Search in tag:** and **Search method:** are initialized with values that are appropriate to the type of metadata search.

Use the drop-down list in the **Search in tag:** field to limit the search to a specific XML element within a metadata document. The list is populated with element names that are appropriate for the selected metadata type. When the value **--Any tag--** is showing, the search will look at all elements within the document type. When the **XMP** metadata type is selected, searches are limited to Description elements within the parent RDF element. If the metadata document is properly constructed, selecting **RDF/Description** in this field should search all relevant metadata within XMP documents.

In the **Search method:** field, select the radio button **Contains** to specify a search where an element contains the search string. Select the radio button **Equals** to specify a search where element values are matched exactly to the search string. For searches in XMP metadata, only the **Contains** search method is available.

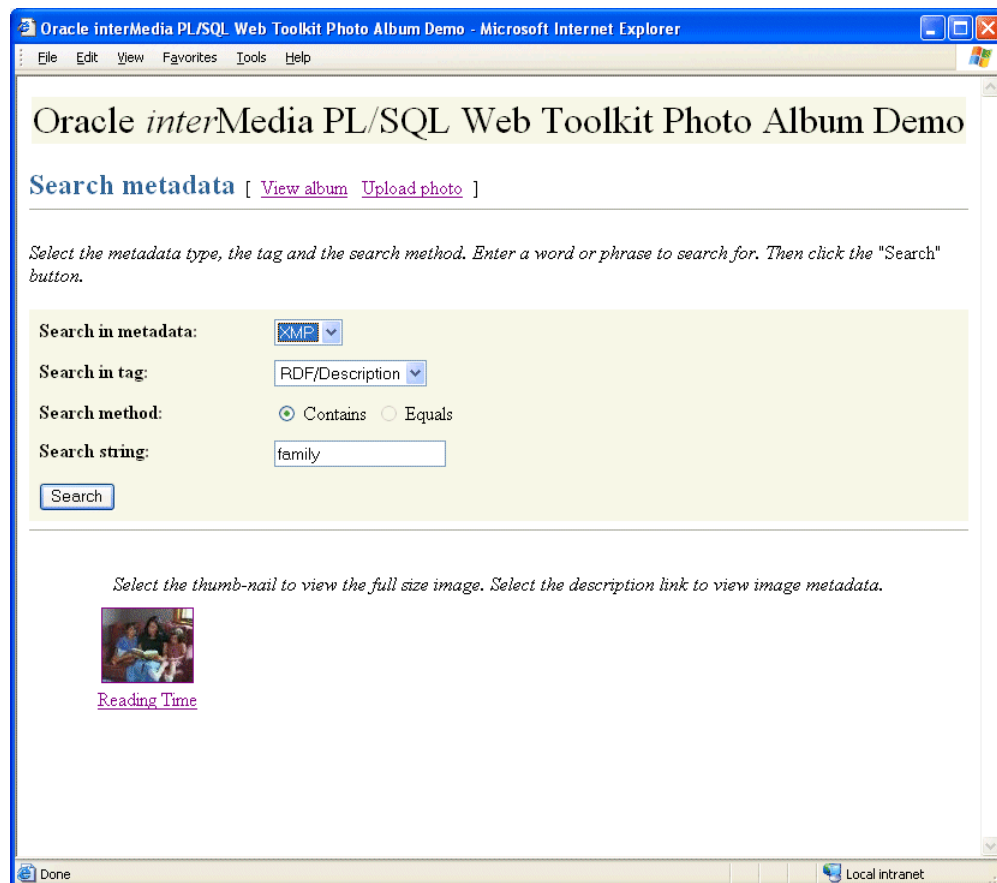
Finally, enter a keyword or phrase in the **Search string:** field and click **Search**. If the search is successful, the thumbnail versions of the matching images are displayed in a four-column table. Select the thumbnail image to view the full-size version of an image. Or, select the description link below the thumbnail image to view the metadata for the image. If the search fails, the message "No photos matched the search criteria." is displayed.



At the top of the **Search metadata** page, there is a navigation bar, which includes links to other photo album pages. From the **Search metadata** page, you can return to the **View album** page or select the **Upload photo** page. These pages are described in [Section 3.1.2.1](#) and [Section 3.1.2.2](#), respectively.

[Figure 3-7](#) shows a **Search metadata** page that contains sample search criteria and results from a successful search operation.

**Figure 3-7 Completed Search metadata Page for an Uploaded Image**



The PL/SQL procedure `search_metadata` receives the form input fields from the Web browser. The search parameters are used to build a query to find images that contain the desired metadata. The search is accomplished using the SQL `EXISTS` operator. The `EXISTS` operator is used to search an XML document for content that matches the given search predicate. The operator returns 1 if the document matched the search, and 0 otherwise. The operator takes three arguments. The first argument is the name of the XMLType column. In this application, the column to search is determined by the metadata type selected in the pull-down menu for the **Search in metadata:** field. The second argument is an XPATH expression that specifies the content to be searched and how the matching is evaluated. The **Search in tag:** and **Search method:** fields are used to build the XPATH expression. The third argument is a string to define the XML namespaces to be used to process the document and the XPATH expression.

For example, assume that the `search_metadata` procedure receives input that specifies to search the `caption` tag in IPTC metadata for an exact match of the word "farm". The query to accomplish this search is as follows:

```

SELECT id, description, thumb
FROM photos
WHERE EXISTSNODE( metaIptc,
                  '/iptcMetadata//caption="farm"',
                  'xmlns:ora="http://xmlns.oracle.com/xdb" ' ||
                  'xmlns="http://xmlns.oracle.com/ord/meta/iptc" ) = 1;

```

The second argument to the EXISTSNODE operator, `'/iptcMetadata//caption="farm"',` specifies a search for all `<caption>` elements under the root element `<iptcMetadata>` where the `<caption>` content is equal to `"farm"`.

For more information about the EXISTSNODE operator, see *Oracle XML DB Developer's Guide*. For more information about the contains() text search operator, see *Oracle Text Application Developer's Guide*.

[Example 3-11](#) contains some relevant lines of code in the `search_metadata` procedure.

#### **Example 3-11 Procedure `search_metadata`**

```

-- set up search variables for EXIF documents
IF mtype = 'exif' THEN
  IF op = 'equals' THEN
    xpath := '/exifMetadata// ' || tag || '=' || c_search || '';
  ELSE -- default to contains
    xpath := '/exifMetadata// ' || tag ||
             '[ora:contains(text(), "' || c_search || "')>0]';
  END IF;

  nspace := 'xmlns:ora="http://xmlns.oracle.com/xdb" ' ||
            'xmlns="http://xmlns.oracle.com/ord/meta/exif"';

  OPEN album_cur FOR
    SELECT id, description, thumb
    FROM photos
    WHERE existsnode( metaExif, xpath, nspace ) = 1;

-- set up search variables for IPTC documents
ELSIF mtype = 'iptc' THEN
  IF op = 'equals' THEN
    xpath := '/iptcMetadata// ' || tag || '=' || c_search || '';
  ELSE -- default to contains
    xpath := '/iptcMetadata// ' || tag ||
             '[ora:contains(text(), "' || c_search || "')>0]';
  END IF;

  nspace := 'xmlns:ora="http://xmlns.oracle.com/xdb" ' ||
            'xmlns="http://xmlns.oracle.com/ord/meta/iptc"';

  OPEN album_cur FOR
    SELECT id, description, thumb
    FROM photos
    WHERE existsnode( metaIptc, xpath, nspace ) = 1;

-- set up search variables for XMP documents
ELSIF mtype = 'xmp' THEN
  -- default to contains
  xpath := '/xmpMetadata/rdf:RDF/rdf:Description/*[ora:contains(text(), "'
           || c_search || "')>0]';

```

```

-- add rdf namespace prefix
namespace := 'xmlns:ora="http://xmlns.oracle.com/xdb" ' ||
             'xmlns="http://xmlns.oracle.com/ord/meta/xmp" ' ||
             'xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"';

OPEN album_cur FOR
  SELECT id, description, thumb
  FROM photos
  WHERE existsnode( metaXMP, xpath, namespace ) = 1;

ELSE
  errorMsg := 'Search domain is invalid: ' || htf.escape_sc(mtype);
END IF;

print_search_form( mtype, tag, op, c_search );
htp.print('<hr size="-1">');
print_album( album_cur, 'No photos matched the search criteria.' );

```

## 3.2 Oracle Multimedia Java Servlet Photo Album Sample Application

The Oracle Multimedia Java Servlet Photo Album sample application demonstrates the use of Oracle Multimedia Servlets and JSP Java API to upload and retrieve multimedia data to and from the database. Users access this photo album application to view the contents of the photo album, including thumbnail versions of each photograph, to view the full-size version of any photograph, and to upload new photographs into the album.

This photo album application demonstrates the use of the Oracle Multimedia image object type to upload and retrieve media data stored in Oracle Database.

When installed, this photo album application creates a table named `photos` and a sequence named `photos_sequence`.

The `photos` table is described by the following CREATE TABLE statement:

```

CREATE TABLE photos( id          NUMBER PRIMARY KEY,
                     description VARCHAR2(40) NOT NULL,
                     location   VARCHAR2(40),
                     image      ORDSYS.ORDIMAGE,
                     thumb      ORDSYS.ORDIMAGE )
--
-- store full-size images and thumbnail images as SecureFile LOBs
--
LOB(image.source.localdata) STORE AS SECUREFILE
LOB(thumb.source.localdata) STORE AS SECUREFILE;

```

Note that the data type for the `image` and `thumb` columns are defined as Oracle Multimedia image object types to store the full-size images and the generated thumbnail images.

The `photos_sequence` sequence is defined by the following CREATE SEQUENCE statement:

```
CREATE SEQUENCE photos_sequence;
```

The sample application files and `README.txt` file are located at:

<ORACLE\_HOME>/ord/http/demo/servlet (on Linux and UNIX)

<ORACLE\_HOME>\ord\http\demo\servlet (on Windows)

The following subsections describe how to run the Java Servlet Photo Album application. See the `README.txt` file for additional requirements and instructions on installing and using this sample application.

### 3.2.1 Running the Java Servlet Photo Album Application

After you have completed the setup tasks and have built the Java Servlet Photo Album application, you can run the application by entering the following URL in the address field of your Web browser:

- Default installation of Oracle Application Server or Oracle Database

```
<protocol><hostname:port-number>/servlet/PhotoAlbumServlet
```

- Default installation of Tomcat 3.2 on Windows

```
<protocol><hostname:port-number>/examples/servlet/PhotoAlbumServlet
```

The `<protocol>` field is `http://`, and the `<hostname:port-number>` field is the host name and port number of the system where your HTTP server is running.

When first invoked, this photo album application displays any images that are currently stored in the album. By default, the photo album is empty when first installed. To upload a new photograph, select **Upload new photo**. Enter a description of the photograph, the location where the photograph was taken, and the name of the image file (or browse to its directory location), then click **Upload photo**. The contents of the photo album are displayed along with a picture of the new photograph. Click the thumbnail image to view the full-size version of the photograph.

When this photo album application displays the text **view image** instead of its thumbnail image, the image format that was uploaded was not recognized by Oracle Multimedia. Click **view image** to display the full-size image.

You can now begin to load your photo album application with your favorite photographs.

### 3.2.2 Description of the Java Servlet Photo Album Application

The Java Servlet Photo Album application combines both business logic and the presentation into a single servlet, which when compiled, creates two class files, `PhotoAlbumServlet.class` and `PhotoAlbumRequest.class`.

To follow along with the description of tasks, users should refer to a copy of the `PhotoAlbumServlet.java` file, which can be found in:

```
<ORACLE_HOME>/ord/http/demo/servlet (on Linux and UNIX)
```

```
<ORACLE_HOME>\ord\http\demo\servlet (on Windows)
```

#### PhotoAlbumServlet Class

The `PhotoAlbumServlet` class performs the following tasks:

- Extends the `HttpServlet` and contains the user-entered connection information.

```
public class PhotoAlbumServlet extends HttpServlet
```

- Accepts connection information by allowing you to select the connection method, supply the necessary connection information, and optionally change the user name and password to connect to a schema other than `scott/tiger`.

```
private final static String JDBC_CONNECT_STRING =  
// "jdbc:oracle:oci:@<SQL*Net TNS name>"; // 9i JDBC OCI driver
```

```
// "jdbc:oracle:oci8:@<SQL*Net TNS name>"; // 8i JDBC OCI driver
    "jdbc:oracle:thin:@<host>:<port>:<sid>"; // JDBC Thin driver
```

```
private final static String JDBC_USER_NAME = "scott";
private final static String JDBC_PASSWORD = "tiger";
```

- Instantiates a Java stack used to implement a simple connection-pooling mechanism.

```
private static Stack connStack = new Stack();
```

- Defines a flag to indicate whether or not the JDBC Thin driver has been loaded.

```
private static boolean driverLoaded = false;
```

- Defines a servlet initialization method.

```
public void init( ServletConfig config ) throws ServletException
{
    super.init(config);
}
```

- Defines a doGet() method to process an HTTP GET request containing an HttpServletRequest object and HttpServletResponse object, and instantiates a PhotoAlbumRequest object to process the request to deliver either a full-size or thumbnail image to the browser, or to display an upload form or the contents of the photo album as thumbnail images.

```
public void doGet( HttpServletRequest request,
                  HttpServletResponse response )
    throws ServletException, IOException
{
    Connection conn = null;

    //
    // Use a try-block to ensure that JDBC connections are always returned
    // to the pool.
    //
    try
    {
        //
        // Get a JDBC connection from the pool.
        //
        conn = getConnection();

        //
        // Instantiate a PhotoAlbumRequest object to process the request.
        //
        PhotoAlbumRequest albumRequest =
            new PhotoAlbumRequest( conn, request, response );

        //
        // Figure out what to do based on query string parameters.
        //
        String view_media = request.getParameter( "view_media" );
        if ( view_media != null )
        {
            //
            // Deliver a full-sized or thumbnail image to the browser.
            //
            albumRequest.viewMedia( view_media );
        }
    }
}
```

```
        return;
    }
    else if ( request.getParameter( "view_form" ) != null )
    {
        //
        // Display the HTML upload form.
        //
        albumRequest.viewUploadForm();
    }
    else if ( request.getParameter( "view_entry" ) != null )
    {
        //
        // Display full-sized photo image.
        //
        albumRequest.viewPhoto();
    }
    else
    {
        //
        // Display album contents with thumbnail images by default.
        //
        albumRequest.viewAlbum();
    }
}
catch ( SQLException e )
{
    //
    // Log what went wrong.
    //
    e.printStackTrace( System.out );

    //
    // Turn SQL exceptions into ServletExceptions.
    //
    throw new ServletException( e.toString() );
}
finally
{
    //
    // If we have a JDBC connection, then return it to the pool.
    //
    freeConnection( conn );
}
}
```

- Defines a `doPost()` method to process an HTTP POST request used to upload a new photograph into the album by instantiating a `PhotoAlbumRequest` object to process the request and then calling the `insertNewPhoto()` method.

```
public void doPost( HttpServletRequest request,
                   HttpServletResponse response )
    throws ServletException, IOException
{
    Connection conn = null;

    //
    // Use a try-block to ensure that JDBC connections are always returned
    // to the pool.
    //
    try
```

```

{
    //
    // Get a JDBC connection from the pool.
    //
    conn = getConnection();

    //
    // Instantiate a PhotoAlbumRequest object to process the request.
    //
    PhotoAlbumRequest albumRequest =
        new PhotoAlbumRequest( conn, request, response );

    //
    // Insert the photo into the album.
    //
    albumRequest.insertNewPhoto();
}
catch ( SQLException e )
{
    //
    // Log what went wrong.
    //
    e.printStackTrace( System.out );

    //
    // Turn SQL exceptions into ServletExceptions.
    //
    throw new ServletException( e.toString() );
}
finally
{
    //
    // If we have a JDBC connection, then return it to the pool.
    //
    freeConnection( conn );
}
}

```

- Defines a `getConnection()` method.

```

private Connection getConnection()
    throws SQLException
{
    OracleConnection conn = null;

    //
    // Synchronize on the stack object. Load the JDBC driver if not yet
    // done. If there's a free connection on the stack, then pop it off
    // the stack and return it to the caller. Otherwise, create a new
    // connection object and call the version compatibility initialization
    // method.
    //
    synchronized( connStack )
    {
        if ( !driverLoaded )
        {
            DriverManager.registerDriver(
                new oracle.jdbc.driver.OracleDriver() );
            driverLoaded = true;
        }
        if ( connStack.empty() )
    }
}

```

```
        {
            conn = (OracleConnection)DriverManager.getConnection
                ( JDBC_CONNECT_STRING, JDBC_USER_NAME, JDBC_PASSWORD );
        }
        else
        {
            conn = (OracleConnection)connStack.pop();
        }
    }

    //
    // Enable auto-commit by default.
    //
    conn.setAutoCommit( true );

    return conn;
}
```

- Defines a `freeConnection()` method.

```
private void freeConnection( Connection conn )
{
    //
    // Synchronize on the stack object, then push the connection onto the
    // stack.
    //
    if ( conn != null )
    {
        synchronized( connStack )
        {
            connStack.push( conn );
        }
    }
}
```

In summary, the `PhotoAlbumServlet` class responds to the HTTP GET and POST requests by allocating a JDBC connection from a connection pool. Each HTTP GET or POST request is assigned its own JDBC connection from the pool to ensure that multiple requests can be serviced concurrently. An HTTP GET request is used to retrieve image data from the photo album and an HTTP POST request is used to upload image data into the photo album. Then, an instance of the `PhotoAlbumRequest` class is created to execute the request, executes the request, then releases the JDBC connection back to the pool at the end of the request.

### **PhotoAlbumRequest Class**

The `PhotoAlbumRequest` class does the actual processing of an HTTP GET or POST request, and defines the `getPreferredFormat()` function and the following methods:

```
viewAlbum()
viewPhoto()
viewMedia()
viewUploadForm()
insertNewPhoto()
printPageHeader()
printPageTrailer()
printMessage()
printHeading()
printLink()
```



In the `viewMedia()` and `insertNewPhoto()` methods, three objects, `OrdHttpResponseHandler`, `OrdHttpUploadFormData`, and `OrdHttpUploadFile`, are instantiated. These objects are used to call the methods of the respective `OrdHttpResponseHandler`, `OrdHttpUploadFormData`, `OrdHttpUploadFile` classes of Oracle Multimedia Servlets and JSP Java API. For example, in the `viewMedia()` method, the `OrdHttpResponseHandler` object is instantiated and used to call the `sendImage()` method as shown in the following code:

```
OrdHttpResponseHandler handler =
    new OrdHttpResponseHandler( request, response );
handler.sendImage( img );
```

The `viewAlbum()`, `viewPhoto()`, `viewMedia()`, and `insertNewPhoto()` methods use the `ORADData` (formerly `getCustomDatum`) and `ORADDataFactory` (formerly `getFactory`) interfaces supplied by Oracle to get the image or thumbnail `OrdImage` object from the result set to obtain height and width information, to retrieve an image from an `OrdImage` Java object and deliver it to the browser, and to upload an image in an `OrdImage` Java object and to also update it in the `photos` table. For example, the following code excerpt is from the `viewAlbum()` method:

```
OrdImage img =
    (OrdImage)rset.getORADData( 4, OrdImage.getORADDataFactory() );
.
.
.
out.print( "<td headers=\"image\"><a href=\"" + servletUri +
    "?view_entry=yes&id=" + id + "\">" );
if ( img.getContentLength() > 0 )
{
    if (img.getMimeType().startsWith("image/"))
    {
out.print( "<img src=\"" + servletUri +
    "?view_media=thumb&id=" + id + "\" +
    " height=" + img.getHeight() +
    " width=" + img.getWidth() +
    " alt=\"" + description + "\" +
    " border=1>" );
    }
}
else
{
    out.print( "[view image]" );
}
out.println( "</a></td>" );
out.println( "</tr>" );
```

What follows is a more detailed description of each method and what it does:

- The `viewAlbum()` method does the following:
  - Initializes the row count to zero.
  - Writes a common page header on the HTML page using the function `printPageHeader()`.
  - Executes a `SELECT` statement to fetch all the thumbnail images in the photo album, order them by description, and display the description and location information for each along with the thumbnail image if it exists, and returns the results in a result set.

- Displays the thumbnail images in an HTML table with column headers labeled `Description`, `Location`, and `Image`.
- Within a `while` block, reads the contents of the result set by reading the first row's contents beginning with the `id` value, displays the description and location values, then gets the thumbnail `OrdImage` object and builds the height and width attributes for each thumbnail image.
- Displays the thumbnail image using an HTML anchor tag that can be used to display the full-size image. When a user clicks the thumbnail image or **view image**, the full-size image is displayed.
- Displays the contents of the photo album within an HTML anchor tag using the tag `<IMG SRC="<servlet-path>?view_media=thumb&id=... ">` to display the thumbnail images, where `<servlet-path>` is the value of `servletUri`. If the thumbnail image was not created because the image format was not supported by Oracle Multimedia, the text **view image** is displayed instead.
- Increments the row count to see if the photo album is empty; if so, it displays the message "The photo album is empty".
- Displays an HTML anchor tag near the bottom of the HTML page using the `printLink()` function with the text **Upload new photo**.
- Writes a common trailer at the bottom of the HTML page by calling the `printPageHeader()` function, however, in this case, sets the Boolean argument to false to not display the common page trailer.
- Closes the result set and the statement.
- The `viewPhoto()` method displays the full-size version of a photograph and does the following:
  - Writes a common page header on the HTML page using the function `printPageHeader()`.
  - Gets the value of the `id` column for the entry being viewed.
  - Executes a SQL `SELECT` statement to fetch the entry's description, location, and full-size image where the value of `id` in the `where` clause is a parameter marker and returns the results in a result set.
  - Gets the image `OrdImage` object from the result set in order to later build the image height and width attributes within the `<IMG SRC=... >` image tag.
  - Displays the full-size image in an HTML table beginning with the column names `Description` and `Location`, and displays the entry's values for these two columns.
  - Builds the URL to fetch a full-size image for this entry by using an image tag `<IMG SRC="<servlet-path>?view_media=image&id=... ">` to display an image in the column labeled `Photo`, where `<servlet-path>` is the value of `servletUri`.
  - Displays the full-size images height and width by calling the `getHeight()` and `getWidth()` Oracle Multimedia object methods. If the image format is not recognized by Oracle Multimedia, height and width values will be zero and will not be displayed.
  - Writes a common page trailer at the bottom of the HTML page by calling the `printPageHeader()` function and setting its Boolean argument to true to display the common page trailer.

- Closes the result set and the statement.
- The `viewMedia()` method is invoked by both thumbnail and full-size image URLs to retrieve a thumbnail or full-size image from the `photos` table and deliver it to the browser using the `OrdHttpResponseHandler` class. This method does the following:
  - Executes a SQL `SELECT` statement to fetch either the thumbnail or full-size image where the value of `id` in the `where` clause is a parameter marker and returns the results in a result set. The SQL `SELECT` statement is built dynamically with the string `media` equating to either the thumbnail image column or the full-size image column.
  - Fetches a row from the result set.
  - Gets the `OrdImage` object from the result set.
  - Uses the `OrdHttpResponseHandler` class to create an `OrdHttpResponseHandler` object to retrieve the image from the `OrdImage` object and deliver it to the browser using the `sendImage()` method. The `sendImage()` method supports browser content caching by supporting the `If-Modified-Since` and `Last-Modified` headers.
  - Closes the result set and the statement.
- The `viewUploadForm()` method displays an HTML form that allows users to upload new photographs and does the following:
  - Calls the `printPageHeader()` function to produce the common page header.
  - Defines the form action as a `multipart/form-data` POST request.
  - Calls the `upload_form_fields` static string containing the contents of the upload form. The upload form is defined as a table with rows labeled `Description` and `Location`, with an input type of `text` and named `description` and `location` respectively, followed by a row labeled `File name:`, with an input type of `file` and named `photo`, and finally a row with no label, an input type of `submit`, and a value of `Upload photo`.
  - Calls the `printPageTrailer()` function to produce the common page trailer.
- The `insertNewPhoto()` method does the following:
  - Uses the `OrdHttpUploadFormData` class to parse a `multipart/form-data` POST request containing an uploaded photograph.
  - Uses the `OrdHttpUploadFile` class to upload the new photograph into the database.
  - Executes a SQL `SELECT photos_sequence.nextval` statement to get the next value of the `id` column for the new row to be inserted into the `photos` table.
  - Executes a SQL `INSERT` statement to insert a new row in the `photos` table.
  - Executes a SQL `SELECT...FOR UPDATE` statement to fetch the initialized full-size and thumbnail image objects from the `photos` table.
  - Calls the `loadImage()` method in the `OrdHttpUploadFile` class to populate an `OrdImage` object named `image` with the full-size image and sets the properties or attribute values of the image object based on the image contents.
  - Checks to see what the image format is and if it is an image format that may not be displayed inline by a browser, such as a BMP image format, then calls the `getPreferredFormat()` method to convert a BMP image format and return the preferred image format.

- Calls the `ProcessCopy()` method in the `OrdImage` class to process the full-size image, create a thumbnail image, and populate an `OrdImage` object named `thumb`.
- Executes a SQL `UPDATE` statement to update the full-size and thumbnail images in the database.
- Displays a photo upload success message and then directs the browser to refresh the page.
- A `getPreferredFormat()` private function, in this sample application, converts a BMP image format and returns the preferred image file format based on the number of colors in the image; returns a `MONOCHROME` image format if there are no colors, or a `JPEG` if there are more than 8 colors, or a `GIF` if there are greater than 0 and fewer than 8 colors.
- A `printPageHeader()` private function displays an HTML header that is common to all HTML responses.
- A `printPageTrailer()` private function displays an HTML trailer that is common to all HTML responses.
- A `printMessage()` private function prints a message on the HTML page.
- A `printHeading()` private function prints a header on the HTML page.
- A `printLink()` function produces an HTML anchor tag in a standard format.

### 3.3 Oracle Multimedia JSP Photo Album Sample Application

The Oracle Multimedia JSP Photo Album sample application is a JavaServer Pages (JSP) application that demonstrates the use of Oracle Multimedia Servlets and JSP Java API to upload and retrieve multimedia data to and from a database. Users access the JSP files that constitute the application to view the contents of the photo album, including thumbnail versions of each photograph, to view the full-size version of any photograph, and to upload new photographs into the album.

This photo album application demonstrates the use of the Oracle Multimedia image object type to upload and retrieve media data stored in Oracle Database.

This photo album application, when installed, creates a table named `photos` and a sequence named `photos_sequence`.

The `photos` table is described by the following `CREATE TABLE` statement:

```
CREATE TABLE photos( id          NUMBER PRIMARY KEY,
                    description VARCHAR2(40) NOT NULL,
                    location    VARCHAR2(40),
                    image       ORDSYS.ORDIMAGE,
                    thumb       ORDSYS.ORDIMAGE )
--
-- store full-size images and thumbnail images as SecureFile LOBs
--
LOB(image.source.localdata) STORE AS SECUREFILE
LOB(thumb.source.localdata) STORE AS SECUREFILE;
```

Note that the data type for the `image` and `thumb` columns are defined as Oracle Multimedia image object types to store the full-size images and the generated thumbnail images.

The `photos_sequence` sequence is defined by the following `CREATE SEQUENCE` statement:

```
CREATE SEQUENCE photos_sequence;
```

The sample application files and README.txt file are located at:

```
<ORACLE_HOME>/ord/http/demo/jsp (on Linux and UNIX)
```

```
<ORACLE_HOME>\ord\http\demo\jsp (on Windows)
```

The following subsections describe how to run the JSP Photo Album application. See the README.txt file for additional requirements and instructions on installing and using this sample application.

### 3.3.1 Running the JSP Photo Album Application

After you have completed the setup tasks and have built the JSP Photo Album application, you can run the application by entering the following URL in the address field of your Web browser:

- Default installation of Oracle Application Server or Oracle Database

```
<protocol><hostname:port-number>/demo/PhotoAlbum.jsp
```

- Default installation of Tomcat 3.2 on Windows

```
<protocol><hostname:port-number>/examples/jsp/PhotoAlbum.jsp
```

The *<protocol>* field is `http://`, and the *<hostname:port-number>* field is the host name and port number of the system where your HTTP server is running.

When first invoked, this photo album application displays any images that are currently stored in the album. By default, the photo album is empty when first installed. To upload a new photograph, select **Upload new photo**. Enter a description of the photograph, the location where the photograph was taken, and the name of the image file or browse to its directory location, then click **Upload photo**. The contents of the photo album are displayed along with a picture of the new photograph. Click the thumbnail image to view the full-size version of the photograph.

When this photo album application displays the text **view image** instead of its thumbnail image, the image format that was uploaded was not recognized by Oracle Multimedia. Click **view image** to display the full-size image.

You can now begin to load your photo album application with your favorite photographs.

### 3.3.2 Description of the JSP Photo Album Application

The JSP Photo Album application separates the business logic from the presentation by having a JavaBean containing methods that are accessed from each of five JSP files. When compiled, the application creates the `PhotoAlbumBean.class` file, which contains the user-entered connection information and defines the functions: `getId()`, `getDescription()`, `getLocation()`, and `getPreferredFormat()` and the following methods:

```
selectTable()
selectRowById()
fetch()
insertNewPhoto()
release()
getConnection()
freeConnection()
setId()
```

```
setDescription()  
setLocation()  
getImage()  
getThumb()
```

To follow along with the description of tasks, users should refer to a copy of each JSP file, which can be found in:

```
<ORACLE_HOME>/ord/http/demo/jsp (on Linux and UNIX)
```

```
<ORACLE_HOME>\ord\http\demo\jsp (on Windows)
```

In the `PhotoAlbumEntryViewer`, `PhotoAlbumMediaViewer`, `PhotoAlbum`, and `PhotoAlbumInsertPhoto` JSP files, the `jsp:useBean` action tag is used to establish an ID and association with the `PhotoAlbumBean` class and the `OrdHttpJspResponseHandler` and `OrdHttpUploadFormData` classes of Oracle Multimedia Servlets and JSP Java API. For example, the following code appears in the `PhotoAlbumInsertPhoto` JSP file:

```
<jsp:useBean id="album" scope="page" class="PhotoAlbumBean"/>  
<jsp:useBean id="handler" scope="page"  
            class="oracle.ord.im.OrdHttpJspResponseHandler"/>  
<jsp:useBean id="formData" scope="page"  
            class="oracle.ord.im.OrdHttpUploadFormData"/>
```

This `jsp:useBean` action tag is used so these objects can be referenced by their respective ID values (`album`, `handler`, and `formData`) to call the methods of these classes.

The `OrdHttpUploadFile` class of Oracle Multimedia Servlets and JSP Java API is defined as an object with the name `uploadPhoto` in the `insertNewPhoto()` method in the `PhotoAlbumBean.java` file and then used to call its `loadImage()` method to load the photograph into the `photos` table as shown in the following code excerpts:

```
public void insertNewPhoto( OrdHttpUploadFile uploadPhoto )  
    throws SQLException, ServletException, IOException  
{  
    .  
    .  
    .  
    uploadPhoto.loadImage( image );  
    .  
    .  
    .  
}
```

The `insertNewPhoto()` method defined in the `PhotoAlbumBean.java` file, uses the `ORADData` (formerly `getCustomDatum`) and `ORADDataFactory` (formerly `getFactory`) interfaces supplied by Oracle to upload an image and a thumbnail image in an `OrdImage` Java object. First, the method executes a SQL `SELECT...FOR UPDATE` statement to select the row for update, and then, executes a SQL `UPDATE` statement to update the `image` and `thumb` columns for that row in the `photos` table as shown in the following code excerpts:

```
stmt = (OraclePreparedStatement)conn.prepareStatement(  
    "select image,thumb from photos where id = ? for update" );  
stmt.setString( 1, id );  
rset = (OracleResultSet)stmt.executeQuery();  
if ( !rset.next() )  
{  
    throw new ServletException( "new row not found in table" );  
}  
image = (OrdImage)rset.getORADData( 1, OrdImage.getORADDataFactory());
```

```

thumb = (OrdImage)rset.getORADData( 2, OrdImage.getORADDataFactory());

rset.close();
stmt.close();
.
.
.
//
// Prepare and execute a SQL statement to update the full-sized and
// thumbnail images in the database.
//
stmt = (OraclePreparedStatement)conn.prepareStatement(
    "update photos set image = ?, thumb = ? where id = ?" );
stmt.setORADData( 1, image );
stmt.setORADData( 2, thumb );
stmt.setString( 3, id );
stmt.execute();
stmt.close();

//
// Commit the changes.
//
conn.commit();
}

```

The `fetch()` method defined in the `PhotoAlbumBean.java` file or the `PhotoAlbumBean` JavaBean, fetches the next row from the result set using the `ORADData` and `ORADDataFactory` interfaces to retrieve the image and the thumbnail image from an `OrdImage` Java object, and delivers each to the browser, as shown in the following example:

```

public boolean fetch()
    throws SQLException
{
    if ( rset.next() )
    {
        id = rset.getString( 1 );
        description = rset.getString( 2 );
        location = rset.getString( 3 );
        image = (OrdImage)rset.getORADData( 4, OrdImage.getORADDataFactory() );
        thumb = (OrdImage)rset.getORADData( 5, OrdImage.getORADDataFactory() );
        return true;
    }
    else
    {
        rset.close();
        stmt.close();
        return false;
    }
}
}

```

What follows is a more detailed description of each JSP file.

### PhotoAlbum.jsp

This JSP file is the entry point to the JSP Photo Album application and does the following:

- Uses the `PhotoAlbumBean` JavaBean to access the contents of the `photos` table.

- Uses the `OrdHttpJspResponseHandler` class to facilitate the retrieval of image data from the `photos` table and its delivery to a browser or other HTTP client from a Java servlet.
- Displays the title of the page in the HTML header and in the common page header.
- Displays the thumbnail images in a table using column headers labeled, `Description`, `Location`, and `Image`.
- Uses a `try/catch` block to ensure the JDBC connection is released.
- Calls the `selectTable()` method to select all the rows in the `photos` table.
- Initializes the row count to zero.
- Displays an entry in the photo album by calling the `getDescription()` method, then the `getLocation()` method, and then printing the values in the appropriate columns. If the location information is blank, print a space in the `Location` column.
- Displays the contents of the photo album as thumbnail images using an HTML anchor tag to call the `PhotoAlbumEntryViewer.jsp` file to get the ID value by calling the `getID()` function.
- Calls the `getThumb()` method to get the thumbnail image and calls the `getContentLength()` method to determine the image length.
- Tests to see if the value returned for the image length is greater than 0, and if so uses an image tag of the form `<IMG SRC="PhotoAlbumMediaViewer.jsp?media=thumb&...">` to display the thumbnail image; otherwise, prints the link **view image** in the column header labeled `Image`, which, when clicked, retrieves the full-size image.
- Displays a message "The photo album is empty" if the photo album is empty. If the photo album is not empty, the following message is displayed "Select the thumbnail to view the full-sized image".
- Ends the `try/catch` block with a `finally` clause and releases the JDBC connection by calling the `release()` method.
- Displays a link to the upload form with the text **Upload new photo** at the bottom of the page that calls the `PhotoAlbumUploadForm.jsp` file.

### **PhotoAlbumEntryViewer.jsp**

This JSP file is called by the `PhotoAlbum.jsp` file that displays one full-size version of a photograph in the album. This JSP file does the following:

- Uses the `PhotoAlbumBean` JavaBean to access the contents of the `photos` table.
- Uses the `OrdHttpJspResponseHandler` class to facilitate the retrieval of image data from the `photos` table and its delivery to a browser or other HTTP client from a Java servlet.
- Displays the title of the page in the HTML header and in the common page header.
- Defines a string named `id` that calls the `getParameter()` method to get the `id` value.
- Displays a message "Malformed URL, no id parameter" in the event the value of the `id` string is null.
- Uses a `try/catch` block to ensure the JDBC connection is released.



- Calls the `selectRowById()` method with the value of `id` to select the entry to be displayed. If the next row to be fetched for that `id` value is not found, display a message "Entry not found: <id value>".
- Displays the entry in the album by calling the `getDescription()` method and displaying its value under the header `Description`, calling the `getLocation()` method and displaying its value under the `Location` header.
- Displays one full-size version of a photograph in the album using an image tag in the form `<IMG SRC="PhotoAlbumMediaViewer.jsp?media=image&...">` under the `Photo` header.
- Displays the full-size images height and width by calling the `getHeight()` and `getWidth()` methods. If the image format is not recognized by Oracle Multimedia, height and width values will be zero and will not be displayed.
- Displays a link at the bottom of the page **Return to photo album** that calls the `PhotoAlbum.jsp` file.
- Ends the `try/catch` block, and with a `finally` clause, releases the JDBC connection by calling the `release()` method.

### **PhotoAlbumMediaViewer.jsp**

This JSP file is called by the `PhotoAlbum.jsp` and `PhotoAlbumEntryViewer.jsp` files and retrieves a single thumbnail or full-size image from the `photos` table using the `PhotoAlbumBean` JavaBean and delivers it to the browser using the `OrdHttpResponseHandler` class. This JSP file does the following:

- Uses the `PhotoAlbumBean` JavaBean to access the contents of the `photos` table.
- Uses the `OrdHttpJspResponseHandler` class to facilitate the retrieval of image data from the `photos` table and its delivery to a browser or other HTTP client from a Java servlet.
- Defines a string named `id` that calls the `getParameter()` method to get the `id` value.
- Defines a string named `media` that calls the `getParameter()` method to get the `media` value.
- Sets a condition to proceed as long as the value of the string `id` and the value of the string `media` is not null.
- Uses a `try/catch` block to ensure the JDBC connection is released.
- Calls the `selectRowById()` method to select a specific row from the `photos` table for the value of `id`.
- Delivers the full-size or thumbnail image by first calling the `setPageContext()` method of the `OrdHttpJspResponseHandler` class to specify the page context object; then, calling the `getImage()` method to return the image to the `OrdImage` object; then, calling the `sendImage()` method of the `OrdHttpResponseHandler` class to retrieve the image from the `OrdImage` object and deliver it to the browser. If the value of `media` is `image`, an image is delivered to the browser; if the value of `media` is `thumb`, a thumbnail image is delivered to the browser. The `sendImage()` method supports browser content caching by supporting the `If-Modified-Since` and `Last-Modified` headers.
- Ends the `try/catch` block with a `finally` clause and releases the JDBC connection by calling the `release()` method.

- Displays the following message in the event the request is not understood "PhotoAlbumMediaViewer.jsp - malformed URL".

### **PhotoAlbumUploadForm.jsp**

This JSP file is called by the `PhotoAlbum.jsp` file that displays an HTML form to allow users to upload new photographs into the album. This JSP file does the following:

- Displays the title of the page in the HTML header and in its common page header.
- Displays any error message under the header "Error message" from a previous attempt to upload an image to determine whether or not the value of a string is not null after calling the `getParameter()` method with an argument of `error`.
- Displays a header with the text **Upload a new photo**.
- Defines the form action specifying the `PhotoAlbumInsertPhoto.jsp` file to process the upload request as a multipart/form-data POST request.
- Displays the upload form with rows labeled `Description`, `Location`, and `File name`:
- Displays the contents of the upload form defined as a table with rows labeled `Description` and `Location`, both with an input type of `text` and named `description` and `location` respectively, followed by a row labeled `File name`: with an input type of `file` and named `photo`, and finally followed by a row with no label and an input type of `submit` and a value of `Upload photo`.
- Displays a link at the bottom of the page **Return to photo album** that calls the `PhotoAlbum.jsp` file.

### **PhotoAlbumInsertPhoto.jsp**

This JSP file is called by the `PhotoAlbumUploadForm.jsp` file that uses the `OrdHttpUploadFormData` class to parse the POST data in a POST request containing the uploaded photograph. This JSP file does the following:

- Uses the `PhotoAlbumBean` JavaBean to access the contents of the `photos` table.
- Uses the `OrdHttpJspResponseHandler` class to facilitate the retrieval of image data from the `photos` table and its delivery to a browser or other HTTP client from a JSP file.
- Uses the `OrdHttpUploadFormData` class to facilitate the processing of POST requests by parsing the POST data containing the multipart/form-data encoding, and making the contents of regular form fields and uploaded files readily accessible to a JSP file.
- Sets the value of the strings `description` and `location` to `null` and the `OrdHttpUploadFile` object `uploadPhoto` to `null`.
- Uses a `try/catch` block to ensure the JDBC connection is released.
- Passes an `OrdHttpUploadFile` object to the `PhotoAlbumBean` class to store the photograph in the database.
- Calls the `setServletRequest()` method of the `OrdHttpUploadFormData` class to specify the `ServletRequest` object for the request.
- Tests to see if the request is encoded using the multipart/form-data encoding by calling the `isUploadRequest()` method of the `OrdHttpUploadFormData` class.
- Forwards the request to the `PhotoAlbumUploadForm.jsp` file if the call to the `isUploadRequest()` method returns a Boolean expression of not false.

- Parses the form data by calling the `parseFormData()` method of the `OrdHttpUploadFormData` class.
- Gets the form field values for description and location by calling the `getParameter()` method of the `OrdHttpUploadFormData` class, and also gets the name of the file to be uploaded by calling the `getFileParameter()` method of the same class.
- Tests to make sure the file name is not null from the `getFileParameter()` method call of the `OrdHttpUploadFormData` class, then calls the `getOriginalFileName()` method of the `OrdHttpUploadFile` class to ensure that the original file name as provided by the browser is not null, or that the content length of the file is empty by calling the `getContentLength()` method of the `OrdHttpUploadFile` class.
- Forwards the request to the `PhotoAlbumUploadForm.jsp` file if there is a valid image file.
- If the description is null or empty, uses the file name as the description by calling the `getSimpleFileName()` method of the `OrdHttpUploadFile` class.
- Inserts the new entry into the `photos` table by calling the `setDescription()`, `setLocation()`, and `insertNewPhoto()` methods in the `PhotoAlbumBean.java` `JavaBean`.
- Ends the `try/catch` block with a `finally` clause and releases the JDBC connection by calling the `release()` method and releases all resources held by the `OrdHttpUploadFormData` object by calling its `release()` method.
- Displays the updated photo album by displaying the title of the page in the HTML header and in its common page header, directing the browser to the main page by calling the `PhotoAlbum.jsp` file, then displays the header "Photo successfully uploaded into photo album" and the instruction, "Please click on link below or wait for the browser to refresh the page".
- Displays a link at the bottom of the main page **Return to photo album** that calls the `PhotoAlbum.jsp` file.

### **PhotoAlbumBean.java**

This is a `JavaBean` used by the JSP files to access the database.

The first call to the `JavaBean` for a request causes it to allocate a JDBC connection from a connection pool. Subsequent calls by the same request reuse the same connection. At the end of a request, each JSP file is responsible for calling the `JavaBean` to release the JDBC connection back to the pool. Each HTTP GET or POST request is assigned its own JDBC connection from the pool to ensure that multiple requests can be serviced concurrently.

The following methods are defined:

- The `selectTable()` method selects all the rows in the `photos` table, orders them by location, and returns the results in a result set.
- The `selectRowById()` method selects a specific row from the `photos` table where the value of `id` in the `where` clause is a parameter marker and returns the results in a result set.
- The `fetch()` method fetches the next row from the result set.
- The `insertNewPhoto()` method does the following:
  - Uses the `OrdHttpUploadFile` class to upload the new photograph into the database.

- Disables auto-commit by calling the `setAutoCommit()` method with an argument of `false`.
- Executes a SQL `SELECT photos_sequence.nextval` statement to get the next value for the value of the `id` column for the new row to be inserted into the `photos` table.
- Executes a SQL `INSERT` statement to insert a new row in the `photos` table.
- Executes a SQL `SELECT...FOR UPDATE` statement to fetch the initialized full-size and thumbnail image objects from the `photos` table.
- Loads the image by calling the `loadImage()` method in the `OrdHttpUploadFile` class to populate an `OrdImage` object named `image` with the full-size image, and sets the properties or attribute values of the image object based on the image contents.
- Gets the image file format by calling the `getContentTypeFormat()` method and if it is not null, and if the MIME type is BMP, then tries to process the image by calling the `process()` method and calling the `getPreferredFormat()` method to convert it to a MONOCHROME, GIF, or JPEG image format, based on the number of colors in the image.
- Tries to copy the full-size image and process it to create the thumbnail image by calling the `processCopy()` method in the `OrdImage` class and populate the `OrdImage` object named `thumb`.
- Executes a SQL `UPDATE` statement to update the full-size and thumbnail images in the database.
- Commits the changes.
- A `release()` method to release the result set and statement objects, and places the JDBC connection back on the free list or stack.
- Get methods (`getId()`, `getDescription()`, `getLocation()`, `getImage()`, and `getThumb()`) and the set methods (`setId()`, `setDescription()`, and `setLocation()`) are used to get or set attributes for all attributes or columns.
- A `getConnection()` private function implements a simple JDBC connection pool.
- A `freeConnection()` private function releases the JDBC connection back to the pool at the end of the request.
- A `getPreferredFormat()` private function returns the preferred image file format based on the number of bits of color in the BMP image; returns a MONOCHROME image if there are no bits of color, returns JPEG if there are more than 8 bits of color, or returns GIF if there are between 1 and 8 bits of color.

## 3.4 Oracle Multimedia ASP/VBScript Photo Album Sample Application

The Oracle Multimedia ASP/VBScript Photo Album sample application is an ASP/VBScript application that demonstrates how to upload, retrieve, and process multimedia data stored using the Oracle Multimedia `ORDImage` type and Oracle Objects for OLE. Users access the application to view the contents of the photo album, including thumbnail versions of each photograph, to view the full-size version of any photograph, and to upload new photographs into the album.

This photo album application demonstrates the use of the Oracle Multimedia image object type to upload and retrieve media data stored in Oracle Database.

When installed, this photo album application creates a table named `photos` and a sequence named `photos_sequence`.

The `photos` table is described by the following CREATE TABLE statement:

```
CREATE TABLE photos( id          NUMBER PRIMARY KEY,
                    description VARCHAR2(40) NOT NULL,
                    location   VARCHAR2(40),
                    image      ORDSYS.ORDIMAGE,
                    thumb      ORDSYS.ORDIMAGE )
--
-- store full-size images and thumbnail images as SecureFile LOBs
--
LOB(image.source.localdata) STORE AS SECUREFILE
LOB(thumb.source.localdata) STORE AS SECUREFILE;
```

Note that the data type for the `image` and `thumb` columns are defined as Oracle Multimedia image object types to store the full-size images and the generated thumbnail images.

The `photos_sequence` sequence is defined by the following CREATE SEQUENCE statement:

```
CREATE SEQUENCE photos_sequence;
```

The sample application files and `README.txt` file are located at:

```
<ORACLE_HOME>/ord/http/demo/asp (on Linux and UNIX)
```

```
<ORACLE_HOME>\ord\http\demo\asp (on Windows)
```

The following subsections describe how to run the ASP/VBScript Photo Album application. See the `README.txt` file for additional requirements and instructions on installing and using this sample application.

### 3.4.1 Running the ASP/VBScript Photo Album Application

After you have installed and configured the ASP/VBScript Photo Album application in Microsoft IIS and configured the application connection parameters, you are ready to run the application.

To use it, enter the photo album URL into the location bar of your Web browser, for example:

```
http://<hostname:port>/photoAlbum
```

When first invoked, the application displays any images that are currently stored in the album. By default, the photo album is empty when first installed. To upload a new photograph, click **Upload new photo**. Enter a description of the photograph, the location where the photograph was taken, and the name of the image file or browse to its directory location, then click **Upload new photo**. The contents of the photo album are then displayed along with a picture of the new photograph. Click the thumbnail image to view the full-size version of the photograph.

When this photo album application displays the text **view image** instead of its thumbnail image, the image format that was uploaded was not recognized by Oracle Multimedia. Click **view image** to display the full-size image.

You can now begin to load your photo album application with your favorite photographs.

### 3.4.2 Description of the ASP/VBScript Photo Album Application

The top-level files that implement the ASP/VBScript Photo Album application are: `viewAlbum.asp`, `viewEntry.asp`, and `uploadPhoto.asp`. In addition, the `getPhoto.asp` file retrieves the images from the database and the `insertPhoto.asp` file inserts a new image into the database.

#### viewAlbum.asp

The **viewAlbum** page displays the contents of the photo album in a tabular format with columns labeled `Description`, `Location`, and `Image`.

Thumbnail images are ordered by description in the SQL SELECT statement and displayed with an anchor tag that is used to display the full-size image, using the tag `` as follows:

```
<A href="viewEntry.asp?entry_id=<%=strId%>">
  <% If objThumb.contentlength > 0 Then %>
    <IMG border = 1
      height="<%=objThumb.height%>"
      width="<%=objThumb.width%>"
      alt="<%=strDescription%>"
      src="getPhoto.asp?media=thumb&entry_id=<%=strId%>"
    >
```

If Oracle Multimedia does not support the image format, then a thumbnail image would not have been created and stored in the database. In this case, the text **view image** is displayed instead of the thumbnail image in the `Image` column header of the table.

Text is displayed on the page stating **Select the thumbnail to view the full-size image**. A link appearing at the bottom of the page **Upload new photo**, calls the `uploadPhoto.asp` file to present an upload form (`uploadForm.inc`) to assist in uploading a new photograph into the database.

#### viewEntry.asp

The **viewEntry** page, which displays the full-size version of a photograph, also uses the tag `` to display an image, as follows:

```
<TD vAlign=top scope="col"><B>Photo:</B></TD>
  <TD scope="col">
    <IMG border=1
      alt="<%=strDescription%>"
      src="getPhoto.asp?media=image&entry_id=<%=strId%>"
    <% If objImage.height > 0 And objImage.width > 0 Then %>
      height="<%=objImage.height%>"
      width="<%=objImage.width%>"
    <% End If %>
  >
</TD>
```

Both URLs will retrieve an image from the database and deliver it to the browser using Oracle Objects for OLE to communicate with the database.

A link appears at the bottom of the page **Return to photo album** that calls the `viewAlbum.asp` file to present the photo album table and its set of thumbnail images to view.

### **uploadPhoto.asp**

The **uploadPhoto** page displays an HTML form (`uploadForm.inc`) that allows a user to upload a new photograph into the database by entering description and location information for the new photograph, and its image file name. The form invokes the **insertPhoto** page as follows:

```
<FORM action="insertPhoto.asp" method="post" enctype="multipart/form-data">
```

### **insertPhoto.asp**

The **insertPhoto** page performs the work of loading the image into the `photos` table and automatically generating a thumbnail version of the image.

Clicking **Upload photo** near the bottom of the **uploadPhoto** page executes the submit input type form action, as follows:

```
<TD colspan=2><INPUT type=submit value="Upload photo"></TD>
```

### **getPhoto.asp**

The **getPhoto** page retrieves the image, either a thumbnail or full-size image, based on its photo column indicator value (`thumb` or `image`), from the database and returns it to the browser. If the image requested is in the browser cache and the cache is valid, then it retrieves the image from cache; otherwise, it sets the MIME type of the image based on its attribute value in the database, then gets the image from the database and delivers it to the browser, as follows:

```
If CacheIsValid( setPhotos(1).value ) Then
    Response.Status = HTTP_STATUS_NOT_MODIFIED
Else
    ' Set the mime type header and deliver the image to the browser.
    SetLastModified( setPhotos(1).value )
    Response.ContentType = objMedia.mimetype
    ReadBlob objMedia.source.localData
End If
```





---

---

# Oracle Multimedia Code Wizard Sample Application

This chapter describes the Oracle Multimedia Code Wizard sample application. The Oracle Multimedia Code Wizard sample application for the PL/SQL Gateway is a media upload and retrieval Web application that uses the Oracle Multimedia image, audio, video, and heterogeneous media object types.

This chapter assumes the following:

- You are familiar with developing PL/SQL applications using the PL/SQL Gateway.
- You have already installed and configured the Oracle Multimedia Code Wizard sample application.

The sample application files and `README.txt` file are located at:

```
<ORACLE_HOME>/ord/http/demo/plsgwycw (on Linux and UNIX)
```

```
<ORACLE_HOME>\ord\http\demo\plsgwycw (on Windows)
```

The following subsections describe how to run the Code Wizard Photo Album application. See the `README.txt` file for additional requirements and instructions on installing and configuring this sample application.

See [Chapter 5](#) for a description of the Oracle Multimedia IMExample sample application. This sample application lets you retrieve, save, play, and delete multimedia data from the Oracle Database sample schemas using Oracle Multimedia Java classes and Oracle Multimedia object types.

See [Chapter 3](#) for descriptions of the following Photo Album sample applications:

- Oracle Multimedia PL/SQL Web Toolkit Photo Album application
- Oracle Multimedia Java Servlet Photo Album application
- Oracle Multimedia JSP Photo Album application
- Oracle Multimedia ASP/VBScript Photo Album application

This suite of Web applications use PL/SQL scripts, Java servlet files, JSP files, and ASP/VBScript files to demonstrate various ways to upload and retrieve media using Oracle Multimedia object types.

## 4.1 Using the Code Wizard Sample Application

The Oracle Multimedia Code Wizard sample application lets you create PL/SQL stored procedures for the PL/SQL Gateway to upload and retrieve media data

(images, audio, video, and general media) stored in a database using Oracle Multimedia object types, `ORDImage`, `ORDAudio`, `ORDVideo`, and `ORDDoc`, and their respective methods. The Code Wizard guides you through a series of self-explanatory steps to create either a media retrieval or a media upload procedure. You can either create and compile standalone media access procedures, or you can create the source of media access procedures for inclusion in a PL/SQL package. This is similar to how the photo album application (see [Section 3.1](#)) uses the `insert_new_photo` procedure as the image upload procedure and the `deliver_media` procedure as the image retrieval procedure in the `photo_album` PL/SQL package. Finally, once created, you can customize the media access procedures as necessary to meet specific application requirements.

The Oracle Multimedia Code Wizard sample application can be installed from the Oracle Database Examples media, available for download from OTN. The application is installed in the following directory:

```
<ORACLE_HOME>/ord/http/demo/plsgwycw
```

The `README.txt` file in this directory contains complete instructions on installing and configuring the Code Wizard sample application. The following discussion assumes that the Code Wizard has been installed into the `ORDSYS` schema.

To use the Code Wizard to create and test media upload and retrieval procedures, you must do the following steps:

1. Create a new database access descriptor (DAD) or choose an existing DAD for use with the Code Wizard.
2. Authorize use of the DAD using the Code Wizard's administration function.
3. Create and test media upload and retrieval procedures.

This section describes these and other related topics in more detail.

### 4.1.1 Creating a New DAD or Choosing an Existing DAD

To create media upload or retrieval procedures, you must select one or more DADs for use with the Code Wizard. To prevent the unauthorized browsing of schema tables and to prevent the unauthorized creation of media access procedures, you must authorize each DAD using the Code Wizard's administration function. Depending on your database and application security requirements, you may choose to create and authorize one or more new DADs specifically for use with the Code Wizard, or you may choose to authorize the use of one or more existing DADs.

Oracle recommends that any DAD authorized for use with the Code Wizard should use some form of user authentication mechanism. The simplest approach is to create or use a DAD that uses database authentication. To use this approach, select **Basic Authentication Mode** and omit the password in the DAD specification. Alternatively, you may choose to use a DAD that specifies an existing application-specific authentication mechanism. For more information about configuring DADs, see *Oracle HTTP Server Administrator's Guide* in the Oracle Application Server Online Documentation Library.

The following example describes how to create a DAD to create and test media upload and retrieval procedures in the `SCOTT` schema.

---

---

**Note:** To test media upload procedures, the name of a document table must be specified in the DAD. When testing an upload procedure, you may choose the DAD you use to create the procedure, or you may use the DAD used to access the application. You may choose a document table name when you create a DAD, edit a DAD to specify the document table name at a later time, or use an existing DAD that already specifies a document table name. This example illustrates specifying the document table name when you create the DAD.

---

---

1. Set your Web browser to the Oracle HTTP Server Home page. Select "PL/SQL Properties" in the "Administration" page. This opens the `mod_plsql` Services page.
2. Scroll to the DAD Status section on the `mod_plsql` Services page. Click **Create**. This opens the DAD Type page.
3. Select the DAD type to be "General". Click **Next**. This opens the Database Connection page.
4. Enter `"/scottw"` in the DAD Name field. Enter SCOTT for the database account, and leave the password blank. Enter the connection information in the Database Connectivity Information section. In the Default page field, enter ORDCWPKG.MENU. Leave the other fields blank. Click **Next**. This opens the Document, Alias, and Session page.
5. Enter `MEDIA_UPLOAD_TABLE` for the Document Table on the Document, Alias, and Session page. Click **Apply**.
6. Restart Oracle HTTP Server for the changes to take effect.

### 4.1.2 Authorizing a DAD

To authorize a DAD for use with the Code Wizard, do the following steps:

1. Enter the Code Wizard's administration URL into your browser's location bar, for example:  
`http://<host-name>:<port-number>/ordcwadmin`
2. Enter the ORDSYS user name and password when prompted by the browser.
3. Select **DAD authorization** from the **Main menu**, as shown in [Figure 4-1](#). Then, click **Next**.

**Figure 4–1 Main Menu for the *interMedia* Code Wizard for the PL/SQL Gateway**

*interMedia* Code Wizard for the PL/SQL Gateway

**Main menu**

Current DAD: ORDCWADMIN  
Current schema: ORDSYS

Select the required function, then click the **Next** button.

Create media retrieval procedure  
 Create media upload procedure  
 Change DAD
 

- Change to ORDCWADMIN
- Change to SCOTTCW

 DAD authorization  
 Logout

Select action:

4. Enter the name of the DAD you wish to authorize together with the user name, as shown in [Figure 4–2](#). Then, click **Apply**.

**Figure 4–2 Authorizing the SCOTTCW DAD**

*interMedia* Code Wizard for the PL/SQL Gateway

**Authorize DADs for use with the *interMedia* Code Wizard for the PL/SQL Gateway**

The following table list the DADs and users currently authorized to use the *interMedia* Code Wizard for the PL/SQL Gateway. To delete a DAD, check the corresponding checkbox. Note that the code wizard administration DAD, ORDCWADMIN, cannot be deleted.

DAD name	User name	Delete
ORDCWADMIN	ORDSYS	<input type="checkbox"/>

Enter a DAD name and user name below to authorize a DAD for use with the *interMedia* Code Wizard for the PL/SQL Gateway.

DAD name:   
User name:

Select action:

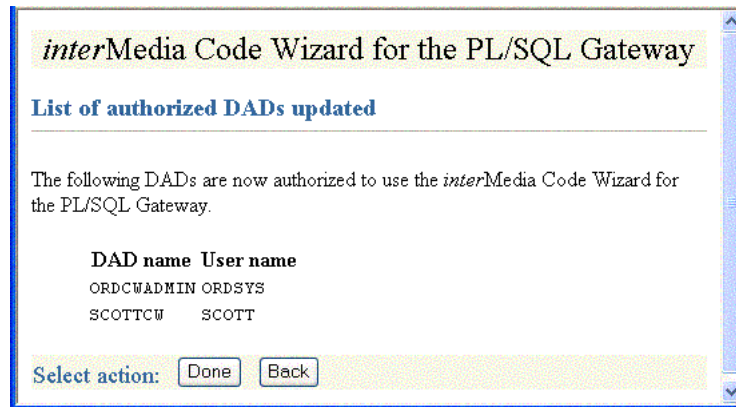
---

**Note:** Duplicate DADs are not allowed, and each authorized DAD must indicate which database schema the user is authorized to access with the Code Wizard, using the DAD. Use this same page to delete the authorization for any existing DADs that no longer need to use the Code Wizard.

---

- Review the updated list of DADs that are authorized to use the Oracle Multimedia Code Wizard, as shown in [Figure 4-3](#). Then, click **Next**.

**Figure 4-3 List of Authorized DADs**



- Select **Logout** from the **Main menu** to log out (clear HTTP authentication information), then click **Next**. The log out operation redirects the request to the PL/SQL Gateway's built-in `logmeoff` function. For more information, see *Oracle HTTP Server mod\_plsql User's Guide*.

### 4.1.3 Creating and Testing Media Upload and Retrieval Procedures

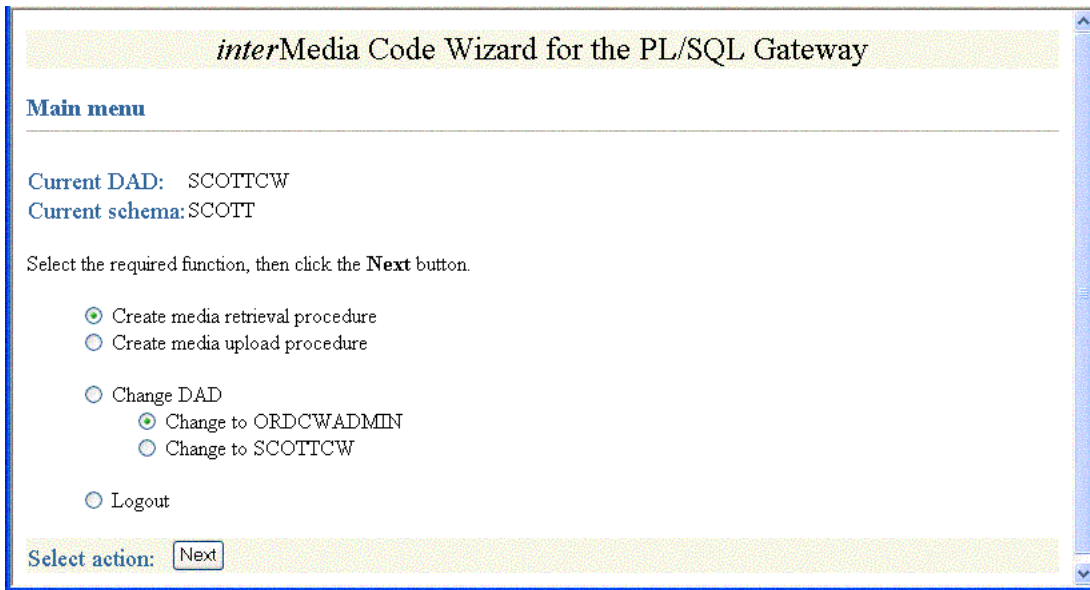
To start the Code Wizard, enter the appropriate URL into your browser's location bar, for example:

```
http://<hostname>:<port-number>/scottcw
```

or

```
http://<hostname>:<port-number>/mediadad/ordcwpkg.menu
```

Then, enter the user name and password when prompted by the browser. The **Main menu** page of the Oracle Multimedia Code Wizard for the PL/SQL Gateway is displayed as shown in [Figure 4-4](#).

**Figure 4–4 Using the SCOTTCW DAD**

If the DAD is configured specifically for use with the Code Wizard, simply enter the DAD name. Alternatively, to use another DAD, enter the DAD name together with the Code Wizard package name and **Main menu** procedure name, `ORDCWPKG.MENU` after the DAD name.

Once you have logged in, you can log out (clear HTTP authentication information) at any time by selecting **Logout** from the **Main menu**, then clicking **Next**. The logout operation redirects the request to the PL/SQL Gateway's built-in `logmeoff` function. For more information, see *Oracle HTTP Server mod\_plsql User's Guide*.

To create a media upload procedure (see [Section 4.1.4](#)) or a media retrieval procedure (see [Section 4.1.5](#)), select the appropriate option from the **Main menu**, then click **Next**. The Code Wizard then guides you through a series of self-explanatory steps to create the procedure.

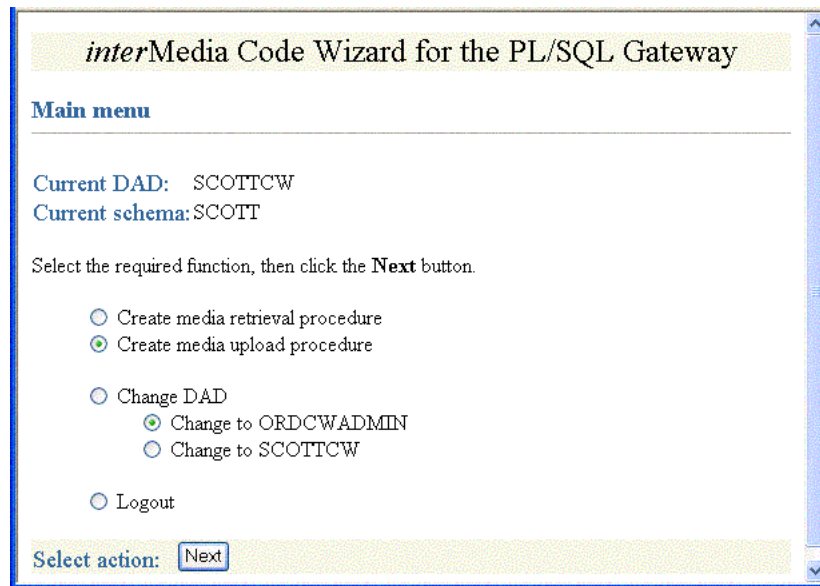
If you create a standalone media upload or retrieval procedure, you will have the opportunity to view the contents of the procedure as well as to test it.

The image and multimedia sample sessions described in [Section 4.2](#) and [Section 4.3](#) respectively, illustrate how to create and test a media upload procedure and a media retrieval procedure.

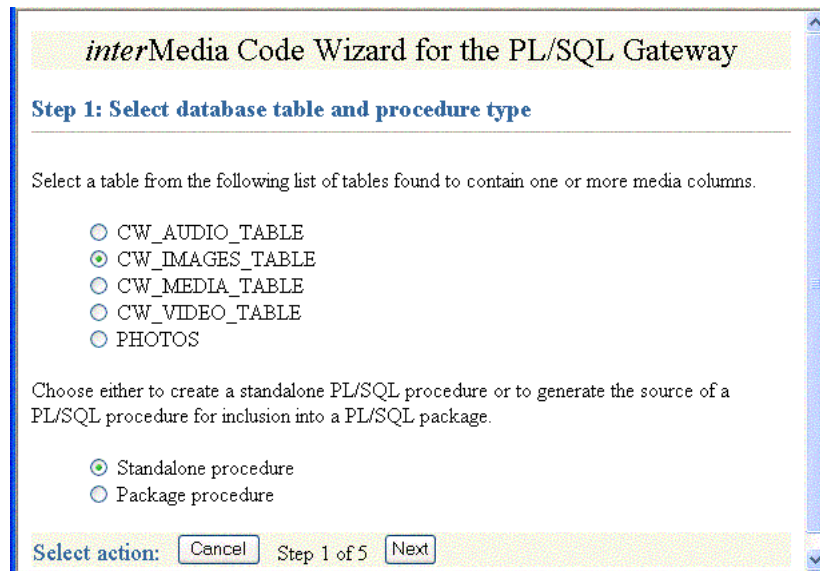
#### 4.1.4 Creating a Media Upload Procedure

To create a media upload procedure using the Oracle Multimedia Code Wizard for the PL/SQL Gateway, do the following steps:

1. Select **Create media upload procedure** from the **Main menu**, as shown in [Figure 4–5](#). Then, click **Next**.

**Figure 4–5 Create a Media Upload Procedure**

2. Select `CW_IMAGES_TABLE` and **Standalone procedure** from **Step 1: Select database table and procedure type**, as shown in [Figure 4–6](#). Then, click Next.

**Figure 4–6 Step 1: Select Database Table and Procedure Type**

3. Select **Use existing document table** and `MEDIA_UPLOAD_TABLE` from **Step 2: Select PL/SQL Gateway document upload table**, as shown in [Figure 4–7](#) and [Figure 4–8](#), because the `SCOTTCW` DAD is configured to use this document table. Then, click Next.

**Figure 4–7 Step 2: Select PL/SQL Gateway Document Upload Table (Part 1)**

*interMedia* Code Wizard for the PL/SQL Gateway

**Step 2: Select PL/SQL Gateway document upload table**

All files uploaded using the PL/SQL Gateway are uploaded into a document table. The media upload procedure created by this code wizard moves uploaded media from the specified document table to the application's table. To avoid transient files appearing temporarily in a document table used by another application component, choose a document table that is not being used to store documents permanently.

**Note:** Be sure to specify the selected document table in the application's Database Access Descriptor (DAD). If the DAD already specifies a different document table, create a new DAD for media uploads.

The document table currently specified for the SCOTTGW DAD is: MEDIA\_UPLOAD\_TABLE

Choose either to select an existing document table or to create a new document table.

Use existing document table

Select an existing PL/SQL Gateway document table from the following list.

- CW\_IMAGE\_UPLOAD\_TABLE
- CW\_MEDIA\_UPLOAD\_TABLE
- CW\_SAMPLE\_UPLOAD\_TABLE
- MEDIA\_UPLOAD\_TABLE
- PHOTOS\_UPLOAD

**Figure 4–8 Step 2: Select PL/SQL Gateway Document Upload Table (Part 2)**

Create new document table

Enter a table name below to create a new document table.

Table name:

Select action:   Step 2 of 5

4. Check **IMAGE (ORDIMAGE)**, select **ID (Primary key)**, and select **Conditional insert or update** from **Step 3: Select data access and media column(s)**, as shown in [Figure 4–9](#). Then, click **Next**.



**Figure 4–9 Step 3: Select Data Access and Media Column(s)**

*interMedia* Code Wizard for the PL/SQL Gateway

**Step 3: Select data access and media column(s)**

Select the column or columns to which media data is to be uploaded from the following list of media columns found in the CW\_IMAGES\_TABLE table. If the table contains multiple media columns, you may select multiple columns to allow more than one media item to be uploaded from a single HTML form.

IMAGE (ORDIMAGE)

Select the column to be used to locate the media data from the following list of columns found in the CW\_IMAGES\_TABLE table.

ID (Primary key)  
 DESCRIPTION  
 LOCATION

Choose how the generated procedure will access the table to store uploaded media data. You may choose to insert a new row into the table, to update an existing row in the table, or to conditionally insert a new row if an existing row does not exist.

Insert new row  
 Update existing row  
 Conditional insert or update

Select action:   Step 3 of 5

5. Check **DESCRIPTION**, accept the default procedure name, **UPLOAD\_CW\_IMAGES\_TABLE\_IMAGE**, and select **Create procedure in the database** from **Step 4: Select additional columns and procedure name**, as shown in [Figure 4–10](#). Then, click **Next**.

**Figure 4–10 Step 4: Select Additional Columns and Procedure Name**

*interMedia* Code Wizard for the PL/SQL Gateway

**Step 4: Select additional columns and procedure name**

Optionally select any additional columns to be stored in the table along with the media data. The primary key and any columns with a unique or not-null constraint are selected automatically. If updating an existing row, simply clear any columns you do not wish to be stored. Note that the key column selected in the previous step is always included.

DESCRIPTION  
 LOCATION

Choose a name for the media upload procedure. You can accept the default provided or supply a different name.

Procedure name:

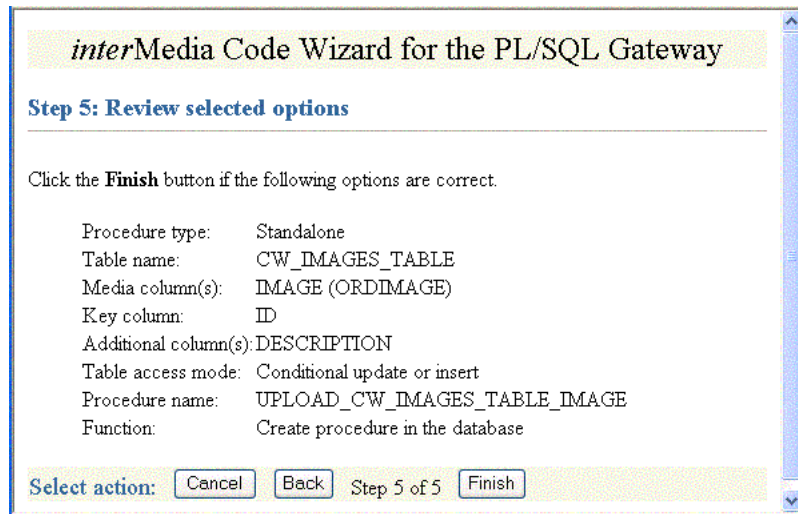
Choose either to create the procedure in the database or to generate the procedure source code only. In either case you will subsequently have the opportunity to view the generated source code.

Create procedure in the database  
 Generate procedure source only

Select action:   Step 4 of 5

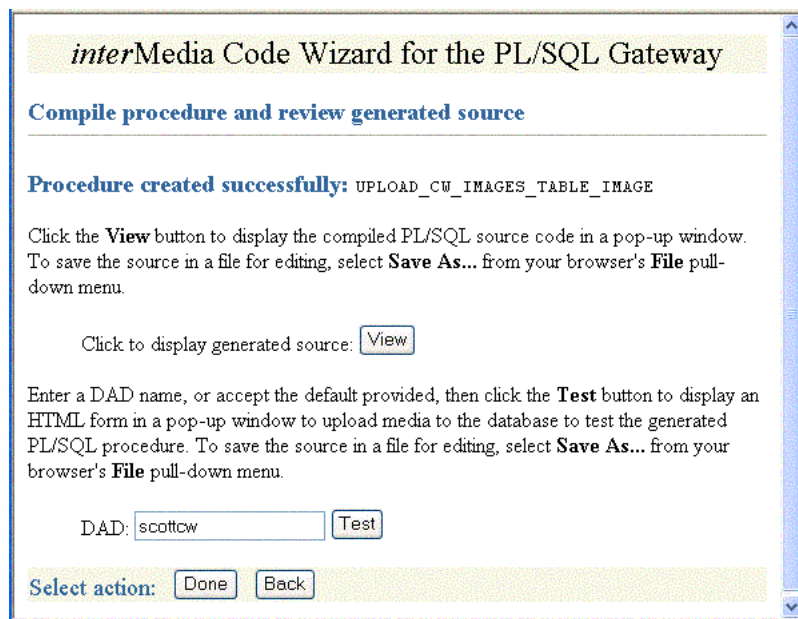
- Review the options you selected from **Step 5: Review selected options**, as shown in [Figure 4-11](#). If the options selected are correct, click **Finish**.

**Figure 4-11 Step 5: Review Selected Options**



- Note the message, Procedure created successfully: UPLOAD\_CW\_IMAGES\_TABLE\_IMAGE on the **Compile procedure and review generated source** window, as shown in [Figure 4-12](#). To review the compiled PL/SQL source code in another window, click **View** (see Step 5 in [Section 4.2](#) for a copy of the generated upload procedure). Assuming you have configured the SCOTTCW DAD and specified MEDIA\_UPLOAD\_TABLE as the document table, in the **DAD:** field, the DAD name scottcw is displayed by default. To test the PL/SQL procedure created, click **Test**.

**Figure 4-12 Compile Procedure and Review Generated Source**



- Enter the value 1 in the **ID** field on the *interMedia Code Wizard: Template Upload Form* window, browse for and select the image you want to upload in the

**IMAGE** field, and enter a brief description of the image to be uploaded in the **DESCRIPTION** field, as shown in [Figure 4–13](#). Then, click **Upload media**.

**Figure 4–13** *Template Upload Form*

9. The image is uploaded into the table row and a message is displayed, as shown in [Figure 4–14](#).

**Figure 4–14** *Template Upload Procedure -- Media Uploaded Successfully Message*

10. Return to the **Compile procedure** and review generated source window. If you are finished testing, click **Done** to return to the **Main menu**.

### 4.1.5 Creating a Media Retrieval Procedure

To create a media retrieval procedure using the Oracle Multimedia Code Wizard for the PL/SQL Gateway, do the following steps:

1. Select **Create media retrieval procedure** from the **Main menu**, as shown in [Figure 4–15](#). Then, click **Next**.

**Figure 4–15 Create a Media Retrieval Procedure**

*interMedia Code Wizard for the PL/SQL Gateway*

**Main menu**

Current DAD: SCOTTCW  
Current schema: SCOTT

Select the required function, then click the **Next** button.

- Create media retrieval procedure
- Create media upload procedure
- Change DAD
  - Change to ORDCWADMIN
  - Change to SCOTTCW
- Logout

Select action:

2. Select **CW\_IMAGES\_TABLE** and **Standalone procedure** from **Step 1: Select database table and procedure type**, as shown in [Figure 4–16](#). Then, click **Next**.

**Figure 4–16 Step 1: Select Database Table and Procedure Type**

*interMedia Code Wizard for the PL/SQL Gateway*

**Step 1: Select database table and procedure type**

Select a table from the following list of tables found to contain one or more media columns.

- CW\_AUDIO\_TABLE
- CW\_IMAGES\_TABLE
- CW\_MEDIA\_TABLE
- CW\_VIDEO\_TABLE
- PHOTOS

Choose either to create a standalone PL/SQL procedure or to generate the source of a PL/SQL procedure for inclusion into a PL/SQL package.

- Standalone procedure
- Package procedure

Select action:  Step 1 of 4

3. Select **IMAGE (ORDIMAGE)** and **ID (Primary key)** from **Step 2: Select media column and key column**, as shown in [Figure 4–17](#). Then, click **Next**.

**Figure 4–17 Step 2: Select Media Column and Key Column**

*interMedia Code Wizard for the PL/SQL Gateway*

**Step 2: Select media column and key column**

Select the column from which to retrieve the media data from the following list of media columns found in the CW\_IMAGES\_TABLE table.

IMAGE (ORDIMAGE)

Select the column to be used to locate the media data from the following list of columns found in the CW\_IMAGES\_TABLE table.

ID (Primary key)  
 ROWID (Unique)  
 DESCRIPTION  
 LOCATION

Select action:   Step 2 of 4

4. Accept the default procedure name, GET\_CW\_IMAGES\_TABLE\_IMAGES, the default parameter name, MEDIA\_ID, and **Create procedure in the database** from **Step 3: Select procedure name and parameter name**, as shown in [Figure 4–18](#). Then, click Next.

**Figure 4–18 Step 3: Select Procedure Name and Parameter Name**

*interMedia Code Wizard for the PL/SQL Gateway*

**Step 3: Select procedure name and parameter name**

Choose a name for the media retrieval procedure. You can accept the default provided or supply a different name.

Procedure name:

Choose a name for the parameter used to supply the key value. You can accept the default provided or supply a different name. The parameter name is used in a media retrieval URL as follows:  
http://host/pls/D&D/proc-name?param-name=key-value

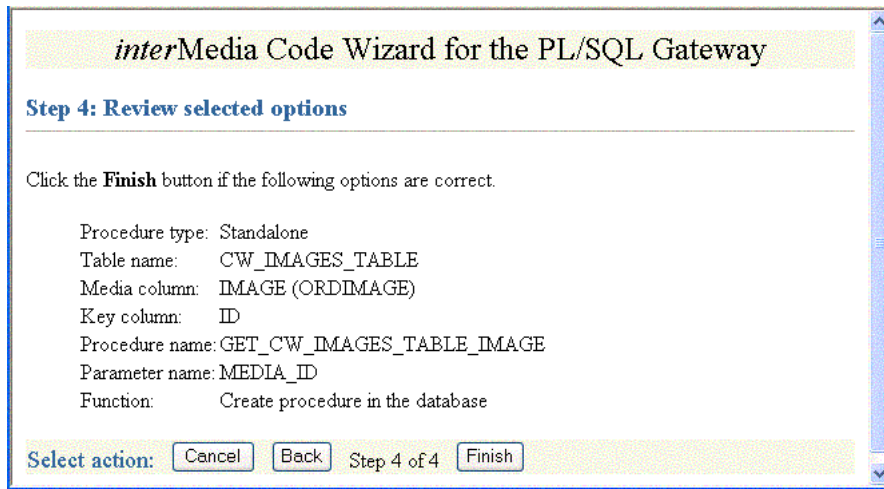
Parameter name:

Choose either to create the procedure in the database or to generate the procedure source code only. In either case you will subsequently have the opportunity to view the generated source code.

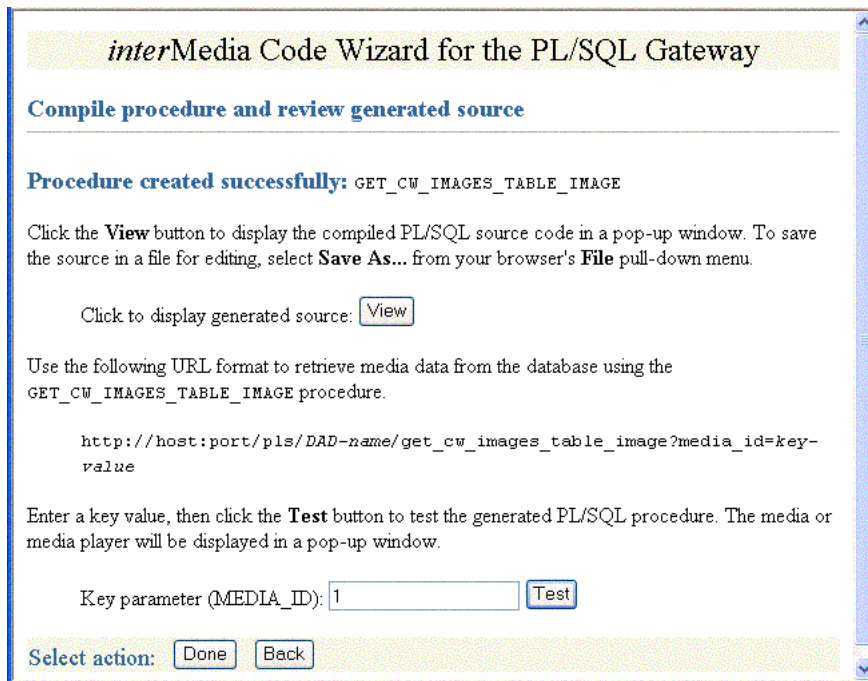
Create procedure in the database  
 Generate procedure source only

Select action:   Step 3 of 4

5. Review the options you selected from **Step 4: Review Selected Options**, as shown in [Figure 4–19](#). If the options selected are correct, click **Finish**.

**Figure 4–19 Step 4: Review Selected Options**

6. Note the message Procedure created successfully: GET\_CW\_IMAGES\_TABLE\_IMAGE from the **Compile procedure and review generated source** window, as shown in Figure 4–20. To review the compiled PL/SQL source code in another window, click **View** (see Step 6 in Section 4.2 for a copy of the generated retrieval procedure). To test the PL/SQL procedure created, assuming you have an image already loaded in the database with an ID value of 1, enter the value 1 for the Key parameter (MEDIA\_ID), then click **Test**. The image is retrieved from the table row and is displayed as shown in Figure 4–21. Click **Done** to return to the **Main menu**.

**Figure 4–20 Compile Procedure and Review Generated Source**

**Figure 4–21 Displayed Image 1981**

### 4.1.6 Using the PL/SQL Gateway Document Table

All files uploaded using the PL/SQL Gateway are stored in a document table. Media upload procedures created by the Code Wizard automatically move uploaded media from the specified document table to the application's table. To avoid transient files from appearing temporarily in a document table used by another application component, use a document table that is not being used to store documents permanently.

Be sure to specify the selected document table in the application's database access descriptor (DAD). If the DAD already specifies a different document table, create a new DAD for media upload procedures. If you choose to create a new document table, the Code Wizard will create a table with the following format:

```
CREATE TABLE document-table-name
( name          VARCHAR2(256) UNIQUE NOT NULL,
  mime_type     VARCHAR2(128) ,
  doc_size      NUMBER,
  dad_charset   VARCHAR2(128) ,
  last_updated  DATE,
  content_type  VARCHAR2(128) ,
  blob_content  BLOB )
--
-- store BLOBs as SecureFile LOBs
--
LOB(blob_content) STORE AS SECUREFILE;
```

For more information about file upload and document tables, see *Oracle HTTP Server mod\_plsql User's Guide*.

### 4.1.7 How Time Zone Information Is Used to Support Browser Caching

User response times are improved and network traffic is reduced if a browser can cache resources received from a Web server and subsequently use those cached resources to satisfy future requests. This section describes at a very high level, how the browser caching mechanism works and how the Code Wizard utility package is used to support that mechanism. When reading this discussion, note that all HTTP date and time stamps are expressed in Coordinated Universal Time (UTC).

All HTTP responses include a Date header, which indicates the date and time when the response was generated. When a Web server sends a resource in response to a request from a browser, it can also include the Last-Modified HTTP response header, which indicates the date and time when the requested resource was last modified. It is important to note that the Last-Modified header must not be later than the Date header.

After receiving and caching a resource, if a browser needs to retrieve the same resource again, it sends a request to the Web server with the If-Modified-Since request header specified as the value of the Last-Modified date, which was returned by the application server when the resource was previously retrieved and cached. When the Web server receives the request, it compares the date in the If-Modified-Since request header with the last update time of the resource. Assuming the resource still exists, if

the resource has not changed since it was cached by the browser, the Web server responds with an HTTP 304 Not Modified status with no response body, which indicates that the browser can use the resource currently stored in its cache. Assuming once again the resource still exists, if the request does not include an If-Modified-Since header or if the resource has been updated since it was cached by the browser, the Web server responds with an HTTP 200 OK status and sends the resource to the browser. See the HTTP specification (<http://www.w3.org/Protocols/>) for more information.

The ORDImage, ORDAudio, ORDVideo, and ORDDoc objects all possess an updateTime attribute stored as a DATE in the embedded ORDSrc object. Although the DATE data type has no support for time zones or daylight savings time, the Oracle9i and later database versions do support time zones and also provide functions for converting a DATE value stored in a database to UTC. See *Oracle Database Administrator's Guide* for more information about how to set a time zone for a database. See *Oracle Database SQL Language Reference* for more information about date and time functions.

When a response is first returned to a browser, a media retrieval procedure sets the Last-Modified HTTP response header based on the updateTime attribute. If a request for media data includes an If-Modified-Since header, the media retrieval procedure compares the value with the updateTime attribute and returns an appropriate response. If the resource in the browser's cache is still valid, an HTTP 304 Not Modified status is returned with no response body. If the resource has been updated since it was cached by the browser, then an HTTP 200 OK status is returned with the media resource as the response body.

Media retrieval procedures created by the Code Wizard call the utility package to convert a DATE value stored in the database to UTC. The utility package uses the time zone information stored with an Oracle9i or later database and the date and time functions to convert database date and time stamps to UTC. To ensure the resulting date conforms to the rule for the Last-Modified date described previously, the time zone information must be specified correctly. See *Oracle Database Administrator's Guide* for more information about how to set a time zone for a database.

## 4.2 Sample Session Using Images

The following sample session uses the SCOTT schema with the password tiger to illustrate the creation of image media upload and retrieval procedures. Substitute a different schema name, password, or both if you want to use a different schema or if you have changed the password for the SCOTT schema.

This sample session assumes the Oracle Multimedia Code Wizard has been installed in the ORDSYS schema.

Perform the following steps:

### **Step 1 Create a table to store images for the application by starting SQL\*Plus and connecting to the SCOTT schema in the database.**

For example:

```
sqlplus SCOTT/tiger[@<connect_identifer>]

SQL> CREATE TABLE cw_images_table( id NUMBER PRIMARY KEY,
                                     description VARCHAR2(30) NOT NULL,
                                     location VARCHAR2(30),
                                     image ORDSYS.ORDIMAGE )
--
```



```
-- store media as SecureFile LOBs
--
LOB(image.source.localdata) STORE AS SECUREFILE;
```

### Step 2 Create the SCOTT<sub>CW</sub> DAD to be used to create the procedures.

1. Set your Web browser to the Oracle HTTP Server Home page. Select "PL/SQL Properties" in the "Administration" page. This opens the mod\_plsql Services page.
2. On the mod\_plsql Services page, scroll to the DAD Status section. Click **Create**. This opens the DAD Type page.
3. Select the DAD type to be "General". Click **Next**. This opens the Database Connection page.
4. Enter "/scottw" in the DAD Name field. Enter SCOTT for the database account, and leave the password blank. Enter the connection information in the Database Connectivity Information section. In the Default page field, enter ORDCWPKG.MENU. Leave the other fields blank. Click **Next**. This opens the Document, Alias, and Session page.
5. On the Document, Alias, and Session page, enter MEDIA\_UPLOAD\_TABLE for the Document Table. Click **Apply**.
6. Restart Oracle HTTP Server for the changes to take effect.

### Step 3 Authorize the use of the SCOTT<sub>CW</sub> DAD and SCOTT schema with the Code Wizard.

1. Enter the Code Wizard's administration URL into your browser's location bar, then enter the ORDSYS user name and password when prompted by the browser, for example:
 

```
http://<hostname>:<port-number>/ordcwadmin
```
2. Select the DAD authorization function from the Code Wizard's **Main menu** and click **Next**. Enter the name of the demonstration DAD, SCOTT<sub>CW</sub>, and the user name SCOTT, then click **Apply**. Click **Done** when the confirmation window is displayed.

### Step 4 Change DADs to the SCOTT<sub>CW</sub> DAD.

1. Click **Change DAD** from the Code Wizard's **Main menu**.
2. Click Change to SCOTT<sub>CW</sub>, if it is not already selected, then click **Next**.
3. Enter the user name SCOTT and the password tiger when prompted for the user name and password, then click **OK**.

The **Main menu** now displays the current DAD as SCOTT<sub>CW</sub> and the current schema as SCOTT.

### Step 5 Create and test the media upload procedure.

Click **Create media upload procedure** from the **Main menu**, then click **Next**.

1. Select the database table and procedure type.
  - a. Click the CW\_IMAGES\_TABLE database table.
  - b. Click **Standalone procedure**.
  - c. Click **Next**.

2. Select the PL/SQL document upload table.

If there are no document tables in the SCOTT schema, the Code Wizard displays a message indicating this situation. In this case, accept the default table name provided, `CW_SAMPLE_UPLOAD_TABLE`, then click **Next**.

If there are existing document tables, but the `CW_SAMPLE_UPLOAD_TABLE` is not among them, click **Create new document table**, accept the default table name provided, `CW_SAMPLE_UPLOAD_TABLE`, then click **Next**.

If the `CW_SAMPLE_UPLOAD_TABLE` document table already exists, ensure that the **Use existing document table** and the `CW_SAMPLE_UPLOAD_TABLE` options are selected. Click **Next**.

3. Select the data access and media columns.

- a. Click **IMAGE (ORDIMAGE)**.
- b. Click **ID (Primary key)**.
- c. Click **Conditional insert or update**.
- d. Click **Next**.

4. Select additional columns and procedure names.

- a. Ensure that **DESCRIPTION** checkmarked because this column has a NOT NULL constraint. (The **LOCATION** column is not checkmarked by default as there are no constraints on this column.)
- b. Accept the procedure name provided, `UPLOAD_CW_IMAGES_TABLE_IMAGE`.
- c. Click **Create procedure in the database**.
- d. Click **Next**.

5. Review the following selected procedure creation options that are displayed:

```

Procedure type:      Standalone
Table name:         CW_IMAGES_TABLE
Media column(s):    IMAGE (ORDIMAGE)
Key column:         ID
Additional column(s): DESCRIPTION
Table access mode:  Conditional update or insert
Procedure name:     UPLOAD_CW_IMAGES_TABLE_IMAGE
Function:           Create procedure in the database

```

Click **Finish**.

6. Compile the procedure and review the generated source information.

The Code Wizard displays the following message: "Procedure created successfully: `UPLOAD_CW_IMAGES_TABLE_IMAGE`".

- a. At the option **Click to display generated source**, click **View** to view the generated source in another window. A copy of the generated source is shown at the end of Step 5, substep 6g.
- b. Close the window after looking at the generated source.
- c. Accept the **DAD:** name provided, `SCOTT.CW`, then click **Test** to produce another window that displays a template file upload form that you can use to test the generated procedure.
- d. To customize the template file upload form, select **Save As...** from your browser's **File** pull-down menu to save the HTML source for editing.

- e. To test the template upload form, enter the following information:
  - For the **ID:** column, enter the number 1 as the row's primary key.
  - For the **IMAGE** column, click **Browse...** and choose an image file to upload to the database.
  - For the **DESCRIPTION** column, enter a brief description of the image.
  - Click **Upload media**.

The Code Wizard displays a template completion window with the heading **interMedia Code Wizard: Template Upload Procedure**, and, if the procedure is successful, the message: Media uploaded successfully.

- f. Close the window.
- g. Click **Done** on the **Compile procedure and review generated source** window to return to the **Main menu** of the Code Wizard.

A copy of the generated image upload procedure is as follows:

```
CREATE OR REPLACE PROCEDURE UPLOAD_CW_IMAGES_TABLE_IMAGE
( in_ID IN VARCHAR2,
  in_IMAGE IN VARCHAR2 DEFAULT NULL,
  in_DESCRIPTION IN VARCHAR2 DEFAULT NULL )
AS
local_IMAGE ORDSYS.ORDIMAGE := ORDSYS.ORDIMAGE.init();
local_ID CW_IMAGES_TABLE.ID%TYPE := NULL;
upload_size      INTEGER;
upload_mimetype  VARCHAR2( 128 );
upload_blob      BLOB;
BEGIN
--
-- Update the existing row.
--
UPDATE CW_IMAGES_TABLE mtbl
SET mtbl.IMAGE = local_IMAGE,
    mtbl.DESCRPTION = in_DESCRIPTION
WHERE mtbl.ID = in_ID
RETURN mtbl.ID INTO local_ID;
--
-- Conditionally insert a new row if no existing row is updated.
--
IF local_ID IS NULL
THEN
--
-- Insert the new row into the table.
--
INSERT INTO CW_IMAGES_TABLE ( ID, IMAGE, DESCRIPTION )
VALUES ( in_ID, local_IMAGE, in_DESCRIPTION );
END IF;
--
-- Select interMedia object(s) for update.
--
SELECT mtbl.IMAGE INTO local_IMAGE
FROM CW_IMAGES_TABLE mtbl WHERE mtbl.ID = in_ID FOR UPDATE;
--
-- Store media data for the column in_IMAGE.
--
IF in_IMAGE IS NOT NULL
THEN
SELECT dtbl.doc_size, dtbl.mime_type, dtbl.blob_content INTO
```

```

        upload_size, upload_mimetype, upload_blob
    FROM CW_IMAGE_UPLOAD_TABLE dtbl WHERE dtbl.name = in_IMAGE;
IF upload_size > 0
THEN
    dbms_lob.copy( local_IMAGE.source.localData,
                  upload_blob,
                  upload_size );
    local_IMAGE.setLocal();
BEGIN
    local_IMAGE.setProperties();
EXCEPTION
    WHEN OTHERS THEN
        local_IMAGE.contentLength := upload_size;
        local_IMAGE.mimeType := upload_mimetype;
END;
END IF;
DELETE FROM CW_IMAGE_UPLOAD_TABLE dtbl WHERE dtbl.name = in_IMAGE;
END IF;
--
-- Update interMedia objects in the table.
--
UPDATE CW_IMAGES_TABLE mtbl
SET mtbl.IMAGE = local_IMAGE
WHERE mtbl.ID = in_ID;
--
-- Display the template completion message.
--
http.print( '<html>' );
http.print( '<title>interMedia Code Wizard: Template Upload
Procedure</title>' );
http.print( '<body>' );
http.print( '<h2>&lt;i>inter</i>Media Code Wizard:
Template Upload Procedure</h2>' );
http.print( 'Media uploaded successfully.' );
http.print( '</body>' );
http.print( '</html>' );
END UPLOAD_CW_IMAGES_TABLE_IMAGE;

```

This sample image upload procedure declares the following input parameters and variables:

1. In the declaration section, the procedure declares three input parameters: `in_ID`, `in_IMAGE`, and `in_DESCRIPTION`, then initializes the latter two to `NULL`.
2. In the subprogram section, the following variables are declared:
  - The variable `local_IMAGE` is assigned the data type `ORDSYS.ORDIMAGE` and initialized with an empty BLOB using the `ORDIMAGE.init()` method.
  - The variable `local_ID` takes the same data type as the `ID` column in the table `CW_IMAGES_TABLE` and is initialized to `NULL`.
  - Three additional variables are declared `upload_size`, `upload_mimetype`, and `upload_blob`, which are later given values from comparable column names `doc_size`, `mime_type`, and `blob_content` from the document table `CW_IMAGE_UPLOAD_TABLE`, using a `SELECT` statement in preparation for copying the content of the image BLOB data to the `ORDSYS.ORDIMAGE.source.localData` attribute.

Within the outer BEGIN...END executable statement section, the following operations are executed:

1. Update the existing row in the table `CW_IMAGES_TABLE` for the `IMAGE` and `DESCRIPTION` columns and return the value of `local_ID` where the value of the `ID` column is the value of the `in_ID` input parameter.
2. If the value returned of `local_ID` is `NULL`, conditionally insert a new row into the table `CW_IMAGES_TABLE` and initialize the instance of the `ORDImage` object type in the `image` column with an empty `BLOB`.
3. Select the `ORDImage` object column `IMAGE` in the table `CW_IMAGES_TABLE` for update where the value of the `ID` column is the value of the `in_ID` input parameter.
4. Select a row for the `doc_size`, `mime_type`, and `blob_content` columns from the document table and pass the values to the `upload_size`, `upload_mimetype`, and `upload_blob` variables where the value of the document table `Name` column is the value of the `in_IMAGE` input parameter.
5. Perform a `DBMS_LOB` copy of the `BLOB` data from the table `CW_IMAGE_UPLOAD_TABLE` into the `ORDSYS.ORDIMAGE.source.localData` attribute, then call the `setLocal()` method to indicate that the image data is stored locally in the `BLOB`, and `ORDImage` methods should look for corresponding data in the `source.localData` attribute.
6. In the inner executable block, call the `ORDImage setProperties()` method to read the image data to get the values of the object attributes and store them in the image object attributes for the `ORDImage` object.
7. If the `setProperties()` call fails, catch the exception and call the `contentLength()` method to get the size of the image and call the `mimeType()` method to get the `MIME` type of the image.
8. Delete the row of data from the document table `CW_IMAGE_UPLOAD_TABLE` that was copied to the row in the table `CW_IMAGES_TABLE` where the value of the `Name` column is the value of the `in_IMAGE` input parameter.
9. Update the `ORDImage` object `IMAGE` column in the table `CW_IMAGES_TABLE` with the content of the variable `local_IMAGE` where the value of the `ID` column is the value of the `in_ID` input parameter.
10. Display a completion message on the `HTML` page to indicate that the media uploaded successfully using the `http.print` function from the `PL/SQL Web Toolkit`.

### Step 6 Create and test a media retrieval.

Select **Create media retrieval procedure** from the **Main menu**, then click **Next**.

1. Select the database table and procedure type.
  - a. Click **CW\_IMAGES\_TABLE**.
  - b. Click **Standalone procedure**.
  - c. Click **Next**.
2. Select the media column and key column.
  - a. Click **IMAGE (ORDIMAGE)**.
  - b. Click **ID (Primary key)**.
  - c. Click **Next**.

3. Select the procedure name and parameter name.
  - a. Accept the procedure name provided, `GET_CW_IMAGES_TABLE_IMAGE`.
  - b. Accept the parameter name provided, `MEDIA_ID`.
  - c. Click **Create procedure in the database**.
  - d. Click **Next**.
4. Review the following selected procedure creation options:

```

Procedure type:      Standalone
Table name:         CW_IMAGES_TABLE
Media column(s):   IMAGE (ORDIMAGE)
Key column:        ID
Procedure name:    GET_CW_IMAGES_TABLE_IMAGE
Parameter Name:    MEDIA_ID
Function:          Create procedure in the database

```

Click **Next**.

5. Compile the procedure and review the generated source.

The Code Wizard displays the following message: Procedure created successfully: `GET_CW_IMAGES_TABLE_IMAGE`

- a. Click **View** to view the generated source in another window. Close the window after looking at the generated source. A copy of the generated source is shown at the end of Step 6, substep 5e.
- b. Review the URL format used to retrieve images using the `GET_CW_IMAGES_TABLE_IMAGE` procedure.
- c. Enter the number 1 as the Key parameter, then click **Test** to test the procedure by retrieving the image uploaded previously.

The retrieved image is displayed in another window.

- d. Close the window.
- e. Click **Done** to return to the **Main menu**.

A copy of the generated image retrieval procedure is as follows:

```

CREATE OR REPLACE PROCEDURE GET_CW_IMAGES_TABLE_IMAGE (
  MEDIA_ID IN VARCHAR2 )
AS
  localObject ORDSYS.ORDIMAGE;
  localBlob  BLOB;
  localBfile BFILE;
  httpStatus NUMBER;
  lastModDate VARCHAR2(256);
BEGIN
  --
  -- Retrieve the object from the database into a local object.
  --
  BEGIN
    SELECT mtbl.IMAGE INTO localObject FROM CW_IMAGES_TABLE mtbl
    WHERE mtbl.ID = MEDIA_ID;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      ordplsgwyutil.resource_not_found( 'MEDIA_ID', MEDIA_ID );
    RETURN;
  END;
END;

```

```

--
-- Check the update time if the browser sent an If-Modified-Since header.
--
IF ordplsgwyutil.cache_is_valid( localObject.getUpdateTime() )
THEN
  owa_util.status_line( ordplsgwyutil.http_status_not_modified );
  RETURN;
END IF;

--
-- Figure out where the image is.
--
IF localObject.isLocal() THEN
  --
  -- Data is stored locally in the localData BLOB attribute.
  --
  localBlob := localObject.getContent();
  owa_util.mime_header( localObject.getMimeType(), FALSE );
  ordplsgwyutil.set_last_modified( localObject.getUpdateTime() );
  owa_util.http_header_close();
  IF owa_util.get_cgi_env( 'REQUEST_METHOD' ) &lt;&gt; 'HEAD' THEN
    wpg_docload.download_file( localBlob );
  END IF;
ELSIF UPPER( localObject.getSourceType() ) = 'FILE' THEN

  --
  -- Data is stored as a file from which ORDSOURCE creates
  -- a BFILE.
  --
  localBfile := localObject.getBFILE();
  owa_util.mime_header( localObject.getMimeType(), FALSE );
  ordplsgwyutil.set_last_modified( localObject.getUpdateTime() );
  owa_util.http_header_close();
  IF owa_util.get_cgi_env( 'REQUEST_METHOD' ) &lt;&gt; 'HEAD' THEN
    wpg_docload.download_file( localBfile );
  END IF;

ELSIF UPPER( localObject.getSourceType() ) = 'HTTP' THEN
  --
  -- The image is referenced as an HTTP entity, so we have to
  -- redirect the client to the URL that ORDSOURCE provides.
  --
  owa_util.redirect_url( localObject.getSource() );
ELSE
  --
  -- The image is stored in an application-specific data
  -- source type for which no default action is available.
  --
  NULL;
END IF;
END GET_CW_IMAGES_TABLE_IMAGE;

```

This sample image retrieval procedure declares the following input parameters and variables:

1. In the declaration section, the procedure declares one input parameter: MEDIA\_ID.
2. In the subprogram section, the following variables are declared:

- The variable `localObject` is assigned the data type `ORDSYS.ORDIMAGE`.
- The variable `localBlob` is a BLOB data type, the variable `localBfile` is a BFILE data type, `httpStatus` is a NUMBER, and `lastModDate` is a VARCHAR2 with a maximum size of 256 characters.

Within the outer BEGIN...END executable statement section, the following operations are executed:

1. Select the `ORDImage` object column `IMAGE` in the table `CW_IMAGES_TABLE` where the value of the `ID` column is the value of the `MEDIA_ID` input parameter.
2. In the inner executable block, when no data is found, raise an exception and call the `resource_not_found` function of the PL/SQL Gateway and get the value of the `MEDIA_ID` input parameter.
3. Check the update time if the browser sent an If-Modified-Since header by calling the `getUpdateTime()` method passed into the `cache_is_valid` function of the PL/SQL Gateway.
4. If the cache is valid, send an HTTP status code to the client using the PL/SQL Web Toolkit `owa_util` package `status_line` procedure passing in the call to the `http_status_not_modified` function.
5. Determine where the image data is stored; call the `ORDImage isLocal()` method, which returns a Boolean expression of true if the image data is stored locally in the BLOB, then get the handle to the local BLOB.
  - If the value is true, assign the variable `localBlob` the `ORDImage getContent()` method to get the handle of the local BLOB containing the image data.
  - Call the `ORDImage getMimeType()` method to determine the image's MIME type and pass this to the `owa_util.mime_header` procedure and keep the HTTP header open.
  - Call the `ORDImage getUpdateTime()` method to get the time the image was last modified and pass this to the `ordplsgwyutil.set_last_modified` procedure.
  - Close the HTTP header by calling the `owa_util.http_header_close()` procedure.
  - Call the `owa_util.get_cgi_env` procedure and if the value of the request method is not `HEAD`, then use the `wpg_docload.download_file` procedure to pass in the value of `localBlob` that contains the LOB locator of the BLOB containing the image data to download the image from the database.
6. If the `ORDImage isLocal()` method returns false, call the `ORDImage getSourceType()` method to determine if the value is `FILE`; if so, then the image data is stored as an external file on the local file system. Then, get the LOB locator of the BFILE containing the image data.
  - Assign the variable `localBfile` the `ORDImage getBfile()` method to get the LOB locator of the BFILE containing the image data.
  - Call the `ORDImage getMimeType()` method to determine the image's MIME type and pass this to the `owa_util.mime_header` procedure and keep the HTTP header open.



- Call the `ORDImage updateTime()` method to get the time the image was last modified and pass this to the `ordplsawyutil.set_last_modified` procedure.
  - Close the HTTP header by calling the `owa_util.http_header_close()` procedure.
  - Call the `owa_util.get_cgi_env` procedure and if the value of the request method is not `HEAD`, then use the `wpg_docload.download_file` procedure to pass in the value of `localBfile` that contains the LOB locator of the BFILE containing the image data to download the image from the file.
7. If the `ORDImage isLocal()` method returns false, call the `ORDImage getSourceType()` method to determine if the value is `HTTP`; if so, then the image data is stored at an HTTP URL location, which then redirects the client to the URL that `ORDSource` provides using the `owa_util.redirect_url` procedure.
  8. If the `ORDImage isLocal()` method returns false, call the `ORDImage getSourceType()` method to determine if the value is `FILE` or `HTTP`; if it is neither, then the image is stored in an application-specific data source type that is not recognized or supported by Oracle Multimedia.

### 4.3 Sample Session Using Multiple Object Columns

The following sample session uses the `SCOTT` schema with the password `tiger` to illustrate the creation of a multimedia upload (multiple Oracle Multimedia object columns) and single media retrieval procedures. Substitute a different schema name, password, or both if you want to use a different schema or if you have changed the password for the `SCOTT` schema.

This sample session assumes the Oracle Multimedia Code Wizard has been installed.

Perform the following steps:

#### **Step 1 Create a table to store audio for the application by starting SQL\*Plus and connecting to the `SCOTT` schema in the database.**

For example:

```
sqlplus SCOTT/tiger[@<connect_identifer>]

SQL> CREATE TABLE cw_media_table( id NUMBER PRIMARY KEY,
                                   description VARCHAR2(30) NOT NULL,
                                   location VARCHAR2(30),
                                   image ORDSYS.ORDIMAGE,
                                   thumb ORDSYS.ORDIMAGE,
                                   audio ORDSYS.ORDAUDIO,
                                   video ORDSYS.ORDVIDEO,
                                   media ORDSYS.ORDDOC )
--
-- store media as SecureFile LOBs
--
LOB(image.source.localdata) STORE AS SECUREFILE
LOB(thumb.source.localdata) STORE AS SECUREFILE
LOB(audio.source.localdata) STORE AS SECUREFILE
LOB(video.source.localdata) STORE AS SECUREFILE
LOB(media.source.localdata) STORE AS SECUREFILE;
```

**Step 2 Use the SCOTTW DAD you created in Step 2. Then, authorize the use of it in Step 3.**

If you have not created the SCOTTW DAD and authorized the use of this DAD, perform Steps 2 and 3 in this section, then continue to next step that follows in this section.

**Step 3 Change DADs to the SCOTTCW DAD.**

1. Enter the Code Wizard's administration URL into your browser's location bar, then enter the ORDSYS user name and password when prompted by the browser, for example:

```
http://<hostname>:<port-number>/ordcwadmin
```

2. Click **Change DAD** from the Code Wizard's **Main menu**.
3. Click Change to SCOTTCW, if it is not already selected, then click **Next**.
4. Enter the user name SCOTT and the password tiger when prompted for the user name and password, then press **OK**.

The **Main menu** now displays the current DAD as SCOTTCW and the current schema as SCOTT.

**Step 4 Create and test the media upload procedure.**

Click **Create media upload procedure** from the **Main menu**, then click **Next**.

1. Select the database table and procedure Type.
  - a. Click **CW\_MEDIA\_TABLE**.
  - b. Click **Standalone procedure**.
  - c. Click **Next**.
2. Select the PL/SQL document upload table.

If there are no document tables in the SCOTT schema, the Code Wizard displays a message indicating this situation. In this case, accept the default table name provided, CW\_MEDIA\_UPLOAD\_TABLE, then click **Next**.

If there are existing document tables, but the table CW\_MEDIA\_UPLOAD\_TABLE is not among them, click **Create new document table**, accept the default table name provided, CW\_MEDIA\_UPLOAD\_TABLE, then click **Next**.

If the CW\_MEDIA\_UPLOAD\_TABLE document table already exists, select **Use existing document table** and CW\_MEDIA\_UPLOAD\_TABLE, then click **Next**.

3. Select the data access and media columns.
  - a. Ensure that **IMAGE (ORDIMAGE)**, **THUMB (ORDIMAGE)**, **AUDIO (ORDAUDIO)**, **VIDEO (ORDVIDEO)**, and **MEDIA (ORDDOC)** are all checkmarked.
  - b. Click **ID (Primary key)**.
  - c. Click **Conditional insert or update**.
  - d. Click **Next**.
4. Select additional columns and procedure names.
  - a. Ensure that **DESCRIPTION** is checkmarked because this column has a NOT NULL constraint. (The **LOCATION** column is not checkmarked by default as there are no constraints on this column.)
  - b. Accept the procedure name provided, UPLOAD\_CW\_MEDIA\_TABLE\_IMAGE.

- c. Click **Create procedure in the database**.
  - d. Click **Next**.
5. Review the following selected procedure creation options that are displayed:

```

Procedure type:      Standalone
Table name:         CW_MEDIA_TABLE
Media column(s):   IMAGE (ORDIMAGE)
                  THUMB (ORDIMAGE)
                  AUDIO (ORDAUDIO)
                  VIDEO (ORDVIDEO)
                  MEDIA (ORDDOC)
Key column:        ID
Additional column(s): DESCRIPTION
Table access mode: Conditional update or insert
Procedure name:    UPLOAD_CW_MEDIA_TABLE_IMAGE
Function:          Create procedure in the database

```

Click **Finish**.

6. Compile the procedure and review the generated source information.

The Code Wizard displays the following message: "Procedure created successfully: UPLOAD\_CW\_MEDIA\_TABLE\_IMAGE".

- a. At the option **Click to display generated source**, click **View** to view the generated source in another window. A copy of the generated source is shown at the end of Step 4, substep 6g.
- b. Close the window after looking at the generated source.
- c. Accept the **DAD:** name provided, *SCOTT*CW, then click **Test** to display in another window a template file upload form that you can use to test the generated procedure.
- d. To customize the template file upload form, select **Save As...** from your browser's **File** pull-down menu to save the HTML source for editing.
- e. To test the template upload form, enter the following information:
  - For the **ID:** column, enter the number 1 as the row's primary key.
  - For each Oracle Multimedia object column, click **Browse...** and choose the appropriate media to upload to each column of the table. You can choose one or more or all columns to test.
  - For the **DESCRIPTION** column, enter a brief description of the media.
  - Click **Upload media**.

The Code Wizard displays a template completion window with the heading *interMedia Code Wizard: Template Upload Procedure*, and, if the procedure is successful, the message: *Media uploaded successfully*.

- f. Close the window.
- g. Click **Done** on the **Compile procedure and review generated source** window to return to the **Main menu** of the Code Wizard.

A copy of the generated multimedia upload procedure is as follows:

```

CREATE OR REPLACE PROCEDURE UPLOAD_CW_MEDIA_TABLE_IMAGE
( in_ID IN VARCHAR2,
  in_IMAGE IN VARCHAR2 DEFAULT NULL,
  in_THUMB IN VARCHAR2 DEFAULT NULL,

```

```

        in_AUDIO IN VARCHAR2 DEFAULT NULL,
        in_VIDEO IN VARCHAR2 DEFAULT NULL,
        in_MEDIA IN VARCHAR2 DEFAULT NULL,
        in_DESCRIPTION IN VARCHAR2 DEFAULT NULL )
AS
    local_IMAGE ORDSYS.ORDIMAGE := ORDSYS.ORDIMAGE.init();
    local_THUMB ORDSYS.ORDIMAGE := ORDSYS.ORDIMAGE.init();
    local_AUDIO ORDSYS.ORDAUDIO := ORDSYS.ORDAUDIO.init();
    local_AUDIO_ctx RAW( 64 );
    local_VIDEO ORDSYS.ORDVIDEO := ORDSYS.ORDVIDEO.init();
    local_VIDEO_ctx RAW( 64 );
    local_MEDIA ORDSYS.ORDDOC := ORDSYS.ORDDOC.init();
    local_MEDIA_ctx RAW( 64 );
    local_ID CW_MEDIA_TABLE.ID%TYPE := NULL;
    upload_size      INTEGER;
    upload_mimetype VARCHAR2( 128 );
    upload_blob      BLOB;
BEGIN
    --
    -- Update the existing row.
    --
    UPDATE CW_MEDIA_TABLE mtbl
        SET mtbl.IMAGE = local_IMAGE,
            mtbl.THUMB = local_THUMB,
            mtbl.AUDIO = local_AUDIO,
            mtbl.VIDEO = local_VIDEO,
            mtbl.MEDIA = local_MEDIA,
            mtbl.DESCRPTION = in_DESCRIPTION
        WHERE mtbl.ID = in_ID
        RETURN mtbl.ID INTO local_ID;
    --
    -- Conditionally insert a new row if no existing row is updated.
    --
    IF local_ID IS NULL
    THEN
        --
        -- Insert a new row into the table.
        --
        INSERT INTO CW_MEDIA_TABLE ( ID, IMAGE, THUMB, AUDIO, VIDEO, MEDIA,
DESCRIPTION )
            VALUES ( in_ID, local_IMAGE, local_THUMB, local_AUDIO,
                    local_VIDEO, local_MEDIA, in_DESCRIPTION );
    END IF;
    --
    -- Select interMedia object(s) for update.
    --
    SELECT mtbl.IMAGE, mtbl.THUMB, mtbl.AUDIO, mtbl.VIDEO, mtbl.MEDIA INTO
local_IMAGE, local_THUMB, local_AUDIO, local_VIDEO, local_MEDIA
    FROM CW_MEDIA_TABLE mtbl WHERE mtbl.ID = in_ID FOR UPDATE;
    --
    -- Store media data for the column in_IMAGE.
    --
    IF in_IMAGE IS NOT NULL
    THEN
        SELECT dtbl.doc_size, dtbl.mime_type, dtbl.blob_content INTO
            upload_size, upload_mimetype, upload_blob
            FROM MEDIA_UPLOAD_TABLE dtbl WHERE dtbl.name = in_IMAGE;
        IF upload_size > 0
        THEN
            dbms_lob.copy( local_IMAGE.source.localData,

```

```

                upload_blob,
                upload_size );
local_IMAGE.setLocal();
BEGIN
    local_IMAGE.setProperties();
EXCEPTION
    WHEN OTHERS THEN
        local_IMAGE.contentLength := upload_size;
        local_IMAGE.mimeType := upload_mimetype;
    END;
END IF;
DELETE FROM MEDIA_UPLOAD_TABLE dtbl WHERE dtbl.name = in_IMAGE;
END IF;
--
-- Store media data for the column in_THUMB.
--
IF in_THUMB IS NOT NULL
THEN
    SELECT dtbl.doc_size, dtbl.mime_type, dtbl.blob_content INTO
        upload_size, upload_mimetype, upload_blob
        FROM MEDIA_UPLOAD_TABLE dtbl WHERE dtbl.name = in_THUMB;
    IF upload_size > 0
    THEN
        dbms_lob.copy( local_THUMB.source.localData,
            upload_blob,
            upload_size );
        local_THUMB.setLocal();
        BEGIN
            local_THUMB.setProperties();
        EXCEPTION
            WHEN OTHERS THEN
                local_THUMB.contentLength := upload_size;
                local_THUMB.mimeType := upload_mimetype;
            END;
        END IF;
        DELETE FROM MEDIA_UPLOAD_TABLE dtbl WHERE dtbl.name = in_THUMB;
    END IF;
--
-- Store media data for the column in_AUDIO.
--
IF in_AUDIO IS NOT NULL
THEN
    SELECT dtbl.doc_size, dtbl.mime_type, dtbl.blob_content INTO
        upload_size, upload_mimetype, upload_blob
        FROM MEDIA_UPLOAD_TABLE dtbl WHERE dtbl.name = in_AUDIO;
    IF upload_size > 0
    THEN
        dbms_lob.copy( local_AUDIO.source.localData,
            upload_blob,
            upload_size );
        local_AUDIO.setLocal();
        BEGIN
            local_AUDIO.setProperties(local_AUDIO_ctx);
        EXCEPTION
            WHEN OTHERS THEN
                local_AUDIO.mimeType := upload_mimetype;
            END;
        END IF;
        DELETE FROM MEDIA_UPLOAD_TABLE dtbl WHERE dtbl.name = in_AUDIO;
    END IF;

```

```

--
-- Store media data for the column in_VIDEO.
--
IF in_VIDEO IS NOT NULL
THEN
  SELECT dtbl.doc_size, dtbl.mime_type, dtbl.blob_content INTO
         upload_size, upload_mimetype, upload_blob
  FROM MEDIA_UPLOAD_TABLE dtbl WHERE dtbl.name = in_VIDEO;
  IF upload_size > 0
  THEN
    dbms_lob.copy( local_VIDEO.source.localData,
                  upload_blob,
                  upload_size );
    local_VIDEO.setLocal();
  BEGIN
    local_VIDEO.setProperties(local_VIDEO_ctx);
  EXCEPTION
    WHEN OTHERS THEN
      local_VIDEO.mimeType := upload_mimetype;
  END;
  END IF;
  DELETE FROM MEDIA_UPLOAD_TABLE dtbl WHERE dtbl.name = in_VIDEO;
END IF;
--
-- Store media data for the column in_MEDIA.
--
IF in_MEDIA IS NOT NULL
THEN
  SELECT dtbl.doc_size, dtbl.mime_type, dtbl.blob_content INTO
         upload_size, upload_mimetype, upload_blob
  FROM MEDIA_UPLOAD_TABLE dtbl WHERE dtbl.name = in_MEDIA;
  IF upload_size > 0
  THEN
    dbms_lob.copy( local_MEDIA.source.localData,
                  upload_blob,
                  upload_size );
    local_MEDIA.setLocal();
  BEGIN
    local_MEDIA.setProperties(local_MEDIA_ctx, FALSE);
  EXCEPTION
    WHEN OTHERS THEN
      local_MEDIA.contentLength := upload_size;
      local_MEDIA.mimeType := upload_mimetype;
  END;
  END IF;
  DELETE FROM MEDIA_UPLOAD_TABLE dtbl WHERE dtbl.name = in_MEDIA;
END IF;
--
-- Update interMedia objects in the table.
--
UPDATE CW_MEDIA_TABLE mtbl
  SET mtbl.IMAGE = local_IMAGE,
      mtbl.THUMB = local_THUMB,
      mtbl.AUDIO = local_AUDIO,
      mtbl.VIDEO = local_VIDEO,
      mtbl.MEDIA = local_MEDIA
  WHERE mtbl.ID = in_ID;
--
-- Display the template completion message.
--

```

```

    http.print( '&lt;html&gt;' );
    http.print( '&lt;title&gt;interMedia Code Wizard: Template Upload
Procedure&lt;/title&gt;' );
    http.print( '&lt;body&gt;' );
    http.print( '&lt;h2&gt;&lt;i&gt;inter&lt;/i&gt;Media Code Wizard:
Template Upload Procedure&lt;/h2&gt;' );
    http.print( 'Media uploaded successfully.' );
    http.print( '&lt;/body&gt;' );
    http.print( '&lt;/html&gt;' );

END UPLOAD_CW_MEDIA_TABLE_IMAGE;

```

This sample multimedia upload procedure declares the following input parameters and variables:

1. In the declaration section, the procedure declares seven input parameters: `in_ID`, `in_IMAGE`, `in_THUMB`, `in_AUDIO`, `in_VIDEO`, `in_MEDIA`, and `in_DESCRIPTION`, then initializes the last six to `NULL`.
2. In the subprogram section, the following variables are declared:
  - The variables `local_IMAGE` and `local_THUMB` are assigned the data type `ORDSYS.ORDIMAGE` and initialized with an empty BLOB using the `ORDIMAGE.init()` method.
  - The variable `local_AUDIO` is assigned the data type `ORDSYS.ORDAUDIO` and initialized with an empty BLOB using the `ORDAUDIO.init()` method. Also a context variable `local_AUDIO_ctx` is assigned the data type `RAW(64)`.
  - The variable `local_VIDEO` is assigned the data type `ORDSYS.ORDVIDEO` and initialized with an empty BLOB using the `ORDVIDEO.init()` method. Also, a context variable `local_VIDEO_ctx` is assigned the data type `RAW(64)`.
  - The variable `local_MEDIA` is assigned the data type `ORDSYS.ORDDOC` and initialized with an empty BLOB using the `ORDDOC.init()` method. Also, a context variable `local_MEDIA_ctx` is assigned the data type `RAW(64)`.
  - The variable `local_ID` takes the same data type as the `ID` column in the table `CW_MEDIA_TABLE` and is initialized to `NULL`.
  - Three additional variables are declared `upload_size`, `upload_mimetype`, and `upload_blob`, which are later given values from comparable column names `doc_size`, `mime_type`, and `blob_content` from the document table `MEDIA_UPLOAD_TABLE` using a `SELECT` statement. This is all in preparation for copying the content of the image, thumb, audio, video, and media BLOB data to the respective `ORDSYS.ORDIMAGE.source.localData`, `ORDSYS.ORDIMAGE.source.localData`, `ORDSYS.ORDAUDIO.source.localData`, `ORDSYS.ORDVIDEO.source.localData`, and `ORDSYS.ORDDOC.source.localData` attributes.

Within the outer `BEGIN...END` executable statement section, the following operations are executed:

1. Update the existing row in the table `CW_MEDIA_TABLE` for the `IMAGE`, `THUMB`, `AUDIO`, `VIDEO`, `MEDIA`, and `DESCRIPTION` columns and return the value of

`local_ID` where the value of the `ID` column is the value of the `in_ID` input parameter.

2. If the value returned of `local_ID` is `NULL`, conditionally insert a new row into the table `CW_MEDIA_TABLE` and initialize the instance of the `ORDImage` object type in the `IMAGE` column with an empty `BLOB`, the instance of the `ORDImage` object type in the `THUMB` column with an empty `BLOB`, the instance of the `ORDAudio` object type in the `AUDIO` column with an empty `BLOB`, the instance of the `ORDVideo` object type in the `VIDEO` column with an empty `BLOB`, and the instance of the `ORDDoc` object type in the `MEDIA` column with an empty `BLOB`.
3. Select the `ORDImage` object column `IMAGE`, `ORDImage` object column `THUMB`, `ORDAudio` object column `AUDIO`, `ORDVideo` object column `VIDEO`, and `ORDDoc` object column `MEDIA` in the table `CW_MEDIA_TABLE` for update where the value of the `ID` column is the value of the `in_ID` input parameter.
4. Select a row for the `doc_size`, `mime_type`, and `blob_content` columns from the document table and pass the values to the `upload_size`, `upload_mimetype`, and `upload_blob` variables where the value of the `Name` column is the value of one of the following input parameters `in_IMAGE`; `in_THUMB`; `in_AUDIO`; `in_VIDEO`; or `in_MEDIA`.
5. Perform a DBMS LOB copy of the `BLOB` data from the table `MEDIA_UPLOAD_TABLE` into the `ORDSYS.ORDIMAGE.source.localData`, `ORDSYS.ORDIMAGE.source.localData`, `ORDSYS.ORDAUDIO.source.localData`, `ORDSYS.ORDVIDEO.source.localData`, and `ORDSYS.ORDDoc.source.localData` attribute, then call the `setLocal()` method to indicate that the image, audio, and video data are stored locally in the `BLOB`, and `ORDImage`, `ORDAudio`, `ORDVideo`, and `ORDDoc` methods should look for corresponding data in the `source.localData` attribute.
6. In the inner executable block, call the respective `ORDImage`, `ORDAudio`, `ORDVideo`, and `ORDDoc` `setProperties()` method to read the image, audio, and video data to get the values of the object attributes and store them in the `image`, `audio`, `video`, and `media` object attributes for the `ORDImage`, `ORDAudio`, `ORDVideo`, and `ORDDoc` objects.
7. If the `setProperties()` call fails, catch the exception and call the `contentLength()` method to get the size of the media data and call the `mimeType()` method to get the MIME type of the media data.
8. Delete the row of data from the document table `MEDIA_UPLOAD_TABLE` that was copied to the row in the table `CW_MEDIA_TABLE` where the value of the `Name` column is the value of the respective `in_IMAGE`, `in_THUMB`, `in_AUDIO`, `in_VIDEO`, and `in_MEDIA` input parameter.
9. Update the `ORDImage` object `IMAGE` column, the `ORDImage` object `THUMB` column, the `ORDAudio` object `AUDIO` column, the `ORDVideo` object `VIDEO` column, and the `ORDDoc` object `MEDIA` column in the table `CW_MEDIA_TABLE` with the content of the variables `local_IMAGE`, `local_THUMB`, `local_AUDIO`, `local_VIDEO`, and `local_MEDIA` respectively, where the value of the `ID` column is the value of the `in_ID` input parameter.
10. Display a completion message on the HTML page to indicate that the media uploaded successfully using the `http.print` function from the PL/SQL Web Toolkit.



**Step 5 Create and test a media retrieval.**

Select **Create media retrieval procedure** from the **Main menu**, then click **Next**.

1. Select the database table and procedure type.
  - a. Click **CW\_MEDIA\_TABLE**.
  - b. Click **Standalone procedure**.
  - c. Click **Next**.
2. Select the media column and key column.
  - a. Ensure that one the following object columns is checkmarked. For example, if you loaded media data into the media column in Step 4, substep 6e, then select the **MEDIA (ORDDOC)** column.
  - b. Click **ID (Primary key)**.
  - c. Click **Next**.
3. Select the procedure name and parameter name.
  - a. Accept the procedure name provided, **GET\_CW\_MEDIA\_TABLE\_IMAGE**.
  - b. Accept the parameter name provided, **MEDIA\_ID**.
  - c. Click **Create procedure in the database**.
  - d. Click **Next**.

4. Review the following selected procedure creation options:

Procedure type:	Standalone
Table name:	CW_MEDIA_TABLE
Key column:	ID
Media column:	IMAGE (ORDDOC)
Procedure name:	GET_CW_MEDIA_TABLE_IMAGE
Parameter name:	MEDIA_ID
Function:	Create procedure in the database

Click **Finish**.

5. Compile the procedure and review the generated source.

The Code Wizard displays the following message: Procedure created successfully: GET\_CW\_MEDIA\_TABLE\_IMAGE.

- a. Click **View** to view the generated source in another window. Close the window after looking at the generated source. A copy of the generated source is shown at the end of this step.
- b. Review the URL format used to retrieve images using the **GET\_CW\_MEDIA\_TABLE\_IMAGE** procedure.
- c. Enter the number 1 as the Key parameter, then click **Test** to test the procedure by retrieving the image uploaded previously.
- d. The retrieved image is displayed in another window.
- e. Close the window.
- f. Click **Done** to return to the **Main menu**.

---



---

**Note:** A generated media retrieval script, unlike the multiple media upload script shown at the end of Step 4, handles only the type of media data designed for that Oracle Multimedia object type. To retrieve media data stored in other Oracle Multimedia object types, generate a retrieval script for each desired media data type and add it to your PL/SQL package.

---



---

A copy of the generated media retrieval procedure is as follows:

```

CREATE OR REPLACE PROCEDURE GET_CW_MEDIA_TABLE_MEDIA ( MEDIA_ID
  IN VARCHAR2 )
AS
  localObject ORDSYS.ORDDOC;
  localBlob  BLOB;
  localBfile BFILE;
  httpStatus NUMBER;
  lastModDate VARCHAR2(256);

BEGIN
  --
  -- Retrieve the object from the database into a local object.
  --
  BEGIN
    SELECT mtbl.MEDIA INTO localObject FROM CW_MEDIA_TABLE mtbl
      WHERE mtbl.ID = MEDIA_ID;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      ordplsgwyutil.resource_not_found( 'MEDIA_ID', MEDIA_ID );
      RETURN;
  END;
  --
  -- Check the update time if the browser sent an If-Modified-Since header.
  --
  IF ordplsgwyutil.cache_is_valid( localObject.getUpdateTime() )
  THEN
    owa_util.status_line( ordplsgwyutil.http_status_not_modified );
    RETURN;
  END IF;
  --
  -- Figure out where the image is.
  --
  IF localObject.isLocal() THEN
    --
    -- Data is stored locally in the localData BLOB attribute.
    --
    localBlob := localObject.getContent();
    owa_util.mime_header( localObject.getMimeType(), FALSE );
    ordplsgwyutil.set_last_modified( localObject.getUpdateTime() );
    owa_util.http_header_close();
    IF owa_util.get_cgi_env( 'REQUEST_METHOD' ) &lt;&gt; 'HEAD' THEN
      wpg_docload.download_file( localBlob );
    END IF;
  ELSIF UPPER( localObject.getSourceType() ) = 'FILE' THEN
    --
    -- Data is stored as a file from which ORDSource creates
    -- a BFILE.
    --

```

```

localBfile := localObject.getBFILE();
owa_util.mime_header( localObject.getMimeType(), FALSE );
ordplsgwyutil.set_last_modified( localObject.getUpdateTime() );
owa_util.http_header_close();
IF owa_util.get_cgi_env( 'REQUEST_METHOD' ) &lt;&gt; 'HEAD' THEN
    wpg_docload.download_file( localBfile );
END IF;

ELSIF UPPER( localObject.getSourceType() ) = 'HTTP' THEN
    --
    -- The image is referenced as an HTTP entity, so we have to
    -- redirect the client to the URL that ORDSource provides.
    --
    owa_util.redirect_url( localObject.getSource() );
ELSE
    --
    -- The image is stored in an application-specific data
    -- source type for which no default action is available.
    --
    NULL;
END IF;
END GET_CW_MEDIA_TABLE_MEDIA;

```

See the description at the end of the generated image retrieval procedure in [Section 4.2](#), Step 6, after substep 5e. The only difference between these two retrieval procedures is the type of object being retrieved, an `ORDImage` object type versus an `ORDDoc` object type.

## 4.4 Known Restrictions of the Oracle Multimedia Code Wizard

The following restrictions are known for the Oracle Multimedia Code Wizard:

- Tables with composite primary keys are not supported.

To use a table with a composite primary key, create an upload or download procedure, then edit the generated source to support all the primary key columns. For example, for a media retrieval procedure, this might involve adding an additional parameter, then specifying that parameter in the `where` clause of the `SELECT` statement.

- User object types containing embedded Oracle Multimedia object types are not recognized by the Oracle Multimedia Code Wizard.



---

---

# Oracle Multimedia IMExample Sample Application

This chapter describes the Oracle Multimedia IMExample sample application and explains how Oracle Multimedia Java Classes are used to create this Java sample application.

This chapter assumes the following:

- You are familiar with developing Java applications using Oracle Multimedia Java classes.
- You have already installed and configured the Oracle Multimedia IMExample sample application.

The sample application files and `README.txt` file are located at:

`<ORACLE_HOME>/ord/im/demo/java` (on Linux and UNIX)

`<ORACLE_HOME>\ord\im\demo\java` (on Windows)

The following subsections describe how to run the IMExample application. See the `README.txt` file for additional requirements and instructions on installing, compiling, and running this sample application.

See [Chapter 3](#) for descriptions of the following Photo Album sample applications:

- Oracle Multimedia PL/SQL Web Toolkit Photo Album application
- Oracle Multimedia Java Servlet Photo Album application
- Oracle Multimedia JSP Photo Album application
- Oracle Multimedia ASP/VBScript Photo Album application

This suite of Web applications use PL/SQL scripts, Java servlet files, JSP files, and ASP/VBScript files to demonstrate various ways to upload and retrieve media using Oracle Multimedia object types.

See [Chapter 4](#) for a description of the Oracle Multimedia Code Wizard sample application, a media upload and retrieval Web application for the PL/SQL Gateway.

## 5.1 Overview of the IMExample Application

The IMExample application lets you retrieve multimedia data from the Oracle Database sample schemas, save to a file, play, and delete from the sample schema image, audio, video, and testimonial data using the respective Oracle Multimedia object types, `OrdImage`, `OrdAudio`, `OrdVideo`, and `OrdDoc`, by product ID for rows in the `PM.ONLINE_MEDIA` table. [Section 5.2](#) describes the class files and shows code

examples that illustrate how Oracle Multimedia object types and methods and other Oracle objects are used.

## 5.2 Description of the IMExample Application

The IMExample sample application, when compiled, creates the following class files:

- `IMExample` -- creates the sample application frame and maintains the only connection to the database. This class is the entry point of this sample application.
- `IMExampleFrame` -- extends the `JFrame` class and displays the main frame.
- `IMLoginDialog` -- extends the `JDialog` class, displays the login dialog box, and creates the connection to the database.
- `IMExampleQuery` -- performs the SQL `SELECT` statement to retrieve rows of the `OE.PRODUCT_INFORMATION` table and displays the content of the table by product ID.
- `IMProductDialog` -- extends the `JDialog` class, shows a dialog box to display detailed information for a particular product, including the product ID, product name, and product description. The `IMProductDialog` class also retrieves and displays the product photo, audio, video, and testimonial data within the appropriate panel. It supports retrieving, saving, deleting, and playing the media data. And, it allows for applying changes or rolling back changes to the media objects.
- `IMImagePanel` -- extends the `IMMediaPanel` class, displays the product photo and its attributes: MIME type, image height, image width, and content length, and if it applies, generates and displays the thumbnail image and displays lists for reading and writing metadata.
- `IMAudioPanel` -- extends the `IMMediaPanel` class and displays the product audio and its attributes: MIME type, duration of the audio, and content length.
- `IMVideoPanel` -- extends the `IMMediaPanel` class and displays the product video and its attributes: MIME type, frame height, frame width, duration of the video, and content length.
- `IMDocPanel` -- extends the `IMMediaPanel` class and displays the product testimonials and its attributes: MIME type and content length.
- `IMLoadFile` -- loads a media stream (photo, video, audio, and testimonials), from a file to the `PM.ONLINE_MEDIA` table in the database, and if necessary, inserts a row and initializes the media objects, then updates the media data, sets the media attributes, and generates and updates the thumbnail image if loading a photo.
- `IMSaveFile` -- saves a media stream from the database to a target file.
- `IMMediaPanel` -- extends the `JPanel` class, lays out the common components for the photo, audio, video, and doc panel with load, save, delete, and play check boxes, initializes the MIME configuration file for each operating system that lists plug-in players and media and their associated MIME types, plays the data stream associated with the MIME type of the media, and allows users to specify their own player to play the media data stream.

The major flow among these class files is: `IMExample` to `IMExampleFrame` to `IMLoginDialog` (login) to `IMExampleFrame.showDefaultTable( )` to `IMExampleQuery` to `IMProductDialog` to one group of classes (`IMImagePanel`, `IMAudioPanel`, `IMVideoPanel`, `IMDocPanel`), and finally to the last group of classes (`IMLoadFile`, `IMSaveFile`, `IMMediaPanel`).

The remaining class files in this sample application include:

- `IMUtil` -- includes common utilities such as a method to generate and update thumbnail images, wrapper methods for each `setProperty()` method of each media object type to separate the exceptions caused by unrecognizable formats, and cleanup methods to close the following: `ResultSet`, `Statement`, input stream and its reader, and output stream and its writer.
- `IMMIME` -- loads and stores the mapping between plug-in players and the MIME type.
- `IMResultSetTableModel` -- extends the `AbstractTableModel` class and controls the display of the `OE.PRODUCT_INFORMATION` table.
- `IMMessage` -- displays various messages for the sample application and classifies the message level as error, warning, or suggestion.
- `IMMessageResource` -- extends the `java.util.ListResourceBundle` class and contains the actual message text for all messages.
- `IMOptionPane` -- extends and puts into subclasses the `JOptionPane` class in order to add an accessible description message to the displayed dialog box.
- `IMGetMetadataDialog` -- extends the `JDialog` class, and retrieves the metadata from an image into an XML document and then displays the XML document in a `JTree` form.
- `IMPutMetadataDialog` -- extends the `JDialog` class, and constructs an XMP packet to write into an image from user inputs.
- `XMLTreeNode` -- extends the `DefaultMutableTreeNode` class, and creates a tree representation of an XML node.
- `XMLTreeView` -- extends the `JPanel` class, and displays an XML document as a tree.
- `IMFileChooser` -- extends the `JFileChooser` class, and inherits from the `JFileChooser` class in order to add the button mnemonic and accessible description.
- `IMConstants` -- describes the `IMConstants` interface, which contains all the constants for column names, media types, message types, and message dialog titles.
- `IMAttrTableModel` -- extends and puts into subclasses the `DefaultTableModel` class in order to provide the table model for displaying media attributes, and overwrites the `isCellEditable()` method to make the cells uneditable.
- `FocusedJTextField` -- extends and puts into subclasses the `JTextField` class and overwrites the `isFocusTraversable()` method to allow it to gain focus when it is set to uneditable.
- `FocusedJTextArea` -- extends and puts into subclasses the `JTextArea` class and overwrites the `isFocusTraversable()` method to allow it to gain focus when it is set to uneditable; also overrides the `isManagingFocus()` method to force the `JTextArea` class not to handle a TAB key operation.
- `FocusedJPanel` -- extends and puts into subclasses the `JPanel` class and overwrites the `isFocusTraversable()` method to allow it to gain focus.
- `FocusedJLabel` -- extends and puts into subclasses the `JLabel` class, overwrites the `isFocusTraversable()` method, and adds a focus listener to allow it to gain focus.

- `BooleanRenderer` -- extends the `JCheckBox` class and renders Boolean objects as `JCheckBox` (a checkbox) in a `JTable` (two-dimensional table format). This class also sets the `AccessibleName` and `AccessibleDescription` properties by setting the tooltip to support accessibility.
- `IMStreamAbsorber` -- extends the `Thread` class and runs as a separate thread to consume an input stream. This is useful when a plug-in application is loaded and it writes something out to, for example, a standard error, without consuming the application's output, the application may be unable to continue.
- `IMTable` -- extends and puts into subclasses the `JTable` class and overwrites the `isManagingFocus()` method to avoid letting the table handle a TAB key operation.
- `IMTableRenderer` -- extends the `DefaultTableCellRenderer` class and renders the `PRODUCT_ID`, `PRODUCT_NAME`, and `PRODUCT_DESCRIPTION` columns to add accessibility information, and sets the customized display.
- `IMUIUtil` -- includes common GUI utilities.

## 5.2.1 IMProductDialog Class

This class defines the following methods followed by a description of what each method does:

- The `loadMedia()` method to retrieve the media objects from the database. This method performs a SQL `SELECT...FOR UPDATE` statement on the `PM.ONLINE_MEDIA` table where the `PRODUCT_ID` column is a parameter marker; then this class uses the `getORADData` and `getORADDataFactory` interfaces to get the media data objects from the result set.
- The `displayMedia()` method to display the media data, which in turn calls the corresponding media display methods `displayImage()`, `displayAudio()`, `displayVideo()`, and `displayDoc()`.
- The `displayImage()` method calls the `IMImagePanel.display()` method to display the image data attributes, display the thumbnail image, and display the full sized image using a media player that supports this MIME type.
- The `displayAudio()` method calls the `IMAudioPanel.display()` method to display the audio data attributes and play the audio stream using a media player that supports this MIME type.
- The `displayVideo()` method calls the `IMVideoPanel.display()` method to display the video data attributes and play the video stream using a media player that supports this MIME type.
- The `displayDoc()` method calls the `IMDocPanel.display()` method to display the testimonial data attributes and play the testimonial data using a media player that supports this MIME type.

The following code example shows the `loadMedia()`, `displayMedia()`, `displayImage()`, `displayAudio()`, `displayVideo()`, and `displayDoc()` methods, and highlights in bold the SQL query statements and areas in the code where Oracle Multimedia and other Oracle object types and methods are used.

```
private void loadMedia() throws SQLException, IOException
{
    String sQuery =
        "select product_photo, product_thumbnail, product_audio, product_video, " +
        "product_testimonials from pm.online_media where product_id = ? for update";
```



```

OracleConnection conn = null;
OracleResultSet rs = null;
OraclePreparedStatement pstmt = null;
boolean isInsertNeeded = false;
byte[] ctx[] = new byte[1][64];

try
{
    conn = IMExample.getDBConnection();

    pstmt = (OraclePreparedStatement)conn.prepareStatement(sQuery);
    pstmt.setInt(1, m_iProdId);
    rs = (OracleResultSet)pstmt.executeQuery();
    if (rs.next() == true)
    {
        m_img = (OrdImage)rs.getORADData(1, OrdImage.getORADDataFactory());
        m_imgThumb = (OrdImage)rs.getORADData(2, OrdImage.getORADDataFactory());
        m_aud = (OrdAudio)rs.getORADData(3, OrdAudio.getORADDataFactory());
        m_vid = (OrdVideo)rs.getORADData(4, OrdVideo.getORADDataFactory());
        m_doc = (OrdDoc)rs.getORADData(5, OrdDoc.getORADDataFactory());
    }

    displayMedia();

    rs.close();
    pstmt.close();
}
finally
{
    IMUtil.cleanup(rs, pstmt);
}
}

private void displayMedia() throws SQLException, IOException
{
    displayImage();
    displayAudio();
    displayVideo();
    displayDoc();
}

/**
 * Add the product photo panel.
 */
private void displayImage() throws SQLException, IOException
{
    m_jImgPanel = new IMImagePanel(this,
        m_img, m_imgThumb, m_iProdId, m_colorFieldBg);
    m_jImgPanel.display();
    m_jImgPanel.getAccessibleContext().setAccessibleName
        ("Product photo panel");
    m_jImgPanel.getAccessibleContext().setAccessibleDescription
        ("Product photo panel with an image icon on the left, " +
        "image attribute panel in the middle and image control" +
        "panel on the right.");

    m_jMediaPanel.add(m_jImgPanel);

    Component jImgFocus = m_jImgPanel.getFirstFocusComponent();
}

```

```
/**
 * Add the product audio panel.
 */
private void displayAudio() throws SQLException, IOException
{
    m_jAudPanel = new IMAudioPanel(this, m_aud, m_iProdId, m_colorFieldBg);
    m_jAudPanel.display();
    m_jAudPanel.getAccessibleContext().setAccessibleName
        ("Product audio panel");
    m_jAudPanel.getAccessibleContext().setAccessibleDescription(
        "Product audio panel with an audio icon at the left, " +
        "audio attribute panel in the middle and audio control" +
        "panel at the right.");
    m_jMediaPanel.add(m_jAudPanel);
}

/**
 * Add the product video panel.
 */
private void displayVideo() throws SQLException, IOException
{
    m_jVidPanel = new IMVideoPanel(this, m_vid, m_iProdId, m_colorFieldBg);
    m_jVidPanel.display();
    m_jVidPanel.getAccessibleContext().setAccessibleName
        ("Product audio panel");
    m_jVidPanel.getAccessibleContext().setAccessibleDescription(
        "Product audio panel with an video icon at the left, " +
        "video attribute panel in the middle and video control" +
        "panel at the right.");
    m_jMediaPanel.add(m_jVidPanel);
}

/**
 * Add the product testimonials panel.
 */
private void displayDoc() throws SQLException, IOException
{
    m_jDocPanel = new IMDocPanel(this, m_doc, m_iProdId, m_colorFieldBg);
    m_jDocPanel.display();
    m_jDocPanel.getAccessibleContext().setAccessibleName
        ("Product testimonials panel");
    m_jDocPanel.getAccessibleContext().setAccessibleDescription(
        "Product testimonials panel with an document icon at the left, " +
        "testimonials attribute panel in the middle and testimonials control" +
        "panel at the right.");
    m_jMediaPanel.add(m_jDocPanel);
}
```

See [IMImagePanel Class](#), [IMAudioPanel Class](#), [IMVideoPanel Class](#), and [IMDocPanel Class](#) for code examples of the corresponding `m_jXxxPanel.display()` methods, where `Xxx` represents the particular media data type, `Img`, `Aud`, `Vid`, or `Doc`.

## 5.2.2 IMImagePanel Class

This class displays the image panel, the product photo and its attributes, and the thumbnail image, and lists for reading and writing metadata. What follows is a more detailed description of each of the methods that are defined and what each method does:

- The `display()` method, which first calls the `insertProperty()` method, which calls the Oracle Multimedia image object type methods `getMimeType()`, `getHeight()`, `getWidth()`, and `getContentlength()` to get the attributes of the image to display in a table, and lays out the user interface components for reading and writing image metadata.
- For supported formats, the class displays the product photo thumbnail image, which is generated by calling the `IMUtil.generateThumbnail()` method to create the thumbnail image from the product photo.
- The `addThumbnail()` method to show the new thumbnail image.
- The `changeThumbnail()` method to change the thumbnail image.
- The `saveToFile()` method to save the photo to a file.
- The `deleteMedia()` method to delete the product photo image and its thumbnail image from the database by setting the image object type columns to empty using the `OrdImage.init()` method.
- The `play()` media method to show the image using a media player.
- The `setMedia()` method to set the photo and thumbnail object.
- The `notExist()` method checks to see if the image data exists and returns true if the BLOB is empty or is not associated with an existing BFILE; otherwise, it returns false.
- The `getDataInByteArray()` method retrieves image data into a byte array by calling the Oracle Multimedia `importData()` method first for the BFILE and returns the results of calling the Oracle Multimedia `getDataInByteArray()` method.
- The `refreshPanel()` method refreshes the display when updating the photo image, attributes, and thumbnail image.
- The `getFirstFocusComponent()` method enforces the correct focus order.
- The `emptyPanel()` method clears the icon and attribute panel.
- The `showMetadata()` method to pop up a window for displaying metadata for the selected type.
- The `writeMetadata()` method to display the write metadata dialog.

The following code example includes the `display()`, `insertProperty()`, `notExist()`, `getDataInByteArray()`, and `refreshPanel()` methods, and highlights in bold any SQL query statements and areas in the code where Oracle Multimedia and other Oracle object types and methods are used:

```
void display() throws IOException, SQLException
{
    addControlPane();

    if (notExist(m_img))
    {
        // The image does not exist.
        m_hasMedia = false;
        layoutEmpty(s_sNotExist);
    }
    else
    {
        m_hasMedia = true;
    }
}
```

```
// If image exists, try to show the attributes.
if (insertProperty())
{
    // Show the thumbnail image.
    // If the thumbnail image does not exist, generate it first.
    if (m_imgThumb != null)
    {
        String sFormat = m_imgThumb.getFormat();

        if (notExist(m_imgThumb) ||
            ( !"JFIF".equalsIgnoreCase(sFormat) ) &&
            !"GIF".equalsIgnoreCase(sFormat)
            ))
        {
            m_imgThumb = IMUtil.generateThumbnail(m_iProdId, m_img, m_imgThumb);
        }

        byte[] thumbnail = getDataInByteArray(m_imgThumb);
        addThumbnail(thumbnail);
    }
    else
    {
        m_imgThumb = IMUtil.generateThumbnail(m_iProdId, m_img, m_imgThumb);
        byte[] thumbnail = getDataInByteArray(m_imgThumb);
        addThumbnail(thumbnail);
    }
}
}
}
.
.
.
boolean insertProperty() throws SQLException
{
    boolean isFormatSupported = false;
    String sMimeType = m_img.getMimeType();

    if (sMimeType == null)
        isFormatSupported = IMUtil.setProperties(m_img);
    else
        isFormatSupported = true;

    if (!isFormatSupported)
    {
        layoutEmpty(s_sNotSupported);
    }
    else
    {
        Object[][] data =
        {
            {"MIME Type", m_img.getMimeType()},
            {"Height", new Integer(m_img.getHeight()).toString()},
            {"Width", new Integer(m_img.getWidth()).toString()},
            {"Content Length", new Integer(m_img.getContentLength()).toString()}
        };
        .
        .
        .
    }
}
```

```

        return isFormatSupported;
    }
    .
    .
    .
    static boolean notExist(OrdImage img) throws SQLException, IOException
    {
        if (img == null)
            return true;
        else
        {
            if (img.isLocal() && (img.getDataInByteArray() == null))
                return true;
            else if (!img.isLocal() && (":///".equals(img.getSource())))
                return true;
            else
            {
                if (!img.isLocal())
                {
                    BFILE bfile = img.getBFILE();
                    if (!bfile.fileExists())
                        return true;
                    else
                        return false;
                }
                else
                    return false;
            }
        }
    }
    .
    .
    .
    static byte[] getDataInByteArray(OrdImage img) throws SQLException, IOException
    {
        if (notExist(img))
            return null;
        else
        {
            if (!img.isLocal())
            {
                byte[] ctx[] = new byte[1][4000];
                try
                {
                    img.importData(ctx);
                }
                catch (SQLException e)
                {
                    new IMessage(IMConstants.ERROR, "MEDIA_SOURCE_ERR", e);
                    return null;
                }
            }
            return img.getDataInByteArray();
        }
    }
    .
    .
    .
    void refreshPanel(boolean isFormatSupported) throws SQLException, IOException

```

```
{
    m_hasMedia = true;
    if (isFormatSupported)
    {
        if (m_jAttrTbl == null)
        {
            m_jAttrPane.remove(m_jEmpty);
            m_jIconPane.remove(m_jIcon);

            byte[] thumbnail = getDataInByteArray(m_imgThumb);
            addThumbnail(thumbnail);

            insertProperty();
        }
        else
        {
            byte[] thumbnail = getDataInByteArray(m_imgThumb);
            changThumbnail(thumbnail);

            m_jAttrTbl.setValueAt(m_img.getMimeType(), 0, 1);
            m_jAttrTbl.setValueAt(new Integer(m_img.getHeight()).toString(), 1, 1);
            m_jAttrTbl.setValueAt(new Integer(m_img.getWidth()).toString(), 2, 1);
            m_jAttrTbl.setValueAt(new Integer(m_img.getContentLength()).toString(), 3, 1);
        }
    }
    .
    .
    .
}
```

### 5.2.3 IMGetMetadataDialog Class

This class shows a dialog to display detailed information for metadata in a product photograph. This class also defines the `displayMetadata()` method and describes what it does.

The `displayMetadata()` method retrieves metadata from the image by using the Oracle Multimedia `OrdImage` `getMetadata()` method, and then displays the metadata.

The following code example includes the `displayMetadata()` method, and highlights in bold any SQL query statements and areas in the code where Oracle Multimedia and other Oracle object types and methods are used:

```
private void displayMetadata(String sMetaType)
{
    XMLDocument doc = null;
    try
    {
        //
        // Retrieves the metadata into an XMLType array
        //
        XMLType xmlList[] = m_img.getMetadata(sMetaType);

        if (xmlList.length == 1)
        {
            DOMParser parser = new DOMParser();
            parser.setValidationMode(XMLConstants.NONVALIDATING);
            parser.setPreserveWhitespace(false);
            parser.parse(new StringReader(XMLType.createXML(xmlList[0]).getStringVal()));
            doc = parser.getDocument();
        }
    }
}
```

```

    }
  }
  .
  .
  .
}

```

## 5.2.4 IMPutMetadataDialog Class

This class shows a dialog to write metadata into a product photograph. This class also defines the `writeMetadata()` method and describes what it does.

The `writeMetadata()` method writes XMP metadata into the image metadata by using the Oracle Multimedia `OrdImage` `putMetadata()` method.

The following code example includes the `writeMetadata()` method, and highlights in bold any SQL query statements and areas in the code where Oracle Multimedia and other Oracle object types and methods are used:

```

void writeMetadata()
{
    try
    {
        //
        // Let the StringBuffer to hold the XMP packet
        //
        StringBuffer sb = new StringBuffer(
            "<xmpMetadata xmlns=\"http://xmlns.oracle.com/ord/meta/xmp\" "
            "+ \" xsi:schemaLocation=\"http://xmlns.oracle.com/ord/meta/xmp \"
            "+ \" http://xmlns.oracle.com/ord/meta/xmp\" "
            "+ \" xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" > "
            "+ \" <rdf:RDF xmlns:rdf=\"http://www.w3.org/1999/02/22-rdf-syntax-ns#\"> "
            "+ \" <rdf:Description about=\"\" xmlns:dc=\"http://purl.org/dc/elements/1.1/\"> "
            );

        String str = null;
        if ( (str=m_jTitleField.getText()) != null)
            sb.append("<dc:title>" + str + "</dc:title>");
        if ( (str=m_jCreatorField.getText()) !=null)
            sb.append("<dc:creator>" + str + "</dc:creator>");
        if ( (str=m_jDateField.getText()) !=null)
            sb.append("<dc:date>" + str + "</dc:date>");
        if ( (str=m_jDescriptionField.getText()) !=null)
            sb.append("<dc:description>" + str + "</dc:description>");
        if ( (str=m_jCopyrightField.getText()) !=null)
            sb.append("<dc:rights>" + str + "</dc:rights>");

        sb.append("</rdf:Description></rdf:RDF></xmpMetadata>");

        XMLType xmp = XMLType.createXML(IMEExample.getDBConnection(), sb.toString(),
            "\"http://xmlns.oracle.com/ord/meta/xmp\", true, true);

        //
        // Make sure the image data is local
        //
        if (!m_img.isLocal())
        {
            byte[] ctx[] = new byte[1][4000];
            m_img.importData(ctx);
        }
    }
}

```

```
//  
// Call Ordimage.putMetadata  
//  
m_img.putMetadata(xmp, "XMP", "utf-8");  
  
this.dispose();  
}  
.  
.  
.  
}
```

## 5.2.5 IMVideoPanel Class

This class displays the video panel, the product video, and its attributes. This class is identical in structure and functions similarly to the `IMImagePanel` class. See [IMImagePanel Class](#) for descriptions of methods.

The following code example includes the `display()`, `insertProperty()`, `notExist()`, `getDataInByteArray()`, and `refreshPanel()` methods, and highlights in bold any SQL query statements and areas in the code where Oracle Multimedia and other Oracle object types and methods are used:

```
void display() throws IOException, SQLException  
{  
    addControlPane();  
  
    // Set the video icon.  
    m_jIcon = new JLabel(new ImageIcon(IMExampleFrame.class.getResource("OrdVideo.gif")));  
    m_jIcon.setLabelFor(m_jAttrPane);  
  
    m_jIconPane.add(m_jIcon, BorderLayout.CENTER);  
  
    if (notExist())  
    {  
        // The video does not exist.  
        m_hasMedia = false;  
        layoutEmpty(s_sNotExist);  
    }  
    else  
    {  
        m_hasMedia = true;  
        // If the video exists, try to show the attributes.  
        insertProperty();  
    }  
}  
.  
.  
.  
  
boolean insertProperty() throws SQLException  
{  
    boolean isFormatSupported = false;  
    String sMimeType = m_vid.getMimeType();  
  
    if (sMimeType == null)  
        isFormatSupported = IMUtil.setProperties(m_vid);  
    else  
        isFormatSupported = true;
```



```

if (!isFormatSupported)
{
    layoutEmpty(s_sNotSupported);
}
else
{
    Object[][] data =
    {
        {"MIME Type", m_vid.getMimeType()},
        {"Height", new Integer(m_vid.getHeight().toString())},
        {"Width", new Integer(m_vid.getWidth().toString())},
        {"Duration", new Integer(m_vid.getVideoDuration().toString())},
        {"Content Length", new Integer(m_vid.getContentLength().toString())}
    };
    .
    .
    .
}

return isFormatSupported;
}
.
.
.
boolean notExist() throws SQLException, IOException
{
    if (m_vid == null)
        return true;
    else
    {
        if (m_vid.isLocal() && (m_vid.getDataInByteArray() == null))
            return true;
        else if (!m_vid.isLocal() && (":///".equals(m_vid.getSource())))
            return true;
        else
        {
            if (!m_vid.isLocal())
            {
                BFILE bfile = m_vid.getBFILE();
                if (!bfile.fileExists())
                    return true;
                else
                    return false;
            }
            else
                return false;
        }
    }
}
.
.
.
byte[] getDataInByteArray(OrdVideo vid) throws SQLException, IOException
{
    if (!m_hasMedia)
        return null;
    else
    {
        if (!vid.isLocal())

```

```
    {
        byte[] ctx[] = new byte[1][4000];
        try
        {
            vid.importData(ctx);
        }
        catch (SQLException e)
        {
            new IMessage(IMConstants.ERROR, "MEDIA_SOURCE_ERR", e);
            return null;
        }
    }
    return vid.getDataInByteArray();
}
}
.
.
.
void refreshPanel(boolean isFormatSupported) throws SQLException, IOException
{
    m_hasMedia = true;

    if (isFormatSupported)
    {
        if (m_jAttrTbl == null)
        {
            m_jAttrPane.remove(m_jEmpty);
            insertProperty();
        }
        else
        {
            m_jAttrTbl.setValueAt(m_vid.getMimeType(), 0, 1);
            m_jAttrTbl.setValueAt(new Integer(m_vid.getHeight()).toString(), 1, 1);
            m_jAttrTbl.setValueAt(new Integer(m_vid.getWidth()).toString(), 2, 1);
            m_jAttrTbl.setValueAt(new Integer(m_vid.getVideoDuration()).toString(), 3, 1);
            m_jAttrTbl.setValueAt(new Integer(m_vid.getContentLength()).toString(), 4, 1);
        }
    }
    .
    .
    .
}
```

## 5.2.6 IMAudioPanel Class

This class displays the audio panel, the product audio, and its attributes. This class is identical in structure and functions similarly to the `IMImagePanel` class. See [IMImagePanel Class](#) for descriptions of methods.

The following code example includes the `display()`, `insertProperty()`, `notExist()`, `getDataInByteArray()`, and `refreshPanel()` methods, and highlights in bold any SQL query statements and areas in the code where Oracle Multimedia and other Oracle object types and methods are used:

```
void display() throws IOException, SQLException
{
    addControlPane();

    // Set the audio icon.
```

```

m_jIcon = new JLabel(new ImageIcon(IMExampleFrame.class.getResource("OrdAudio.gif")));
m_jIcon.setLabelFor(m_jAttrPane);

    m_jIconPane.add(m_jIcon, BorderLayout.CENTER);

if (notExist())
{
    // The audio does not exist.
    m_hasMedia = false;
    layoutEmpty(s_sNotExist);
}
else
{
    m_hasMedia = true;

    // If the audio exists, try to show the attributes.
    insertProperty();
}
}
.
.
.
boolean insertProperty() throws SQLException
{
    boolean isFormatSupported = false;
    String sMimeType = m_aud.getMimeType();

    if (sMimeType == null)
        isFormatSupported = IMUtil.setProperties(m_aud);
    else
        isFormatSupported = true;

    if (!isFormatSupported)
    {
        layoutEmpty(s_sNotSupported);
    }
    else
    {
        Object[][] data =
        {
            {"MIME Type", m_aud.getMimeType()},
            {"Duration", new Integer(m_aud.getAudioDuration().toString())},
            {"Content Length", new Integer(m_aud.getContentLength().toString())}
        };

        .
        .
        .
    }

    return isFormatSupported;
}
.
.
.
boolean notExist() throws SQLException, IOException
{
    if (m_aud == null)
        return true;
    else

```

```
{
    if (m_aud.isLocal() && (m_aud.getDataInByteArray() == null))
        return true;
    else if (!m_aud.isLocal() && (":///".equals(m_aud.getSource()))
        return true;
    else
    {
        if (!m_aud.isLocal())
        {
            BFILE bfile = m_aud.getBFILE();
            if (!bfile.fileExists())
                return true;
            else
                return false;
        }
        else
            return false;
    }
}
}
.
.
.
byte[] getDataInByteArray(OrdAudio aud) throws SQLException, IOException
{
    if (!m_hasMedia)
        return null;
    else
    {
        if (!aud.isLocal())
        {
            byte[] ctx[] = new byte[1][4000];
            try
            {
                aud.importData(ctx);
            }
            catch (SQLException e)
            {
                new IMessage(IMConstants.ERROR, "MEDIA_SOURCE_ERR", e);
                return null;
            }
        }
        return aud.getDataInByteArray();
    }
}
}
.
.
.
void refreshPanel(boolean isFormatSupported) throws SQLException, IOException
{
    m_hasMedia = true;
    if (isFormatSupported)
    {
        if (m_jAttrTbl == null)
        {
            m_jAttrPane.remove(m_jEmpty);
            insertProperty();
        }
        else
        {

```

```

        m_jAttrTbl.setValueAt(m_aud.getMimeType(), 0, 1);
        m_jAttrTbl.setValueAt(new Integer(m_aud.getAudioDuration()).toString(), 1, 1);
        m_jAttrTbl.setValueAt(new Integer(m_aud.getContentLength()).toString(), 2, 1);
    }
}
.
.
.
}

```

## 5.2.7 IMDocPanel Class

This class displays the doc panel, the product testimonials, and its attributes. This class is identical in structure and functions similarly to the `IMImagePanel` class. See [IMImagePanel Class](#) for descriptions of methods.

The following code example includes the `display()`, `insertProperty()`, `notExist()`, `getDataInByteArray()`, and `refreshPanel()` methods, and highlights in bold any SQL query statements and areas in the code where Oracle Multimedia and other Oracle object types and methods are used:

```

void display() throws IOException, SQLException
{
    addControlPane();

    // Set the icon.
    m_jIcon = new JLabel(new ImageIcon(
        IMExampleFrame.class.getResource("OrdDoc.gif")
    ));
    m_jIcon.setLabelFor(m_jAttrPane);
    m_jIconPane.add(m_jIcon, BorderLayout.CENTER);

    if (notExist())
    {
        // The doc does not exist.
        m_hasMedia = false;
        layoutEmpty(s_sNotExist);
    }
    else
    {
        // If the doc exists, show the attribute table.
        m_hasMedia = true;
        insertProperty();
    }
}
.
.
.
boolean insertProperty() throws SQLException
{
    boolean isFormatSupported = false;
    String sMimeType = m_doc.getMimeType();

    if (sMimeType == null)
        isFormatSupported = IMUtil.setProperties(m_doc);
    else
        isFormatSupported = true;

    if (!isFormatSupported)

```

```
    {
        layoutEmpty(s_sNotSupported);
    }
else
{
    Object[][] data =
    {
        {"MIME Type", m_doc.getMimeType()},
        {"Content Length", new Integer(m_doc.getContentLength()).toString()}
    };

    .
    .
    .
}

return isFormatSupported;
}
.
.
.
boolean notExist() throws SQLException, IOException
{
    if (m_doc == null)
        return true;
    else
    {
        if (m_doc.isLocal() && (m_doc.getDataInByteArray() == null))
            return true;
        else if (!m_doc.isLocal() && (":///".equals(m_doc.getSource())))
            return true;
        else
        {
            if (!m_doc.isLocal())
            {
                BFILE bfile = m_doc.getBFILE();
                if (!bfile.fileExists())
                    return true;
                else
                    return false;
            }
            else
                return false;
        }
    }
}
.
.
.
byte[] getDataInByteArray(OrdDoc doc) throws SQLException, IOException
{
    if (!m_hasMedia)
        return null;
    else
    {
        if (!doc.isLocal())
        {
            byte[] ctx[] = new byte[1][4000];
            try
            {
```

```

        doc.importData(ctx, false);
    }
    catch (SQLException e)
    {
        new IMessage(IMConstants.ERROR, "MEDIA_SOURCE_ERR", e);
        return null;
    }
}
return doc.getDataInByteArray();
}
}
.
.
.
void refreshPanel(boolean isFormatSupported) throws SQLException, IOException
{
    m_hasMedia = true;
    if (isFormatSupported)
    {
        if (m_jAttrTbl == null)
        {
            m_jAttrPane.remove(m_jEmpty);
            insertProperty();
        }
        else
        {
            m_jAttrTbl.setValueAt(m_doc.getMimeType(), 0, 1);
            m_jAttrTbl.setValueAt(new Integer(m_doc.getContentLength()).toString(), 1, 1);
        }
    }
}
.
.
.
}

```

## 5.2.8 IMLoadFile Class

This class loads a media stream from a file to a database for each of the media object types. First, it checks to see if this `PRODUCT_ID` column exists in the `PM.ONLINE_MEDIA` table and if not, it inserts a new row into the table. Then, it creates and initializes a new media object for each media object type, updates the media data, that is, loads it into the database if it is not already stored there, and finally, sets the media attributes for each media data object.

In this class, the `IMFileLoad()` method calls the `initFileChooser()` method, then the `initFileChooser()` method calls the `loadNewMedia()` method, which does the row insertion and initializing of the media object type columns, and then calls the `updateMedia()` method to update the media and to set the media attributes.

The following code example includes the `loadNewMedia()` and `UpdateMedia()` methods, and highlights in bold any SQL query statements and areas in the code where Oracle Multimedia and other Oracle object types and methods are used as previously described:

```

private void loadNewMedia()
    throws SQLException, FileNotFoundException, SecurityException, IOException
{
    boolean isInsertNeeded = false;
    String sQuery = null;

```

```
OracleConnection conn = null;
OracleResultSet rs = null;
OraclePreparedStatement pstmt = null;

try
{
    conn = IMExample.getDBConnection();

    if (m_obj == null)
    {
        // First, check whether or not this product exists in the
        // pm.online_media table. If it exists, isInsertNeeded is set to false;
        // or else, isInsertNeeded is set to true.
        sQuery = new String(
            "select product_id from pm.online_media where product_id = ?");
        pstmt = (OraclePreparedStatement) conn.prepareStatement(sQuery);
        pstmt.setInt(1, m_iProdId);
        rs = (OracleResultSet)pstmt.executeQuery();
        if (rs.next() == false)
            isInsertNeeded = true;
        else
            isInsertNeeded = false;
        rs.close();
        pstmt.close();

        if (isInsertNeeded)
        {
            // If this product is not in the pm.online_media table,
            // insert a row in pm.online_media for this product,
            // and initialize the media object at the same time.
            sQuery = new String(
                "insert into pm.online_media (product_id, product_photo, " +
                "product_photo_signature, product_thumbnail, product_video, " +
                "product_audio, product_text, product_testimonials) values (" +
                "?, ORDSYS.ORDImage.init(), ORDSYS.ORDImageSignature.init(), " +
                "ORDSYS.ORDImage.init(), ORDSYS.ORDVideo.init(), " +
                "ORDSYS.ORDAudio.init(), null, ORDSYS.ORDDoc.init())");

            pstmt = (OraclePreparedStatement) conn.prepareCall(sQuery);
            pstmt.setInt(1, m_iProdId);
            pstmt.execute();
            pstmt.close();
        }
    }

    if (!isInsertNeeded)
    {
        // Create a new media object.
        switch (m_iTypeIdentifier)
        {
            case IMG_TYPE:
                sQuery = new String(
                    "update pm.online_media set " + m_sColName +
                    " = ORDSYS.ORDImage.init() where product_id = ?");
                break;
            case AUD_TYPE:
                sQuery = new String(
                    "update pm.online_media set " + m_sColName +
                    " = ORDSYS.ORDAudio.init() where product_id = ?");
                break;
        }
    }
}
```



```

case VID_TYPE:
    sQuery = new String(
        "update pm.online_media set " + m_sColName +
        " = ORDSYS.ORDVideo.init() where product_id = ?");
    break;
case DOC_TYPE:
    sQuery = new String(
        "update pm.online_media set " + m_sColName +
        " = ORDSYS.ORDDoc.init() where product_id = ?");
    break;
default:
    new IMessage(IMConstants.ERROR, "UNKNOWN_TYPE");
    break;
}

pstmt = (OraclePreparedStatement) conn.prepareCall(sQuery);
pstmt.setInt(1, m_iProdId);
pstmt.execute();
pstmt.close();
}

// At this point, there is a row in the online_media table
// for this product and the desired media object is initialized.
// In the following, we update the media object pointer and
// acquire the right to modify it by selecting again from the
// database.
//
sQuery = new String(
    "select " + m_sColName +
    " from pm.online_media where product_id = ? for update");
pstmt = (OraclePreparedStatement) conn.prepareStatement(sQuery);
pstmt.setInt(1, m_iProdId);
rs = (OracleResultSet)pstmt.executeQuery();
if (rs.next() == false)
    throw new SQLException();
else
{
    switch (m_iTypeIdentifier)
    {
        case IMG_TYPE:
            m_img = (OrdImage)rs.getORADData(1, OrdImage.getORADDataFactory());
            break;
        case AUD_TYPE:
            m_aud = (OrdAudio)rs.getORADData(1, OrdAudio.getORADDataFactory());
            break;
        case VID_TYPE:
            m_vid = (OrdVideo)rs.getORADData(1, OrdVideo.getORADDataFactory());
            break;
        case DOC_TYPE:
            m_doc = (OrdDoc)rs.getORADData(1, OrdDoc.getORADDataFactory());
            break;
        default:
            new IMessage(IMConstants.ERROR, "UNKNOWN_TYPE");
            break;
    }
}

// Update the media object.
updateMedia();
}

```

```
        rs.close();
        pstmt.close();
    }
    finally
    {
        IMUtil.cleanup(rs, pstmt);
    }
}

/**
 * Update the media and also set the media properties.
 */
private void updateMedia()
    throws SQLException, FileNotFoundException, SecurityException, IOException
{
    String sQuery = null;
    OracleConnection conn = null;
    byte[] ctx[] = new byte[1][64];
    OraclePreparedStatement pstmt = null;

    boolean isFormatSupported = false;

    try
    {
        conn = IMExample.getDBConnection();
        sQuery = new String(
            "update pm.online_media set " + m_sColName +
            " = ? where product_id = ?");
        pstmt = (OraclePreparedStatement) conn.prepareCall(sQuery);
        pstmt.setInt(2, m_iProdId);

        switch (m_iTypeIdentifier)
        {
            case IMG_TYPE:
                m_img.loadDataFromFile(m_jFileChooser.getText());
                isFormatSupported = IMUtil.setProperties(m_img);
                m_img.setLocal();
                pstmt.setORADData(1, m_img);
                break;
            case AUD_TYPE:
                m_aud.loadDataFromFile(m_jFileChooser.getText());
                isFormatSupported = IMUtil.setProperties(m_aud);
                m_aud.setLocal();
                pstmt.setORADData(1, m_aud);

                // We need to update the media pointer for display,
                // because the input media pointer may be null.
                ((IMAudioPanel)m_parent).setMedia(m_aud);
                ((IMAudioPanel)m_parent).refreshPanel(isFormatSupported);
                break;
            case VID_TYPE:
                m_vid.loadDataFromFile(m_jFileChooser.getText());
                isFormatSupported = IMUtil.setProperties(m_vid);
                m_vid.setLocal();
                pstmt.setORADData(1, m_vid);

                ((IMVideoPanel)m_parent).setMedia(m_vid);
                ((IMVideoPanel)m_parent).refreshPanel(isFormatSupported);
                break;
            case DOC_TYPE:

```

```

        m_doc.loadDataFromFile(m_jFileChooser.getText());
        isFormatSupported = IMUtil.setProperties(m_doc);
        m_doc.setLocal();
        pstmt.setORADData(1, m_doc);

        ((IMDocPanel)m_parent).setMedia(m_doc);
        ((IMDocPanel)m_parent).refreshPanel(isFormatSupported);
        break;
    default:
        new IMessage(IMConstants.ERROR, "UNKNOWN_TYPE");
        break;
    }

    pstmt.execute();
    pstmt.close();

    // Update the thumbnail image.
    if (m_iTypeIdentifier == IMG_TYPE)
    {
        if (isFormatSupported)
            m_imgThumb = IMUtil.generateThumbnail(m_iProdId, m_img, m_imgThumb);

        ((IMImagePanel)m_parent).setMedia(m_img, m_imgThumb);
        ((IMImagePanel)m_parent).refreshPanel(isFormatSupported);
    }
    }
    finally
    {
        IMUtil.cleanup(pstmt);
    }
}

```

## 5.2.9 IMUtil Class

This class contains common utilities, such as a `generateThumbnail()` static method, wrapper methods for the `setProperties()` methods for each media object type to separate the exceptions caused by unrecognizable formats, and finally, a number of cleanup methods.

The following code example includes the `generateThumbnail()` method, and highlights in bold any SQL query statements and areas in the code where Oracle Multimedia and other Oracle object types and methods are used:

```

static OrdImage generateThumbnail(int iProdId, OrdImage img, OrdImage imgThumb)
    throws SQLException
    {
        String sQuery = null;
        OracleConnection conn = null;
        OracleResultSet rs = null;
        OraclePreparedStatement pstmt = null;

        try
        {
            conn = IMExample.getDBConnection();

            if (imgThumb == null)
            {
                // The thumbnail media pointer is not initialized.
                // Initialize it first.
            }
        }
    }

```

```
sQuery = new String(
    "update pm.online_media set product_thumbnail = " +
    "ORDSYS.ORDImage.init() where product_id = ?");
pstmt = (OraclePreparedStatement) conn.prepareCall(sQuery);
pstmt.setInt(1, iProdId);
pstmt.execute();
pstmt.close();

// Acquire the new pointer and the permission to update.
sQuery = new String("select product_thumbnail from pm.online_media " +
    "where product_id = ? for update");
pstmt = (OraclePreparedStatement) conn.prepareStatement(sQuery);
pstmt.setInt(1, iProdId);
rs = (OracleResultSet)pstmt.executeQuery();
if (rs.next() == false)
    throw new SQLException();
else
    imgThumb = (OrdImage)rs.getORADData(1, OrdImage.getORADDataFactory());

rs.close();
pstmt.close();
}

// Generate the thumbnail image.
img.processCopy("maxScale=64 64, fileFormat=GIFF", imgThumb);

// Update the thumbnail image in the database.
sQuery = new String(
    "update pm.online_media set product_thumbnail = ? where product_id = ?");
pstmt = (OraclePreparedStatement) conn.prepareCall(sQuery);
pstmt.setORADData(1, imgThumb);
pstmt.setInt(2, iProdId);
pstmt.execute();
pstmt.close();

return imgThumb;
}
finally
{
    IMUtil.cleanup(rs, pstmt);
}
}
```

---

---

## Working with Metadata in Images

Image files can contain information about the content of the images, the image rasters, and image metadata. In general, data about data is referred to as **metadata**. In this case, metadata refers to additional information about the actual images, which is stored in the image files along with the images.

### 6.1 Metadata Concepts

Several types of metadata can be stored in an image file, and each type can serve a different purpose. One type, **technical metadata**, is used to describe an image in a technical sense. For example, technical metadata can include attributes about an image, such as its height and width, in pixels, or the type of compression used to store it. Another type, **content metadata**, can further describe the content of an image, the name of the photographer, and the date and time when a photograph was taken.

Metadata is stored in image files using a variety of mechanisms. Digital cameras and scanners automatically insert metadata into the images they create. Digital photograph processing applications like Adobe Photoshop allow users to add or edit metadata to be stored with the image. Annotating digital images with additional metadata is a common practice in photographic and news gathering applications and for image archiving usages, as well as at the consumer level.

Storing metadata together with image data in the same containing file provides encapsulation. With encapsulation, both types of data can be shared and exchanged reliably as one unit. Metadata that is stored in the image file format is referred to as **embedded metadata**.

### 6.2 Oracle Multimedia Image Metadata Concepts

For a large number of image file formats, Oracle Multimedia can extract and manage a limited set of metadata attributes. These attributes include: height, width, contentLength, fileFormat, contentFormat, compressionFormat, and mimeType. For a limited number of image file formats, Oracle Multimedia can extract a rich set of metadata attributes. This metadata is represented in schema-based XML documents. These XML documents can be stored in a database, indexed, searched, updated, and made available to applications using the standard mechanisms of Oracle Database.

Oracle Multimedia can also write or embed metadata supplied by users into a limited number of image file formats. The application provides the metadata as a schema-based XML document. Oracle Multimedia processes the XML document and writes the metadata into the image file.

## 6.3 Image File Formats

Oracle Multimedia supports metadata extraction and metadata embedding for the GIF, TIFF, and JPEG file formats. See *Oracle Multimedia Reference* for information about the image file formats supported by Oracle Multimedia.

## 6.4 Image Metadata Formats

The term **image metadata format** refers to the standard protocols and techniques used to store image metadata within an image file. The following subsections describe the embedded image metadata formats supported by Oracle Multimedia:

- EXIF
- IPTC-IIM
- XMP

### 6.4.1 EXIF

The Exchangeable Image File Format (EXIF) is the standard for image file storage for digital still cameras. It was developed by the Japan Electronic Industry Development Association (JEIDA) as a standard way of storing images created by digital cameras as well as metadata about the images. EXIF image metadata can be stored in TIFF and JPEG format images. Oracle Multimedia supports the extraction of EXIF metadata from TIFF and JPEG file formats.

### 6.4.2 IPTC-IIM

The International Press Telecommunications Council-Information Interchange Model (IPTC-IIM) Version 4 is a standard developed jointly by the International Press Telecommunications Council and the Newspaper Association of America. This metadata standard is designed to capture information that is important to the activities of news gathering, reporting, and publishing. These information records are commonly referred to as IPTC tags.

The use of embedded IPTC tags in image file formats became widespread with the use of the Adobe Photoshop tool for image editing. IPTC metadata can be stored in TIFF and JPEG format images. Oracle Multimedia supports the extraction of IPTC metadata from TIFF and JPEG file formats.

### 6.4.3 XMP

The Extensible Metadata Platform (XMP) is a standard metadata format, developed by Adobe, for the creation, processing, and interchange of metadata in a variety of applications. XMP uses Resource Description Framework (RDF) technology for data modeling. XMP also defines how the data model is serialized (converted to a byte stream), and embedded within an image file. Oracle Multimedia supports the extraction of XMP metadata from GIF, TIFF, and JPEG file formats. Oracle Multimedia also supports writing XMP data packets into GIF, TIFF, and JPEG file formats.

For more information about XMP, see the Adobe Systems Incorporated Web site at

<http://www.adobe.com/>

For more information about RDF, see the Resource Description Framework Web page on the World Wide Web Consortium Web site at

<http://www.w3.org/RDF/>

## 6.5 Representing Metadata Outside Images

After metadata has been extracted from the binary image file, the next step is to represent the metadata in a form that can be easily stored, indexed, queried, updated, and presented. Oracle Multimedia returns image metadata in XML documents. These documents are based on XML schemas that Oracle Multimedia registers with the database. Each type of image metadata has a separate XML schema. These XML schemas are used by the metadata methods of the `ORDImage` object type. See *Oracle Multimedia Reference* for complete definitions of the XML schemas supported by Oracle Multimedia.

The XML documents can be stored in `XMLType` columns within the database. These documents are easily searched and processed using the wide range of standards-based XML technologies provided by Oracle XML DB. (See *Oracle XML DB Developer's Guide* for more information.)

## 6.6 Oracle Multimedia Image Metadata Examples

The following examples of metadata extraction and embedding make use of the `photos` table. The `photos` table is defined by the Photo Album sample application. The implementation of the Photo Album sample application is defined in the PL/SQL package named `PHOTO_ALBUM`. See [Section 3.1](#) for a complete description of the Oracle Multimedia PL/SQL Web Toolkit Photo Album sample application.

The `photos` table stores two instances of an image: the full-size photograph and a thumbnail image. This table can also store up to four different image metadata documents. These documents are stored in the columns named `metaORDImage`, `metaEXIF`, `metaIPTC`, and `metaXMP`, and represent image metadata from the `ORDImage`, `EXIF`, `IPTC`, and `XMP` metadata formats, respectively. The metadata columns are of type `XMLType`, and they are bound to the corresponding metadata XML schemas that Oracle Multimedia provides.

### 6.6.1 Creating a Table for Metadata Storage

Before you can extract or embed metadata, you must create the table and columns where the metadata will be stored. The following PL/SQL code segment creates the `photos` table with four `XMLType` columns (`metaORDImage`, `metaEXIF`, `metaIPTC`, and `metaXMP`) to store each type of image metadata, and two `ORDImage` columns (`image` and `thumb`) for the original image and the thumbnail image, respectively. Each metadata column is bound to its corresponding metadata schema. For example, the `metaEXIF` column is bound to the XML schema stored at `http://xmlns.oracle.com/ord/meta/exif`, and is defined as the XML element `exifMetadata`.

The code statements where the image metadata columns are defined and bound to XML schemas are highlighted in bold.

```
--
-- Create the PHOTOS table
--
CREATE TABLE photos( id          NUMBER PRIMARY KEY,
                     description  VARCHAR2(40) NOT NULL,
                     metaORDImage XMLType,
                     metaEXIF    XMLType,
                     metaIPTC    XMLType,
                     metaXMP     XMLType,
                     image        ORDSYS.ORDIMAGE,
                     thumb        ORDSYS.ORDIMAGE )
```

```

--
-- store full-size images and thumbnail images as SecureFile LOBs
--
LOB(image.source.localdata) STORE AS SECUREFILE
LOB(thumb.source.localdata) STORE AS SECUREFILE
-- and bind the XMLType columns to the interMedia metadata schemas
XMLType COLUMN metaORDImage
  XMLSCHEMA "http://xmlns.oracle.com/ord/meta/ordimage"
  ELEMENT "ordImageAttributes"
XMLType COLUMN metaEXIF
  XMLSCHEMA "http://xmlns.oracle.com/ord/meta/exif"
  ELEMENT "exifMetadata"
XMLType COLUMN metaIPTC
  XMLSCHEMA "http://xmlns.oracle.com/ord/meta/iptc"
  ELEMENT "iptcMetadata"
XMLType COLUMN metaXMP
  XMLSCHEMA "http://xmlns.oracle.com/ord/meta/xmp"
  ELEMENT "xmpMetadata";

```

## 6.6.2 Extracting Image Metadata

The following PL/SQL procedure extracts metadata from an image and stores it in the specified columns in the `photos` table you created. This procedure demonstrates the `getMetadata()` method, which returns an array of XML documents. The root element of each document is examined to determine the metadata type. The `UPDATE` statement stores the documents in the corresponding columns in the `photos` table.

The code statement where the `getMetadata()` method is called is highlighted in bold.

```

--
-- fetch the metadata and sort the results
--
PROCEDURE extractMetadata(inID IN INTEGER)
IS
  img ORDSYS.ORDIMAGE;
  metav XMLSequenceType;
  meta_root VARCHAR2(40);
  xmlORD XMLType;
  xmlXMP XMLType;
  xmlEXIF XMLType;
  xmlIPTC XMLType;

BEGIN

-- select the image
SELECT image
INTO img
FROM PHOTOS
WHERE id = inID;

-- extract all the metadata
metav := img.getMetadata( 'ALL' );

-- process the result array to discover what types of metadata were
returned
FOR i IN 1..metav.count() LOOP
  meta_root := metav(i).getRootElement();
  CASE meta_root
    WHEN 'ordImageAttributes' THEN xmlORD := metav(i);
    WHEN 'xmpMetadata' THEN xmlXMP := metav(i);

```



```

        WHEN 'iptcMetadata' THEN xmlIPTC := metav(i);
        WHEN 'exifMetadata' THEN xmlEXIF := metav(i);
        ELSE NULL;
    END CASE;
END LOOP;

-- Update metadata columns
--
UPDATE photos
SET metaORDImage = xmlORD,
    metaEXIF = xmlEXIF,
    metaIPTC = xmlIPTC,
    metaXMP = xmlXMP
WHERE id = inID;

END extractMetadata;
```

### 6.6.3 Embedding Image Metadata

The following PL/SQL procedure demonstrates the `putMetadata()` method. This procedure accepts six arguments. The `entry_id` argument identifies the image in the `photos` table to be updated. The remaining arguments (`title`, `creator`, `date`, `description`, and `copyright`) are strings that will be formatted into an XMP packet and embedded within the target image.

This example creates an XML document instance based on the Oracle Multimedia XML schema for XMP metadata. (This schema is preregistered with Oracle XML DB. See *Oracle XML DB Developer's Guide* for more information.) The schema for XMP metadata defines a single, global element `<xmpMetadata>`. The `<xmpMetadata>` element contains a single, well-formed RDF document. The RDF document contains a single `<RDF>` element, which is derived from the `rdf` namespace. This RDF document is constructed using elements defined by the Dublin Core Schema.

The call to the `putMetadata()` method embeds the metadata document into the image file. The `UPDATE` statement stores the new image and the new metadata back in the `photos` table.

The code statement where the `putMetadata()` method is called is highlighted in bold.

```

--
-- write the metadata to the image
--
PROCEDURE write_metadata( entry_id IN VARCHAR2,
                        title IN VARCHAR2,
                        creator IN VARCHAR2,
                        date IN VARCHAR2,
                        description IN VARCHAR2,
                        copyright IN VARCHAR2 )
IS
    img ORDSYS.ORDImage;
    xmp XMLType;
    buf VARCHAR2(5000);
BEGIN
    -- select the image
    SELECT image
    INTO img
    FROM PHOTOS
    WHERE id = entry_id FOR UPDATE;

    -- Create the XMP packet it must be schema valid
```

```

-- to "http://xmlns.oracle.com/ord/meta/xmp"
-- and contain an <RDF> element. This example uses
-- the Dublin Core schema.

/* An example XML instance document

<xmpMetadata xmlns="http://xmlns.oracle.com/ord/meta/xmp"
             xsi:schemaLocation="http://xmlns.oracle.com/ord/meta/xmp
             http://xmlns.oracle.com/ord/meta/xmp"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          <rdf:Description about="" xmlns:dc="http://purl.org/dc/elements/1.1/">
    <dc:title>A Winter Day</dc:title>
    <dc:creator>Frosty S. Man</dc:creator>
    <dc:date>21-Dec-2004</dc:date>
    <dc:description>a sleigh ride</dc:description>
    <dc:copyright>North Pole Inc.</dc:copyright>
  </rdf:Description>
</rdf:RDF>
</xmpMetadata>

*/

buf := '<xmpMetadata xmlns="http://xmlns.oracle.com/ord/meta/xmp"
       xsi:schemaLocation="http://xmlns.oracle.com/ord/meta/xmp
       http://xmlns.oracle.com/ord/meta/xmp"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
       <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       <rdf:Description about="" xmlns:dc="http://purl.org/dc/elements/1.1/">;

IF title IS NOT NULL THEN
  buf := buf || '<dc:title>' || htf.escape_sc(title) || '</dc:title>';
END IF;

IF creator IS NOT NULL THEN
  buf := buf || '<dc:creator>' || htf.escape_sc(creator)
           || '</dc:creator>';
END IF;
IF date IS NOT NULL THEN
  buf := buf || '<dc:date>' || htf.escape_sc(date)
           || '</dc:date>';
END IF;
IF description IS NOT NULL THEN
  buf := buf || '<dc:description>' || htf.escape_sc(description)
           || '</dc:description>';
END IF;
IF copyright IS NOT NULL THEN
  buf := buf || '<dc:copyright>' || htf.escape_sc(copyright)
           || '</dc:copyright>';
END IF;

buf := buf || '
  </rdf:Description>
</rdf:RDF>
</xmpMetadata>';

-- create the XML document
xmp := XMLType.createXML(buf, 'http://xmlns.oracle.com/ord/meta/xmp');

```

```

-- write the metadata
img.putMetadata( xmp, 'XMP' );

-- update the image
UPDATE photos
SET image = img,
    metaXMP = xmp
WHERE id = entry_id;

END write_Metadata;

```

## 6.7 Metadata References

The following Web sites provide information about standards and technologies related to working with metadata in images.

- Dublin Core, a standard schema for Dublin core elements  
<http://dublincore.org/2003/03/24/dces>
- Extensible Metadata Platform  
<http://www.adobe.com/>
- Resource Description Framework (See *RDF Primer*)  
<http://www.w3.org/RDF/>

## 6.8 Extracting and Mapping DICOM Metadata Attributes in Images

The Digital Imaging and Communications in Medicine (DICOM) feature allows Oracle Multimedia to recognize standalone DICOM objects and extract the set of attributes related to patient, study, series, and equipment. Recognizing a DICOM object means that Oracle Multimedia can determine whether given binary data represents a standalone DICOM object. If it does, Oracle Multimedia can extract DICOM metadata from that DICOM object. The data can be stored as the content of an `ORDImage` object, or directly in a `BLOB` or a `BFILE`.

DICOM objects are objects that can store different types of data. Examples include patient administration information, waveforms, images, slices of 3-D volumes, video segments, and time-variant 3-D volumes. A standalone DICOM object must have a file header that conforms to the DICOM standard.

The current release adds a new `ORDDicom` object type. In addition, applications can use Oracle Multimedia Java and PL/SQL APIs to store, manage, and manipulate DICOM objects. For more information about these and other new Oracle Multimedia DICOM enhancements, see *Oracle Multimedia DICOM Developer's Guide* and *Oracle Multimedia DICOM Java API Reference*.

See *Oracle Multimedia Reference* for details about the DICOM methods, encoding rules, image format, and exceptions supported by Oracle Multimedia in Oracle Database 10g Release 2.

## 6.9 DICOM Image File Format

Oracle Multimedia supports metadata extraction for the DICM image file format. DICM is the Oracle Multimedia designation for the Digital Imaging and Communications in Medicine format. See *Oracle Multimedia Reference* for more information about the image file formats supported by Oracle Multimedia.

## 6.10 Oracle Multimedia DICOM Image Metadata Example

The following example of DICOM metadata extraction makes use of the `medicalImages` table, which is defined in the sample PL/SQL code segments described in the following subsections.

The `medicalImages` table stores two instances of an image: the full-size photograph and a thumbnail image. This table can also store one image metadata document. This document is stored in the column named `metadata`, and represents DICOM image metadata from the DICOM metadata format. The `metadata` column is of type `XMLType`, and it is bound to the corresponding metadata XML schema that Oracle Multimedia provides.

### 6.10.1 Creating a Table for DICOM Metadata Storage

Before you can extract DICOM metadata, you must create the table and column where the metadata will be stored. The following PL/SQL code segment creates the `medicalImages` table with one `XMLTYPE` column (`metadata`) to store the DICOM image metadata, and two `ORDIMAGE` columns (`image` and `thumb`) for the original image and the thumbnail image, respectively. The `metadata` column is bound to the XML schema stored at `http://xmlns.oracle.com/ord/meta/dicomImage`, and is defined as the XML element `DICOM_IMAGE`.

The code statements where the image metadata column is defined and bound to the XML schema are highlighted in bold.

```
create table medicalImages( id number primary key,
                           description VARCHAR2(40),
                           metadata XMLType,
                           image ORDSYS.ORDIMAGE,
                           thumb ORDSYS.ORDIMAGE )
--
-- store full-size images and thumbnail images as SecureFile LOBs
--
LOB(image.source.localdata) STORE AS SECUREFILE
LOB(thumb.source.localdata) STORE AS SECUREFILE
-- and bind the XMLType columns to the interMedia metadata columns
XMLType column metadata
  XMLSCHEMA "http://xmlns.oracle.com/ord/meta/dicomImage"
  ELEMENT "DICOM_IMAGE";
```

### 6.10.2 Extracting DICOM Image Metadata

The following PL/SQL procedure extracts DICOM metadata from an image and stores it in the specified column in the `medicalImages` table you created. This procedure demonstrates the `getDicomMetadata()` method, which returns an XML document. The `UPDATE` statement stores the document in the corresponding column in the `medicalImages` table.

The code statement where the `getDicomMetadata()` method is called is highlighted in bold.

```
--
PROCEDURE extractDicomMetadata(inID INTEGER)
IS
  local_image ORDSYS.ORDIMAGE;
  local_id INTEGER;
  dicom_metadata XMLType := NULL;
BEGIN
  SELECT image INTO local_image FROM medicalImages WHERE id = inID;
```

```
-- extract DICOM metadata
dicom_metadata := local_image.getDicomMetadata('imageGeneral');
IF (dicom_metadata IS NULL) THEN
    DBMS_OUTPUT.PUT_LINE('metadata is NULL');
ELSE
    UPDATE medicalImages SET metadata = dicom_metadata where id = inID;
END IF;
-- let us print the namespace of the XML document containing the
-- dicom metadata that we just extracted
DBMS_OUTPUT.PUT_LINE('namespace: ' || dicom_metadata.getNamespace());

END extractDicomMetadata;
--
```

## 6.11 DICOM References

The following Web sites provide information about medical imaging standards and issues related to DICOM.

- NEMA standard: Digital Imaging and Communications in Medicine (DICOM)  
<http://medical.nema.org/>
- Integrating the Healthcare Enterprise (IHE)  
<http://www.ihe.net/>



---

---

## Extending Oracle Multimedia

Oracle Multimedia can be extended to support:

- Other external sources of media data not currently supported (see [Section 7.1](#))
- Other media data formats not currently supported (see [Section 7.2](#))
- A new object type (see [Section 7.3](#))
- Media (audio and video) data processing (see [Section 7.4](#))

For each unique external media data source or each unique ORDAudio, ORDDoc, or ORDVideo data format that you want to support, you must:

1. Design your new data source or new ORDAudio, ORDDoc, or ORDVideo data format.
2. Implement your new data source or new ORDAudio, ORDDoc, or ORDVideo data format.
3. Install your new plug-in in the ORDPLUGINS schema.
4. Grant EXECUTE privileges on your new plug-in to PUBLIC.

### 7.1 Supporting Other External Sources

To implement your new data source, you must implement the required interfaces in the `ORDX_<srcType>_SOURCE` package in the ORDPLUGINS schema (where `<srcType>` represents the name of the new external source type). Use the package body example in [Section 7.1.1.3](#) as a template to create the package body. Then, set the source type parameter in the `setSourceInformation()` call to the appropriate source value to indicate to the ORDAudio, ORDImage, ORDDoc, or ORDVideo object that package `ORDPLUGINS.ORDX_<srcType>_SOURCE` is available as a plug-in. Use the `ORDPLUGINS.ORDX_FILE_SOURCE` and `ORDPLUGINS.ORDX_HTTP_SOURCE` packages as guides when you extend support to other external audio, image, video, or other heterogeneous media data sources.

See [Section 7.1.1.1](#), [Section 7.1.1.2](#), and [Section 7.1.1.3](#) for examples and for more information about extending the supported external sources of audio, image, video, or other heterogeneous media data.

#### 7.1.1 Packages or PL/SQL Plug-ins

This section presents reference information on the packages or PL/SQL plug-ins provided.

Plug-ins must be named as `ORDX_<name>_<module_name>` where the `<module_name>` is `SOURCE` for `ORDSource`. For example, the `FILE` plug-in described in

Section 7.1.1.1, is named `ORDX_FILE_SOURCE` and the HTTP plug-in described in Section 7.1.1.2, is named `ORDX_HTTP_SOURCE` and `<name>` is the source type. Both source type names, `FILE` and `HTTP`, are reserved to Oracle.

Use the `ORDPLUGINS.ORDX_FILE_SOURCE` and `ORDPLUGINS.ORDX_HTTP_SOURCE` packages as a guide in developing your new source type package.

### 7.1.1.1 `ORDPLUGINS.ORDX_FILE_SOURCE` Package

The `ORDPLUGINS.ORDX_FILE_SOURCE` package or PL/SQL plug-in provides support for multimedia stored in the local file system external to the database.

```
CREATE OR REPLACE PACKAGE ORDX_FILE_SOURCE AS
  -- functions/procedures
  FUNCTION processCommand(obj      IN OUT NOCOPY ORDSYS.ORDSource,
                          ctx      IN OUT RAW,
                          cmd      IN VARCHAR2,
                          arglist  IN VARCHAR2,
                          result   OUT RAW)
      RETURN RAW;
  PROCEDURE import(obj      IN OUT NOCOPY ORDSYS.ORDSource,
                   ctx      IN OUT RAW,
                   mimetype OUT VARCHAR2,
                   format   OUT VARCHAR2);
  PROCEDURE import(obj      IN OUT NOCOPY ORDSYS.ORDSource,
                   ctx      IN OUT RAW,
                   dlob     IN OUT NOCOPY BLOB,
                   mimetype OUT VARCHAR2,
                   format   OUT VARCHAR2);
  PROCEDURE importFrom(obj  IN OUT NOCOPY ORDSYS.ORDSource,
                       ctx   IN OUT RAW,
                       mimetype OUT VARCHAR2,
                       format  OUT VARCHAR2,
                       loc     IN VARCHAR2,
                       name    IN VARCHAR2);
  PROCEDURE importFrom(obj  IN OUT NOCOPY ORDSYS.ORDSource,
                       ctx   IN OUT RAW,
                       dlob  IN OUT NOCOPY BLOB,
                       mimetype OUT VARCHAR2,
                       format  OUT VARCHAR2,
                       loc    IN VARCHAR2,
                       name   IN VARCHAR2);
  PROCEDURE export(obj  IN OUT NOCOPY ORDSYS.ORDSource,
                   ctx   IN OUT RAW,
                   slob  IN OUT NOCOPY BLOB,
                   loc   IN VARCHAR2,
                   name  IN VARCHAR2);
  FUNCTION getContentLength(obj  IN ORDSYS.ORDSource,
                             ctx  IN OUT RAW),
      RETURN INTEGER;
  PRAGMA RESTRICT_REFERENCES(getContentLength, WNDS, WNPS, RNDS, RNPS);
  FUNCTION getSourceAddress(obj  IN ORDSYS.ORDSource,
                            ctx   IN OUT RAW,
                            userData IN VARCHAR2)
      RETURN VARCHAR2;
  PRAGMA RESTRICT_REFERENCES(getSourceAddress, WNDS, WNPS, RNDS, RNPS);
  FUNCTION open(obj  IN OUT NOCOPY ORDSYS.ORDSource,
                userArg IN RAW,
                ctx  OUT RAW) RETURN INTEGER;
  FUNCTION close(obj  IN OUT NOCOPY ORDSYS.ORDSource, ctx  IN OUT RAW)
```



```

RETURN INTEGER;
FUNCTION trim(obj IN OUT NOCOPY ORDSYS.ORDSource,
             ctx IN OUT RAW,
             newlen IN INTEGER) RETURN INTEGER;
PROCEDURE read(obj IN OUT NOCOPY ORDSYS.ORDSource,
              ctx IN OUT RAW,
              startPos IN INTEGER,
              numBytes IN OUT INTEGER,
              buffer OUT RAW);
PROCEDURE write(obj IN OUT NOCOPY ORDSYS.ORDSource,
               ctx IN OUT RAW,
               startPos IN INTEGER,
               numBytes IN OUT INTEGER,
               buffer OUT RAW);
END ORDX_FILE_SOURCE;
/

```

Table 7–1 shows the methods supported in the `ORDX_FILE_SOURCE` package and the exceptions raised if you call a method that is not supported.

**Table 7–1 Methods Supported in the `ORDPLUGINS.ORDX_FILE_SOURCE` Package**

Name of Method	Level of Support
<code>processCommand</code>	Not supported - raises exception: <code>METHOD_NOT_SUPPORTED</code>
<code>import</code>	Supported
<code>import</code>	Supported
<code>importFrom</code>	Supported
<code>importFrom</code>	Supported
<code>export</code>	Supported
<code>getContentLength</code>	Supported
<code>getSourceAddress</code>	Supported
<code>open</code>	Supported
<code>close</code>	Supported
<code>trim</code>	Not supported - raises exception: <code>METHOD_NOT_SUPPORTED</code>
<code>read</code>	Supported
<code>write</code>	Not supported - raises exception: <code>METHOD_NOT_SUPPORTED</code>

### 7.1.1.2 `ORDPLUGINS.ORDX_HTTP_SOURCE` Package

The `ORDPLUGINS.ORDX_HTTP_SOURCE` package or PL/SQL plug-in provides support for multimedia stored in any HTTP server and accessed through a URL.

```

CREATE OR REPLACE PACKAGE ORDX_HTTP_SOURCE AS
  -- functions/procedures
  FUNCTION processCommand(obj IN OUT NOCOPY ORDSYS.ORDSource,
                        ctx IN OUT RAW,
                        cmd IN VARCHAR2,
                        arglist IN VARCHAR2,
                        result OUT RAW)

    RETURN RAW;
  PROCEDURE import(obj IN OUT NOCOPY ORDSYS.ORDSource,
                  ctx IN OUT RAW,
                  mimetype OUT VARCHAR2,

```

```

        format    OUT VARCHAR2);
PROCEDURE import(obj      IN OUT NOCOPY ORDSYS.ORDSource,
                 ctx      IN OUT RAW,
                 dlob     IN OUT NOCOPY BLOB,
                 mimetype OUT VARCHAR2,
                 format   OUT VARCHAR2);
PROCEDURE importFrom(obj  IN OUT NOCOPY ORDSYS.ORDSource,
                     ctx  IN OUT RAW,
                     mimetype OUT VARCHAR2,
                     format OUT VARCHAR2,
                     loc   IN VARCHAR2,
                     name  IN VARCHAR2);
PROCEDURE importFrom(obj  IN OUT NOCOPY ORDSYS.ORDSource,
                     ctx  IN OUT RAW,
                     dlob  IN OUT NOCOPY BLOB,
                     mimetype OUT VARCHAR2,
                     format OUT VARCHAR2,
                     loc   IN VARCHAR2,
                     name  IN VARCHAR2);
PROCEDURE export(obj IN OUT NOCOPY ORDSYS.ORDSource,
                 ctx IN OUT RAW,
                 dlob IN OUT NOCOPY BLOB,
                 loc IN VARCHAR2,
                 name IN VARCHAR2);
FUNCTION  getLength(obj IN ORDSYS.ORDSource,
           ctx IN OUT RAW)
        RETURN INTEGER;
PRAGMA RESTRICT_REFERENCES(getLength, WNDS, WNPS, RNDS, RNPS, TRUST);
FUNCTION  getSourceAddress(obj IN ORDSYS.ORDSource,
                          ctx IN OUT RAW,
                          userData IN VARCHAR2)
        RETURN VARCHAR2;
PRAGMA RESTRICT_REFERENCES(getSourceAddress, WNDS, WNPS, RNDS, RNPS);
FUNCTION  open(obj IN OUT NOCOPY ORDSYS.ORDSource, userArg IN RAW,
              ctx OUT RAW) RETURN INTEGER;
FUNCTION  close(obj IN OUT NOCOPY ORDSYS.ORDSource, ctx IN OUT RAW)
        RETURN INTEGER;
FUNCTION  trim(obj IN OUT NOCOPY ORDSYS.ORDSource,
              ctx IN OUT RAW,
              newlen IN INTEGER) RETURN INTEGER;
PROCEDURE read(obj      IN OUT NOCOPY ORDSYS.ORDSource,
               ctx      IN OUT RAW,
               startPos IN INTEGER,
               numBytes IN OUT INTEGER,
               buffer   OUT RAW);
PROCEDURE write(obj      IN OUT NOCOPY ORDSYS.ORDSource,
                ctx      IN OUT RAW,
                startPos IN INTEGER,
                numBytes IN OUT INTEGER,
                buffer   OUT RAW);
END ORDX_HTTP_SOURCE;
/

```

Table 7–2 shows the methods supported in the `ORDX_HTTP_SOURCE` package and the exceptions raised if you call a method that is not supported.

**Table 7–2 Methods Supported in the `ORDPLUGINS.ORDX_HTTP_SOURCE` Package**

Name of Method	Level of Support
<code>processCommand</code>	Not supported - raises exception: <code>METHOD_NOT_SUPPORTED</code>

**Table 7–2 (Cont.) Methods Supported in the ORDPLUGINS.ORDX\_HTTP\_SOURCE**

Name of Method	Level of Support
import	Supported
import	Supported
importFrom	Supported
importFrom	Supported
export	Not supported - raises exception: METHOD_NOT_SUPPORTED
getContentLength	Supported
getSourceAddress	Supported
open	Supported
close	Supported
trim	Not supported - raises exception: METHOD_NOT_SUPPORTED
read	Not supported - raises exception: METHOD_NOT_SUPPORTED
write	Not supported - raises exception: METHOD_NOT_SUPPORTED

### 7.1.1.3 Extending Oracle Multimedia to Support a New Data Source

Extending Oracle Multimedia to support a new data source consists of the following steps:

1. Design your new data source.
2. Implement your new data source and name it, for example, `ORDX_MY_SOURCE.SQL`.
3. Install your new `ORDX_MY_SOURCE.SQL` plug-in in the `ORDPLUGINS` schema.
4. Grant `EXECUTE` privileges on your new plug-in, for example, `ORDX_MY_SOURCE.SQL` plug-in to `PUBLIC`.
5. Set the `srctype` to `my` to cause your plug-in to be invoked.

A package body listing is provided in [Example 7–1](#) to assist you in this operation. Add your variables to the places that say "--Your variables go here" and add your code to the places that say "--Your code goes here".

#### **Example 7–1 Show the Package Body for Extending Support to a New Data Source**

```
CREATE OR REPLACE PACKAGE BODY ORDX_MY_SOURCE
AS
  -- functions/procedures
  FUNCTION processCommand(
    obj IN OUT NOCOPY ORDSYS.ORDSource,
    ctx IN OUT RAW,
    cmd IN VARCHAR2,
    arglist IN VARCHAR2,
    result OUT RAW)
  RETURN RAW
  IS
    --Your variables go here.
  BEGIN
    --Your code goes here.
  END processCommand;
  PROCEDURE import( obj IN OUT NOCOPY ORDSYS.ORDSource,
    ctx IN OUT RAW,
```

```

        mimetype OUT VARCHAR2,
        format   OUT VARCHAR2)
IS
--Your variables go here.
BEGIN
--Your code goes here.
END import;
PROCEDURE import( obj  IN OUT NOCOPY ORDSYS.ORDSource,
                  ctx  IN OUT RAW,
                  dlob IN OUT NOCOPY BLOB,
                  mimetype OUT VARCHAR2,
                  format   OUT VARCHAR2)
IS
--Your variables go here.
BEGIN
--Your code goes here.
END import;
PROCEDURE importFrom( obj      IN OUT NOCOPY ORDSYS.ORDSource,
                     ctx      IN OUT RAW,
                     mimetype  OUT VARCHAR2,
                     format   OUT VARCHAR2,
                     loc      IN VARCHAR2,
                     name     IN VARCHAR2)
IS
--Your variables go here.
BEGIN
--Your code goes here.
END importFrom;
PROCEDURE importFrom( obj      IN OUT NOCOPY ORDSYS.ORDSource,
                     ctx      IN OUT RAW,
                     dlob     IN OUT NOCOPY BLOB,
                     mimetype  OUT VARCHAR2,
                     format   OUT VARCHAR2,
                     loc      IN VARCHAR2,
                     name     IN VARCHAR2)
IS
--Your variables go here.
BEGIN
--Your code goes here.
END importFrom;
PROCEDURE export( obj  IN OUT NOCOPY ORDSYS.ORDSource,
                  ctx  IN OUT RAW,
                  slob IN OUT NOCOPY BLOB,
                  loc  IN VARCHAR2,
                  name IN VARCHAR2)
IS
--Your variables go here.
BEGIN
--Your code goes here.
END export;

FUNCTION getContentLength( obj  IN ORDSYS.ORDSource,
                          ctx  IN OUT RAW)
RETURN INTEGER
IS
--Your variables go here.
BEGIN
--Your code goes here.
END getContentLength;
FUNCTION getSourceAddress(obj  IN ORDSYS.ORDSource,
```

```

                                ctx  IN OUT RAW,
                                userData IN VARCHAR2)

RETURN VARCHAR2
IS
--Your variables go here.
BEGIN
--Your code goes here.
END getSourceAddress;
FUNCTION open(obj IN OUT NOCOPY ORDSYS.ORDSource, userArg IN RAW, ctx OUT RAW)
RETURN INTEGER
IS
--Your variables go here.
BEGIN
--Your code goes here.
END open;
FUNCTION close(obj IN OUT NOCOPY ORDSYS.ORDSource, ctx IN OUT RAW)
RETURN INTEGER
IS
--Your variables go here.
BEGIN
--Your code goes here.
END close;
FUNCTION trim(obj      IN OUT NOCOPY ORDSYS.ORDSource,
              ctx      IN OUT RAW,
              newlen  IN INTEGER)

RETURN INTEGER
IS
--Your variables go here.
BEGIN
--Your code goes here.
END trim;
PROCEDURE read(obj      IN OUT NOCOPY ORDSYS.ORDSource,
              ctx      IN OUT RAW,
              startPos  IN INTEGER,
              numBytes  IN OUT INTEGER,
              buffer    OUT RAW)

IS
--Your variables go here.
BEGIN
--Your code goes here.
END read;
PROCEDURE write(obj      IN OUT NOCOPY ORDSYS.ORDSource,
              ctx      IN OUT RAW,
              startPos  IN INTEGER,
              numBytes  IN OUT INTEGER,
              buffer    OUT RAW)

IS
--Your variables go here.
BEGIN
--Your code goes here.
END write;
END ORDX_MY_SOURCE;
/
show errors;

```

## 7.2 Supporting Other Media Data Formats

To implement your new ORDAudio, ORDDoc, or ORDVideo data format, you must implement the required interfaces in the `ORDPLUGINS.ORDX_<format>_<media>` package in the `ORDPLUGINS` schema (where `<format>` represents the name of the new audio or video, or other heterogeneous media data format and `<media>` represents the type of media ("AUDIO" or "VIDEO", or "DOC"). Use the `ORDPLUGINS.ORDX_DEFAULT_<media>` package as a guide when you extend support to other audio or video data formats or other heterogeneous media data formats. Use the package body examples in [Section 7.2.1.2](#), [Section 7.2.2.2](#) and [Section 7.2.3.2](#) as templates to create the audio or video, or other heterogeneous media data package body, respectively. Then, set the new format parameter in the `setFormat()` call to the appropriate format value to indicate to the ORDAudio, ORDDoc, or ORDVideo object that package `ORDPLUGINS.ORDX_<format>_<media>` is available as a plug-in and should be used for method invocation.

### 7.2.1 Supporting Other ORDAudio Data Formats

[Section 7.2.1.1](#) and [Section 7.2.1.2](#) describe how to extend ORDAudio to support other data formats.

#### 7.2.1.1 ORDPLUGINS.ORDX\_DEFAULT\_AUDIO Package

Use the following `ORDPLUGINS.ORDX_DEFAULT_AUDIO` package provided as a guide in developing your own `ORDPLUGINS.ORDX_<format>_AUDIO` audio format package. This package sets the `mimeType` field in the `setProperties()` method with a MIME type value that is dependent on the file format.

```
CREATE OR REPLACE PACKAGE ORDX_DEFAULT_AUDIO
authid current_user
AS
--AUDIO ATTRIBUTES ACCESSORS

PROCEDURE setProperties(ctx IN OUT RAW,
                      obj IN OUT NOCOPY ORDSYS.ORDAudio,
                      setComments IN NUMBER := 0);
FUNCTION checkProperties(ctx IN OUT RAW, obj IN OUT ORDSYS.ORDAudio)
RETURN NUMBER;
FUNCTION getAttribute(ctx IN OUT RAW,
                    obj IN ORDSYS.ORDAudio,
                    name IN VARCHAR2) RETURN VARCHAR2;
PROCEDURE getAllAttributes(ctx IN OUT RAW,
                          obj IN ORDSYS.ORDAudio,
                          attributes IN OUT NOCOPY CLOB);

--AUDIO PROCESSING METHODS
FUNCTION processCommand(ctx          IN OUT RAW,
                      obj           IN OUT NOCOPY ORDSYS.ORDAudio,
                      cmd            IN VARCHAR2,
                      arguments     IN VARCHAR2,
                      result         OUT RAW)

RETURN RAW;

END;
/
```

[Table 7-3](#) shows the methods supported in the `ORDPLUGINS.ORDX_DEFAULT_AUDIO` package and the exceptions raised if you call a method that is not supported.

**Table 7–3 Methods Supported in the `ORDPLUGINS.ORDX_DEFAULT_AUDIO` Package**

Name of Method	Level of Support
<code>setProperty</code> s	Supported; if the source is local, extract attributes from the local data and set the properties, but if the source is <code>NULL</code> , raise an <code>ORDSYS.ORDSourceExceptions.EMPTY_SOURCE</code> exception; if the source is a <code>BFILE</code> , then extract attributes from the <code>BFILE</code> and set the properties; if the source is neither local nor a <code>BFILE</code> , get the media content into a temporary LOB, extract attributes from the data, and set the properties.
<code>checkProperty</code> s	Supported; if the source is local, extract the attributes from the local data and compare them with the attribute values of the object, but if the source is <code>NULL</code> , raise an <code>ORDSYS.ORDSourceExceptions.EMPTY_SOURCE</code> exception; if the source is a <code>BFILE</code> , extract the attributes from the <code>BFILE</code> and compare them with the attribute values of the object; if the source is neither local nor a <code>BFILE</code> , get the media content into a temporary LOB, extract the attributes from the media content and compare them with the attribute values of the object.
<code>getAttribute</code>	Not supported - raises exceptions: <code>METHOD_NOT_SUPPORTED</code> and <code>AUDIO_PLUGIN_EXCEPTION</code> .
<code>getAllAttributes</code>	Supported; if the source is local, get the attributes and return them, but if the source is <code>NULL</code> , raise an <code>ORDSYS.ORDSourceExceptions.EMPTY_SOURCE</code> exception; otherwise, if the source is external, raise an <code>ORDSYS.ORDAudioExceptions.LOCAL_DATA_SOURCE_REQUIRED</code> exception.
<code>processCommand</code>	Not supported - raises exceptions: <code>METHOD_NOT_SUPPORTED</code> and <code>AUDIO_PLUGIN_EXCEPTION</code> .

### 7.2.1.2 Extending Oracle Multimedia to Support a New Audio Data Format

Extending Oracle Multimedia to support a new audio data format consists of the following steps:

1. Design your new audio data format.
  - a. To support a new audio data format, implement the required interfaces in the `ORDX_<format>_AUDIO` package in the `ORDPLUGINS` schema (where `<format>` represents the name of the new audio data format). See [Section 7.2.1.1](#) for a complete description of the interfaces for the `ORDX_DEFAULT_AUDIO` package. Use the package body example in [Example 7–2](#) as a template to create the audio package body.
  - b. Then, set the new format parameter in the `setFormat()` call to the appropriate format value to indicate to the audio object that package `ORDPLUGINS.ORDX_<format>_AUDIO` is available as a plug-in.
2. Implement your new audio data format and name it, for example, `ORDX_MY_AUDIO.SQL`.
3. Install your new `ORDX_MY_AUDIO.SQL` plug-in in the `ORDPLUGINS` schema.
4. Grant `EXECUTE` privileges on your new plug-in, for example, `ORDX_MY_AUDIO` plug-in, to `PUBLIC`.
5. In an application, set the format attribute to `my` to cause your plug-in to be invoked.

A package body listing is provided in [Example 7-2](#) to assist you in this operation. Add your variables to the places that say "--Your variables go here" and add your code to the places that say "--Your code goes here".

**Example 7-2 Show the Package Body for Extending Support to a New Audio Data Format**

```
CREATE OR REPLACE PACKAGE BODY ORDX_MY_AUDIO
AS
  --AUDIO ATTRIBUTES ACCESSORS
  PROCEDURE setProperties(ctx IN OUT RAW,
                        obj IN OUT NOCOPY ORDSYS.ORDAudio,
                        setComments IN NUMBER :=0)
  IS
  --Your variables go here.
  BEGIN
  --Your code goes here.
  END;
  FUNCTION checkProperties(ctx IN OUT RAW, obj IN OUT ORDSYS.ORDAudio)
  RETURN NUMBER
  IS
  --Your variables go here.
  BEGIN
  --Your code goes here.
  END;
  FUNCTION  getAttribute(ctx IN OUT RAW,
                        obj IN ORDSYS.ORDAudio,
                        name IN VARCHAR2)
  RETURN VARCHAR2
  IS
  --Your variables go here.
  BEGIN
  --Your code goes here.
  END;
  PROCEDURE getAllAttributes(ctx IN OUT RAW,
                            obj IN ORDSYS.ORDAudio,
                            attributes IN OUT NOCOPY CLOB)
  IS
  --Your variables go here.
  BEGIN
  --Your code goes here.
  END;
  -- AUDIO PROCESSING METHODS
  FUNCTION processCommand(
                                ctx          IN OUT RAW,
                                obj          IN OUT NOCOPY ORDSYS.ORDAudio,
                                cmd         IN VARCHAR2,
                                arguments  IN VARCHAR2,
                                result     OUT RAW)
  RETURN RAW
  IS
  --Your variables go here.
  BEGIN
  --Your code goes here.
  END;
END;
/
show errors;
```



## 7.2.2 Supporting Other ORDDoc Data Formats

Section 7.2.2.1 and Section 7.2.2.2 describe how to extend ORDDoc to support other data formats.

### 7.2.2.1 ORDPLUGINS.ORDX\_DEFAULT\_DOC Package

Use the following ORDPLUGINS.ORDX\_DEFAULT\_DOC package provided as a guide in developing your own ORDPLUGINS.ORDX\_<format>\_DOC media format package.

```
CREATE OR REPLACE PACKAGE ORDX_DEFAULT_DOC
authid current_user
AS

PROCEDURE setProperties(ctx IN OUT RAW,
                      obj IN OUT NOCOPY ORDSYS.ORDDoc,
                      setComments IN NUMBER := 0);

END;
/
```

Table 7-4 shows the method supported in the ORDPLUGINS.ORDX\_DEFAULT\_DOC package and the exception raised if the source is NULL.

**Table 7-4 Method Supported in the ORDPLUGINS.ORDX\_DEFAULT\_DOC Package**

Name of Method	Level of Support
setProperties	Supported; if the source is local, extract attributes from the local data and set the properties, but if the source is NULL, raise an ORDSYS.ORDSourceExceptions.EMPTY_SOURCE exception; if the source is a BFILE, then extract attributes from the BFILE and set the properties; if the source is neither local nor a BFILE, get the media content into a temporary LOB, extract attributes from the data, and set the properties.

### 7.2.2.2 Extending Oracle Multimedia to Support a New ORDDoc Data Format

Extending Oracle Multimedia to support a new ORDDoc data format consists of the following steps:

1. Design your new media data format.
  - a. To support a new media data format, implement the required interfaces in the ORDX\_<format>\_DOC package in the ORDPLUGINS schema (where <format> represents the name of the new media data format). See Section 7.2.2.1 for a complete description of the interfaces for the ORDX\_DEFAULT\_DOC package. Use the package body example in Example 7-3 as a template to create the package body.
  - b. Then, set the new format parameter in the setFormat() call to the appropriate format value to indicate to the media object that package ORDPLUGINS.ORDX\_<format>\_DOC is available as a plug-in.
2. Implement your new media data format and name it, for example, ORDX\_MY\_DOC.SQL.
3. Install your new ORDX\_MY\_DOC.SQL plug-in in the ORDPLUGINS schema.
4. Grant EXECUTE privileges on your new plug-in, for example, ORDX\_MY\_DOC plug-in, to PUBLIC.
5. In an application, set the format to my to cause your plug-in to be invoked.

A package body listing is provided in [Example 7-3](#) to assist you in this operation. Add your variables to the places that say "--Your variables go here" and add your code to the places that say "--Your code goes here".

**Example 7-3 Show the Package Body for Extending Support to a New ORDDoc Data Format**

```
CREATE OR REPLACE PACKAGE BODY ORDX_MY_DOC
AS
  --DOCUMENT ATTRIBUTES ACCESSORS
  PROCEDURE setProperties(ctx IN OUT RAW,
                        obj IN OUT NOCOPY ORDSYS.ORDDoc,
                        setComments IN NUMBER :=0)
  IS
  --Your variables go here.
  BEGIN
  --Your code goes here.
  END;
END;
/
show errors;
```

## 7.2.3 Supporting Other Video Data Formats

[Section 7.2.3.1](#) and [Section 7.2.3.2](#) describe how to extend ORDVideo to support other data formats.

### 7.2.3.1 ORDPLUGINS.ORDX\_DEFAULT\_VIDEO Package

Use the following ORDPLUGINS.ORDX\_DEFAULT\_VIDEO package provided as a guide in developing your own ORDPLUGINS.ORDX\_<format>\_VIDEO video format package. This package sets the mimeType field in the setProperties() method with a MIME type value that is dependent on the file format.

```
CREATE OR REPLACE PACKAGE ORDX_DEFAULT_VIDEO
authid current_user
AS
  --VIDEO ATTRIBUTES ACCESSORS
  FUNCTION getAttribute(ctx IN OUT RAW,
                      obj IN ORDSYS.ORDVideo,
                      name IN VARCHAR2)
  RETURN VARCHAR2;
  PROCEDURE setProperties(ctx IN OUT RAW,
                        obj IN OUT NOCOPY ORDSYS.ORDVideo,
                        setComments IN NUMBER := 0);
  FUNCTION checkProperties(ctx IN OUT RAW,obj IN ORDSYS.ORDVideo) RETURN NUMBER;

  -- must return name=value; name=value; ... pairs
  PROCEDURE getAllAttributes(ctx IN OUT RAW,
                            obj IN ORDSYS.ORDVideo,
                            attributes IN OUT NOCOPY CLOB);
  -- VIDEO PROCESSING METHODS
  FUNCTION processCommand(
                        ctx          IN OUT RAW,
                        obj          IN OUT NOCOPY ORDSYS.ORDVideo,
                        cmd          IN VARCHAR2,
                        arguments IN VARCHAR2,
                        result      OUT RAW)
  RETURN RAW;
```

```
END;  
/
```

Table 7–5 shows the methods supported in the `ORDPLUGINS.ORDX_DEFAULT_VIDEO` package and the exceptions raised if you call a method that is not supported.

**Table 7–5 Methods Supported in the `ORDPLUGINS.ORDX_DEFAULT_VIDEO` Package**

Name of Method	Level of Support
<code>getAttribute</code>	Not supported - raises exceptions: <code>METHOD_NOT_SUPPORTED</code> and <code>VIDEO_PLUGIN_EXCEPTION</code>
<code>setProperties</code>	Supported; if the source is local, extract attributes from the local data and set the properties, but if the source is <code>NULL</code> , raise an <code>ORDSYS.ORDSourceExceptions.EMPTY_SOURCE</code> exception; if the source is a <code>BFILE</code> , then extract attributes from the <code>BFILE</code> and set the properties; if the source is neither local nor a <code>BFILE</code> , get the media content into a temporary LOB, extract attributes from the data, and set the properties.
<code>checkProperties</code>	Supported; if the source is local, extract attributes from the local data and compare them with the attribute values of the object, but if the source is <code>NULL</code> , raise an <code>ORDSYS.ORDSourceExceptions.EMPTY_SOURCE</code> exception; if the source is a <code>BFILE</code> , then extract attributes from the <code>BFILE</code> data and compare them with the attribute values of the object; if the source is neither local nor a <code>BFILE</code> , get the media content into a temporary LOB, extract attributes from the media content and compare them with the attribute values of the object.
<code>getAllAttributes</code>	Supported; if the source is local, get the attributes and return them, but if the source is <code>NULL</code> , raise an <code>ORDSYS.ORDSourceExceptions.EMPTY_SOURCE</code> exception; otherwise, if the source is external, raise an <code>ORDSYS.ORDVideoExceptions.LOCAL_DATA_SOURCE_REQUIRED</code> exception.
<code>processCommand</code>	Not supported - raises exceptions: <code>METHOD_NOT_SUPPORTED</code> and <code>VIDEO_PLUGIN_EXCEPTION</code>

### 7.2.3.2 Extending Oracle Multimedia to Support a New Video Data Format

Extending Oracle Multimedia to support a new video data format consists of the following steps:

1. Design your new video data format.
  - a. To support a new video data format, implement the required interfaces in the `ORDX_<format>_VIDEO` package in the `ORDPLUGINS` schema (where `<format>` represents the name of the new video data format). See [Section 7.2.3.1](#) for a complete description of the interfaces for the `ORDX_DEFAULT_VIDEO` package. Use the package body example in [Example 7–4](#) as a template to create the video package body.
  - b. Then, set the new format parameter in the `setFormat()` call to the appropriate format value to indicate to the video object that package `ORDPLUGINS.ORDX_<format>_VIDEO` is available as a plug-in.
2. Implement your new video data format and name it, for example, `ORDX_MY_VIDEO.SQL`.
3. Install your new `ORDX_MY_VIDEO.SQL` plug-in in the `ORDPLUGINS` schema.
4. Grant `EXECUTE` privileges on your new plug-in, for example, `ORDX_MY_VIDEO` plug-in, to `PUBLIC`.

5. In an application, set the video format to my to cause your plug-in to be invoked.

A package body listing is provided in [Example 7-4](#) to assist you in this operation. Add your variables to the places that say "--Your variables go here" and add your code to the places that say "--Your code goes here".

**Example 7-4 Show the Package Body for Extending Support to a New Video Data Format**

```

CREATE OR REPLACE PACKAGE BODY ORDX_MY_VIDEO
AS
  --VIDEO ATTRIBUTES ACCESSORS
  FUNCTION getAttribute(ctx IN OUT RAW,
                      obj IN ORDSYS.ORDVideo,
                      name IN VARCHAR2)

  RETURN VARCHAR2
  IS
  --Your variables go here.
  BEGIN
  --Your code goes here.
  END;

  PROCEDURE setProperties(ctx IN OUT RAW,
                        obj IN OUT NOCOPY ORDSYS.ORDVideo,
                        setComments IN NUMBER :=0)

  IS
  --Your variables go here.
  BEGIN
  --Your code goes here.
  END;

  FUNCTION checkProperties(ctx IN OUT RAW, obj IN ORDSYS.ORDVideo) RETURN NUMBER
  IS
  --Your variables go here.
  BEGIN
  --Your code goes here.
  END;

  PROCEDURE getAllAttributes(ctx IN OUT RAW,
                            obj IN ORDSYS.ORDVideo,
                            attributes IN OUT NOCOPY CLOB)

  IS
  --Your variables go here.
  BEGIN
  --Your code goes here.
  END;

  -- VIDEO PROCESSING METHODS
  FUNCTION processCommand(
                                ctx          IN OUT RAW,
                                obj          IN OUT NOCOPY ORDSYS.ORDVideo,
                                cmd          IN VARCHAR2,
                                arguments IN VARCHAR2,
                                result OUT RAW)

  RETURN RAW
  IS
  --Your variables go here.
  BEGIN
  --Your code goes here.
  END;
END;
/
show errors;

```

## 7.2.4 Supporting Other Image Data Formats

Oracle Multimedia supports certain other image formats through the `setProperties()` method for foreign images. This method allows other image formats to be recognized by writing the values supplied to the `setProperties()` method for foreign images to the existing `ORDImage` data attributes. See the `setProperties()` for foreign images method in *Oracle Multimedia Reference* for more information and to determine the type of images that can be supported this way.

## 7.3 Extending Oracle Multimedia with a New Type

You can use any of the Oracle Multimedia objects types as the basis for a new type of your own creation as shown in [Example 7-5](#) for the `ORDImage` object type.

---

**Note:** When a type is altered, any dependent type definitions are invalidated. You will encounter this problem if you define a new type that includes an Oracle Multimedia object type attribute and the Oracle Multimedia object type is altered, which always occurs during an Oracle Multimedia installation upgrade.

A workaround to this problem is to revalidate all invalid type definitions with the following SQL statement:

```
SQL> ALTER TYPE <type-name> COMPILE;
```

---

### Example 7-5 Extend Oracle Multimedia `ORDImage` with a New Object Type

```
CREATE TYPE AnnotatedImage AS OBJECT
  ( image ORDSYS.ORDImage,
    description VARCHAR2(2000),
    MEMBER PROCEDURE SetProperties(SELF IN OUT AnnotatedImage),
    MEMBER PROCEDURE Copy(dest IN OUT AnnotatedImage),
    MEMBER PROCEDURE ProcessCopy(command IN VARCHAR2,
                                  dest IN OUT AnnotatedImage)
  );
/

CREATE TYPE BODY AnnotatedImage AS
  MEMBER PROCEDURE SetProperties(SELF IN OUT AnnotatedImage) IS
  BEGIN
    SELF.image.setProperties();
    SELF.description :=
      'This is an example of using Image object as a subtype';
  END SetProperties;
  MEMBER PROCEDURE Copy(dest IN OUT AnnotatedImage) IS
  BEGIN
    SELF.image.copy(dest.image);
    dest.description := SELF.description;
  END Copy;
  MEMBER PROCEDURE ProcessCopy(command IN VARCHAR2,
                                dest IN OUT AnnotatedImage) IS
  BEGIN
    SELF.image.processCopy(command, dest.image);
    dest.description := SELF.description;
  END ProcessCopy;
END;
/
```

After creating the new type, you can use it as you would any other type. For example:

```
CREATE OR REPLACE DIRECTORY ORDIMGDIR AS 'C:\TESTS';

CREATE TABLE my_example(id NUMBER, an_image AnnotatedImage);
INSERT INTO my_example VALUES (1,
    AnnotatedImage(
        ORDSYS.ORDImage.init('file','ORDIMGDIR','plaid.gif'));
COMMIT;
DECLARE
    myimage AnnotatedImage;
BEGIN
    SELECT an_image INTO myimage FROM my_example;
    myimage.SetProperties;
    DBMS_OUTPUT.PUT_LINE('This image has a description of ');
    DBMS_OUTPUT.PUT_LINE(myimage.description);
    UPDATE my_example SET an_image = myimage;
END;
/
```

## 7.4 Supporting Media Data Processing

[Section 7.4.1](#) and [Section 7.4.2](#) describe support for audio and video data processing.

### 7.4.1 Supporting Audio Data Processing

To support audio data processing, that is, the passing of an audio processing command and set of arguments to a format plug-in for processing, use the `processAudioCommand()` method. This method is available only for user-defined formats.

See the `processAudioCommand()` method in *Oracle Multimedia Reference* for a description.

### 7.4.2 Supporting Video Data Processing

To support video data processing, that is, the passing of a command and set of arguments to a format plug-in for processing, use the `processVideoCommand()` method. This method is only available for user-defined formats.

See the `processVideoCommand()` method in *Oracle Multimedia Reference* for a description.

---

---

## Tuning Tips for DBAs

This chapter provides information and advice for Oracle DBAs who want to achieve more efficient storage and management of multimedia data in the database when using Oracle Multimedia.

The goals of your Oracle Multimedia application determine the resource needs and how those resources should be allocated. Because application development and design decisions have the greatest effect on performance, standard tuning methods must be applied to the system planning, design, and development phases of the project to achieve optimal results for your Oracle Multimedia application in a production environment.

Multimedia data consists of a variety of media types including images, audio clips, video clips, line drawings, and so on. All these media types are typically stored in LOBs. LOBs can be either internal BLOBs (stored in an internal database tablespace) or BFILEs (external LOBs in operating system files outside of the database tablespaces). This chapter discusses the management of audio, image, and video data stored in BLOBs only.

The following general topics will help you to better manage your Oracle Multimedia LOB data:

- Understanding the performance profile of Oracle Multimedia operations (see [Section 8.1](#))
- Choosing LOB storage parameters for Oracle Multimedia objects (see [Section 8.2](#))
- Setting database initialization parameters (see [Section 8.3](#))

For more information about using LOBs in Oracle Database, see *Oracle Database SecureFiles and Large Objects Developer's Guide*.

### 8.1 Understanding the Performance Profile of Oracle Multimedia Operations

Multimedia data, as well as the operations that can be performed on that data, differs significantly from traditional types of data commonly stored in relational databases. A basic understanding of the performance profile of Oracle Multimedia operations can help you make better decisions when tuning your database for media performance.

The following tables summarize the general performance profiles for a set of commonly performed operations. There are two primary components to each profile. The I/O pattern is a general characterization of the primary type of I/O access and of how much of the media data the operation will read or write. Because some operations involve two media objects, the I/O pattern is described for both the source and

destination media objects. The second component is a general characterization of the level of CPU usage for the operation.

---



---

**Note:** The information in these tables describes general characterizations and I/O patterns, thus CPU usage may vary considerably for some media formats.

---



---

Table 8–1 shows the profile for loading and retrieving data, which applies to all Oracle Multimedia media types.

**Table 8–1 Performance Profile For All Multimedia Types**

Operation	I/O Pattern (Source)	I/O Pattern (Destination)	I/O Pattern (Amount)	CPU Usage
Load new media data into a database	N/A	Sequential write	All	Low
Retrieve media from a database	Sequential read	N/A	All	Low

Table 8–2 shows the profile for commonly used methods of the ORDIImage type.

**Table 8–2 Performance Profile For ORDIImage Methods**

Object Method	I/O Pattern (Source)	I/O Pattern (Destination)	I/O Pattern (Amount)	CPU Usage
setProperty( )	Sequential read	N/A	Media header	Low to medium
getMetadata( )	Sequential read	N/A	Media header	Low to medium
putMetadata( )	Sequential read	Sequential write	All	Low to medium
process( )	Sequential read	Sequential write	All	High
processCopy( )	Sequential read	Sequential write	All	High

Table 8–3 shows the profile for commonly used methods of the ORDDicom type.

**Table 8–3 Performance Profile For ORDDicom Methods**

Object Method	I/O Pattern (Source)	I/O Pattern (Destination)	I/O Pattern (Amount)	CPU Usage
setProperty( )	Sequential read	N/A	Media header	Low to medium
extractMetadata( )	Sequential read	N/A	Media header	Low to medium
writeMetadata( )	Sequential read	Sequential write	All	Low to medium
makeAnonymous( )	Sequential read	Sequential write	All	Low to medium
process( )	Sequential read	Sequential write	All	High
processCopy( )	Sequential read	Sequential write	All	High

Table 8–4 shows the profile for commonly used methods of the ORDAudio and ORDVideo types.



**Table 8–4 Performance Profile For ORDAudio and ORDVideo Methods**

Object Method	I/O Pattern (Source)	I/O Pattern (Destination)	I/O Pattern (Amount)	CPU Usage
setProperty( )	Sequential read	N/A	Media header	Low
getProperty( )	Sequential read	N/A	Media header	Low

## 8.2 Choosing LOB Storage Parameters for Oracle Multimedia Objects

The choices you make for specifying LOB storage attributes during table creation can significantly affect the performance of media load, retrieval, and processing operations. This section describes the most important options to consider and shows how the performance profile of Oracle Multimedia operations can affect the choice of LOB storage parameters. For detailed information about LOBs, see *Oracle Database SecureFiles and Large Objects Developer's Guide*.

### 8.2.1 SecureFile LOBs and BasicFile LOBs

SecureFile LOBs (SecureFiles) were introduced in Oracle Database 11g Release 1 (11.1) to supplement the original BasicFile LOBs implementation that is identified by the SQL parameter BASICFILE. The performance of SecureFile LOBs is significantly better than that of BasicFile LOBs, especially for large media data. Oracle recommends using SecureFile LOBs for storing media data whenever possible. SecureFile LOBs are identified by specifying the SQL parameter SECUREFILE.

### 8.2.2 TABLESPACE

You can achieve the best performance for LOBs by specifying storage for LOBs in a different tablespace than the one used for the table that contains the LOB. If many different LOBs will be accessed frequently, you may also want to specify a separate tablespace for each LOB column or attribute to reduce device contention.

### 8.2.3 CACHE, NOCACHE, and CACHE READS

The cache option is a part of the STORE AS clause, and determines whether LOB pages are stored in the buffer cache.

- When the option has the value `CACHE`, Oracle places LOB pages in the buffer cache where they can be shared among multiple users. Over time and if the LOB pages are no longer accessed, the pages will eventually be removed from the buffer cache.
- For the value `NOCACHE`, LOB pages are not placed in the buffer cache.
- For the value `CACHE READS`, LOB pages are placed in the cache for read operations only.

If your application will perform multiple read operations on a media object (for example: invoking the `setProperty( )` method and then generating a thumbnail image), enable read caching for the source media object.

### 8.2.4 LOGGING and NOLOGGING

The logging option is a part of the STORE AS clause and determines if REDO data is logged when a LOB is updated. If the `[NO]LOGGING` clause is omitted, neither

NOLOGGING nor LOGGING is specified and the logging attribute of the table or table partition defaults to the logging attribute of the tablespace in which it resides.

There is another alternative depending on how the cache option is specified.

- If CACHE is specified and [NO]LOGGING is omitted, LOGGING is automatically implemented (because you cannot have CACHE NOLOGGING).
- If CACHE is not specified and [NO]LOGGING is omitted, the [NO]LOGGING value is obtained from the tablespace in which the LOB segment resides.

NOLOGGING should be used only when you do not care about media recovery. However, if the disk, tape, or storage media should fail, you will not be able to recover your changes from the log because those changes were not logged.

NOLOGGING can be useful for bulk loading of media data. For instance, when loading data into the LOB, if you do not care about the redo operation and you can start the load over if it fails, set the LOB data segment storage characteristics to NOCACHE NOLOGGING. This option provides good performance for the initial loading of data.

After you finish loading data, if necessary, you can use the ALTER TABLE statement to modify the LOB storage characteristics for the LOB data segment for normal LOB operations (for example: to CACHE or NOCACHE LOGGING).

---



---

**Note:** Oracle Data Guard Redo Apply technology uses logging to populate the standby database. Thus, do not specify NOLOGGING with this Data Guard technology.

---



---

## 8.2.5 CHUNK

The CHUNK option applies only to BasicFile LOBs. It is part of the STORE AS clause, and indicates the size of the minimum unit of storage for the LOB data. CHUNK must be an integer multiple of the block size, and it must have a maximum value of 32K bytes.

Accessing LOBs in bigger chunks is more efficient. For the most efficient storage of media objects, which are almost always much larger than 32K in size, choose the maximum value of 32K.

## 8.2.6 Example of Setting LOB Storage Options

This section describes a simple example that shows how to use the performance profiles of Oracle Multimedia operations (see Tables 8-1 through 8-4 in [Section 8.1](#)) to guide your usage of LOB storage options.

In this example, Company X wants to build an archive for digital images. The archive will store a full resolution copy of the original image, and two Web-ready, JPEG format versions of the original at reduced scales, one at 50% of the original size and another at 25% of the original size. The database team plans to use the SQL\*Loader utility to bulk load all the initial images. Then, they will use a PL/SQL program to initialize the image data. Initialization consists of setting the properties for the original image and generating the scaled images. After initialization, the table will be prepared for the primary application, which retrieves images for Web-based users.

The following example shows a table definition for storing the images. The table will store the binary image data using SecureFiles in tablespace `tbs2`. All the other table data will be stored in tablespace `tbs1`.

```
create table images(id          integer primary key,
```

```

        original  ordsys.ordimage,
        scale50   ordsys.ordimage,
        scale25   ordsys.ordimage)
tablespace tbs1
lob(original.source.localdata)store as secureFile (tablespace tbs2)
lob(scale50.source.localdata)store as secureFile (tablespace tbs2)
lob(scale25.source.localdata)store as secureFile (tablespace tbs2);

```

After the table is created, the image data can be loaded. Loading image data generates a sequential write pattern to the LOB. Because no applications are reading the data during the load operation, there is no need to cache it. You can also improve load performance by disabling logging for the column that is loaded. The following command dynamically alters the table to prepare the original image column LOB for loading.

```
alter table images modify lob(original.source.localdata) (nocache nologging);
```

After loading, the next step is to set the image properties for the `original` column and generate the scaled versions that will be stored in the `scale50` and `scale25` columns. In this step, the source image will be fully read twice to generate the scaled versions. The scaled images that are generated will be written but not read. The following command dynamically alters the table to enable read caching for the source image, and disables caching and logging for the destination images.

```
alter table images modify lob(original.source.localdata) (cache reads);
alter table images modify lob(scale50.source.localdata) (nocache nologging);
alter table images modify lob(scale25.source.localdata) (nocache nologging);
```

After running the program to set the properties of the `original` image and generate the scaled versions, the LOB storage attributes can be optimized for the main application that retrieves images for users who will view them in a Web browser. Because the archive contains millions of images, it is not expected that users will want to view the same image at the same time. Thus, there is little benefit to caching the image data. The following command re-enables logging for the LOBs and disables caching.

```
alter table images modify lob(original.source.localdata) (nocache logging);
alter table images modify lob(scale50.source.localdata) (nocache logging);
alter table images modify lob(scale25.source.localdata) (nocache logging);
```

## 8.3 Setting Database Initialization Parameters

[Section 8.2](#) points out that you can disable logging of LOB data at the column level to reduce the amount of I/O to the redo log. However, if logging cannot be disabled, additional database tuning may be needed. Specifically, you may need to increase the size of the redo log buffer to prevent load processes from waiting.

The initialization parameter `LOG_BUFFER` specifies the amount of memory (in bytes) that Oracle uses when buffering redo entries to a redo log file. Redo log entries contain a record of the changes that have been made to the database block buffers. The LGWR process writes redo log entries from the log buffer to a redo log file.

If redo log data is written to the redo log buffer faster than the LGWR process can write it to disk, the buffer will become full and user sessions will be forced to wait until space is available. The wait event "log buffer space" indicates the number of times a session had to wait for space in the redo log buffer. You can monitor this event in the `V$SYSTEM_EVENT` dynamic view to learn how many times a session had to wait for log buffer space. If sessions are forced to wait often for log buffer space, consider increasing the value of the `LOG_BUFFER` initialization parameter.

See *Oracle Database Performance Tuning Guide* and *Oracle Database Reference* for comprehensive information on setting database initialization parameters.

---

---

## Oracle Multimedia Examples

This chapter provides examples that show common operations with Oracle Multimedia. The examples are presented by the following data groups:

- Audio (Section 9.1)
- Media (Section 9.2)
- Image (Section 9.3)
- Video (Section 9.4)

These sections include scripts that perform common operations, including these:

- Populating an ORDAudio object with BLOB data (Section 9.1.2)
- Populating an ORDDoc object from a file data source (Section 9.2.1)
- Populating an ORDImage object from a file data source (Section 9.3.1)
- Loading an image table from an HTTP data source (Section 9.3.2)
- Addressing globalization support issues (Section 9.3.3)

These scripts, and other examples, can be found on the Oracle Multimedia Sample Code section of the Oracle Technology Network (OTN) Web site at

<http://www.oracle.com/technology/products/multimedia/>

Select **Sample Code** under **Oracle Multimedia Resources** to go to the Oracle Multimedia Sample Code Web page. On that page, select **Use Multimedia and PL/SQL to manage media content** under **Multimedia Code Samples**.

### 9.1 Audio Data Examples

Audio data examples using Oracle Multimedia include the following common operations:

- Using Oracle Multimedia with object views (See Section 9.1.1)
- Using a set of scripts for populating an ORDAudio object with BLOB data stored in the database (See Section 9.1.2)

Reference information on the methods used in these examples is presented in *Oracle Multimedia Reference*.

#### 9.1.1 Using Audio Types with Object Views

This section describes how to use audio types with object views. Just as a view is a virtual table, an object view is a virtual object table.

Oracle provides object views as an extension of the basic relational view mechanism. By using object views, you can create virtual object tables -- of either built-in or user-defined types -- from data stored in the columns of relational or object tables in the database.

Object views can offer specialized or restricted access to the data and objects in a database. For example, you might use an object view to provide a version of an employee object table that does not have attributes containing sensitive data or a deletion method. Object views also let you try object-oriented programming without permanently converting your tables. Using object views, you can convert data gradually and transparently from relational tables to object-relational tables.

In [Example 9-1](#), consider the following relational table (containing no ORDAudio objects).

**Example 9-1 Define a Relational Table Containing No ORDAudio Object**

```
create table flat (
  id                NUMBER,
  description       VARCHAR2(4000),
  localData        BLOB,
  srcType          VARCHAR2(4000),
  srcLocation      VARCHAR2(4000),
  srcName          VARCHAR2(4000),
  upDateTime       DATE,
  local            NUMBER,
  format           VARCHAR2(31),
  mimeType         VARCHAR2(4000),
  comments         CLOB,
  encoding         VARCHAR2(256),
  numberOfChannels NUMBER,
  samplingRate     NUMBER,
  sampleSize      NUMBER,
  compressionType VARCHAR2(4000),
  audioDuration   NUMBER,
)
--
-- store audio data as SecureFile LOBs
--
LOB(localData) STORE AS SECUREFILE;
```

You can create an object view on the relational table shown in [Example 9-1](#) as follows in [Example 9-2](#).

**Example 9-2 Define an Object View Containing an ORDAudio Object and Relational Columns**

```
create or replace view object_audio_v as
select
  id,
  ORDSYS.ORDAudio(T.description,
  ORDSYS.ORDSource(
    T.localData, T.srctype, T.srcLocation, T.srcName, T.updateTime,
    T.local),
  T.format,
  T.mimeType,
  T.comments,
  T.encoding,
  T.numberOfChannels,
  T.samplingRate,
```

```

        T.sampleSize,
        T.compressionType,
        T.audioDuration)
from flat T;

```

Object views provide the flexibility of looking at the same relational or object data in more than one way. Therefore, you can use different in-memory object representations for different applications without changing the way you store the data in the database. See *Oracle Database Concepts* for more information about defining, using, and updating object views.

## 9.1.2 Scripts for Populating an ORDAudio Object with BLOB Data

The scripts presented in this section demonstrate how to populate an Oracle Multimedia ORDAudio object from an existing BLOB stored in the database.

[Table 9-1](#) lists each script by name, along with a brief description of the operations it performs. Each script is included and described in further detail in the sections that follow.

**Table 9-1 Audio Scripts**

Script Name	Operations Performed
<code>create_mediadir.sql</code> <a href="#">Example 9-3</a>	Creates an audio data load directory. (See <a href="#">Section 9.1.2.1</a> )
<code>create_soundtable.sql</code> <a href="#">Example 9-4</a>	Creates and populates the <code>soundtable</code> table. (See <a href="#">Section 9.1.2.2</a> )
<code>create_audtable.sql</code> <a href="#">Example 9-5</a>	Creates and populates the <code>audio_table</code> table. (See <a href="#">Section 9.1.2.3</a> )
<code>import_aud.sql</code> <a href="#">Example 9-6</a>	Loads the audio data. This script imports the audio data from the <code>soundtable</code> table into the <code>audio_table</code> table using the ORDAudio <code>import()</code> method. (See <a href="#">Section 9.1.2.4</a> )
<code>copy_audblob.sql</code> <a href="#">Example 9-7</a>	Copies the BLOB data from the <code>soundtable</code> table to the <code>audio_table</code> table using a SQL UPDATE statement. (See <a href="#">Section 9.1.2.5</a> )
<code>showprop_aud.sql</code> <a href="#">Example 9-8</a>	Displays the properties of the loaded audio data stored in the <code>audio_table</code> table. (See <a href="#">Section 9.1.2.6</a> )
<code>setup_audsample.sql</code> <a href="#">Example 9-9</a>	Automates the process by running the previous audio scripts in the required order. (See <a href="#">Section 9.1.2.7</a> )
<code>cleanup_audsample.sql</code> <a href="#">Example 9-10</a>	Cleans up the process by removing the sample tables, directories, and procedures from your database. (See <a href="#">Section 9.1.2.8</a> )

### 9.1.2.1 Create an Audio Data Load Directory

The `create_mediadir.sql` script creates the audio data load directory. This script is shown in [Example 9-3](#).

To successfully load the audio data, you must have a directory created on your system. [Example 9-3](#) uses the `mediadir` directory. This directory must contain your sample audio clip file. The following examples use the sample file `aud1.wav`. Actually, you can copy any supported audio files to the `mediadir` directory to run the scripts in these examples. The directory path and disk drive of the `mediadir` directory must be specified in the `CREATE DIRECTORY` statement in the file `create_mediadir.sql`.

---

**Note:** You must have the `CREATE ANY DIRECTORY` privilege to run this script. If you wish to delete previous instances of the audio data load directory, you will also need the `DROP ANY DIRECTORY` privilege. You must edit the `create_mediadir.sql` file to replace the directory path in the `CREATE DIRECTORY` statement. If you run the `create_mediadir.sql` script under a different user than you run the other demo scripts, you must also replace the user in the `GRANT READ` statement. See the following script for detailed instructions.

---

### **Example 9-3** `create_mediadir.sql`

```
-- create_mediadir.sql
-- You must have the CREATE ANY DIRECTORY privilege to run this script.
--
-- You must edit this script to:
--   o replace the directory path in the CREATE DIRECTORY statement
--     with your own (see below for details)
--   o uncomment the GRANT READ ON DIRECTORY statement and grant to the
--     user under which you will run this demo

SET SERVEROUTPUT ON;
SET ECHO ON;

-- You need DROP ANY DIRECTORY privilege to delete a directory. If there
-- is no need to delete the directory, then leave the next line
-- commented out.
-- DROP DIRECTORY mediadir;

-- Create the mediadir load directory, the directory where the image
-- clips reside. You must specify the disk drive and path in the
-- CREATE DIRECTORY statement below. Edit this file to replace the
-- directory specification with your own.
CREATE OR REPLACE DIRECTORY mediadir AS 'C:/mediadir';

-- If you intend to run this demo under a user other than the user
-- that just created this directory, edit the following grant to grant
-- READ on the directory just created to the user under which you will
-- run the demo scripts. For example, if you will run the demo scripts
-- under user 'SCOTT', then replace the string "<USER>" with "SCOTT".
-- Then uncomment the following GRANT statement. There is no need to
-- do this if the user creating the directory will also be used to run
-- the other demo scripts.
-- GRANT READ ON DIRECTORY mediadir TO <USER>;
```

#### **9.1.2.2 Create and Populate the soundtable Table**

The `create_soundtable.sql` script creates and populates the `soundtable` table. This table contains a `BLOB` column; it shows how to populate a table with an Oracle Multimedia `ORDAudio` column from a table with a `BLOB` column. The `soundtable` table is created for demonstration purposes only. This script is shown in [Example 9-4](#).



To demonstrate how to populate a table with an Oracle Multimedia ORDAudio column from a table with a BLOB column, first we must have a table with a BLOB column. The `soundtable` table is our sample table with a BLOB column. This script creates the `soundtable` table, inserts a row with an empty BLOB, loads the BLOB with audio data, and then checks the length of the BLOB data.

Be sure to change the data file name in the `create_soundtable.sql` script to correspond with the name of the data file you use.

---



---

**Note:** This script must be run from a user with the CREATE TABLE privilege.

---



---

#### **Example 9-4** `create_soundtable.sql`

```
-- create_soundtable.sql
--
-- This script must be run from a user with CREATE TABLE privilege.
--
-- Create the soundtable table. This table is used ONLY to show
-- how to copy data from a BLOB column to an ORDAudio column.
--
-- Insert a row into the table with an empty BLOB.
-- Load the row with BLOB data by pointing to the audio file to
-- be loaded from the directory specified using the BFILE data
-- type.
-- Close the files and commit the transaction.
-- Check the length of the BLOB loaded. Is the length
-- what you are expecting?
--

SET SERVEROUTPUT ON;
SET ECHO ON;

DROP TABLE soundtable PURGE;
CREATE TABLE soundtable ( id number,
                          sound BLOB
                          default EMPTY_BLOB() )
--
-- store audio data as SecureFile LOBs
--
LOB(sound) STORE AS SECUREFILE;
--
INSERT INTO soundtable(id, sound) VALUES (1, EMPTY_BLOB());
COMMIT;
DECLARE
    f_lob BFILE := BFILENAME('MEDIADIR', 'aud1.wav');
    b_lob BLOB;
    Lob BLOB;
    Length INTEGER;
BEGIN

    SELECT sound INTO b_lob FROM soundtable WHERE id=1 FOR UPDATE;

-- Open the LOBs.
    dbms_lob.open(f_lob, dbms_lob.file_readonly);
    dbms_lob.open(b_lob, dbms_lob.lob_readwrite);

-- Populate the BLOB from the 'aud1.wav' file in the BFILE
    dbms_lob.loadfromfile
```

```
(b_lob, f_lob, dbms_lob.getlength(f_lob));

-- Close the LOBs.
dbms_lob.close(b_lob);
dbms_lob.close(f_lob);
COMMIT;

-- Select the LOB:
SELECT sound INTO Lob FROM soundtable
WHERE ID = 1;

-- Opening the LOB is optional.
DBMS_LOB.OPEN (Lob, DBMS_LOB.LOB_READONLY);
-- Get the length of the LOB and verify length is not null.
length := DBMS_LOB.GETLENGTH(Lob);
IF length IS NULL THEN
  DBMS_OUTPUT.PUT_LINE('LOB is null. ');
ELSE
  DBMS_OUTPUT.PUT_LINE('The length is ' || length);
END IF;
-- Closing the LOB is mandatory if you have opened it.
DBMS_LOB.CLOSE (Lob);
END;
/
```

### 9.1.2.3 Create and Populate the audio\_table Table

The `create_audtable.sql` script creates and populates the `audio_table` table. This table contains an `ORDAudio` column. This script creates the `audio_table` table with two columns (`id`, `audio`), inserts two rows, and initializes the `audio` column. This script is shown in [Example 9-5](#).

---

---

**Note:** Calling the `ORDAudio.init()` method initializes the blob attribute of the `ORDAudio` object. This is necessary to populate the BLOB with data.

This script must be run from a user with the `CREATE TABLE` privilege.

---

---

#### **Example 9-5** *create\_audtable.sql*

```
-- create_audtable.sql
--
-- This script must be run from a user with the CREATE TABLE privilege.
--
-- Create the audio_table table.
-- Insert two rows with empty BLOBs and initialize object attributes.

SET SERVEROUTPUT ON;
SET ECHO ON;

DROP TABLE audio_table PURGE;

CREATE TABLE audio_table ( id NUMBER,
                           audio ORDAudio )
LOB(audio.source.localData) STORE AS SECUREFILE;
```

```
-- Insert rows with an empty BLOB and initialize the object attributes.

INSERT INTO audio_table VALUES(1,ORDAudio.init());
INSERT INTO audio_table VALUES(2,ORDAudio.init());

COMMIT;
```

#### 9.1.2.4 Load the Audio Data

The `import_aud.sql` script imports audio data from an audio file (in the `soundtable` table) into the `ORDAudio` column in the `audio_table` table using the `ORDAudio import()` method. To import data into a blob attribute where the audio clip will be stored, the audio column in the row must be selected for update. This script is shown in [Example 9-6](#).

To successfully run this script, you must copy one audio clip to your `mediadir` directory using the names specified in this script, or modify this script to match the file names of your audio clips.

This script should load the same audio clip that was loaded by the `create_soundtable.sql` script. It is used later in the `showprop_aud.sql` script to show that data loaded with the `import()` method matches the data copied from the BLOB column of the `soundtable` table.

---



---

**Note:** This script should be run from the same user as the scripts in [Example 9-4](#) and [Example 9-5](#).

---



---

#### Example 9-6 `import_aud.sql`

```
--import_aud.sql

DECLARE
  obj ORDAUDIO;
  ctx RAW(64) := NULL;

BEGIN

  -- selects the audio column for update
  SELECT audio INTO obj FROM audio_table WHERE id = 1 FOR UPDATE;

  --import audio clip aud1.wav from mediadir
  obj.setSource('FILE', 'MEDIADIR', 'aud1.wav');
  obj.import(ctx);

  --set properties
  obj.setProperties(ctx);

  --update table with audio object
  UPDATE audio_table SET audio = obj WHERE id = 1;

  COMMIT;

END;
/
```

### 9.1.2.5 Copy the BLOB Data to the ORDAudio Object

The `copy_audblob.sql` script copies the audio data in the `sound` column of the `soundtable` table into the ORDAudio object column of the `audio_table` table for a row with `id=2`. The script uses a SQL UPDATE statement to copy the BLOB data from the `soundtable` table into the `T.audio.source.localData` attribute in the `audio_table` table, which performs the copy operation. It also sets the properties and updates the time stamp for the new BLOB stored in the ORDAudio object. This script is shown in [Example 9-7](#).

---

**Note:** This script must be run from the same user as the scripts in [Example 9-4](#), [Example 9-5](#), and [Example 9-6](#).

---

#### Example 9-7 `copy_audblob.sql`

```
--copy_audblob.sql
--

SET SERVEROUTPUT ON;
SET ECHO ON;

-- Use the SQL UPDATE statement to set the contents of
-- T.audio.source.localData to be the same as the BLOB stored
-- in the sound column of the soundtable table. This is an easy way
-- to copy a BLOB stored in the database into a row containing
-- a column defined as an interMedia ORDAudio object type.
--
-- In this case, the BLOB (an audio clip), which was stored in
-- a row in the soundtable table containing a sound column
-- defined as a BLOB data type for an ID=1 is copied to a row
-- in the audio_table table containing an audio column defined as
-- an ORDAudio object type in which the ID=2. The audio
-- clip is referenced through the source attribute of the
-- ORDAudio object type to the underlying localData attribute
-- of the ORDSource object type.
--
-- Then (1) Call setProperties() and (2) call setUpdateTime()
-- for this new BLOB stored in the ORDAudio object type.
-- Create a procedure to do this.

CREATE OR REPLACE PROCEDURE update_proc IS

    obj ORDAudio;
    ctx RAW(64) :=NULL;

BEGIN
    UPDATE audio_table T SET T.audio.source.localData = (SELECT sound FROM
        soundtable S WHERE S.id = 1) WHERE T.id=2;
    COMMIT;

    SELECT audio INTO obj FROM audio_table WHERE id = 2 FOR UPDATE;
    obj.setProperties(ctx);
    obj.setUpdateTime(SYSDATE);
    UPDATE audio_table SET audio = obj WHERE id = 2;
    COMMIT;

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Operation failed');
```

```

END;
/

EXECUTE UPDATE_PROC;

```

### 9.1.2.6 Show the Properties of the Loaded Audio Data

The `showprop_aud.sql` script displays the properties of the audio data clips stored in the `audio_table` table in rows with `id=1` and `id=2`. They should be identical. Different load methods were used to load the same audio clip from the `soundtable` table into two rows in the `audio_table` table. This script verifies that audio data loaded using the `ORDAudio import()` method matches audio data copied from a BLOB column of the `soundtable` table. This script is shown in [Example 9-8](#).

---

**Note:** This script must be run from the same user as the scripts in [Example 9-4](#), [Example 9-5](#), [Example 9-6](#), and [Example 9-7](#).

---

#### **Example 9-8** *showprop\_aud.sql*

```

-- showprop_aud.sql
--

SET SERVEROUTPUT ON;
SET ECHO ON;

--Query audio_table for ORDAudio content.
--
-- This script verifies that the properties of the audio data
-- in the row with ID=1 (which was populated with the ORDAudio
-- import() method), match the properties of the audio data
-- in the row with ID=2 (which was populated by copying data
-- from a BLOB in the soundtable table.
--
--

DECLARE
  audio ORDAudio;
  idnum integer;
  properties_match BOOLEAN;
  ctx RAW(64) :=NULL;

BEGIN
  -- Check the properties of the audio data clip imported into the
  -- ORDAudio object type. Properties for ID=1 should be identical
  -- with ID=2.

  dbms_output.put_line(' Properties of these audio clips are identical:');

  SELECT id, audio INTO idnum, audio FROM audio_table WHERE id=1;
  dbms_output.put_line('Properties for id: ' || idnum);

  dbms_output.put_line('audio encoding: ' || audio.getEncoding);
  dbms_output.put_line('audio number of channels: ' ||
    audio.getNumberOfChannels);
  dbms_output.put_line('audio MIME type: ' || audio.getMimeType);
  dbms_output.put_line('audio file format: ' || audio.getFormat);
  dbms_output.put_line('BLOB Length: ' ||
    TO_CHAR(audio.getContentLength(ctx)));

  dbms_output.put_line

```

```

('-----');

-- Check the properties of the audio data clip copied into the
-- ORDAudio object type from a BLOB stored in the database.
-- Properties for ID=1 should be identical with ID=2.

SELECT id, audio INTO idnum, audio FROM audio_table WHERE id=2;
dbms_output.put_line('Properties for id: ' || idnum);
dbms_output.put_line('audio encoding: ' || audio.getEncoding);
dbms_output.put_line('audio number of channels: ' ||
    audio.getNumberOfChannels);
dbms_output.put_line('audio MIME type: ' || audio.getMimeType);
dbms_output.put_line('audio file format: ' || audio.getFormat);
dbms_output.put_line('BLOB Length: ' ||
    TO_CHAR(audio.getContentLength(ctx)));
dbms_output.put_line
    ('-----');

END;
/

```

The results from running the script `showprop_aud.sql` show that the properties are identical for each stored audio clip.

Properties of these audio clips are identical:

```

Properties for id:          1

audio encoding:           MS_PCM
audio number of channels: 1
audio MIME type:          audio/x-wav
audio file format:        WAVE
BLOB Length:              15932
-----
Properties for id:          2

audio encoding:           MS_PCM
audio number of channels: 1
audio MIME type:          audio/x-wav
audio file format:        WAVE
BLOB Length:              15932
-----

```

PL/SQL procedure successfully completed.

### 9.1.2.7 Automate the ORDAudio Examples

The `setup_audsample.sql` script runs each of the previous audio scripts in the correct order to automate this process. This script is shown in [Example 9-9](#).

---



---

**Note:** This script must be run from a user with CREATE ANY DIRECTORY and CREATE TABLE privileges.

---



---

#### **Example 9-9** *setup\_audsample.sql*

```

-- setup_audsample.sql
--
-- This script automates the demo by invoking each script in
-- the required order.

```

```

--

-- Create the mediadir load directory
@create_mediadir.sql

-- Create a soundtable table and populate it with
-- an audio clip:
@create_soundtable.sql

-- Create an audtable table
@create_audtable.sql

--import an audio clip
@import_aud.sql

-- Copy a BLOB into an ORDAudio object, set the properties,
-- and update the time:
@copy_audblob.sql

-- Check the properties of the audio clips. The properties
-- should be identical:
@showprop_aud.sql

--exit;

```

### 9.1.2.8 Clean Up the ORDAudio Examples

The `cleanup_audsample.sql` script removes the sample tables, directories, and procedures created by the previous audio scripts from your database. This script is shown in [Example 9-10](#).

---



---

**Note:** This script must be run from a user with DROP ANY DIRECTORY privilege. This script should be run from the user that was used to run the previous audio scripts.

---



---

#### **Example 9-10** *cleanup\_audsample.sql*

```

-- cleanup_audsample.sql
--
-- This script removes all tables, procedures, and directories
-- created by this demonstration. You must have the DROP ANY
-- DIRECTORY privilege to drop the audio load directory. This
-- script should be run under the same user as the demo was run
-- under.

-- drop the audio load directory.
DROP DIRECTORY mediadir;

-- Drop the tables created by the demo.
DROP TABLE soundtable PURGE;
DROP TABLE audio_table PURGE;

-- Drop the Update procedure.
DROP PROCEDURE update_proc;

commit;

```

## 9.2 Media Data Examples

Media data examples using Oracle Multimedia include the following common operation on heterogeneous data:

- Using a set of scripts for populating an ORDDoc object from a file data source (See [Section 9.2.1](#))

Reference information on the methods used in these examples is presented in *Oracle Multimedia Reference*.

### 9.2.1 Scripts for Populating an ORDDoc Object from a File Data Source

The scripts presented in this section demonstrate how to populate an ORDDoc object from an existing file.

[Table 9–2](#) lists each script by name, along with a brief description of the operations it performs. Each script is included and described in further detail in the sections that follow.

**Table 9–2 Media Scripts**

Script Name	Operations Performed
<code>create_mediadir.sql</code> <a href="#">Example 9–11</a>	Creates a media data load directory. (See <a href="#">Section 9.2.1.1</a> )
<code>create_doctable.sql</code> <a href="#">Example 9–12</a>	Creates and populates the <code>doc_table</code> table. (See <a href="#">Section 9.2.1.2</a> )
<code>import_doc.sql</code> <a href="#">Example 9–13</a>	Loads the media data. This script imports the media data from a file into the <code>doc_table</code> table using the ORDDoc <code>import()</code> method. (See <a href="#">Section 9.2.1.3</a> )
<code>read_doc.sql</code> <a href="#">Example 9–14</a>	Reads the media data from a BLOB using a stored procedure. (See <a href="#">Section 9.2.1.4</a> )
<code>showprop_doc.sql</code> <a href="#">Example 9–15</a>	Displays the properties of the loaded media data stored in the <code>doc_table</code> table. (See <a href="#">Section 9.2.1.5</a> )
<code>setup_docsample.sql</code> <a href="#">Example 9–16</a>	Automates the process by running the previous media scripts in the required order. (See <a href="#">Section 9.2.1.6</a> )
<code>cleanup_docsample.sql</code> <a href="#">Example 9–17</a>	Cleans up the process by removing the sample tables, directories, and procedures from your database. (See <a href="#">Section 9.2.1.7</a> )

#### 9.2.1.1 Create a Media Data Load Directory

The `create_mediadir.sql` script creates the media data load directory. This script is shown in [Example 9–11](#).

To successfully load the media data, you must have a directory created on your system. [Example 9–11](#) uses the `mediadir` directory. This directory must contain your sample audio media files. The following examples use the sample files `aud1.wav` and `aud2.mp3`. Actually, you can copy any supported media files to the `mediadir` directory to run the scripts in these examples. The directory path and disk drive of the



mediadir directory must be specified in the CREATE DIRECTORY statement in the create\_mediadir.sql file.

---

**Note:** You must have the CREATE ANY DIRECTORY privilege to run this script. If you wish to delete previous instances of the media data load directory, you will also need the DROP ANY DIRECTORY privilege. You must edit the create\_mediadir.sql file to replace the directory path in the CREATE DIRECTORY statement. If you run the create\_mediadir.sql script under a different user than you run the other demo scripts, you must also replace the user in the GRANT READ statement. See the following script for detailed instructions.

---

#### **Example 9-11 create\_mediadir.sql**

```
-- create_mediadir.sql
-- You must have the CREATE ANY DIRECTORY privilege to run this script.
--
-- You must edit this script to:
--   o replace the directory path in the CREATE DIRECTORY statement
--     with your own (see below for details)
--   o uncomment the GRANT READ ON DIRECTORY statement and grant to the
--     user under which you will run this demo

SET SERVEROUTPUT ON;
SET ECHO ON;

-- You need DROP ANY DIRECTORY privilege to delete a directory. If there
-- is no need to delete the directory, then leave the next line
-- commented out.
-- DROP DIRECTORY mediadir;

-- Create the mediadir load directory, the directory where the media
-- data resides. You must specify the disk drive and path in the
-- CREATE DIRECTORY statement below. Edit this file to replace the
-- directory specification with your own.
CREATE OR REPLACE DIRECTORY mediadir AS 'C:/mediadir';

-- If you intend to run this demo under a user other than the user
-- that just created this directory, edit the following grant to grant
-- READ on the directory just created to the user under which you will
-- run the demo scripts. For example, if you will run the demo scripts
-- under user 'SCOTT', then replace the string "<USER>" with "SCOTT".
-- Then uncomment the following GRANT statement. There is no need to
-- do this if the user creating the directory will also be used to run
-- the other demo scripts.
-- GRANT READ ON DIRECTORY mediadir TO <USER>;
```

#### **9.2.1.2 Create and Populate the doc\_table Table**

The create\_doctable.sql script creates and populates the doc\_table table. This table contains an ORDDoc column. This script creates the doc\_table table with two columns (id, document), inserts two rows, and initializes the document column. This script is shown in [Example 9-12](#).

---

---

**Note:** Calling the `ORDDoc.init()` method initializes the blob attribute of the `ORDDoc` object. This is necessary to populate the BLOB with data.

This script must be run from a user with the `CREATE TABLE` privilege.

---

---

**Example 9–12** *create\_doctable.sql*

```
-- create_doctable.sql
--
-- This script must be run from a user with the CREATE TABLE privilege.
--
-- Create the doc_table table.
-- Insert two rows with empty ORDDoc columns.
--

SET SERVEROUTPUT ON;
SET ECHO ON;

DROP TABLE doc_table PURGE;
CREATE TABLE doc_table ( id NUMBER,
                        document ORDDoc )
LOB(document.source.localData) STORE AS SECUREFILE;

-- Insert a row with an empty ORDDoc object.
INSERT INTO doc_table VALUES(1,ORDDoc.init());
-- Insert a row with an empty BLOB.
INSERT INTO doc_table VALUES(2,ORDDoc.init());

COMMIT;
```

### 9.2.1.3 Load the Media Data

The `import_doc.sql` script imports media data from a media file into the `ORDDoc` column in the `doc_table` table using the `ORDDoc import()` method. To import data into a blob attribute where the media data will be stored, the `document` column in the row must be selected for update. This script is shown in [Example 9–13](#).

To successfully run this script, you must copy two media files to your `mediadir` directory using the names specified in this script, or modify this script to match the file names of your media files.

---

---

**Note:** This script should be run from the same user as the script in [Example 9–12](#).

---

---

**Example 9–13** *import\_doc.sql*

```
-- import_doc.sql
--
-- This script uses the import method to load the media data into the
-- ORDDoc column. It then extracts properties from the media using
-- the setProperties method.
--
-- To successfully run this script, you must copy two media files to your
-- MEDIADIR directory using the names specified in this script, or modify
-- this script to match the file names of your media.
--
```

```

SET SERVEROUTPUT ON;
SET ECHO ON;

DECLARE
    obj ORDDOC;
    ctx RAW(64) := NULL;

BEGIN
-- This imports the audio file aud1.wav from the MEDIADIR directory
-- on a local file system (srcType=file) and sets the properties.

    SELECT document INTO obj FROM doc_table WHERE id = 1 FOR UPDATE;
    obj.setSource('FILE', 'MEDIADIR', 'aud1.wav');
    obj.import(ctx, TRUE);

    UPDATE doc_table SET document = obj WHERE id = 1;
    COMMIT;

-- This imports the audio file aud2.mp3 from the MEDIADIR directory
-- on a local file system (srcType=file) and sets the properties.
    SELECT document INTO obj FROM doc_table WHERE id = 2 FOR UPDATE;
    obj.setSource('FILE', 'MEDIADIR', 'aud2.mp3');
    obj.import(ctx, TRUE);

    UPDATE doc_table SET document = obj WHERE id = 2;
    COMMIT;
END;
/

```

#### 9.2.1.4 Read the Media Data from the BLOB

The `read_doc.sql` script reads media data from a BLOB by creating the stored procedure `readdocument`. This procedure reads a specified amount of media data from the BLOB attribute, beginning at a particular offset, until all the media data is read. This script is shown in [Example 9–14](#).

---

**Note:** This script must be run from the same user as the scripts in [Example 9–12](#) and [Example 9–13](#).

---

#### **Example 9–14** *read\_doc.sql*

```

--read_doc.sql

SET SERVEROUTPUT ON
SET ECHO ON

----Read from the OrdDoc object column in the doc_table.
create or replace procedure readdocument as
obj ORDDoc;
buffer RAW (32767);
numBytes BINARY_INTEGER := 32767;
startpos integer := 1;
read_cnt integer := 1;
ctx RAW(64) := NULL;
BEGIN
    select document into obj from doc_table where id = 1;
    LOOP

```

```

obj.readFromSource(ctx,startPos,numBytes,buffer);
DBMS_OUTPUT.PUT_LINE('BLOB Length: ' || TO_CHAR(obj.getContentLength()));
DBMS_OUTPUT.PUT_LINE('start position: ' || startPos);
DBMS_OUTPUT.PUT_LINE('doing read: ' || read_cnt);
startpos := startpos + numBytes;
read_cnt := read_cnt + 1;
END LOOP;

-- Note: Add your own code here to process the media data being read;
-- this routine just reads the data into the buffer 32767 bytes
-- at a time, then reads the next chunk, overwriting the first
-- buffer full of data.
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('End of data ');
  WHEN ORDSYS.ORDSourceExceptions.METHOD_NOT_SUPPORTED THEN
    DBMS_OUTPUT.PUT_LINE('ORDSourceExceptions.METHOD_NOT_SUPPORTED caught');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('EXCEPTION caught');
END;
/
show errors

```

To execute the stored procedure, enter the following SQL statements:

```

SQL> set serveroutput on;
SQL> execute readdocument
Content Length: 93594
start position: 1
doing read: 1
start position: 32768
doing read: 2
start position: 65535
doing read: 3
-----
End of data

PL/SQL procedure successfully completed.

```

### 9.2.1.5 Show the Properties of the Loaded Media Data

The `showprop_doc.sql` script displays the properties of the media data loaded into the `doc_table` table. This script is shown in [Example 9-15](#).

---



---

**Note:** This script must be run from the same user as the scripts in [Example 9-12](#), [Example 9-13](#), and [Example 9-14](#).

---



---

#### **Example 9-15** *showprop\_doc.sql*

```

-- showprop_doc.sql
--

SET SERVEROUTPUT ON;
SET ECHO ON;

--
-- Query doctable for ORDDoc.

DECLARE
  document ORDDoc;

```

```

idnum integer;
ctx RAW(64) := NULL;
BEGIN
FOR I IN 1..2 LOOP
SELECT id, document INTO idnum, document FROM doc_table WHERE id=I;
dbms_output.put_line('document id: ' || idnum);
dbms_output.put_line('document MIME type: ' || document.getMimeType());
dbms_output.put_line('document file format: ' || document.getFormat());
dbms_output.put_line('BLOB Length: ' || TO_CHAR(document.getContentLength()));
dbms_output.put_line('-----');
END loop;
END;
/

```

The results from running the script `showprop_doc.sql` are the following:

```

SQL> @showprop_doc.sql
document id:          1
document MIME type:   audio/xwav
document file format: WAVE
BLOB Length:         93594
-----
document id:          2
document MIME type:   audio/mpeg
document file format: MPGA
BLOB Length:         51537
-----
PL/SQL procedure successfully completed.

```

### 9.2.1.6 Automate the ORDDoc Examples

The `setup_docsample.sql` script runs each of the previous media scripts in the correct order to automate this process. This script is shown in [Example 9-16](#).

---



---

**Note:** This script must be run from a user with CREATE ANY DIRECTORY and CREATE TABLE privileges.

---



---

#### **Example 9-16** *setup\_docsample.sql*

```

-- setup_docsample.sql
--
-- This script automates the demo by invoking each script in
-- the required order.
--
-- Create the mediadir load directory
@create_mediadir.sql
-- Create the media table:
@create_doctable.sql
--Import 2 media clips and set properties:
@import_doc.sql
--Display the properties of the media clips:
@showprop_doc.sql
--create stored procedure to read from ordDoc
@read_doc.sql

```

```
--Execute stored procedure
execute readdocument

--exit;
```

### 9.2.1.7 Clean Up the ORDDoc Examples

The `cleanup_docsample.sql` script removes the sample tables, directories, and procedures created by the previous media scripts from your database. This script is shown in [Example 9–17](#).

---

---

**Note:** This script must be run from a user with DROP ANY DIRECTORY privilege. This script should be run from the user that was used to run the previous media scripts.

---

---

#### **Example 9–17** *cleanup\_docsample.sql*

```
-- cleanup_docsample.sql
--
-- This script removes all tables and directories created by this
-- demonstration. You must have the DROP ANY DIRECTORY privilege
-- to drop the doc load directory. This script should be run under
-- the same user as the demo was run under.

-- drop the doc load directory.
DROP DIRECTORY mediadir;

-- Drop the table created by the demo.
DROP TABLE doc_table PURGE;
```

## 9.3 Image Data Examples

Image data examples using Oracle Multimedia include the following common operations:

- Using a set of scripts for populating an ORDImage object from a file data source (See [Section 9.3.1](#))
- Using a set of scripts for loading an image table from an HTTP data source (See [Section 9.3.2](#))
- Addressing globalization support issues

Reference information on the methods used in these examples is presented in *Oracle Multimedia Reference*.

### 9.3.1 Scripts for Populating an ORDImage Object from a File Data Source

The scripts presented in this section demonstrate how to populate an Oracle Multimedia ORDImage object from an existing file.

[Table 9–3](#) lists each script by name, along with a brief description of the operations it performs. Each script is included and described in further detail in the sections that follow.

**Table 9–3 Image Scripts**

Script Name	Operations Performed
<code>create_mediadir.sql</code> <a href="#">Example 9–18</a>	Creates an image data load directory. (See <a href="#">Section 9.3.1.1</a> )
<code>create_imgtable.sql</code> <a href="#">Example 9–19</a>	Creates and populates the <code>image_table</code> table. (See <a href="#">Section 9.3.1.2</a> )
<code>import_img.sql</code> <a href="#">Example 9–20</a>	Loads the image data. This script imports the image data from a file into the <code>image_table</code> table using the <code>ORDImage import()</code> method. (See <a href="#">Section 9.3.1.3</a> )
<code>read_image.sql</code> <a href="#">Example 9–21</a>	Reads the image data from a BLOB using a stored procedure. (See <a href="#">Section 9.3.1.4</a> )
<code>showprop_img.sql</code> <a href="#">Example 9–22</a>	Displays the properties of the loaded image data stored in the <code>image_table</code> table. (See <a href="#">Section 9.3.1.5</a> )
<code>setup_imgsample.sql</code> <a href="#">Example 9–23</a>	Automates the process by running the previous image scripts in the required order. (See <a href="#">Section 9.3.1.6</a> )
<code>cleanup_imgsample.sql</code> <a href="#">Example 9–24</a>	Cleans up the process by removing the sample tables, directories, and procedures from your database. (See <a href="#">Section 9.3.1.7</a> )

### 9.3.1.1 Create an Image Data Load Directory

The `create_mediadir.sql` script creates the image data load directory. This script is shown in [Example 9–18](#).

To successfully load the image data, you must have a `mediadir` directory created on your system. This directory must contain your sample image media files, `img71.gif` and `img50.gif`, which are installed in the `<ORACLE_HOME>/ord/img/demo` directory. Actually, you can copy any supported image files to the `mediadir` directory to run this script. This directory path and disk drive must be specified in the `CREATE DIRECTORY` statement in the `create_mediadir.sql` file.

---

**Note:** You must have the `CREATE ANY DIRECTORY` privilege to run this script. If you wish to delete previous instances of the image data load directory, you will also need the `DROP ANY DIRECTORY` privilege. You must edit the `create_mediadir.sql` file to replace the directory path in the `CREATE DIRECTORY` statement. If you run the `create_mediadir.sql` script under a different user than you run the other demo scripts, you must also replace the user in the `GRANT READ` statement. See the following script for detailed instructions.

---

#### **Example 9–18 `create_mediadir.sql`**

```
-- create_mediadir.sql
-- You must have the CREATE ANY DIRECTORY privilege to run this script.
--
-- You must edit this script to:
```

```

-- o replace the directory path in the CREATE DIRECTORY statement
--   with your own (see below for details)
-- o uncomment the GRANT READ ON DIRECTORY statement and grant to the
--   user under which you will run this demo

SET SERVEROUTPUT ON;
SET ECHO ON;

-- You need DROP ANY DIRECTORY privilege to delete a directory. If there
-- is no need to delete the directory, then leave the next line
-- commented out.
-- DROP DIRECTORY mediadir;

-- Create the mediadir load directory, the directory where the image
-- clips reside. You must specify the disk drive and path in the
-- CREATE DIRECTORY statement below. Edit this file to replace the
-- directory specification with your own.
CREATE OR REPLACE DIRECTORY mediadir AS 'C:/mediadir';

-- If you intend to run this demo under a user other than the user
-- that just created this directory, edit the following grant to grant
-- READ on the directory just created to the user under which you will
-- run the demo scripts. For example, if you will run the demo scripts
-- under user 'SCOTT', then replace the string "<USER>" with "SCOTT".
-- Then uncomment the following GRANT statement. There is no need to
-- do this if the user creating the directory will also be used to run
-- the other demo scripts.
-- GRANT READ ON DIRECTORY mediadir TO <USER>;

```

### 9.3.1.2 Create and Populate the image\_table Table

The `create_imgtable.sql` script creates and populates the `image_table` table. This table contains an `ORDImage` column. This script creates the `image_table` table with two columns (`id`, `image`), inserts two rows, and initializes the `image` column. This script is shown in [Example 9-19](#).

---



---

**Note:** Calling the `ORDImage.init()` method initializes the `blob` attribute of the `ORDImage` object. This is necessary to populate the `BLOB` with data.

This script must be run from a user with the `CREATE TABLE` privilege.

---



---

#### **Example 9-19** *create\_imgtable.sql*

```

-- create_imgtable.sql
--
-- This script must be run from a user with the CREATE TABLE privilege.
--
-- Create the image_table table.
-- Insert two rows with empty OrdImage columns and initialize object attributes.

SET SERVEROUTPUT ON;
SET ECHO ON;

DROP TABLE image_table PURGE;
CREATE TABLE image_table ( id NUMBER,
                           image ORDImage )
LOB(image.source.localData) STORE AS SECUREFILE;

```



```
-- Insert rows with empty OrdImage columns and initialize the object attributes.

INSERT INTO image_table VALUES(1,ORDImage.init());
INSERT INTO image_table VALUES(2,ORDImage.init());

COMMIT;
```

### 9.3.1.3 Load the Image Data

The `import_img.sql` script imports image data from an image file into the `ORDImage` column in the `image_table` table using the `ORDImage import()` method. To import data into a blob attribute where the image will be stored, the `image` column in the row must be selected for `update`. This script is shown in [Example 9–20](#).

To successfully run this script, you must copy two image files to your `mediadir` directory using the file names specified in this script, or modify this script to match the file names of your image files.

---



---

**Note:** This script should be run from the same user as the script in [Example 9–19](#).

---



---

#### **Example 9–20** `import_img.sql`

```
-- import_img.sql
--
SET SERVEROUTPUT ON
SET ECHO ON

-- Import the two files into the database.

DECLARE
    obj ORDIMAGE;
    ctx RAW(64) := NULL;
BEGIN
    -- This imports the image file img71.gif from the MEDIADIR directory
    -- on a local file system (srcType=file).
    -- the import method also sets the object properties by reading the image blob.
    select Image into obj from image_table where id = 1 for update;
    obj.setSource('file','MEDIADIR','img71.gif');
    obj.import(ctx);
    update image_table set image = obj where id = 1;
commit;

    -- This imports the image file img50.gif from the MEDIADIR directory
    select Image into obj from image_table where id = 2 for update;
    obj.setSource('file','MEDIADIR','img50.gif');
    obj.import(ctx);
    update image_table set image = obj where id = 2;
commit;
END;
/
```

### 9.3.1.4 Read the Image Data from the BLOB

The `read_image.sql` script reads image data from a BLOB by creating the stored procedure `readimage`. This procedure reads a specified amount of image data from

the BLOB attribute, beginning at a particular offset, until all the image data is read. This script is shown in [Example 9-21](#).

---

---

**Note:** This script must be run from the same user as the scripts in [Example 9-19](#) and [Example 9-20](#).

---

---

**Example 9-21** *read\_image.sql*

```
-- read_image.sql
set serveroutput on
set echo on
create or replace procedure readimage as
-- Note: ORDImage has no readFromSource method like ORDAudio
-- and ORDVideo; therefore, you must use the DBMS_LOB package to
-- read image data from a BLOB.
buffer RAW (32767);
src BLOB;
obj ORDImage;
amt BINARY_INTEGER := 32767;
pos integer := 1;
read_cnt integer := 1;
BEGIN
  Select t.image.getcontent() into src from image_table t where t.id = 1;
  Select image into obj from image_table t where t.id = 1;
  DBMS_OUTPUT.PUT_LINE('Content length is: '||          TO_
  CHAR(obj.getContentLength()));
  LOOP
  DBMS_LOB.READ(src,amt,pos,buffer);
  DBMS_OUTPUT.PUT_LINE('start position: '|| pos);
  DBMS_OUTPUT.PUT_LINE('doing read '|| read_cnt);
  pos := pos + amt;
  read_cnt := read_cnt + 1;
  -- Note: Add your own code here to process the image data being read;
  -- this routine just reads data into the buffer 32767 bytes
  -- at a time, then reads the next chunk, overwriting the first
  -- buffer full of data.
  END LOOP;
  EXCEPTION
  WHEN NO_DATA_FOUND THEN
  DBMS_OUTPUT.PUT_LINE('-----');
  DBMS_OUTPUT.PUT_LINE('End of data ');
  END;
  /
show errors
```

To execute the stored procedure, enter the following SQL statements:

```
SQL> set serveroutput on;
SQL> execute read_image;
Content length is: 1124
start position: 1
doing read 1
-----
End of data

PL/SQL procedure successfully completed.
```

### 9.3.1.5 Show the Properties of the Loaded Image Data

The `showprop_img.sql` script displays the properties of the image data loaded into the `image_table` table. This script is shown in [Example 9–22](#).

---

**Note:** This script must be run from the same user as the scripts in [Example 9–19](#), [Example 9–20](#), and [Example 9–21](#).

---

#### Example 9–22 `showprop_img.sql`

```
-- showprop_img.sql
--

SET SERVEROUTPUT ON;
SET ECHO ON;

-- This script displays properties of the image stored in image_table
--

DECLARE
  image ORDImage;
  idnum integer;
  rowcount integer;
BEGIN

  FOR I IN 1..2 LOOP
    SELECT id, image into idnum, image from image_table where id=I;
    dbms_output.put_line('Image properties:');
    dbms_output.put_line('image id: '|| idnum);
    dbms_output.put_line('image height: '|| image.getHeight());
    dbms_output.put_line('image width: '|| image.getWidth());
    dbms_output.put_line('image MIME type: '|| image.getMimeType());
    dbms_output.put_line('image file format: '|| image.getFileFormat());
    dbms_output.put_line('BLOB Length: '|| TO_CHAR(image.getContentLength()));
    dbms_output.put_line('-----');
  END loop;
END;
/
```

The results from running the script `showprop_img.sql` are the following:

```
SQL> @showprop_img.sql
image id:          1

image height:      15
image width:       43
image MIME type:   image/gif
image file format: GIFF
BLOB Length:      1124
-----
image id:          2

image height:      32
image width:       110
image MIME type:   image/gif
image file format: GIFF
BLOB Length:      686
-----
```

PL/SQL procedure successfully completed.

### 9.3.1.6 Automate the ORDImage Examples

The `setup_imgsample.sql` script runs each of the previous image scripts in the correct order to automate this process. This script is shown in [Example 9–23](#).

---

---

**Note:** This script must be run from a user with CREATE ANY DIRECTORY and CREATE TABLE privileges.

---

---

#### **Example 9–23** *setup\_imgsample.sql*

```
-- setup_imgsample.sql
--
-- This script automates the demo by invoking each script in
-- the required order.
--
-- Create the imgdir load directory
@create_mediadir.sql
-- Create image table:
@create_imgtable.sql
--Import images into image_table
@import_img.sql
--Show properties of images
@showprop_img.sql
--create stored procedure to read from ordImage
@read_image.sql
--Execute stored procedure
execute readimage
--exit;
```

### 9.3.1.7 Clean Up the ORDImage Examples

The `cleanup_imgsample.sql` script removes the sample tables, directories, and procedures created by the previous image scripts from your database. This script is shown in [Example 9–24](#).

---

---

**Note:** This script must be run from a user with DROP ANY DIRECTORY privilege. This script should be run from the user that was used to run the previous image scripts.

---

---

#### **Example 9–24** *cleanup\_imgsample.sql*

```
-- cleanup_imgsample.sql
--
-- This script removes all tables, procedures, and directories
-- created by this demonstration. You must have the DROP ANY
-- DIRECTORY privilege to drop the image load directory. This
-- script should be run under the same user as the demo was run
-- under.
```

```

-- drop the image load directory.
DROP DIRECTORY mediadir;

-- Drop the tables created by the demo.
DROP TABLE image_table PURGE;

-- Drop the Update procedure.
DROP PROCEDURE read_image;

commit;
exit;

```

### 9.3.2 Script for Loading an Image Table from an HTTP Data Source

The `import_imghttp.sql` script imports the image data from an HTTP data source. This script inserts two rows into the `image_table` table and loads the image data from the specified HTTP data source (source type `HTTP`, URL location, and HTTP object name). This script is shown in [Example 9-25](#).

To successfully run this script, you must modify it to point to two images located on your Web site.

#### **Example 9-25 `import_imghttp.sql` Script**

```

--import_imghttp.sql
-- Import the two HTTP images from a Web site into the database.
-- Prerequisites:
--   You will need to do the following before running this script
--       1. Run create_imgdir.sql
--       2. Run create_imgtable.sql
--       3. Modify the HTTP URL and object name to point to two images on
--          your own Web site.

SET SERVEROUTPUT ON
SET ECHO ON

-- Import two images from HTTP source URLs.

-- Insert two rows with an empty BLOB.
-- See above section on pre requisites
insert into image_table values (7,ORDImage.init(
'http', 'http://your_website/images', 'image1.jpg'));

insert into image_table values (8,ORDImage.init(
'http', 'http://your_website/images', 'image2.gif'));

commit;

DECLARE
    obj ORDIMAGE;
    ctx RAW(64) := NULL;
BEGIN
-- This imports the image file image1.gif from the HTTP source URL
-- (srcType=HTTP), and automatically sets the properties.
    select Image into obj from image_table where id = 7 for update;
    obj.import(ctx);
    update image_table set image = obj where id = 7;
    commit;
-- This imports the image file image2.gif from the HTTP source URL

```

```
-- (srcType=HTTP), and automatically sets the properties.
select Image into obj from image_table where id = 8 for update;
obj.import(ctx);
update image_table set image = obj where id = 8;
commit;
END;
/
```

### 9.3.3 Addressing Globalization Support Issues

[Example 9-26](#) shows how to use the `processCopy()` method with language settings that use the comma as the decimal point. For example, when the territory is FRANCE, the decimal point is expected to be a comma. Notice that the `" , 75 "` is specified as the scale factor. This example addresses globalization support issues.

#### **Example 9-26 Address a Globalization Support Issue**

```
ALTER SESSION SET NLS_LANGUAGE = FRENCH;
ALTER SESSION SET NLS_TERRITORY = FRANCE;
DECLARE
myimage ORDImage;

BEGIN

SELECT image into myimage from image_table where id=1 for update;

myimage.process('scale=" , 75 " ');
UPDATE image_table SET image = myimage where id=1;
COMMIT;
END;
/
```

Run the `showprop_img.sql` script to see the properties of the scaled image.

## 9.4 Video Data Examples

See *Oracle Multimedia Reference* for video data examples.

---

---

# Oracle Multimedia Sample Applications

Oracle Multimedia includes a number of scripts and sample applications that you can use. These consist of OCI, SQL, Java, PL/SQL, and ASP/VBScript sample applications (demos). These applications are described briefly in the following sections:

- [Oracle Multimedia Image OCI C Sample Application](#)
- [Oracle Multimedia Java Sample Applications](#)
- [Oracle Multimedia PL/SQL Sample Applications](#)
- [Oracle Multimedia ASP/VBScript Sample Application](#)
- [Other Oracle Multimedia Sample Applications](#)

The sample applications are available in the following directories after you install Oracle Multimedia:

On Linux and UNIX

```
<ORACLE_HOME>/ord/img/demo  
<ORACLE_HOME>/ord/im/demo/java  
<ORACLE_HOME>/ord/http/demo/
```

On Windows

```
<ORACLE_HOME>\ord\img\demo  
<ORACLE_HOME>\ord\im\demo\java  
<ORACLE_HOME>\ord\http\demo\
```

## A.1 Oracle Multimedia Image OCI C Sample Application

Once you have installed Oracle Multimedia, you can run the Oracle Multimedia Image OCI C sample application to modify images. Or, you can use this application as a test to confirm successful installation. The sample application files and `README.txt` file are located in the Oracle home directory at:

```
<ORACLE_HOME>/ord/img/demo (on Linux and UNIX)  
<ORACLE_HOME>\ord\img\demo (on Windows)
```

See the `README.txt` file for more information about Oracle Multimedia image features. This section describes how to run the Oracle MultimediaImage OCI C sample application.

### Running the Image OCI C Sample Application

The file `imgdemo.c` is a sample application that shows how Oracle Multimedia image features can be used from within a program. The application is written in C and uses OCI, Oracle Call Interface, to access the database and use Oracle Multimedia image features.

The application operates on `imgdemo.bin`, which is a bitmap (BMP) image in the `demo` directory. Optionally, you can supply an image file name on the command line, provided the file resides in the same directory as the application. In either case, once the image has been manipulated by Oracle Multimedia, the resulting image is written to the file `imgdemo.out` and can then be viewed with common rendering tools that you supply.

When the application is run, it deletes and re-creates a table named `IMGDEMOTAB` in the `SCOTT` schema of the default database. This table is used to hold the application data. Once the table is created, a reference to the image file is inserted into the table. The data is then loaded into the table and converted to JFIF using the `processCopy()` method of `ORDImage`.

The image properties are extracted within the database using the `setProperties()` method. An `UPDATE` statement is issued after the `setProperties()` invocation. This is required to make the object attributes permanent because the `setProperties()` invocation has updated only a local copy of the type attributes.

Upon completion, the application leaves the `imgdemo.out` file in the current directory. It also leaves the table `IMGDEMOTAB` in the `SCOTT` schema of the database.

Next, the `process()` method is used to cut and scale the image within the database. This is followed by an update that commits the change. The application cuts a portion of the image 100 pixels wide by 100 pixels high, starting from pixel location (100,100). This subimage is scaled to twice its original size and the resulting image is written out to the file system in a file named `imgdemo.out`.

Execute the application by typing `imgdemo` on the command line, as follows:

```
$ imgdemo <optional-image-filename>
```

The application displays a number of messages describing its progress, along with any errors encountered in the event that something was not set up correctly. Expect to see the following messages:

```
Dropping table IMGDEMOTAB...
Creating and populating table IMGDEMOTAB...
Loading data into cartridge...
Modifying image characteristics...
Writing image to file imgdemo.out...
Disconnecting from database...
Logged off and detached from server.
Demo completed successfully.
```

If the application encounters any errors, it is likely that either Oracle Multimedia software has not been installed correctly, or the database has not been started. If the application completes successfully, the original image and the resulting image, which has undergone the cutting and scaling described earlier, can be viewed with common image rendering tools.



## A.2 Oracle Multimedia Java Sample Applications

Three Java sample applications are available after installing Oracle Multimedia. These include:

- Oracle Multimedia IMExample sample application

The IMExample application has been provided to help you learn to use the audio, video, image, and media (ORDDoc) client-side Java classes so you can build your own Java applications. In this Java sample application, the sample schema is used to demonstrate the use of the OrdAudio, OrdVideo, OrdImage, and OrdDoc Java objects. The sample application files and README.txt file are located at:

```
<ORACLE_HOME>/ord/im/demo/java (on Linux and UNIX)
```

```
<ORACLE_HOME>\ord\im\demo\java (on Windows)
```

See the README.txt file for requirements and instructions on how to run this sample application. See [Chapter 5](#) for more information about installing and using this application.

- Oracle Multimedia Java Servlet Photo Album sample application

The Java Servlet Photo Album application shows you how to use Oracle Multimedia Servlets and JSP Java API to upload and retrieve multimedia data. The sample application files and README.txt file are located at:

```
<ORACLE_HOME>/ord/http/demo/servlet (on Linux and UNIX)
```

```
<ORACLE_HOME>\ord\http\demo\servlet (on Windows)
```

See the README.txt file for requirements and instructions on how to run this sample application. See [Section 3.2](#) for more information about installing and using this application.

- Oracle Multimedia JavaServer Pages (JSP) Photo Album sample application

The JSP Photo Album application shows you how to use Oracle Multimedia Servlets and JSP Java API to upload and retrieve multimedia data. The sample application files and README.txt file are located at:

```
<ORACLE_HOME>/ord/http/demo/jsp (on Linux and UNIX)
```

```
<ORACLE_HOME>\ord\http\demo\jsp (on Windows)
```

See the README.txt file for requirements and instructions on how to run this sample application. See [Section 3.3](#) for more information about installing and using this application.

See *Oracle Application Server 10g Multimedia Tag Library for JSP User's Guide and Reference* in the Oracle Application Server Online Documentation Library for information about using Multimedia Tag Library.

## A.3 Oracle Multimedia PL/SQL Sample Applications

Two PL/SQL sample applications are available after installing Oracle Multimedia. These include:

- Oracle Multimedia PL/SQL Web Toolkit Photo Album sample application

The PL/SQL Web Toolkit Photo Album application shows how to upload and retrieve image data using the PL/SQL Web Toolkit and PL/SQL Gateway. The SQL scripts and README.txt file are located at:

```
<ORACLE_HOME>/ord/http/demo/plsqlwtk (on Linux and UNIX)
```

<ORACLE\_HOME>\ord\http\demo\plsqlwtk (on Windows)

See the README.txt file for requirements and instructions on how to run this sample application. See [Section 3.1](#) for more information about installing and using this application.

- Oracle Multimedia Code Wizard sample application for the PL/SQL Gateway

The Code Wizard application for the PL/SQL Gateway is an example of a tool that lets you create PL/SQL procedures for the PL/SQL Gateway to upload and retrieve media data stored in the database using any of the Oracle Multimedia object types. The SQL scripts and README.txt file are located at:

<ORACLE\_HOME>/ord/http/demo/plsgwycw (on Linux and UNIX)

<ORACLE\_HOME>\ord\http\demo\plsgwycw (on Windows)

See the README.txt file for requirements and instructions on how to run this sample application. See [Chapter 4](#) for more information about installing and using this application.

## A.4 Oracle Multimedia ASP/VBScript Sample Application

Oracle Multimedia also provides the Oracle Multimedia ASP/VBScript Photo Album sample application, which shows how to upload and retrieve multimedia data using Microsoft Active Server Pages (ASP) and Visual Basic Scripting (VBScript). The sample application files and README.txt file are located at:

<ORACLE\_HOME>/ord/http/demo/asp (on Linux and UNIX)

<ORACLE\_HOME>\ord\http\demo\asp (on Windows)

See the README.txt file for requirements and instructions on how to run this sample application. See [Section 3.4](#) for more information about installing and using this application.

## A.5 Other Oracle Multimedia Sample Applications

Additional Oracle Multimedia sample applications are available from the Oracle Multimedia Web page on the Oracle Technology Network at

[http://www.oracle.com/technology/sample\\_code/products/intermedia/](http://www.oracle.com/technology/sample_code/products/intermedia/)

---

---

## Managing Oracle Multimedia Installations

This appendix describes the manual installation of Oracle Multimedia as well as the manual upgrading of an installed version of Oracle Multimedia. In addition, this appendix provides information about how to verify and downgrade an installed version of Oracle Multimedia.

These topics are presented in the following sections:

- [Installing Oracle Multimedia](#)
- [Upgrading an Installed Version of Oracle Multimedia](#)
- [Verifying an Installed Version of Oracle Multimedia](#)
- [Downgrading an Installed Version of Oracle Multimedia](#)
- [Removing Oracle Multimedia](#)

---

---

**Note:** See the Oracle Multimedia `README.txt` file located in `<ORACLE_HOME>/ord/im/admin` for the latest information.

---

---

### B.1 Installing Oracle Multimedia

Oracle Multimedia is installed and configured with Oracle Database. If, for some reason, you need to install Oracle Multimedia manually, you can follow the instructions in this section, according to the following topics:

- Installation decisions
- Preinstallation steps
- Installation steps

---

---

**WARNING:** Oracle-supplied Oracle Multimedia data types are installed under the database users ORDSYS, ORDPLUGINS, and SI\_INFORMTN\_SCHEMA. These users are created during database installation, and they are subject to change in future releases.

The functions, packages, types, and views supplied by Oracle Multimedia are reserved by Oracle, and they are subject to change in future releases. The ORDSYS and SI\_INFORMTN\_SCHEMA users and their contents must not be modified by the customer. Do not connect to these users or add your own tables, functions, packages, types, or views to these users. In addition, the packages supplied by Oracle Multimedia in the ORDPLUGINS user must not be modified by the customer. Doing so could cause internal errors and security violations in the database management system.

---

---

### B.1.1 Installation Decisions

The installation procedure creates the ORDSYS, ORDPLUGINS, SI\_INFORMTN\_SCHEMA, and MDSYS users. These user IDs are the standard Oracle Database account with special privileges.

**Decision:** Decide which tablespace to use for Oracle Multimedia users (ORDSYS, ORDPLUGINS, and SI\_INFORMTN\_SCHEMA), and which tablespace to use for the Spatial/Oracle Multimedia Location Services user (MDSYS). Oracle suggests you use the SYSAUX tablespace for both.

**Decision:** Decide on passwords for the ORDSYS, ORDPLUGINS, SI\_INFORMTN\_SCHEMA, and MDSYS users. The installation uses default passwords for ORDSYS, ORDPLUGINS, SI\_INFORMTN\_SCHEMA, and MDSYS. Then, it locks the accounts and expires the passwords. You must change these passwords and unlock the accounts after the installation completes if you want to log in to these accounts directly.

The default password for the ORDSYS user during automatic installation is ORDSYS, for ORDPLUGINS is ORDPLUGINS, for SI\_INFORMTN\_SCHEMA is SI\_INFORMTN\_SCHEMA, and for MDSYS is MDSYS.

The installation process grants the EXECUTE privilege to the user group PUBLIC for the Oracle Multimedia packages and objects installed in the ORDSYS, ORDPLUGINS, and SI\_INFORMTN\_SCHEMA schemas.

### B.1.2 Preinstallation Tasks

Perform the following preinstallation tasks before installing and configuring Oracle Multimedia manually. For instructions, see *Oracle Database Installation Guide* for your operating system:

1. Install Oracle Database, including PL/SQL, Oracle JVM, and Oracle XML DB.
2. Create the database.
3. Start the database.
4. Verify that Oracle JVM is installed and is valid.

You can verify that Oracle JVM is correctly installed by running SQL\*Plus, connecting as SYSDBA, and issuing the following query:

```
SQL> select version, status from dba_registry where comp_id='JAVAVM';
```

Ensure that the version is correct and the status is `VALID`.

### B.1.3 Installation Steps

Perform the following mandatory installation steps to configure Oracle Multimedia manually. These steps are *not* required if you use the Database Configuration Assistant.

References to `<ORACLE_HOME>` in these instructions represent the Oracle home directory.

1. Use Oracle Universal Installer to install the files that make up Oracle Multimedia on your system.
2. Create the users and grant the appropriate privileges.

Start SQL\*Plus.

```
% sqlplus
```

Connect as SYSDBA.

```
SQL> CONNECT / as sysdba
```

Invoke `ordinst.sql` with two parameters for `interMedia` tablespace and `Location Services` tablespace.

```
SQL> @<ORACLE_HOME>/ord/admin/ordinst.sql SYSAUX SYSAUX (on Linux and UNIX)
      @<ORACLE_HOME>\ord\admin\ordinst.sql SYSAUX SYSAUX (on Windows)
```

3. Install Oracle Multimedia types and packages.

```
SQL> @<ORACLE_HOME>/ord/im/admin/catim.sql (on Linux and UNIX)
      @<ORACLE_HOME>\ord\im\admin\catim.sql (on Windows)
```

4. Start the listener.

The listener must be configured to use external procedure calls. Check the `tnsnames.ora` file for an entry called `extproc_connection_data`. Then check the `listener.ora` file for an entry called `extproc`.

See your network documentation for details. If this step is not done correctly, Oracle Multimedia will *not* work for all supported formats.

Once these mandatory installation steps have been completed, Oracle Multimedia is ready for use.

## B.2 Upgrading an Installed Version of Oracle Multimedia

If you upgrade a database from an earlier release of Oracle Database, Oracle Multimedia will be upgraded automatically if detected in the source database. See *Oracle Database Upgrade Guide* for detailed instructions.

## B.3 Verifying an Installed Version of Oracle Multimedia

After installing or upgrading Oracle Multimedia, you can verify the Oracle Multimedia installation by invoking the Oracle Multimedia verification procedure.

To run the Oracle Multimedia verification procedure, connect as SYSDBA and invoke `validate_ordim` as follows:

1. Start SQL\*Plus.

```
% sqlplus
```

2. Connect as SYSDBA.

```
SQL> CONNECT / as sysdba
```

3. Execute `validate_ordim`.

```
SQL> execute validate_ordim;
```

On Windows

```
SQL> @<ORACLE_HOME>\ord\im\admin\imchk.sql
```

The validation procedure will silently set the Oracle Multimedia registry entry to `VALID` if Oracle Multimedia is valid. If invalid objects are detected, the `validate` procedure will list the first few invalid objects and set the registry entry to `INVALID`.

To query the registry entry for Oracle Multimedia, run SQL\*Plus and connect as SYSDBA again (follow steps 1 and 2 in the verification procedure). Then, enter the following query:

```
SQL> select version, status from dba_registry where comp_id='ORDIM';
```

Ensure that the version is correct and the status is `VALID`.

## B.4 Downgrading an Installed Version of Oracle Multimedia

Oracle Multimedia is automatically downgraded when you downgrade a database with the Oracle Multimedia feature installed. See *Oracle Database Upgrade Guide* for detailed instructions.

## B.5 Removing Oracle Multimedia

Oracle Multimedia ships an installation removal script (`imremov.sql`) that you can use to manually remove Oracle Multimedia. First, the removal script invokes a removal checking script (`imremchk.sql`), which checks for tables that contain columns of one of the Oracle Multimedia object types, or columns that are bound to one of the Oracle Multimedia XML schemas. Next, the removal checking script displays a message that indicates whether or not Oracle Multimedia is in use.

---

---

**Note:** The `imremchk.sql` script does not check whether dependent products, such as Oracle Spatial, are installed.

---

---

Then, the installation removal script (`imremov.sql`) prompts to ask if you really want to remove Oracle Multimedia. If the first character of your response is `Y` or `y`, Oracle Multimedia will be removed with `force`, regardless of whether or not it is in use.

Follow these steps to manually remove Oracle Multimedia:

1. Start SQL\*Plus, and connect as SYSDBA (see Steps 1 and 2 in [Section B.3](#)).
2. Call the `imremov.sql` script.

```
SQL> @<ORACLE_HOME>/ord/im/admin/imremov.sql (on Linux and UNIX)
@<ORACLE_HOME>\ord\im\admin\imremov.sql (on Windows)
```

---

---

# Glossary

## **codecs**

Digital compression and decompression schemes.

## **content metadata**

Data that describes the content of image media, such as the name of the photographer, and the date and time when a photograph was taken.

## **embedded metadata**

Metadata that is stored with image data in the image file format.

## **image interchange format**

A well-defined organization and use of image attributes, data, and often compression schemes that allows different applications to create, exchange, and use images. Interchange formats are often stored as disk files.

## **image metadata format**

Standard protocols and techniques used to store image metadata within an image file. Formats include: EXIF, IPTC-IIM, and XMP.

## **Java servlets**

Java classes that dynamically process HTTP requests and construct HTTP responses.

## **JavaServer Pages (JSP)**

See JSP.

## **JSP**

JavaServer Pages, Java text-based documents that execute as Java servlets, but which allow a more natural approach to creating static content than using servlets.

## **lossless compression schemes**

Compression schemes that squeeze an image so that when it is decompressed, the resulting image is bit-for-bit identical with the original.

## **lossy compression schemes**

Compression schemes that do not result in an identical image when decompressed, but rather, one in which the changes may be imperceptible to the human eye. Lossy schemes generally provide higher compression than lossless compression schemes.

## **media data**

Data from audio, image, video, or heterogeneous media.

---

**metadata**

Information about media data, such as object length, compression type, or format.

**methods**

Procedures that can be performed on objects, such as `getContent()` or `setProperties()`.

**Oracle *interMedia***

In Oracle Database 11g Release 1 (11.1), the name Oracle *interMedia* has been changed to Oracle Multimedia.

**ORDAudio**

Object relational type for audio data characteristics.

**ORDDoc**

Object relational type for heterogeneous data characteristics.

**ORDImage**

Object relational type for image data characteristics.

**ORDSource**

Object relational type that stores data source information for audio, heterogeneous, image, and video data characteristics.

**ORDVideo**

Object relational type for video data characteristics.

**portlets**

Summarized versions of applications and Web content situated in defined regions of a Web page that are accessible through enterprise portals.

**protocols**

Image interchange formats exchanged in a sequential fashion over a network.

**technical metadata**

Data that describes image media in a technical sense, such as the height and width of an image, in pixels, or the type of compression used to store the image.



---

---

# Index

## A

---

- application development, 2-1
  - class libraries, 2-1
  - Oracle development tools, 2-2
  - PL/SQL Gateway feature, 2-2
- ASP/VBScript Photo Album sample application, 3-42, A-4
- ASP/VBScript sample applications
  - ASP/VBScript Photo Album application, A-4
- audio data examples, 9-1

## C

---

- class libraries, 2-1
- Code Wizard for the PL/SQL Gateway sample application, A-4
- Code Wizard sample application, 4-1
- codecs (compression and decompression schemes), 1-3
- compression, 1-3
- compression formats
  - audio, 1-3
  - image, 1-3
  - video, 1-8
- compression schemes, 1-3, 1-6
- content metadata, 6-1

## D

---

- data
  - loading multimedia, 1-10
- data formats, 1-5
- DBA tuning tips, 8-1
- decompression schemes, 1-3
- DICM
  - DICOM image file format, 6-7
- DICOM
  - defined, 6-7
  - extracting metadata, 1-6
  - information about, 6-9
  - medical images, 6-7
- DICOM examples
  - creating a table, 6-8
  - extracting DICOM metadata, 6-8
- digital camera images, 6-2

- downgrading an installed version of Oracle Multimedia, B-4

## E

---

- embedded metadata, 6-1
- embedding metadata, 1-6
- exception handling
  - Java, 2-17
  - PL/SQL, 2-7
- EXIF standard, 6-2
- extending Oracle Multimedia
  - audio default format, 7-8
  - document default format, 7-11
  - new audio format, 7-9, 7-11
  - new data source, 7-5
  - new document format, 7-11
  - new image object type, 7-15
  - new video format, 7-13
  - video default format, 7-12

## G

---

- globalization support
  - image data examples, 9-26

## H

---

- handling exceptions
  - Java, 2-17
  - PL/SQL, 2-7
- HTML output, 2-8

## I

---

- image data examples, 9-18
  - globalization support issues, 9-26
- image file formats
  - DICM, 6-7
- image file storage standards
  - EXIF, 6-2
  - IPTC-IIM, 6-2
  - XMP, 6-2
- image metadata format
  - defined, 6-2
- image OCI C sample application, A-1

IMExample sample application, 5-1, A-3  
installing Oracle Multimedia, B-1  
interchange format, 1-6  
interchanging metadata, 6-2  
IPTC-IIM standard, 6-2

## J

---

Java database connectivity (JDBC), 2-12  
Java sample applications  
    IMExample application, A-3  
    Java Servlet Photo Album application, A-3  
    JSP Photo Album application, A-3  
Java Servlet Photo Album sample application, 3-25, A-3  
JSP Photo Album sample application, 3-34, A-3

## L

---

loading data  
    multimedia, 1-10  
    using PL/SQL, 1-10  
    using SQL\*Loader, 1-10  
loading media data  
    example, 2-15  
lossless compression, 1-6  
lossy compression, 1-6

## M

---

media data examples, 9-12  
media delivery components  
    Java servlet example, 2-20  
    JavaServer Pages example, 2-20  
media queries  
    PL/SQL, 2-5  
medical imaging, 1-6  
medical imaging standards  
    DICOM, 6-7  
metadata, 6-1  
    embedding, 6-2  
        extracting metadata, 1-6  
        embedding in XML, 3-2  
        embedding metadata, 6-5  
        extracting, 1-6, 3-2, 6-2  
        extracting DICOM metadata, 6-8  
        extracting metadata, 6-4  
        information about, 6-7  
        searching, 3-2  
        storing, 3-2  
        XML DB, 6-3, 6-5  
        XML documents, 6-3  
metadata examples  
    creating a table, 6-3  
    embedding metadata, 6-5  
    extracting metadata, 6-4

## N

---

news media images, 6-2

## O

---

object relational technology, 1-1  
Oracle development tools, 2-2  
Oracle *interMedia* *See* Oracle Multimedia  
Oracle Multimedia, 1-1  
    media data storage model, 1-2  
    objects types, 1-2  
Oracle Multimedia Java Classes  
    Java client applications, 2-12  
Oracle Multimedia Java Classes for Servlets and JSP  
    Java-based Web applications, 2-19  
Oracle Technology Network  
    sample applications, A-4  
ORDPLUGINS.ORDX\_DEFAULT\_AUDIO  
    package, 7-8  
ORDPLUGINS.ORDX\_DEFAULT\_DOC  
    package, 7-11  
ORDPLUGINS.ORDX\_DEFAULT\_VIDEO  
    package, 7-12  
ORDPLUGINS.ORDX\_FILE\_SOURCE package, 7-2  
ORDPLUGINS.ORDX\_HTTP\_SOURCE package, 7-3

## P

---

packages  
    ORDPLUGINS.ORDX\_DEFAULT\_AUDIO, 7-8  
    ORDPLUGINS.ORDX\_DEFAULT\_DOC, 7-11  
    ORDPLUGINS.ORDX\_DEFAULT\_VIDEO, 7-12  
    ORDPLUGINS.ORDX\_FILE\_SOURCE, 7-2  
    ORDPLUGINS.ORDX\_HTTP\_SOURCE, 7-3  
packages or PL/SQL plug-ins, 7-1  
PL/SQL  
    client applications, 2-4  
    configuring your environment, 2-4  
    HTML tags, 2-8  
    loading data, 1-10  
    media queries, 2-5  
    retrieving media, 2-5, 2-9  
    uploading media, 2-6, 2-8  
    Web applications, 2-8  
PL/SQL Gateway feature, 2-2  
PL/SQL packages, 2-8  
PL/SQL sample applications  
    Code Wizard for the PL/SQL Gateway  
        application, A-4  
    PL/SQL Web Toolkit Photo Album  
        application, A-3  
PL/SQL Web Toolkit Photo Album sample  
    application, 3-2, A-3  
protocol, 1-6

## R

---

related documents, xii  
removing Oracle Multimedia, B-4  
retrieving media  
    Oracle Multimedia Java Classes, 2-14  
    Oracle Multimedia Java Classes for Servlets and  
        JSP, 2-19  
    PL/SQL, 2-5, 2-9

## **S**

---

sample applications, A-1  
  ASP/VBScript Photo Album, 3-42  
  Code Wizard, 4-1  
  Image OCI C, A-1  
  IMExample, 5-1  
  Java Servlet Photo Album, 3-25  
  JSP Photo Album, 3-34  
  located on Oracle Technology Network, A-4  
  location, A-1  
  PL/SQL Web Toolkit Photo Album, 3-2  
SQL\*Loader  
  loading data, 1-10

## **T**

---

technical metadata, 6-1

## **U**

---

upgrading an Oracle Multimedia installation, B-3  
uploading media  
  Oracle Multimedia Java Classes, 2-15  
  Oracle Multimedia Java Classes for Servlets and  
    JSP, 2-21  
  PL/SQL, 2-6, 2-8

## **V**

---

verifying an Oracle Multimedia installation, B-3  
video data examples, 9-26

## **X**

---

XML  
  representing metadata, 6-3  
XMP standard, 6-2

