Oracle® *inter*Media

User's Guide

10*g* Release 1 (10.1)

Part No. B10840-01

December 2003

Oracle *inter*Media 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 *inter*Media extends Oracle Database reliability, availability, and data management to multimedia content in traditional, Internet, electronic commerce, and media-rich applications.

ORACLE®

Oracle *inter*Media User's Guide, 10*g* Release 1 (10.1)

Part No. B10840-01

Primary Author: Rod Ward

Contributors: Susan Mavris, Simon Oxbury, Robert Abbott, Guozhong Wang, Dongbai Guo, Fengting Chen, Dong Lin, Melliyal Annamalai, Manjari Yalavarthy, Rajiv Chopra, Joseph Mauro, Joseph Meeks, Rabah Mediouni, Bill Voss, Susan Kotsovolos, Rosanne Toohe, Bill Beauregard, Susan Shepard, Deborah Owens

Contents

List of Examples

List of Figures

List of Tables

Send Us Your Comments

Oracle *inter*Media User's Guide, 10*g* Release 1 (10.1)

Part No. B10840-01

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

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

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

- Electronic mail: nedc-doc_us@oracle.com
- FAX: 603.897.3825 Attn: Oracle *inter*Media Documentation
- Postal service:
  Oracle Corporation
  Oracle *inter*Media Documentation
  One Oracle Drive
  Nashua, NH 03062
  USA

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

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

Preface

This guide describes how to use Oracle *inter*Media ("*inter*Media").

Oracle *inter*Media ships with Oracle Database Standard and Enterprise Editions.

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

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.

If you are interested in an overview and how to use Oracle *inter*Media, see Chapter 1 for general introductory information.

For tuning tips for storing media files, see Chapter 8.

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. Standards will continue to evolve over time, and Oracle Corporation 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 additional information,

visit the Oracle Accessibility Program Web site at
`http://www.oracle.com/accessibility/`.

Accessibility of Code Examples in Documentation JAWS, a Windows screen reader, 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, JAWS 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 Corporation does not own or control. Oracle Corporation neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Organization

This guide contains the following chapters and appendixes:

Chapter 1, "Introduction to Oracle interMedia"

Introduces multimedia and Oracle *inter*Media; explains multimedia-related concepts.

Chapter 2, "Application Development"

Describes Web application development environments and using *inter*Media in these environments.

Chapter 3, "Developing Media Upload and Retrieval Applications"

Describes developing an *inter*Media photo album Web application.

Chapter 4, "IMExample Java Sample Application"

Describes the IMExample Java sample application for loading, retrieving, and playing media data using the *inter*Media image, audio, video, and general media object types.

Chapter 5, "Content-Based Retrieval Concepts"

Explains concepts, operations, and techniques related to content-based retrieval using the *inter*Media image signature object type.

Chapter 6, "Custom DataSource and DataSink for JMF Versions 2.0 and 2.1"

Describes how to install, register, and use the *inter*Media Custom DataSource and DataSink feature, which is an extension to Java Media Framework (JMF) version 2.0 and 2.1 developed by Sun Microsystems.

Chapter 7, "Extending Oracle interMedia"

Describes how to extend *inter*Media to support other external sources of media data, other media data formats, and audio and video data processing.

Chapter 8, "Tuning Tips for the DBA"

Provides tuning tips for the DBA for more efficient storage of multimedia data.

Chapter 9, "interMedia Examples"

Provides basic examples of using *inter*Media object types and methods.

Appendix A, "Sample Programs"

Describes the sample scripts and sample programs and how to run them.

Appendix B, "Installing and Upgrading Oracle interMedia"

Describes how to install and upgrade Oracle *inter*Media manually.

Related Documentation

> **Note:** For information added after the release of this guide, refer to the online `README.txt` file in your *ORACLE_HOME* directory. Depending on your operating system, this file may be in:
>
> *ORACLE_HOME*`/ord/im/admin/README.txt`
>
> Please see your operating system-specific installation guide for more information.

For more information about using *inter*Media in a development environment, see the following documents in the Oracle Database documentation set for 10*g* Release 1 (10.1):

- *Oracle Call Interface Programmer's Guide*
- *Oracle Database Application Developer's Guide - Fundamentals*

- *Oracle Database Application Developer's Guide - Large Objects*
- *Oracle Database Concepts*
- *PL/SQL User's Guide and Reference*
- *Oracle interMedia Java Classes Reference*
- *Oracle interMedia Reference*
- *Oracle interMedia Annotator User's Guide and Reference* (available only from OTN)

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

For reference information on both Oracle *inter*Media Java Classes and Oracle *inter*Media Java Classes for Servlets and JSP in Javadoc format, see the Oracle API documentation (also known as Javadoc). The API documentation is available on the Oracle Database 10*g* Documentation Library CD-ROM and also from the documentation section of the Oracle Technology Network (OTN) Web site at

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

For more information on Java, see the API documentation provided by Sun Microsystems at

`http://java.sun.com/docs`

For more information on the Java Advanced Imaging (JAI) API, see the following Web site (which is maintained by Sun Microsystems)

`http://java.sun.com/products/java-media/jai/index.html`

Printed documentation is available for sale in the Oracle Store at

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

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

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

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

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

Conventions

In this guide, Oracle *inter*Media is sometimes referred to as *inter*Media.

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.

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 conventions are also used in this guide:

Convention	Meaning
. . .	Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.
. . .	Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted.
boldface text	Boldface text indicates a term defined in the text.
italic text	Italic text is used for emphasis, book titles, and variable names.
< >	Angle brackets enclose user-supplied names.
[ ]	Brackets enclose optional clauses from which you can choose one or none.

Changes to This Guide

This is a new guide for this release. It contains several chapters that were originally in *Oracle interMedia User's Guide and Reference* in the Oracle9*i* (9.0.1) release, as well as a chapter originally in *Oracle interMedia Java Classes User''s Guide and Reference* in the Oracle9*i* (9.2) release.

1 Introduction to Oracle *inter*Media

Oracle *inter*Media ("*inter*Media") 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 *inter*Media extends Oracle Database reliability, availability, and data management to multimedia content in traditional, Internet, electronic commerce, and media-rich applications. Oracle *inter*Media does not control media capture or output devices; this function is left to application software.

*inter*Media manages multimedia content by providing the following:

- Media and application metadata management (see Section 1.4, Section 1.5, Section 1.6, Section 1.7, and Section 1.8.4)
- Storage and retrieval (see Section 1.8.1, Section 1.9 and Section 1.10)
- Support for popular formats (see the audio, image, and video data format appendixes in *Oracle interMedia Reference*)
- Access through traditional and Web interfaces (see Section 1.8.3 and Section 1.10)
- Querying using associated relational data
- Querying using extracted metadata
- Querying using media content with optional specialized indexing

*inter*Media provides media content services to Oracle JDeveloper 10*g*, 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 *inter*Media, 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 on using BLOBs and BFILEs:

- *Oracle Database Application Developer's Guide - Large Objects*
- *Oracle Database Concepts* -- see the chapter on Object Views.

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

1.2 SQL/MM Still Image Standard Support

*inter*Media 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

In addition, the following image-matching features are not specified by the SQL/MM Still Image Standard, and therefore are not available for StillImage objects:

- Image matching based on shape
- Indexing (averagecolor, texture, positionalcolor, and colorhistogram)

Finally, the SI_Score methods do not provide the same performance as the *inter*Media ORDImageSignature methods for image matching.

See *Oracle interMedia Reference* for more information on the SQL/MM Still Image Standard object types. The remainder of this chapter applies to the ORDAudio, ORDVideo, ORDDoc, ORDImage, and ORDSource object types.

1.3 Multimedia Content Management

The capabilities of *inter*Media include the storage, retrieval, management, and manipulation of multimedia data managed by Oracle Database. *inter*Media supports multimedia storage, retrieval, and management of:

- Binary large objects (BLOBs) stored locally in the database and containing audio, image, or video data, or other heterogeneous media data
- File-based large objects, or BFILEs, stored locally in operating system-specific file systems and containing audio, image, or video data, or other heterogeneous media data
- URLs containing audio, image, or video data or other heterogeneous media data, stored on any HTTP server such as Oracle Application Server or Oracle Database, Netscape Application Server, Microsoft Internet Information Server (IIS), Apache HTTPD server, and Spyglass servers
- Streaming audio or video data stored on specialized media

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

*inter*Media 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 *inter*Media through modern class library interfaces, or PL/SQL and Oracle Call Interface (OCI).

*inter*Media supports storage of the popular file formats, including desktop publishing image, and streaming audio and video formats in databases. *inter*Media 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. *inter*Media developers can use the basic functions provided here to build specialized multimedia applications.

*inter*Media 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 the object, such as getContent( ) and setProperties( ).

The *inter*Media 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 1.9 describes several ways of loading multimedia data into a database.

Media metadata is stored in the database under *inter*Media control. Whether media data is stored within or outside the database, *inter*Media 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 and video metadata, or other heterogeneous media metadata in XML
- 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

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.

*inter*Media 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 on extending *inter*Media.

*inter*Media 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 *inter*Media are:

- Internet music stores that provide music samplings of CD quality
- Digital sound repositories
- Dictation and telephone conversation repositories
- Audio archives and collections (for example, for musicians)
- Digital art galleries
- Real estate marketing
- Document imaging
- Photograph collections (for example, for professional photographers)

- Internet video stores and digital video-clip previews
- Digital video sources for streaming video delivery systems
- Digital video libraries, archives, and repositories
- Libraries of digital video training programs
- Digital video repositories (for example, for motion picture production, television broadcasting, documentaries, advertisements, and so forth)

1.4 Audio Concepts

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

1.4.1 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.

1.4.2 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 interMedia 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.

1.5 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.

1.5.1 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.

1.5.2 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. *inter*Media 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 interMedia Reference* for a list of supported data formats from which *inter*Media can extract and store attributes. ORDDoc is extensible and can be made to recognize and support other heterogeneous media data formats.

1.6 Image Concepts

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

1.6.1 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 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.

1.6.2 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 forth, 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. ORDImage can process and automatically extract properties of images of a variety of popular data formats. See *Oracle interMedia Reference* for a list of supported data formats for which ORDImage can process and extract metadata. In addition, certain foreign images (formats not natively supported by ORDImage) have limited support for image processing. See *Oracle interMedia 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.

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 a **protocol**. There are many application subdomains within the digitized imaging world and many applications that create or utilize digitized images within these. ORDImage supports storage and retrieval of all image data formats, and processing and attribute extraction of many image data formats (see *Oracle interMedia Reference*).

1.7 Video Concepts

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

1.7.1 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.

1.7.2 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 identical to what is provided by Oracle *inter*Media Annotator.

  See *Oracle interMedia Reference* for a list of supported data formats from which *inter*Media 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.

1.8 Multimedia Storage

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

The *inter*Media relational interface lets developers use static methods of *inter*Media object types with existing and new media stored in BLOBs and BFILEs to move media data between the local file system and the database, to parse and extract the properties of the media data, and to store these properties in 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 *inter*Media object types to take advantage of this relational interface. *inter*Media static methods can also be used to perform image processing operations such as cut, scale, compress, and convert format. See *Oracle interMedia Reference* for more information.

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

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

1.8.1 Storing Multimedia Data

*inter*Media 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 4K bytes in size 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 4 gigabytes) 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 Application Developer's Guide - Large Objects* and *Oracle Call Interface Programmer's Guide* for detailed information on using BLOBs and BFILEs.

*inter*Media 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

1.8.2 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.

1.8.3 Accessing and Manipulating 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. See *Oracle interMedia Java Classes Reference* for more information about using Java.

The object syntax for accessing attributes within a complex object is the dot notation (except in Java):

```
variable.data_attribute
```

The syntax for invoking methods of a complex object is also the dot notation (except in Java):

```
variable.function(parameter1, parameter2, ...)
```

A complete set of media attribute accessors (get methods) and setters (set methods) are provided for accessing attributes for each media type.

See *Oracle Database Concepts* for information on this and other SQL syntax.

1.8.4 Metadata Extraction

*inter*Media provides the ability to extract content and format metadata from media sources (image, 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 interMedia Reference* for more information.

Oracle *inter*Media Annotator lets you write an application that reads media data, extracts metadata, associates other metadata, and then uploads the media data and all associated metadata into the database. It lets you customize annotations to further describe the data and loads the annotation and the media data into a database. Use Oracle *inter*Media Annotator if you need to extend the formats or metadata supported, or if you want to associate other metadata. See *Oracle interMedia Annotator User's Guide and Reference* for more information.

1.8.5 Image Processing

*inter*Media 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, *inter*Media supports a variety of operators for changing the format characteristics. See *Oracle interMedia Reference* for more information.

1.8.6 Content-Based Retrieval of Images

Content-based retrieval of images with extensible indexing is supported for image matching. An overview of the benefits of content-based retrieval is described in Chapter 5 along with how content-based retrieval works, including definitions and explanation of the visual attributes (color, texture, shape, and location) and why you might emphasize specific attributes in certain situations. In addition, the use of indexing to improve search and retrieval performances is described in Section 5.4.

Note: All *inter*Media features are available with the Standard Edition of Oracle Database, except image indexing, which uses the ORDImageSignature object. The image indexing feature requires bit-mapped indexing, which is available only when you install the Enterprise Edition of Oracle Database.

1.8.7 *inter*Media Speech Mining and Speech Indexing

*inter*Media speech mining and speech indexing adds speech recognition and indexing capabilities to the database. With this feature, third-party speech recognition vendors can easily and tightly integrate their speech recognition technology with a database. Customers can then build their own complete multimedia storage, management, mining, and indexing solutions on top of Oracle and third-party speech recognition software.

Speech-to-text conversion capabilities of third-party speech recognition software integrated with a database enable speech data mining. The Oracle speech index introduces a new index type that lets users build indexes on the results of speech processing so they can use text-based queries to retrieve data.

This technology is designed to facilitate the mining, indexing, and searching of recorded speech data. It is not designed to be used for real-time, speech-enabled user interfaces, or for voice transcriptions, such as a call center application.

This feature can be downloaded from the Oracle *inter*Media Software section of the Oracle Technology Network Web site

`http://otn.oracle.com/products/intermedia`

1.9 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 *inter*Media object type columns.

- PL/SQL

  A procedural extension to SQL, PL/SQL is an advanced fourth-generation programming language (4GL) of Oracle Corporation. You can write PL/SQL procedures to load multimedia data from BLOB, file system, and URL media data sources into *inter*Media 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 Section 8.3 for a description of a sample control file. See also *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 Section 8.3 for a description of a sample PL/SQL multimedia data load script. See also *PL/SQL User's Guide and Reference* for more information.

Loading Multimedia Data Using Oracle *inter*Media Annotator

You can use Oracle *inter*Media Annotator to upload media data and an associated annotation into a database. Oracle *inter*Media Annotator does this using a PL/SQL Upload Template, which contains both PL/SQL calls and Annotator-specific keywords. Oracle *inter*Media Annotator extracts content and format attributes from media sources (image, audio, and video files), and organizes the attributes into an XML formatted annotation.

Advanced users with PL/SQL experience can create their own PL/SQL Upload Templates in a text editor.

See *Oracle interMedia Annotator User's Guide and Reference* for more information.

1.10 Accessing Multimedia Data

Section 1.10.1 through Section 1.10.4 describe ways in which applications, Oracle development tools, and third-party development tools can access multimedia data stored in the database using *inter*Media object types.

1.10.1 Oracle *inter*Media Java Classes

Oracle *inter*Media Java Classes enables 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 *inter*Media Java Classes makes it possible for Java database connectivity (JDBC) result sets to include both traditional relational data and *inter*Media media objects. This support enables applications to easily select and operate on a result set that contains sets of *inter*Media columns plus other relational data. These classes also enable access to object attributes and invocation of object methods. See *Oracle interMedia Java Classes Reference* for more information.

1.10.2 Streaming Content from Oracle Database

You can stream content stored in a database using an *inter*Media 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 *inter*Media Plug-in for RealNetworks Servers

Oracle *inter*Media Plug-in for RealNetworks, RealSystem RealServer 7.0, RealSystem iQ Server 8.0, and Helix Universal Server allows these RealNetworks servers to stream multimedia data to a client directly out of the database. This plug-in is installed in the RealNetworks server and defined in the RealNetworks server configuration file. The data is requested with a URL, which contains information necessary to select the multimedia data from the database.

For information on RealNetworks servers, see the following URL

`http://www.real.com/`

See *Oracle interMedia Plug-in 2.0 for RealNetworks Streaming Servers Readme* for more information. The Oracle *inter*Media Plug-in for RealNetworks Streaming Servers can be downloaded from the Oracle *inter*Media Software section of the Oracle Technology Network Web site

`http://otn.oracle.com/products/intermedia`

1.10.3 Support for Web Technologies

Using *inter*Media 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 *inter*Media Java Classes for Servlets and JSP

Oracle *inter*Media Java Classes for servlets and JSP facilitates the upload and retrieval of multimedia data stored in a database using the *inter*Media OrdAudio, OrdDoc, OrdImage, and OrdVideo object types. Oracle *inter*Media Java Classes for servlets and JSP uses Oracle *inter*Media Java Classes to access data stored in the *inter*Media object types. However, Oracle *inter*Media Java Classes for servlets and JSP 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 a Java servlet. 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 interMedia Java Classes 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 JavaServer Pages. 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.

To read the Javadoc documentation that describes how to use Oracle *inter*Media Java Classes for servlets and JSP, see the API documentation, which can be found on the Oracle Database 10*g* Documentation Library CD-ROM as Oracle *inter*Media Java Classes for Servlets and JSP and as Oracle *inter*Media Java Classes.

Also, see *Oracle interMedia Java Classes Reference* for more information.

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 *inter*Media 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: Oracle Application Server Forms Services and Oracle Application Server Reports Services. 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 Oracle Application Server Forms Services 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 *inter*Media objects.

In addition to forms, Oracle Application Server Portal offers a report component. The Oracle Application Server Reports Services 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 an interactive demonstration on the use of Oracle Application Server Portal and *inter*Media, see Building Portlets Using Oracle *inter*Media and an *inter*Media/Portal Interactive Demonstration in the *inter*Media Training section on the Oracle Technology Network Web site

```
http://otn.oracle.com/products/intermedia
```

Integration with BC4J

For rapid development of media-rich Web applications, Oracle offers developers a Java integrated development environment (IDE) (Oracle JDeveloper 10*g*) and an application framework (Oracle Business Components for Java (BC4J)), that utilize rich media data type support in a database. Oracle JDeveloper 10*g* enables developers to build multitier, component-based Internet applications in Java quickly and efficiently. Oracle BC4J is an XML-powered framework for creating reusable business logic. An Oracle *inter*Media/BC4J integration package includes media-specific domain classes and a set of utilities. The domain classes are wrappers of the classes of Oracle *inter*Media Java Classes, and inherit all the underlying multimedia retrieval, upload, and manipulation methods. The domain classes support the BC4J framework 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 do the following:

- Write and test business logic in components that automatically integrate with relational databases.

- Reuse business logic through multiple SQL-based views of data, supporting different application tasks.
- Access and update the views from servlets, JavaServer Pages, and thin Java Swing clients.
- Customize application functions in layers without requiring modification of the delivered application.

For more information, see the Oracle *inter*Media/BC4J Interactive Demonstration in the *inter*Media Training section on the Oracle Technology Network Web site

```
http://otn.oracle.com/products/intermedia
```

WebDAV-Compliant Client Applications

You can also use a WebDAV-compliant client application, such as Adobe GoLive, Macromedia UltraDev, Microsoft's Web Folders, and other available software products to retrieve multimedia objects, such as audio, video, and image data, from a database. Using WebDAV, client applications have read-only access to multimedia content in the database.

To configure WebDAV access to multimedia data, using Oracle HTTP Server and the mod_oradav component (OraDAV), download the necessary software from the Oracle *inter*Media Software section of the Oracle Technology Network Web site

```
http://otn.oracle.com/products/intermedia
```

1.10.4 *inter*Media Custom DataSource and DataSink Classes for JMF 2.0/2.1

Oracle *inter*Media 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 *inter*Media OrdAudio and OrdVideo objects. See Chapter 6 for more information.

1.10.5 *inter*Media Support for Java Advanced Imaging (JAI)

Oracle *inter*Media Java Classes describes three types of stream objects, which provide interfaces to BLOB and BFILE data, that can be used by Java Advanced imaging (JAI). These classes allow a JAI application to read and write image data stored in a database using *inter*Media OrdImage objects, or in BLOBs or BFILEs. See *Oracle interMedia Java Classes Reference* for more information.

1.11 *inter*Media Architecture

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

The *inter*Media 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 *inter*Media 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 *inter*Media, 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-sized images, converting image formats, and image indexing and matching.

*inter*Media supports a heterogeneous media column, known as the ORDDoc object type. This allows a column to hold a mixture of image, audio, and video data, or other heterogeneous media data. Using *inter*Media import and export methods for each object type and for the relational interface, import and export operations between media objects and operating system files (external file storage) are possible. *inter*Media also supports special delivery types of servers, such as streaming content from a database. Using the Oracle *inter*Media Plug-in for RealNetworks, RealSystem RealServer 7.0, RealSystem iQ Server 8.0, or the Helix Universal 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 run external to the database.

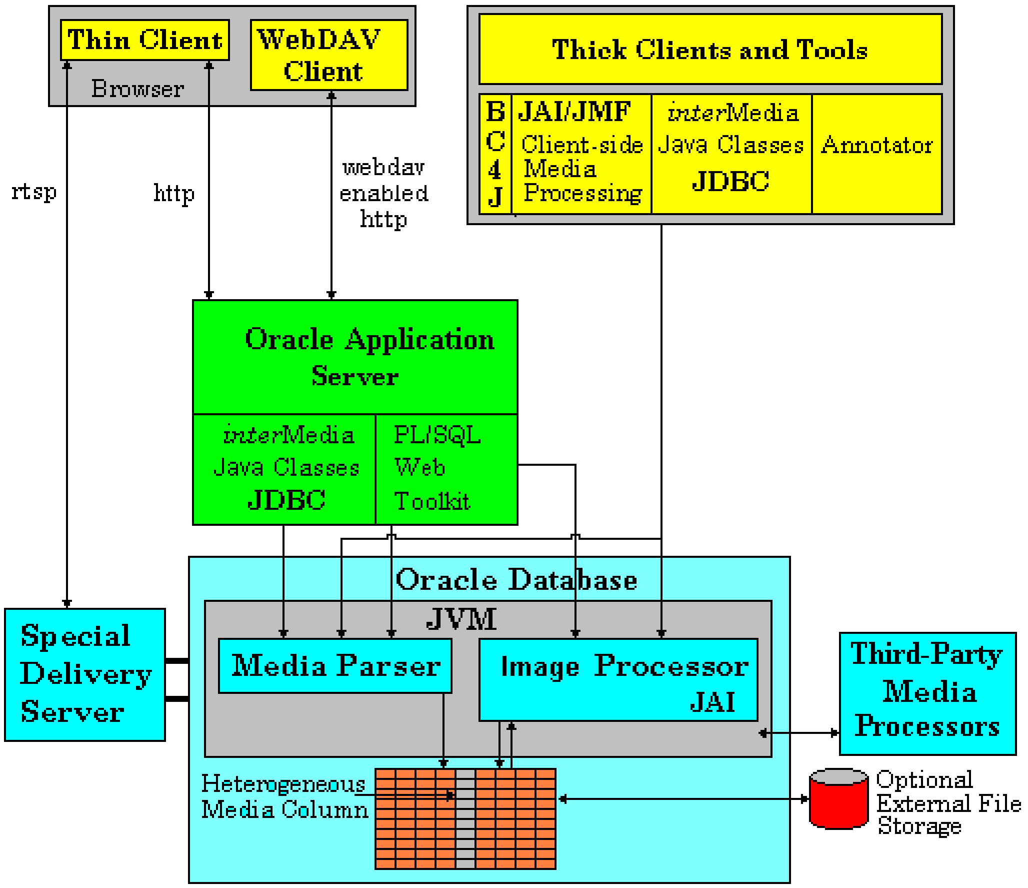
In the second or middle tier, Oracle Application Server provides access to *inter*Media through Oracle *inter*Media Java Classes, which enables Java applications on any tier (client, application server, or database) to access, manipulate, and modify audio, image, and video data stored in a database. Oracle *inter*Media Java Classes makes it possible for JDBC result sets to include both traditional relational data and *inter*Media media objects (OrdAudio, OrdDoc, OrdImage, and OrdVideo).

This support enables applications to easily select and operate on a result set that contains *inter*Media columns plus other relational data. These Java classes also enable access to *inter*Media object attributes and invocation of *inter*Media object methods.

In addition, Oracle *inter*Media Java Classes for servlets and JSP facilitates the upload and retrieval of multimedia data stored in a database using the *inter*Media OrdAudio, OrdDoc, OrdImage, and OrdVideo object types. Oracle *inter*Media Java Classes for servlets and JSP can access data stored in the *inter*Media objects or BLOBs or BFILEs directly.

In the third or client tier, the browser-based WebDAV clients can use the WebDAV-enabled HTTP protocol for communication with the Oracle Application Server tier to access media data in the database. For thick clients and tools, Oracle *inter*Media Annotator is a media parser. In addition, client-side media processing is supported through Java classes, JAI, and the JMF. For rapid development of media-rich Web applications, Oracle offers developers a Java IDE (Oracle JDeveloper 10*g*) and an application framework (Oracle Business Components for Java (BC4J)), that utilize rich media data type support in a database so developers can build scalable, multitier database applications from reusable business components.

Figure 1–1 interMedia Architecture


Thin Client
WebDAV Client
Browser
rtsp
http
webdav enabled http
Thick Clients and Tools
BC4J
JAI/JMF Client-side Media Processing
interMedia Java Classes JDBC
Annotator
Oracle Application Server
interMedia Java Classes JDBC
PL/SQL Web Toolkit
Oracle Database
JVM
Media Parser
Image Processor JAI
Special Delivery Server
Third-Party Media Processors
Heterogeneous Media Column
Optional External File Storage

Oracle *inter*Media features available only on Oracle Technology Network (OTN) `http://otn.oracle.com/products/intermedia` include the following:

- *inter*Media Plug-in for RealNetworks Servers, see Section 1.10.2
- WebDAV-compliant client applications, see Section 1.10.3
- *inter*Media Custom DataSource and DataSink classes JMF 2.0/2.1 (requires JMF 2.0 or higher), see Section 1.10.4
- *inter*Media speech mining and speech indexing, see Section 1.8.7

1.12 Extending Oracle *inter*Media

*inter*Media 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; not supported means *inter*Media can store any format, it just cannot extract the metadata or process the media data for these external formats that are not known to it.
- Audio and video data processing

For more information about extending *inter*Media, see Chapter 7.

2 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 *inter*Media 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 XTHML, 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, *inter*Media provides Internet support for Oracle Application Server 10*g* 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 *inter*Media 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 *inter*Media Java Classes. 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 *inter*Media Java Classes makes it possible for JDBC result sets to

include both traditional relational data with *inter*Media 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 interMedia Java Classes Reference* for more information.

  - Using Oracle *inter*Media Java Classes for servlets and JavaServer Pages (JSP). These additional classes support Web technologies. See Section 1.10.3 for more general information and Section 2.2 for more information about developing Java-based Web applications using JDBC. See Section 3.1.2 for a description of a Java servlet application and Section 3.1.3 for a description of a JSP application that uses these classes. See *Oracle interMedia Java Classes Reference* for more information.
  - Using the *inter*Media/BC4J integration package, which includes the *inter*Media domain classes and a set of utilities. These classes are for use with Oracle JDeveloper 10*g*, the Java integrated development environment (IDE) tool that supports the application framework (Oracle Business Components for Java (BC4J)) that enables you to build multitier, component-based Internet applications. See Section 1.10.3 and Oracle JDeveloper 10*g* help for more information.
  - Using Oracle *inter*Media 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 *inter*Media OrdAudio and OrdVideo object types. See Section 1.10.4 and *Oracle interMedia Java Classes Reference* for more information.
  - Using Java Advanced Imaging (JAI) classes. Oracle *inter*Media Java Classes 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 *inter*Media OrdImage objects, or in BLOBs or BFILEs. See *Oracle interMedia Java Classes Reference* 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 10*g* 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 2.1 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: Oracle Application Server Forms Services and Oracle Application Server Reports Services, which both support rich media content being uploaded or downloaded between the database and the portal framework form or report. See Section 1.10.3 for more information.
  - Oracle JDeveloper 10*g* -- written 100% in Java, is the IDE tool that supports the application framework (Oracle Business Components for Java (BC4J)). An *inter*Media/BC4J integration package includes the *inter*Media domain classes and a set of utilities. The domain classes are wrappers of Oracle *inter*Media Java Classes and inherit all the underlying multimedia retrieval, upload, and manipulation methods. The domain classes support the BC4J framework APIs and provide built-in integrated multimedia capabilities, while the utility classes support the retrieval, rendering, and uploading of multimedia content. See Section 1.10.3 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 or on Oracle Application Server 10*g* or on Oracle Database to call *inter*Media 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 or executing SQL or calling stored PL/SQL procedures for execution in the transaction context of the database.

- Integration with third-party client Web authoring tools for rapid Web site development to allow direct delivery of dynamic multimedia data stored in the database to the Web site. These third-party Web authoring tools include:
  - Oracle *inter*Media Plug-in for RealNetworks servers -- Oracle *inter*Media Plug-in for RealNetworks, RealSystem RealServer 7.0, RealSystem iQ Server 8.0, and Helix Universal Server. This tool lets these RealNetworks servers stream multimedia data to a client directly out of the database. This plug-in is installed in the RealNetworks server and defined in the RealNetworks server configuration file. The data is requested with a URL, which contains information necessary to select the multimedia data from the database. See Section 1.10.2 for more information.

Chapter 3 describes the *inter*Media photo album Web application and how to develop media upload and retrieval applications using either the PL/SQL development environment, the Java IDE, or the Microsoft Active Server Pages (ASP)/Visual Basic (VB) development environment for developing Web applications for the Microsoft IIS Web Server using *inter*Media. In addition, this chapter describes an *inter*Media Code Wizard application that 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 *inter*Media object types. You can either create and compile standalone media access procedures using the Code Wizard, or you can create the source of media access procedures for inclusion in a PL/SQL package.

Chapter 4 describes the IMExample Java sample application (sometimes referred to as a demo) and how the classes of Oracle *inter*Media Java Classes are used to create this sample application that lets you retrieve from the sample schema, save to a file, play, and delete from the sample schema *inter*Media image, audio, video, and testimonial data using the respective *inter*Media object types, OrdImage, OrdAudio, OrdVideo, and OrdDoc.

If you are not familiar with developing PL/SQL Web applications and using the PL/SQL Gateway and PL/SQL Web Toolkit, see Section 2.1.

If you are not familiar with developing Java-based Web applications using JDBC to access *inter*Media objects, see Section 2.2.

2.1 Developing PL/SQL Web Applications

SQL developers familiar with the database can develop Web applications that exclusively use Oracle Application Server 10*g* and Oracle Database using the PL/SQL development environment. 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.

The PL/SQL development environment lets you achieve the best performance for database-intensive applications because PL/SQL is highly optimized for use with the database through its support for and tight integration with SQL, support for processing an entire block of SQL statements at one time, and letting you compile PL/SQL procedures and store them in executable form in the database, to be called later. In addition, as a development environment, SQL developers have support for the object-oriented programming model, can experience higher productivity due to its procedural nature, and 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 10*g* 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 *inter*Media 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 *inter*Media 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 *inter*Media 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. *inter*Media 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.

*inter*Media 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.

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

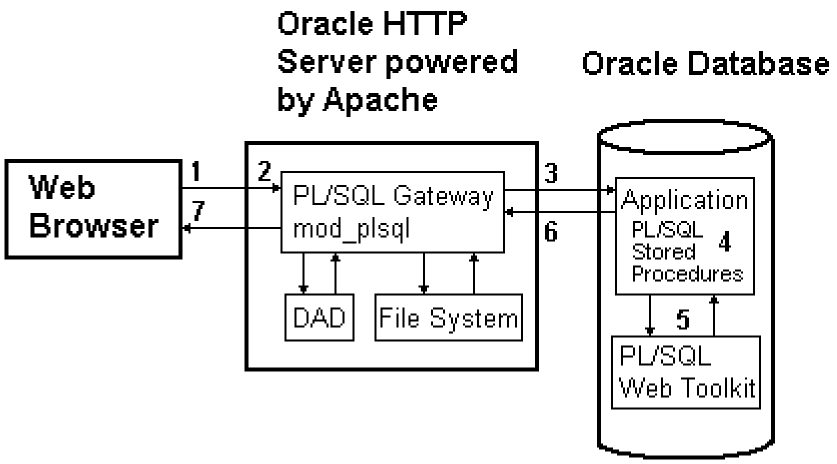
Oracle Application Server 10*g* and Oracle Database install Oracle HTTP Server powered by the Apache HTTPD server that contains the PL/SQL Gateway to communicate directly with a client Web browser.

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 `htp`, `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 10*g* 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 `htp`, `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 2–1.

Example 2–1 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, for the purpose of using the photo album application for *inter*Media and the PL/SQL Web Toolkit described in Section 3.1.1, the URL can be simplified to the format shown in Example 2–2.

Example 2–2 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 2–1, 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 Section 3.2.1 illustrates how this works. For example, to invoke the Code Wizard administration URL, enter the following URL shown in Step 3 in Section 3.2.2:

```
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 `ordcwadmin` 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 Application Developer's Guide - Fundamentals*.

2.2 Developing Java-Based Web Applications Using JDBC

Java database connectivity (JDBC) is a standard Java interface defined by Sun Microsystems, based on the X/Open SQL Call Level Interface that complies with the SQL 92 Entry Level standard, that is used for connecting from Java to relational databases. JDBC supports dynamic SQL, letting a calling program construct SQL statements dynamically at runtime. These are the major benefits of using this Java interface in addition to allowing individual providers, such as Oracle Corporation, to implement and extend their own JDBC drivers.

A database-embedded JVM supports the JDBC interface. Java source code (.java file) is compiled into one or more byte code files (`.class` files) and these class files are interpreted at runtime and executed by the embedded JVM. If your Java application uses objects defined in other packages, you must set the CLASSPATH environment variable and specify the paths to all objects used by your application.

Resulting class files for servlets are usually placed in the directory enabled to run servlets. Class files for JSP are usually placed in the JavaBean directory for your servlet's container, while the JSP files are usually copied to a directory enabled to serve JSP, which by default, is your JSP sample application directory. For servlets, the actual location of your servlet class files depends on the servlet container you are using and how it is configured. For JSP, the actual location of your servlet and JSP files depends on the servlet container and JSP engine you are using, and how the engine is configured. Each of the Java sample applications uses the default location of the servlet containers as required by Oracle HTTP Server powered by Apache for an installation of Oracle Application Server 10*g* or Oracle Database. See the respective `Java sample application readme.txt` files for more information.

To write a Java application that uses the JDBC interface and the embedded JVM, perform the following operations that will access *inter*Media objects in a database table:

1. Establish a JDBC connection from the Java application to the database.

   Call a getConnection( ) method to obtain an OracleConnection object.

2. If your application will modify the *inter*Media object, perform the following operations:

   a. Call the setAutoCommit( ) method to disable auto-commit mode.

   b. Execute a SELECT... FOR UPDATE statement on the database table.

   Create an OracleStatement or OraclePreparedStatement object in your application. Call the executeQuery( ) method to execute the SELECT... FOR UPDATE statement and return an OracleResultSet object, and fetch a row from the result set.

3. If your application will not modify the *inter*Media object, execute a SELECT statement on the database table.

4. Retrieve the *inter*Media object from the result set as an instance of one of the classes of Oracle *inter*Media Java Classes.

5. Perform operations on the Java application object.

Having retrieved the *inter*Media Java object from the result set, your application can now load new data into the object, or your application can retrieve or manipulate existing data in the object.

6. If the *inter*Media object has been modified by the application, update the database object to include the results of the operations, and commit your changes.

   If the application modified the object in the previous step, create an OraclePreparedStatement object that contains a SQL statement that updates the database object, and execute the statement.

   Commit the transaction by calling the commit( ) method.

7. Close the connection to the database table.

   The IMExample Java sample application, which is described in Chapter 4, provides examples of each step in this process.

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

3 Developing Media Upload and Retrieval Applications

This chapter describes the development of the following types of media upload and retrieval applications using *inter*Media object types:

- Section 3.1 describes the development of the following *inter*Media photo album sample Web applications that use the *inter*Media image object type:
  - PL/SQL application that uses the PL/SQL Gateway and PL/SQL Web Toolkit for Oracle Application Server 10*g* and Oracle Database, see Section 3.1.1
  - Java servlet application that uses Oracle *inter*Media Java Classes for servlets and JSP, see Section 3.1.2.
  - JavaServer Pages (JSP) application that uses Oracle *inter*Media Java Classes for servlets and JSP, see Section 3.1.3
  - Active Server Pages (ASP)/Visual Basic (VB) application for the Microsoft Internet Information Server (IIS) Web Server, see Section 3.1.4
- Section 3.2 describes the *inter*Media Code Wizard application for the PL/SQL Gateway that uses the *inter*Media image, audio, video, and heterogeneous media object types.

Section 3.1 and Section 3.2 assume 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 10*g* and Oracle Database.
  - Developing ASP/VB scripts for the Microsoft IIS Web Server.

- You have already installed and configured the following sample application:
  - Oracle *inter*Media PL/SQL Web Toolkit Photo Album application
  - Oracle *inter*Media Java Servlet Photo Album application
  - Oracle *inter*Media JSP Photo Album application
  - Oracle *inter*Media ASP/VBScript Photo Album application
  - Oracle *inter*Media Code Wizard for the PL/SQL Gateway application

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

3.1 *inter*Media Photo Album Sample Applications

This set of four *inter*Media photo album sample applications demonstrates the use of *inter*Media image object type to upload and retrieve media data stored in Oracle Database.

Each of these photo album applications, 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);
```

Note that the data type for the `image` and `thumb` columns are defined as *inter*Media image object types to store the full-sized images and the generated thumbnail images.

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

```
CREATE SEQUENCE photos_sequence;
```

3.1.1 Oracle *inter*Media PL/SQL Web Toolkit Photo Album Sample Application

The *inter*Media PL/SQL Web Toolkit Photo Album sample application shows you how to develop a Web application using PL/SQL Web Toolkit function calls to generate the HTML tags for Web page generation.

During the installation, 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 );
```

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`.

3.1.1.1 Running the Photo Album Application

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

```
<protocol><hostname:port-number>/pls/photo_album_dad
```

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, the 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 photo, select **Upload new photo**. Enter a description of the photo, the location where the photo 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 photo. Click the thumbnail image to view the full-sized version of the photo. When the photo album application displays the text **view image** instead of its thumbnail image, the image format that was uploaded was not recognized by *inter*Media. Click **view image** to display the full-sized image.

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

3.1.1.2 Description of the Photo Album Application

The photo album application is composed of a PL/SQL package specification containing 12 modules: 10 procedures, and 2 functions and with the package body implementation. Each module describes tasks that the photo album application performs and information that the photo album application displays on the Web page. These modules are categorized into specific actions and print operations to reduce redundancy, with each action and print operation being assigned to a specific procedure.

Some examples of action operations are the following:

- Display the contents of the photo album (`view_album`).
- Display an entry in the photo album (`view_entry`).
- Retrieve the media from the database and deliver it to the Web browser (`deliver_media`).
- Display the upload form for inserting a new photo in the photo album (`view_upload_form`).
- Insert a new photo in the photo album (`insert_new_photo`).

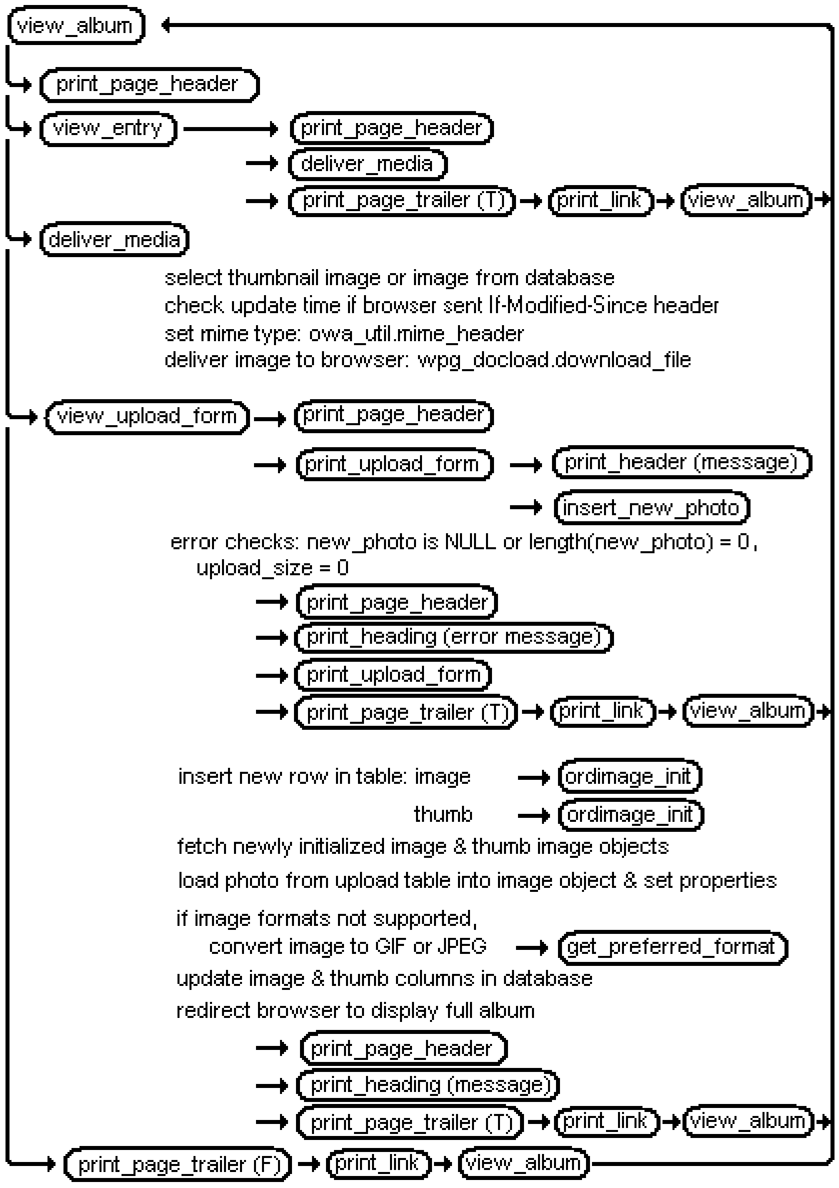
Print operations display some text on the Web page, such as:

- Print the upload form (`print_upload_form`).
- Print the common page header (`print_page_header`).
- Print the common page trailer (`print_page_trailer`).
- Print a heading message (`print_heading`).
- Print a text link (`print_link`).

The function `get_preferred_format` performs a task described later in this section, and returns a value.

Figure 3–1 shows a flow diagram of the names of the procedures and functions, and the order in which each is called to implement a particular operation in the photo album application.

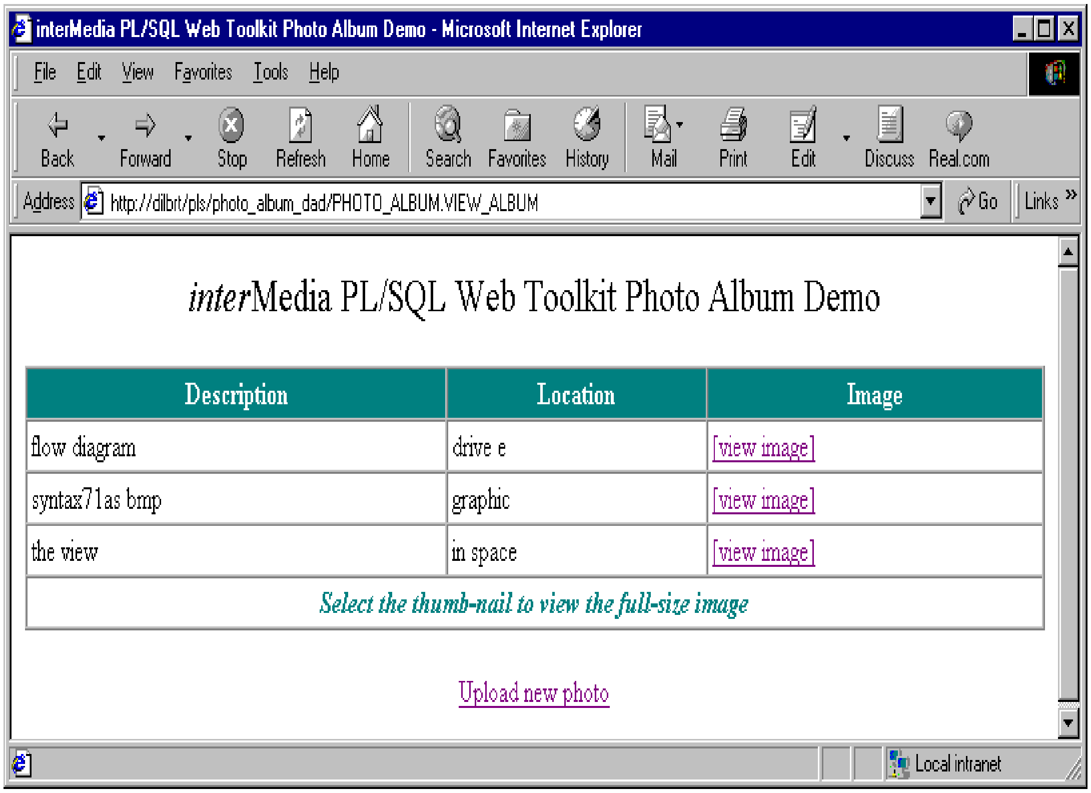
Figure 3–1 Procedure and Function Flow Chart for the Photo Album Application


view_album
print_page_header
view_entry
print_page_header
deliver_media
print_page_trailer (T)
print_link
view_album
deliver_media
select thumbnail image or image from database
check update time if browser sent If-Modified-Since header
set mime type: owa_util.mime_header
deliver image to browser: wpg_docload.download_file
view_upload_form
print_page_header
print_upload_form
print_header (message)
insert_new_photo
error checks: new_photo is NULL or length(new_photo) = 0,
upload_size = 0
print_page_header
print_heading (error message)
print_upload_form
print_page_trailer (T)
print_link
view_album
insert new row in table: image
ordimage_init
thumb
ordimage_init
fetch newly initialized image & thumb image objects
load photo from upload table into image object & set properties
if image formats not supported,
convert image to GIF or JPEG
get_preferred_format
update image & thumb columns in database
redirect browser to display full album
print_page_header
print_heading (message)
print_page_trailer (T)
print_link
view_album
print_page_trailer (F)
print_link
view_album

The implementation of the photo album sample application is defined in the PL/SQL package named PHOTO_ALBUM. The package specification includes the following procedures and functions.

VIEW_ALBUM Procedure

Procedure VIEW_ALBUM displays the contents of the photo album on a Web page as a three column table that includes the image description, location, and a thumbnail image to view if the image format is supported by *inter*Media. See Figure 3–2.



http://dilbrt/pls/photo_album_dad/PHOTO_ALBUM.VIEW_ALBUM

*inter*Media PL/SQL Web Toolkit Photo Album Demo

Description	Location	Image
flow diagram	drive e	[view image]
syntax71as bmp	graphic	[view image]
the view	in space	[view image]
Select the thumb-nail to view the full-size image		

Upload new photo

Figure 3–2 interMedia PL/SQL Web Toolkit Photo Album Application (Demo)

Thumbnail images display as anchor tags that, when selected, are used to display the full-sized image using the following statements:

```
htp.print( '<td headers="image"><a href="PHOTO_ALBUM.VIEW_ENTRY' ||
         '?entry_id=' || entry.id || '">' );
```

```
IF entry.thumb.contentLength > 0
THEN
    htp.print( '<img src="PHOTO_ALBUM.DELIVER_MEDIA?media=thumb' ||
                 ampersand || 'entry_id=' || entry.id || '"' ||
               ' height=' || entry.thumb.height ||
               ' width=' || entry.thumb.width ||
               ' alt="' || htf.escape_sc( entry.description ) || '"' ||
               ' border=1></a></td>' );
ELSE
    htp.print( '[view image]</a></td>' );
END IF;
```

The thumbnail images for all entries are displayed by calling the `PHOTO_ALBUM.DELIVER_MEDIA` procedure using the tag `<IMG SRC="PHOTO_ALBUM.DELIVER_MEDIA?media=thumb...>` within the preceding IF...THEN...ELSE...END IF statement. Notice that in the query string, `?media=thumb`, that the name/value pair indicates that the parameter named `media` declared in the `deliver_media` procedure has a value of `thumb`, which represents the thumbnail image.

In the preceding IF...THEN...ELSE...END IF statement, if the image format is not supported by *inter*Media, then no thumbnail image would have been created when the image was uploaded. In this special case, a text link of **view image** is displayed as indicated in the `ELSE` clause. Selecting **view image** in the `image` column, calls the `PHOTO_ALBUM.VIEW_ENTRY` procedure to display the full-sized image.

At the bottom of the table is **Select the thumbnail to view the full-size image**, which when selected, calls the `PHOTO_ALBUM.VIEW_UPLOAD_FORM` procedure. The Web browser also displays a common page header with the page title **interMedia PL/SQL Web Toolkit Photo Album Demo** by calling the `PHOTO_ALBUM.PRINT_PAGE_HEADER` procedure. In addition, the Web page displays a common page trailer by calling the `PHOTO_ALBUM.PRINT_PAGE_TRAILER` procedure. Note that the page trailer is controlled by a Boolean operation, and in this case, is set to false and so the trailer is not displayed. At the bottom of the page is an **Upload new photo** link to upload a new photo, which when selected, calls the `PHOTO_ALBUM.VIEW_UPLOAD_FORM` procedure.

VIEW_ENTRY Procedure

Procedure `VIEW_ENTRY` displays the description, location, and a full-sized version of the image by calling the `PHOTO_ALBUM.DELIVER_MEDIA` procedure, using the tag `<IMG SRC="PHOTO_ALBUM.DELIVER_MEDIA?media=image...>` shown in the following statement:

```
htp.prn( '<tr><td scope= "col" valign="top"><b>Photo:</b></td>
              <td scope="col">' ||
'<img src="PHOTO_ALBUM.DELIVER_MEDIA?media=image' ||
  ampersand || 'entry_id=' || entry_id || '"' );
htp.prn( ' alt="' || htf.escape_sc( entry.description ) || '"' );
```

Notice that in the query string, `?media=image`, that the name/value pair indicates that the parameter named `media` declared in the `DELIVER_MEDIA` procedure has a value of `image`, which represents the full-sized image.

The `VIEW_ENTRY` procedure is called from the `VIEW_ALBUM` procedure.

The Web page also displays the common page header (the Web page title) by calling the `PHOTO_ALBUM.PRINT_PAGE_HEADER` procedure. In addition, the common page trailer is called with the `PHOTO_ALBUM.PRINT_PAGE_TRAILER` procedure. In this case, the Boolean expression is set to true and displays the **Return to photo album** link, near the bottom of the Web page. When this link is selected, it calls the `PHOTO_ALBUM.VIEW_ALBUM` procedure.

DELIVER_MEDIA Procedure

Procedure `DELIVER_MEDIA` retrieves the image from the database and delivers either a thumbnail or a full-sized image to the Web browser, depending on the value of the field named `MEDIA`. If the value of the `MEDIA` field is `thumb`, a thumbnail image is delivered; otherwise, a full-sized image is delivered.

If the image requested is in the browser cache and the cache is valid, then the image is retrieved from cache; otherwise, the MIME type of the image is set based on the image attribute value in the database, then the image is retrieved from the database and is delivered to the browser, as follows:

```
IF ordplsgwyutil.cache_is_valid( local_image.getUpdateTime() )
THEN
  owa_util.status_line( ordplsgwyutil.http_status_not_modified );
  RETURN;
END IF;
.
.
.
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;
```

Whether the image retrieved is a thumbnail or a full-sized image, two packages of the PL/SQL Web Toolkit are used to do this operation. The first package, owa_util, sets the MIME type of the image, while the second package, wpg_docload, downloads the image from the database to the Web browser.

The DELIVER_MEDIA procedure is called by the PHOTO_ALBUM.VIEW_ALBUM and PHOTO_ALBUM.VIEW_ENTRY procedures.

VIEW_UPLOAD_FORM Procedure

Procedure VIEW_UPLOAD_FORM displays the upload form by calling the PHOTO_ALBUM.PRINT_UPLOAD_FORM procedure. The procedure also displays the common page header by calling the PHOTO_ALBUM.PRINT_PAGE_HEADER procedure, and displays the common page trailer by calling the PHOTO_ALBUM.PRINT_PAGE_TRAILER procedure. The VIEW_UPLOAD_FORM procedure sets the Boolean expression to true to display the common page trailer.

PRINT_UPLOAD_FORM Procedure

Procedure PRINT_UPLOAD_FORM calls the PHOTO_ALBUM.PRINT_HEADING procedure to display the message header **Upload a new photo** on the Web page, then calls the PHOTO_ALBUM.INSERT_NEW_PHOTO procedure to do the insert operation. In addition, the user input for the image description, image location, and image file name is entered by the user upon request. Clicking **Upload photo** initiates the upload operation because the input type is submit.

INSERT_NEW_PHOTO Procedure

Procedure INSERT_NEW_PHOTO inserts a new photo into the photo album. This procedure also checks to make sure a file name was entered as input from the previous PHOTO_ALBUM.PRINT_UPLOAD_FORM procedure call, and that the file has content or is not of zero length. If the DESCRIPTION field is blank, the file name is used as the description. Next, the ORDSYS.ORDIMAGE.INIT( ) function is called to initialize both 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 and full-sized image object type columns for update. A DBMS_LOB.COPY operation loads the image from the upload table into the full-sized image ORDImage object type column, then calls the setProperties( ) method to read the image data, get the values of the image object attributes, and store them in the object attributes for the image column.

Because some browsers cannot display some image formats inline, in this sample application, BMP formatted images are converted to either a JPEG image format (for images with greater 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-sized image to create the thumbnail image, then both the full-sized image and the thumbnail image are updated in the database with a SQL UPDATE statement. After the row is updated in the `photos` table, then the same row containing these new images is deleted from the upload `PHOTOS_UPLOAD` table. The upload operation returns a message "Photo successfully uploaded into photo album". Finally, the `PHOTO_ALBUM.VIEW_ALBUM` procedure is called again to refresh the Web page and the `PHOTO_ALBUM.PRINT_PAGE_TRAILER` procedure is called to display the common page trailer with the Boolean expression set to true to display the trailer.

PRINT_PAGE_HEADER Procedure

Procedure `PRINT_PAGE_HEADER` displays the common page header, **interMedia PL/SQL Web Toolkit Photo Album Demo**.

PRINT_PAGE_TRAILER Procedure

Procedure `PRINT_PAGE_TRAILER` contains a Boolean expression, which when set to true, calls the `PRINT_LINK` procedure; otherwise when set to false, it displays no text link.

PRINT_HEADING Procedure

Procedure `PRINT_HEADING` displays a one line message for the specified value of the `MESSAGE` field when this procedure is called.

PRINT_LINK Procedure

Procedure `PRINT_LINK` displays **Return to photo album** at the bottom of the Web page, which when selected, calls the `PHOTO_ALBUM.VIEW_ALBUM` procedure. The `PRINT_LINK` procedure is called from the `PHOTO_ALBUM.PRINT_PAGE_TRAILER` procedure.

For more information about building applications using PL/SQL, see *Oracle HTTP Server mod_plsql User's Guide*, *Oracle Application Server 10g PL/SQL Web Toolkit Reference*, *Oracle HTTP Server Administrator's Guide*, *Oracle Database Application Developer's Guide - Fundamentals*, *PL/SQL User's Guide and Reference*, and *PL/SQL Packages and Types Reference*.

3.1.2 Oracle *inter*Media Java Servlet Photo Album Sample Application

The *inter*Media Java Servlet Photo Album sample application demonstrates the use of Oracle *inter*Media Java Classes for servlets and JSP to upload and retrieve multimedia data to and from the database. Users access the servlet application to view the contents of the photo album, including thumbnail versions of each photo, to view the full-sized version of any photo, and to upload new photos into the album.

3.1.2.1 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 10*g* or Oracle Database

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

- Default installation of Tomcat 3.2 on Windows NT

  ```
  <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, the 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 photo, select **Upload new photo**. Enter a description of the photo, the location where the photo 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 photo. Click the thumbnail image to view the full-sized version of the photo.

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

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

3.1.2.2 Description of the *inter*Media Java Servlet Photo Album Application

The *inter*Media 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 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-sized 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 photo 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 following methods: viewAlbum( ), viewPhoto( ), viewMedia( ), viewUploadForm( ), insertNewPhoto( ), printPageHeader( ), printPageTrailer( ), printMessage( ), printHeading( ), and printLink( ), and the getPreferredFormat( ) function.

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 the Oracle *inter*Media Java Classes for servlets and JSP. 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 ORAData (formerly getCustomDatum) and ORADataFactory (formerly the 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.getORAData( 4, OrdImage.getORADataFactory() );
.
.
.
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 printPageHeader( ) function.
  - 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-sized image. When a user clicks the thumbnail image or **view image**, the full-sized 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 *inter*Media, the text **view image** is displayed instead.
  - Increments the row count to see if the photo album is empty; if so, display 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-sized version of a photo and does the following:
  - Writes a common page header on the HTML page using the printPageHeader( ) function.
  - 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-sized 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-sized 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-sized 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-sized images height and width by calling the getHeight( ) and getWidth( ) *inter*Media object methods. If the image format is not recognized by *inter*Media, 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-sized image URLs to retrieve a thumbnail or full-sized 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-sized 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-sized 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 photos 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 photo.
  - Uses the `OrdHttpUploadFile` class to upload the new photo 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-sized 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-sized 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-sized image, create a thumbnail image, and populate an OrdImage object named `thumb`.
  - Executes a SQL UPDATE statement to update the full-sized 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 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.

3.1.3 Oracle *inter*Media JavaServer Pages (JSP) Photo Album Sample Application

The *inter*Media JavaServer Pages (JSP) Photo Album sample application is a JSP application that demonstrates the use of Oracle *inter*Media Java Classes for servlets and JSP to upload and retrieve multimedia data to and from a database. Users access the JSP that constitute the application to view the contents of the photo album, including thumbnail versions of each photo, to view the full-sized version of any photo, and to upload new photos into the album.

3.1.3.1 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 JSP photo album application by entering the following URL in the address field of your Web browser:

- Default installation of Oracle Application Server 10*g* or Oracle Database

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

- Default installation of Tomcat 3.2 on Windows NT

  ```
  <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, the 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 photo, select **Upload new photo**. Enter a description of the photo, the location where the photo 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 photo. Click the thumbnail image to view the full-sized version of the photo.

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

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

3.1.3.2 Description of the *inter*Media 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 following methods: selectTable( ), selectRowById( ), fetch( ), insertNewPhoto( ), release( ), getConnection( ), freeConnection( ), setId( ), setDescription( ), setLocation( ), getImage( ), and getThumb( ), and the functions: getId( ), getDescription( ), and getLocation( ), and getPreferredFormat( ).

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 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 *inter*Media Java Classes for servlets and JSP. 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 *inter*Media Java Classes for servlets and JSP 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 photo 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 ORAData (formerly getCustomDatum) and ORADataFactory (formerly the 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.getORAData( 1, OrdImage.getORADataFactory());
thumb = (OrdImage)rset.getORAData( 2, OrdImage.getORADataFactory());

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.setORAData( 1, image );
    stmt.setORAData( 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 ORAData and ORADataFactory 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.getORAData( 4, OrdImage.getORADataFactory() );
        thumb = (OrdImage)rset.getORAData( 5, OrdImage.getORADataFactory() );
        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-sized 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-sized version of a photo 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-sized version of a photo in the album using an image tag in the form `<IMG SRC="PhotoAlbumMediaViewer.jsp?media=image&...">` under the `Photo` header.
- Displays the full-sized images height and width by calling the getHeight( ) and getWidth( ) methods. If the image format is not recognized by *inter*Media, 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-sized 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-sized 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 photos 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 photo. 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 photo into 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 photo 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-sized 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-sized 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 getContentFormat( ) 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-sized 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-sized 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.

3.1.4 Oracle *inter*Media ASP/VBScript Photo Album Sample Application

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

3.1.4.1 Running the ASP/VBScript Photo Album Application

After you have installed and configured this photo album application in Microsoft IIS and configured the application connection parameters, you are ready to run the photo album 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 photo, click **Upload new photo**. Enter a description of the photo, the location where the photo 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 photo. Click the thumbnail image to view the full-sized version of the photo.

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

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

3.1.4.2 Description of the ASP/VBScript Photo Album Application

The top-level files that implement the 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-sized image, using the tag `<img src="getPhoto.asp?entry_id=...">` 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 *inter*Media 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 photo into the database.

viewEntry.asp

The **viewEntry** page, which displays the full-sized version of a photo, also uses the tag `<img src="getPhoto.asp?entry_id...">` 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 photo into the database by entering description and location information for the new photo, 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-sized 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
```

3.2 *inter*Media Code Wizard Sample Application

The *inter*Media 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 *inter*Media 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 3.1.1.2) 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.

3.2.1 Using the Code Wizard

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

1. Create a new 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 each of these topics in more detail as the following topics:

- Creating a New DAD or Choosing an Existing DAD
- Authorizing a DAD
- Creating and Testing Media Upload and Retrieval Procedures
- Using the PL/SQL Gateway Document Table
- How Time Zone Information Is Used to Support Browser Caching

3.2.1.1 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 on configuring DADs, see *Oracle HTTP Server mod_plsql User's Guide*.

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. Enter the URL of the **Gateway Configuration Menu** page into your browser's location bar, for example:

   ```
   http://<host-name>:<port-number>/pls/admin_/gateway.htm
   ```

2. Select **Gateway Database Access Descriptor Settings**.
3. Select **Add Default (blank configuration)**.
4. Enter the following information into the specified sections of the Create DAD Entry form:

   ```
   Add Database Access Descriptor
     Database Access Descriptor Name:      SCOTTCW
     Schema name:                          [leave blank]
   Database Connectivity Information
     Oracle User Name:                     SCOTT
     Oracle Password:                      [leave blank]
     Oracle Connect String:                [leave blank or enter TNS name]
   Authentication Mode
     Authentication Mode:                  Basic
   Session Cookie
     Session Cookie Name:                  [leave blank]
   Package/Session Management State
     Package/Session Management Type       Stateless (Reset Package State)
   Connection Pool Parameters
     Enable Connection Pooling?            Yes
   Default (Home) Page
     Default (Home) Page:                  ORDCWPKG.MENU
   Document Access Information
     Document Table:                       MEDIA_UPLOAD_TABLE
     Document Access Path:                 [leave blank]
     Document Access Procedure:            [leave blank]
     Extensions uploaded as Long Raw:      [leave blank]
   Path Aliasing
     Path Alias:                           [leave blank]
     Path Alias Procedure:                 [leave blank]
   ```

3.2.1.2 Authorizing a DAD

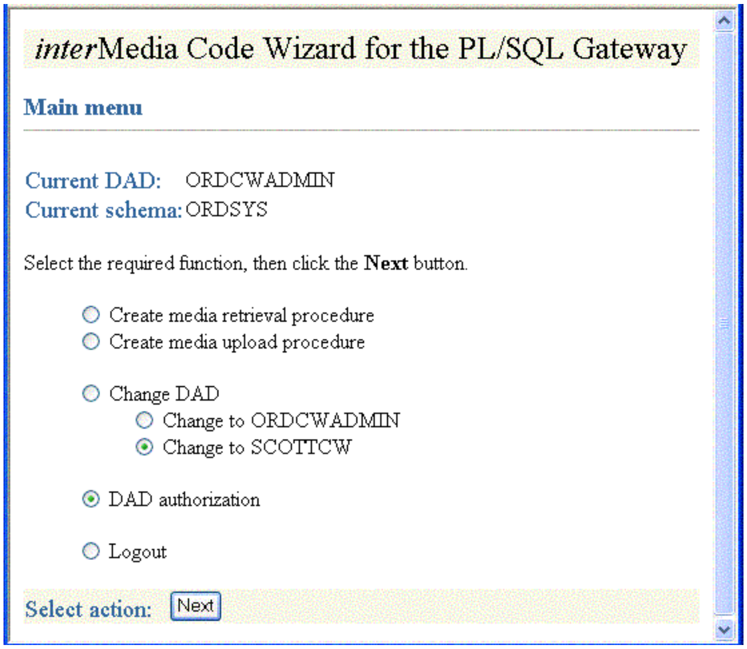
To authorize a DAD for use with the Code Wizard, do the following steps:

1. Enter the Code Wizard's administration URL into your browser's location bar, for example:

   ```
   http://<host-name>:<port-number>/pls/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 3–3, then click **Next**.

Figure 3–3 Main Menu for the interMedia Code Wizard for the PL/SQL Gateway



*inter*Media Code Wizard for the PL/SQL Gateway

Main menu

Current DAD: ORDCWADMIN
Current schema: ORDSYS

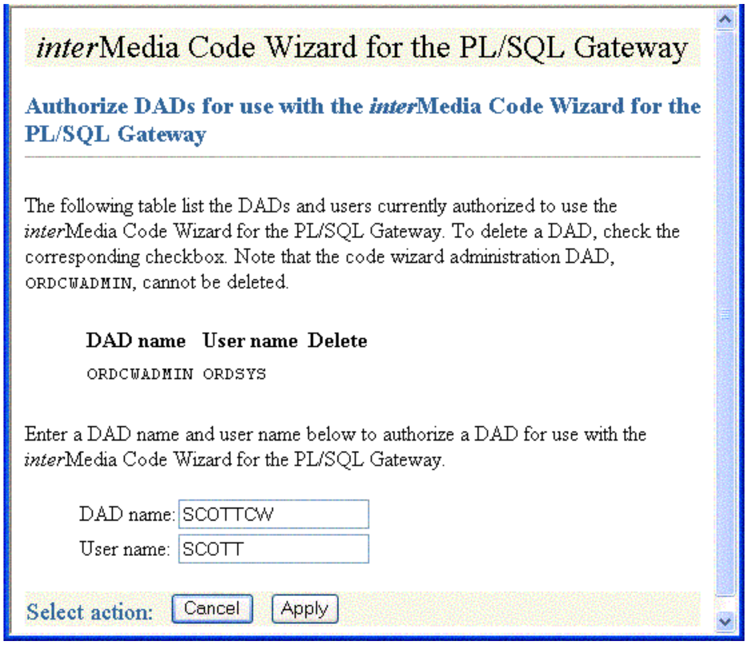
Select the required function, then click the **Next** button.

- Create media retrieval procedure
- Create media upload procedure
- Change DAD
  - Change to ORDCWADMIN
  - [✓ Change to SCOTTCW]
- [✓ DAD authorization]
- Logout

Select action: Next

4. Enter the name of the DAD you wish to authorize together with the user name, as shown in Figure 3–4, then click **Apply**.

Figure 3–4 Authorizing the SCOTTCW DAD



*inter*Media Code Wizard for the PL/SQL Gateway

Authorize DADs for use with the *inter*Media Code Wizard for the PL/SQL Gateway

The following table list the DADs and users currently authorized to use the *inter*Media 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	

Enter a DAD name and user name below to authorize a DAD for use with the *inter*Media Code Wizard for the PL/SQL Gateway.

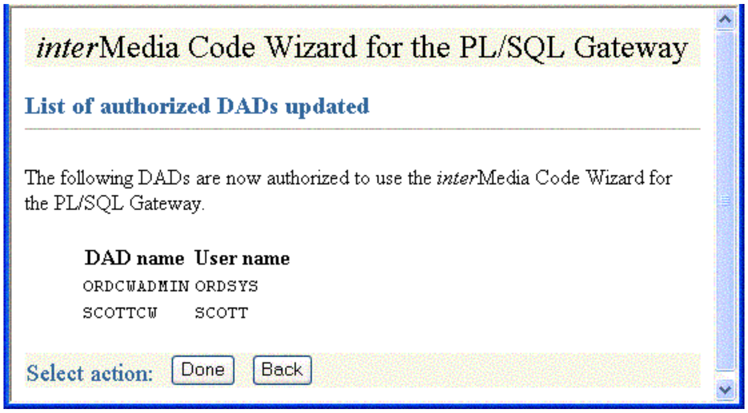
DAD name: SCOTTCW
User name: SCOTT

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.

---

5. Review the updated list of DADs that are authorized to use the *inter*Media Code Wizard, as shown in Figure 3–5, then click **Next**.

Figure 3–5 List of Authorized DADs



*inter*Media Code Wizard for the PL/SQL Gateway

List of authorized DADs updated

The following DADs are now authorized to use the *inter*Media Code Wizard for the PL/SQL Gateway.

DAD name	User name
ORDCWADMIN	ORDSYS
SCOTTCW	SCOTT

6. To log out (clear HTTP authentication information), select **Logout** from the **Main menu**, 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*.

3.2.1.3 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:

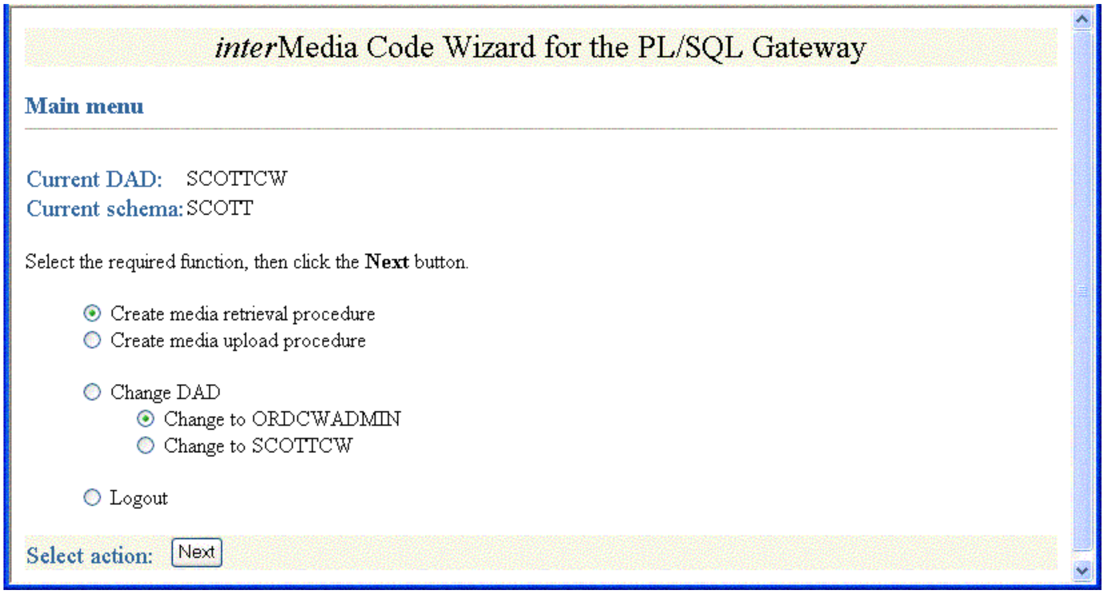
```
http://<hostname>:<port-number>/pls/scottcw

or

http://<hostname>:<port-number>/pls/mediadad/ordcwpkg.menu
```

Then, enter the user name and password when prompted by the browser. The **Main menu** page of the *inter*Media Code Wizard for the PL/SQL Gateway is displayed as shown in Figure 3–6.

Figure 3–6 Using the SCOTTCW DAD



*inter*Media 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

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 3.2.1.4) or a media retrieval procedure (see Section 3.2.1.5), 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.

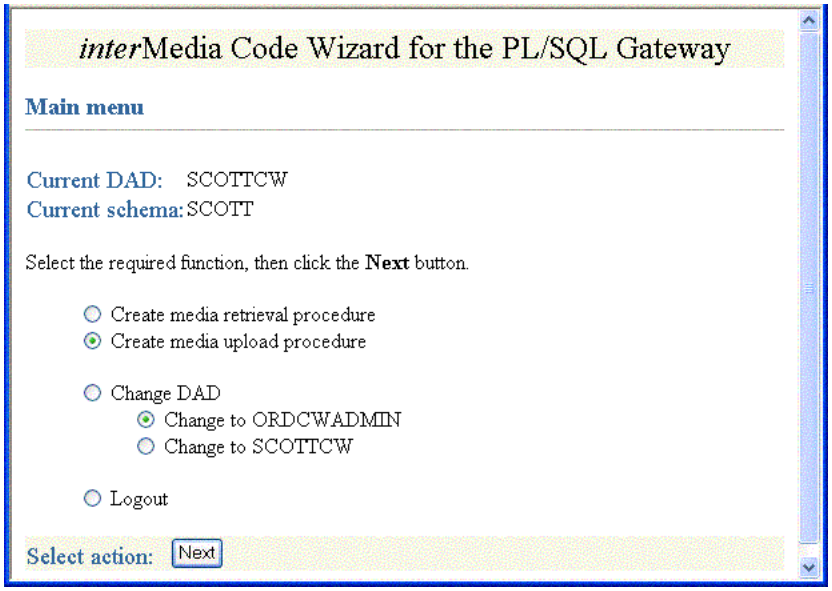
The image and multimedia sample sessions described in Section 3.2.2 and Section 3.2.3 respectively, illustrate how to create and test a media upload procedure and a media retrieval procedure.

3.2.1.4 Creating a Media Upload Procedure

To create a media upload procedure using the *inter*Media Code Wizard for the PL/SQL Gateway, do the following steps:

1. At the **Main menu**, select **Create media upload procedure** as shown in Figure 3–7. Click **Next**.

Figure 3–7 Create a Media Upload Procedure



*inter*Media Code Wizard for the PL/SQL Gateway

Main menu

Current DAD: SCOTTCW
Current schema: SCOTT

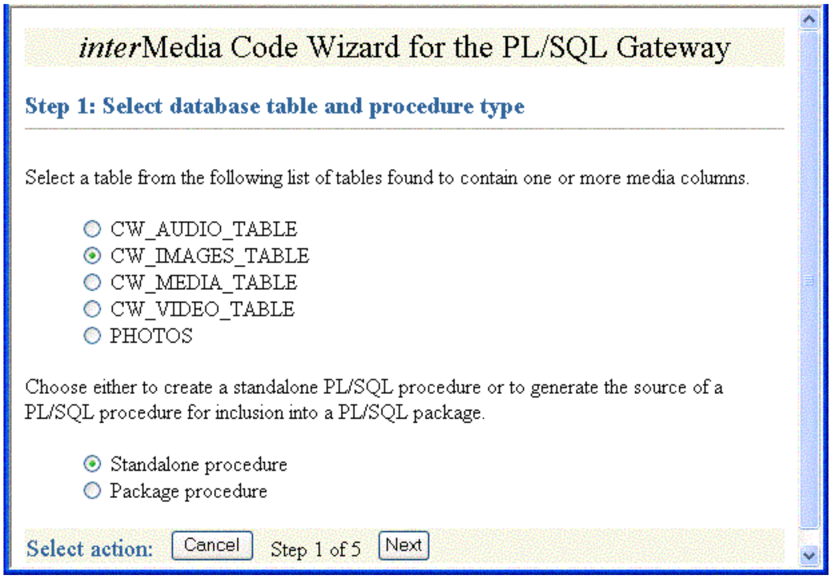
Select the required function, then click the **Next** button.

- ○ Create media retrieval procedure
- ◉ Create media upload procedure
- ○ Change DAD
  - ◉ Change to ORDCWADMIN
  - ○ Change to SCOTTCW
- ○ Logout

Select action: Next

2. At **Step 1: Select database table and procedure type**, select **CW_IMAGES_TABLE** and **Standalone procedure** as shown in Figure 3–8. Click **Next**.

Figure 3–8 Step 1: Select Database Table and Procedure Type



*inter*Media 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.

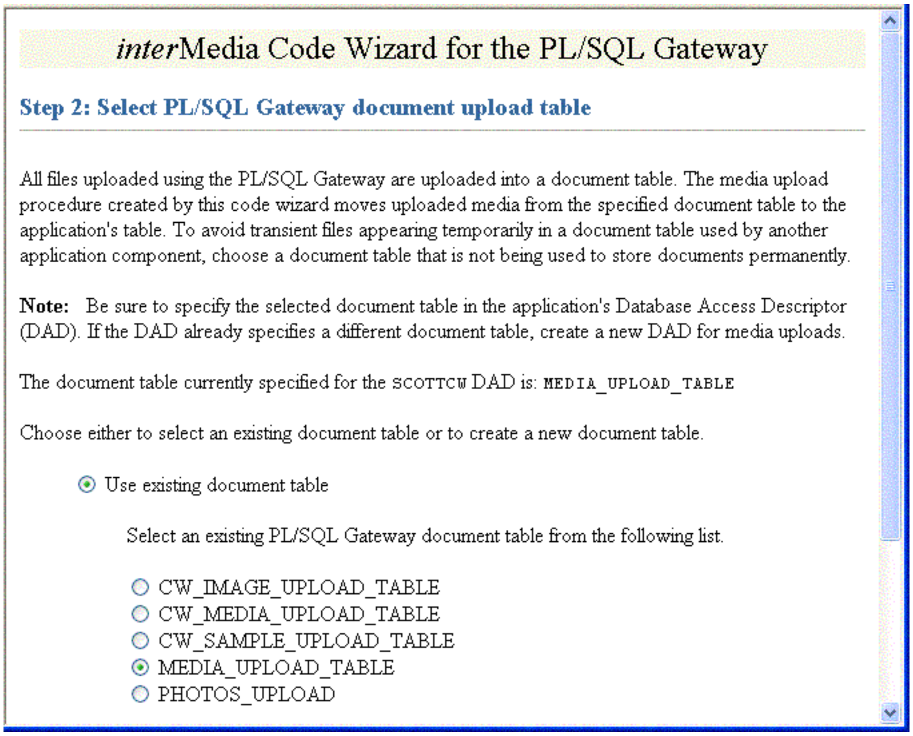
- ○ CW_AUDIO_TABLE
- [✓ CW_IMAGES_TABLE]
- ○ CW_MEDIA_TABLE
- ○ CW_VIDEO_TABLE
- ○ PHOTOS

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

3. At **Step 2: Select PL/SQL Gateway document upload table**, select **Use existing document table** and select **MEDIA_UPLOAD_TABLE** because the `SCOTTCW` DAD is configured to use this document table as shown in Figure 3–9 and Figure 3–10. Click **Next**.

Figure 3–9   Step 2: Select PL/SQL Gateway Document Upload Table (Part 1)


*inter*Media 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 `SCOTTCW` DAD is: `MEDIA_UPLOAD_TABLE`

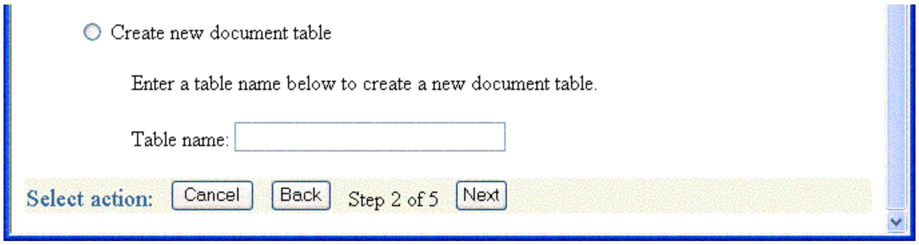
Choose either to select an existing document table or to create a new document table.

[✓ 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 3–10   Step 2: Select PL/SQL Gateway Document Upload Table (Part 2)

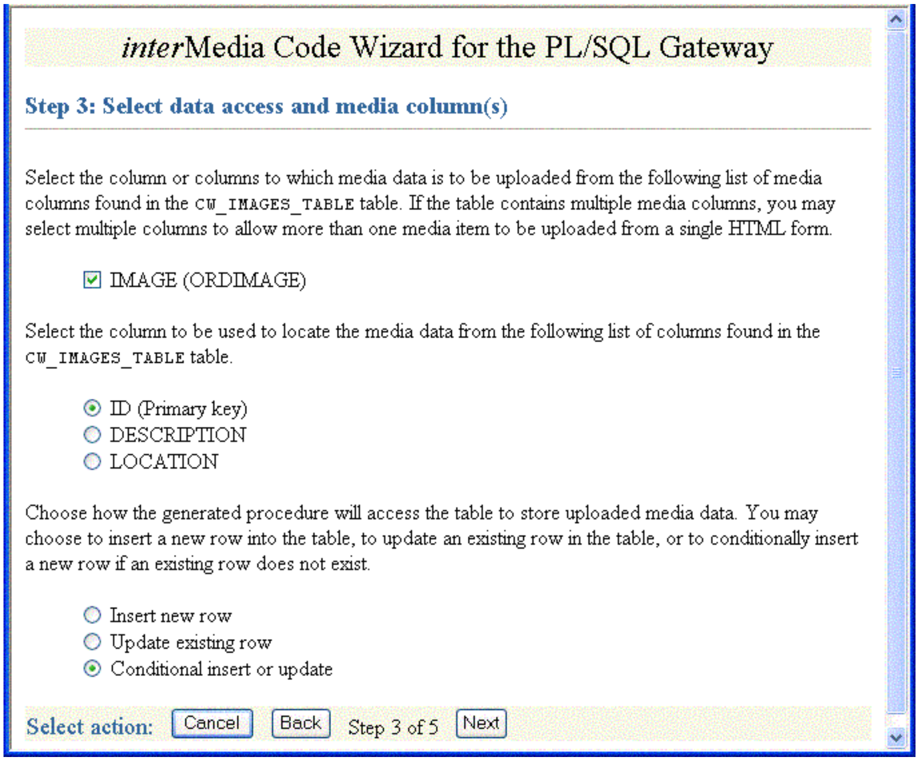

Create new document table

Enter a table name below to create a new document table.

Table name:

4. At **Step 3: Select data access and media column(s)**, ensure **IMAGE (ORDIMAGE)** is checkmarked, that **ID (Primary key)** is selected, and select **Conditional insert or update** as shown in Figure 3–11. Click **Next**.

Figure 3–11 Step 3: Select Data Access and Media Column(s)


*inter*Media 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.

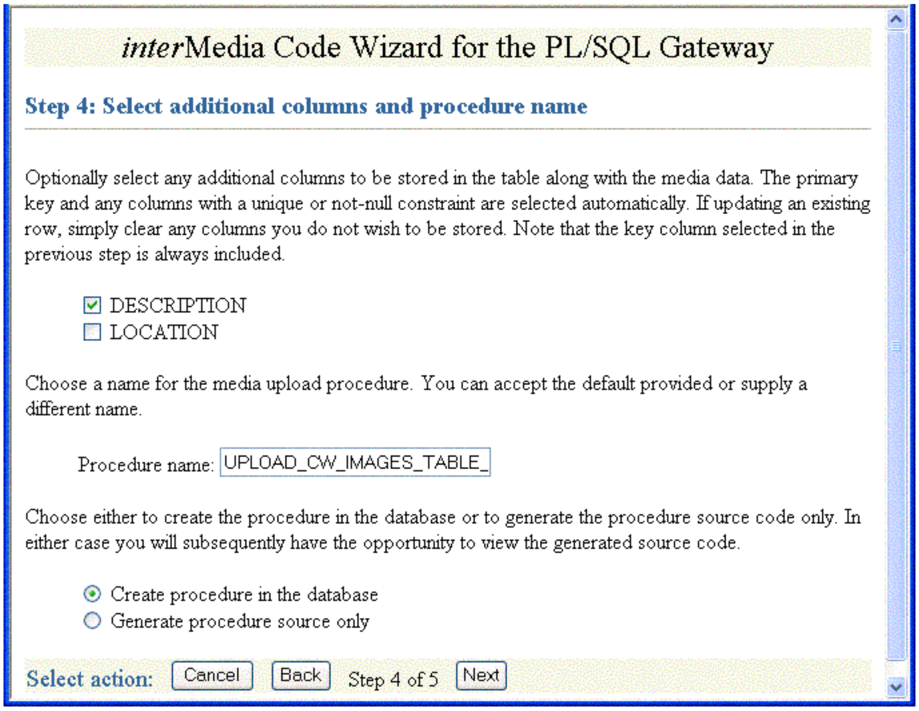
[✓ ID (Primary key)]
DESCRIPTION
LOCATION

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]

5. At **Step 4: Select additional columns and procedure name**, ensure **DESCRIPTION** is checkmarked, accept the default procedure name, **UPLOAD_CW_IMAGES_TABLE_IMAGE**, and select **Create procedure in the database** as shown in Figure 3–12. Click **Next**.

Figure 3–12 Step 4: Select Additional Columns and Procedure Name



*inter*Media 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.

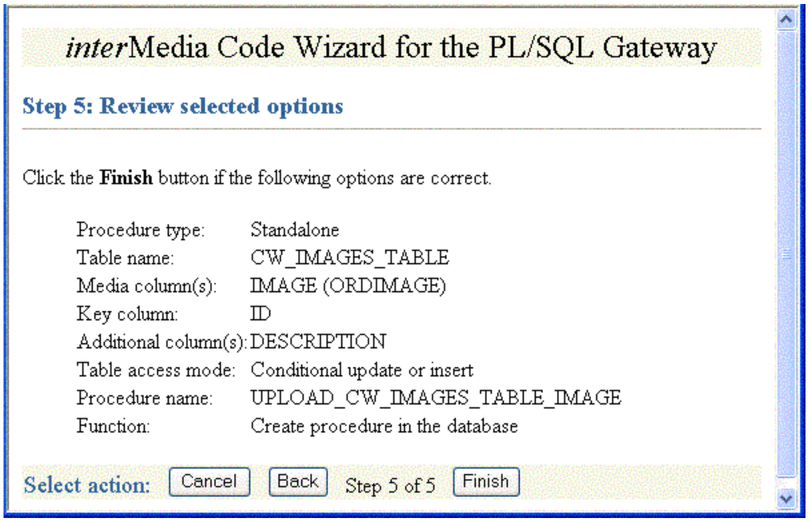
Procedure name: UPLOAD_CW_IMAGES_TABLE_

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

6. At **Step 5: Review selected options**, review the options you have selected as shown in Figure 3–13. If the options selected are correct, click **Finish**.

Figure 3–13 Step 5: Review Selected Options



*inter*Media Code Wizard for the PL/SQL Gateway

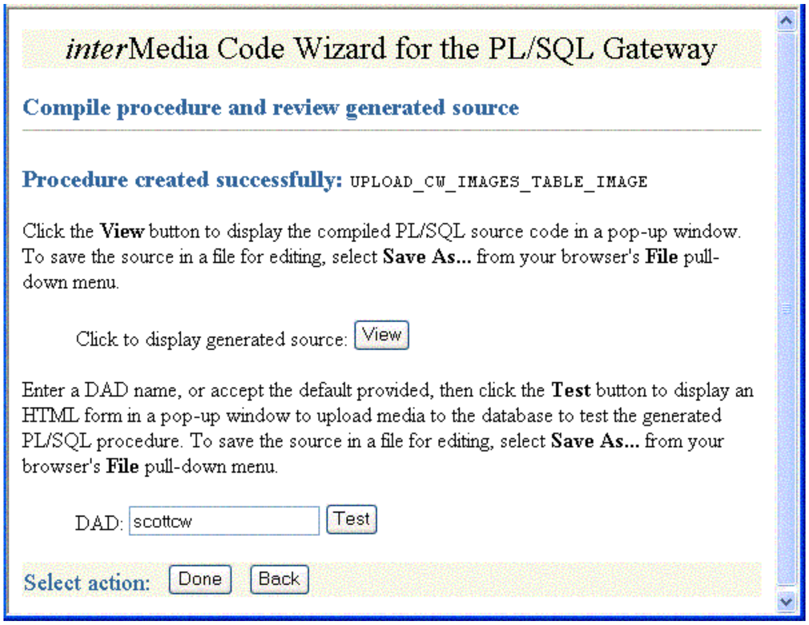
Step 5: Review selected options

Click the **Finish** button if the following options are correct.

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

7. At the **Compile procedure and review generated source** window note the message, `Procedure created successfully: UPLOAD_CW_IMAGES_TABLE_IMAGE` as shown in Figure 3–14. To review the compiled PL/SQL source code in another window, click **View** (see Step 5, substep 6g in Section 3.2.2 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 3–14 Compile Procedure and Review Generated Source



*inter*Media Code Wizard for the PL/SQL Gateway

Compile procedure and review generated source

Procedure created successfully: UPLOAD_CW_IMAGES_TABLE_IMAGE

Click the **View** button to display the compiled PL/SQL source code in a pop-up window. To save the source in a file for editing, select **Save As...** from your browser's **File** pull-down menu.

Click to display generated source: View

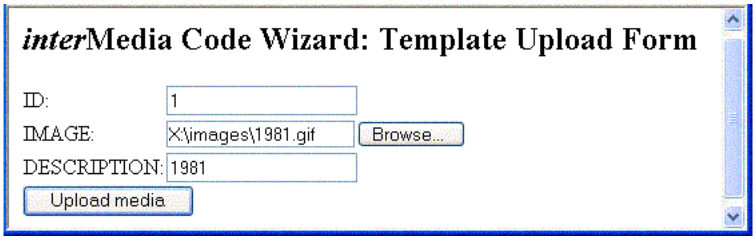
Enter a DAD name, or accept the default provided, then click the **Test** button to display an HTML form in a pop-up window to upload media to the database to test the generated PL/SQL procedure. To save the source in a file for editing, select **Save As...** from your browser's **File** pull-down menu.

DAD: scottcw Test

Select action: Done Back

8. At the ***inter*****Media Code Wizard: Template Upload Form** window, enter the value `1` in the **ID** field, 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 3–15. Click **Upload media**.

Figure 3–15 Template Upload Form



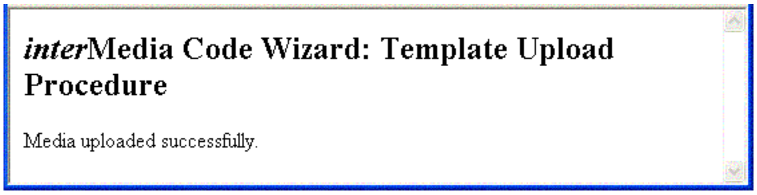
*inter*Media Code Wizard: Template Upload Form

ID:	1
IMAGE:	X:\images\1981.gif Browse...
DESCRIPTION:	1981

Upload media

9. The image is uploaded into the table row and a message is displayed, as shown in Figure 3–16.

Figure 3–16 Template Upload Procedure -- Media Uploaded Successfully Message



***inter*Media Code Wizard: Template Upload Procedure**

Media uploaded successfully.

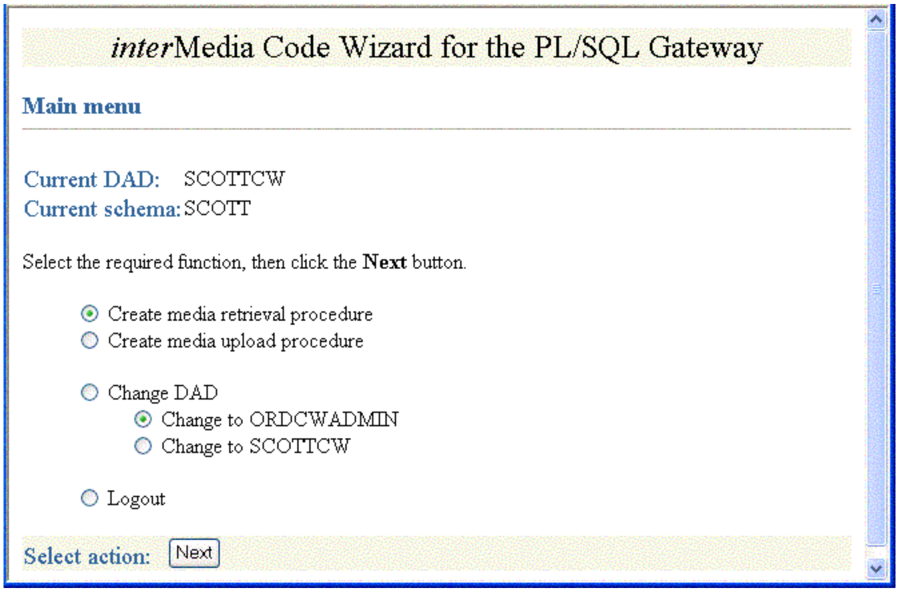
10. Return to the **Compile procedure and review generated source** window. If you are finished testing, click **Done** to return to the **Main menu**.

3.2.1.5 Creating a Media Retrieval Procedure

To create a media retrieval procedure using the *inter*Media Code Wizard for the PL/SQL Gateway, do the following steps:

1. At the **Main menu**, select **Create media retrieval procedure** as shown in Figure 3–17. Click **Next**.

Figure 3–17 Create a Media Retrieval Procedure



*inter*Media Code Wizard for the PL/SQL Gateway

Main menu

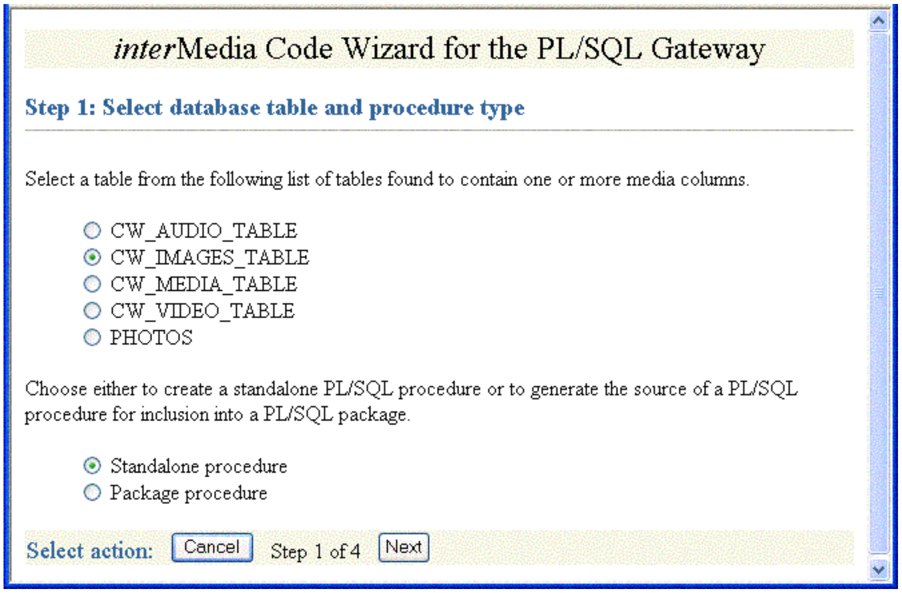
Current DAD:	SCOTTCW
Current schema:	SCOTT

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

2. At **Step 1: Select database table and procedure type**, select **CW_IMAGES_ TABLE** and select **Standalone procedure** as shown in Figure 3–18. Click **Next**.

Figure 3–18 Step 1: Select Database Table and Procedure Type



*inter*Media 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.

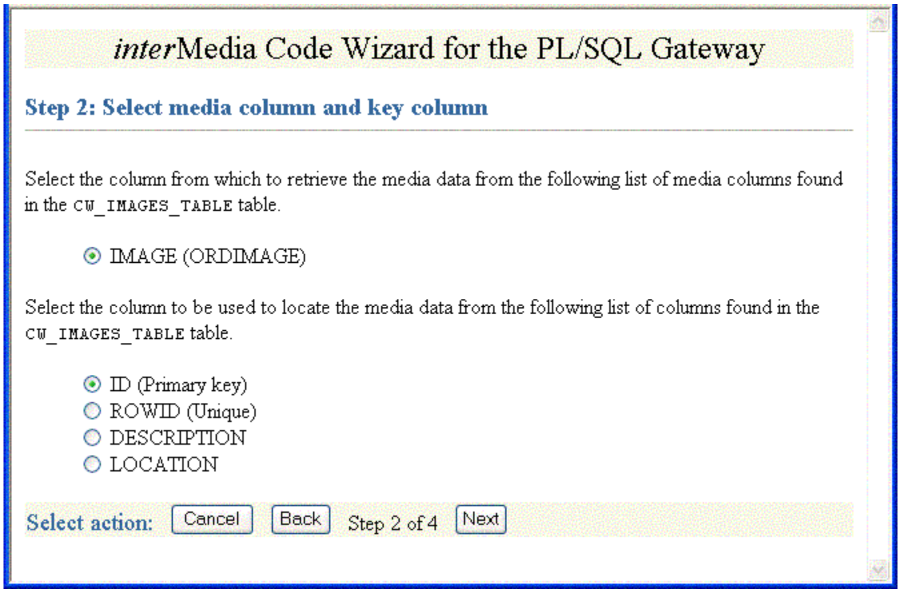
- ○ CW_AUDIO_TABLE
- ● CW_IMAGES_TABLE
- ○ CW_MEDIA_TABLE
- ○ CW_VIDEO_TABLE
- ○ PHOTOS

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

3. At **Step 2: Select media column and key column**, ensure **IMAGE (ORDIMAGE)** and **ID (Primary key)** are selected as shown in Figure 3–19. Click **Next**.

Figure 3–19 Step 2: Select Media Column and Key Column


*inter*Media 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.

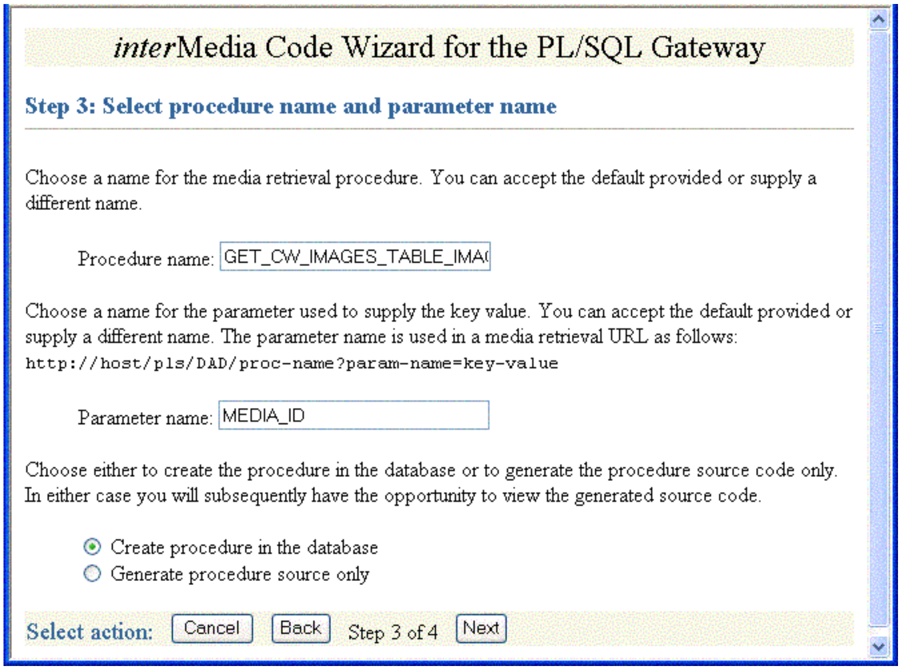
[✓ IMAGE (ORDIMAGE)]

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

4. At **Step 3: Select procedure name and parameter name**, accept the default procedure name, `GET_CW_IMAGES_TABLE_IMAGES`, accept the default parameter name, `MEDIA_ID`, and accept **Create procedure in the database** as shown in Figure 3–20. Click **Next**.

Figure 3–20 Step 3: Select Procedure Name and Parameter Name



*inter*Media 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: GET_CW_IMAGES_TABLE_IMA(

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/DAD/proc-name?param-name=key-value`

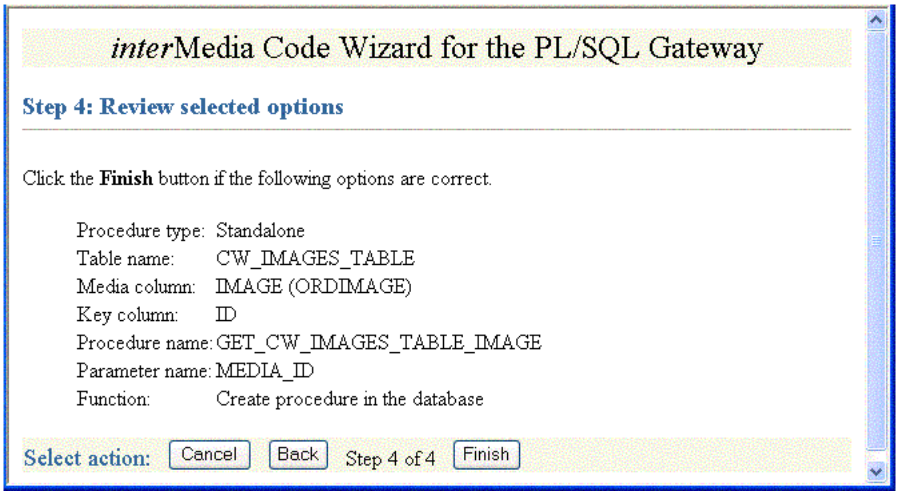
Parameter name: MEDIA_ID

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

5. At **Step 4: Review Selected Options**, review the options you have selected as shown in Figure 3–21. If the options selected are correct, click **Finish**.

Figure 3–21 Step 4: Review Selected Options



*inter*Media Code Wizard for the PL/SQL Gateway

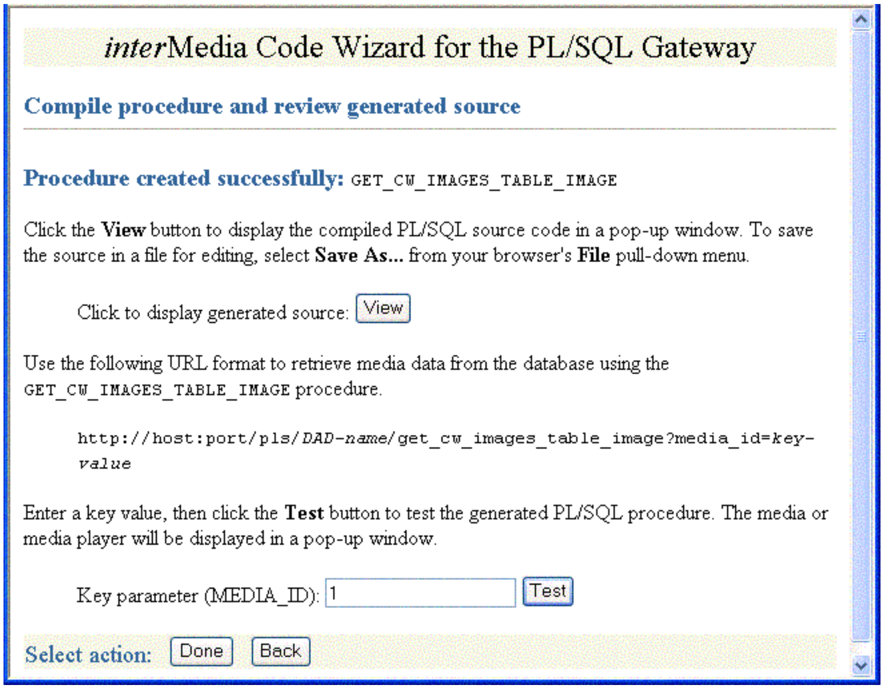
Step 4: Review selected options

Click the **Finish** button if the following options are correct.

Procedure type:	Standalone
Table name:	CW_IMAGES_TABLE
Media column:	IMAGE (ORDIMAGE)
Key column:	ID
Procedure name:	GET_CW_IMAGES_TABLE_IMAGE
Parameter name:	MEDIA_ID
Function:	Create procedure in the database

6. At the **Compile procedure and review generated source** window note the message, `Procedure created successfully: GET_CW_IMAGES_TABLE_IMAGE` as shown in Figure 3–22. To review the compiled PL/SQL source code in another window, click **View** (see Step 6, substep 5e in Section 3.2.2 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 3–23. Click **Done** to return to the **Main menu**.

Figure 3–22 Compile Procedure and Review Generated Source



*inter*Media Code Wizard for the PL/SQL Gateway

Compile procedure and review generated source

Procedure created successfully: `GET_CW_IMAGES_TABLE_IMAGE`

Click the **View** button to display the compiled PL/SQL source code in a pop-up window. To save the source in a file for editing, select **Save As...** from your browser's **File** pull-down menu.

Click to display generated source:

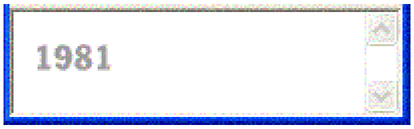
Use the following URL format to retrieve media data from the database using the `GET_CW_IMAGES_TABLE_IMAGE` procedure.

```
http://host:port/pls/DAD-name/get_cw_images_table_image?media_id=key-value
```

Enter a key value, then click the **Test** button to test the generated PL/SQL procedure. The media or media player will be displayed in a pop-up window.

Key parameter (MEDIA_ID): 1

Figure 3–23 The Displayed Image 1981



3.2.1.6 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 );
```

For more information on file upload and document tables, see *Oracle HTTP Server mod_plsql User's Guide*.

3.2.1.7 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 ORDSource object. Although the DATE data type has no support for time zones or daylight savings time, the Oracle9*i* 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 on how to set a time zone for a database. See *Oracle Database SQL Reference* for more information on 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 Oracle9*i* 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 on how to set a time zone for a database.

3.2.2 Sample Session Using Images

The following sample session uses the `SCOTT` schema to illustrate the creation of image media upload and retrieval procedures. Substitute a different schema name if you want to use a different schema.

This sample session assumes the *inter*Media Code Wizard has been installed.

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 );
```

Step 2 Create the `SCOTTCW` DAD to be used to create the procedures.

1. Enter the URL of the PL/SQL Gateway Configuration page into your browser's location bar, for example:

   ```
   http://<hostname>:<port-number>/pls/admin_/gateway.htm
   ```

2. Click **Gateway Database Access Descriptor Settings**.
3. Click **Add Default (blank configuration)**.
4. Enter the following information into the specified sections of the **Create DAD Entry Upload** form:

   ```
   Add Database Access Descriptor
     Database Access Descriptor Name:      SCOTTCW
     Schema name:                          [leave blank]
   Database Connectivity Information
     Oracle User Name:                     SCOTT
     Oracle Password:                      [leave blank]
     Oracle Connect String:                [leave blank or enter TNS name]
   Authentication Mode
     Authentication Mode:                  Basic
   Session Cookie
     Session Cookie Name:                  [leave blank]
   Package/Session Management State
     Package/Session Management Type      Stateless (Reset Package State)
   Connection Pool Parameters
     Enable Connection Pooling?            Yes
   Default (Home) Page
     Default (Home) Page:                  ORDCWPKG.MENU
   Document Access Information
     Document Table:                       CW_SAMPLE_UPLOAD_TABLE
     Document Access Path:                 [leave blank]
     Document Access Procedure:            [leave blank]
     Extensions uploaded as Long Raw:      [leave blank]
   Path Aliasing
     Path Alias:                           [leave blank]
     Path Alias Procedure:                 [leave blank]
   ```

Step 3 Authorize the use of the `SCOTTCW` 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>/pls/ordcwadmin
   ```

2. Select the DAD authorization function from the Code Wizard's **Main menu** and click **Next**. Enter the name of the demonstration DAD, `SCOTTCW`, and the user name `SCOTT`, then click **Apply**. Click **Done** when the confirmation window is displayed.

Step 4 Change DADs to the `SCOTTCW` DAD.

1. Click **Change DAD** from the Code Wizard's **Main menu**.
2. Click **Change to SCOTTCW**, if it is not already selected, then click **Next**.
3. Enter the user name `SCOTT` and password `TIGER` when prompted for user name and password, then click **OK**.

   The **Main menu** now displays the current DAD as `SCOTTCW` 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, `SCOTTCW`, 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 ***inter*****Media 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.DESCRIPTION = 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 &gt; 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.
  --
  htp.print( '&lt;html&gt;' );
  htp.print( '&lt;title&gt;interMedia Code Wizard: Template Upload
Procedure&lt;/title&gt;' );
  htp.print( '&lt;body&gt;' );
  htp.print( '&lt;h2&gt;&lt;i&gt;inter&lt;/i&gt;Media Code Wizard:
Template Upload Procedure&lt;/h2&gt;' );
  htp.print( 'Media uploaded successfully.' );
  htp.print( '&lt;/body&gt;' );
  htp.print( '&lt;/html&gt;' );
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 `htp.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;

  --
  -- 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 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 `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 *inter*Media.

3.2.3 Sample Session Using Multiple Object Columns

The following sample session uses the `SCOTT` schema to illustrate the creation of a multimedia upload (multiple *inter*Media object columns) and single media retrieval procedures. Substitute a different schema name if you want to use a different schema.

This sample session assumes the *inter*Media 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);
```

Step 2 Use the SCOTTW DAD you created in Step 2, and then, authorized the use of it in Step 3, of Section 3.2.2.

If you have not created the SCOTTW DAD and authorized the use of this DAD, perform Steps 2 and 3 in Section 3.2.2, then continue to next step that follows in this section, Step 3.

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>/pls/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 password TIGER when prompted for 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, `SCOTTCW`, 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 *inter*Media 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 ***inter*****Media 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.DESCRIPTION = 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 &gt; 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 &gt; 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 &gt; 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 &gt; 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 &gt; 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.
  --
  htp.print( '&lt;html&gt;' );
  htp.print( '&lt;title&gt;interMedia Code Wizard: Template Upload
 Procedure&lt;/title&gt;' );
  htp.print( '&lt;body&gt;' );
  htp.print( '&lt;h2&gt;&lt;i&gt;inter&lt;/i&gt;Media Code Wizard:
 Template Upload Procedure&lt;/h2&gt;' );
  htp.print( 'Media uploaded successfully.' );
  htp.print( '&lt;/body&gt;' );
  htp.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` hat 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 `htp.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 *inter*Media object type. To retrieve media data stored in other *inter*Media 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 3.2.2, Step 6 "Create and test a media retrieval", 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.

3.2.4 Known Restrictions of the Oracle *inter*Media Code Wizard

The following restrictions are known for the *inter*Media 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 *inter*Media object types are not recognized by the *inter*Media Code Wizard.

4 IMExample Java Sample Application

This chapter describes the IMExample Java sample application. This chapter assumes you have already installed, compiled, and can run this sample application. See the `readme.txt` file for requirements and instructions on how to install, compile, and run this sample application. This chapter describes how Oracle *inter*Media Java Classes is used in creating this sample application.

4.1 Overview

This sample application lets you retrieve multimedia data from the sample schema, save to a file, play, and delete from the sample schema image, audio, video, and testimonial data using the respective *inter*Media object types, OrdImage, OrdAudio, OrdVideo, and OrdDoc by product ID for rows in the `PM.ONLINE_MEDIA` table. Section 4.2 briefly describes how to compile and run the IMExample Java sample application. Section 4.3 describes the class files and shows code examples that illustrate how *inter*Media object types and methods and other Oracle objects are used.

4.2 Compiling and Running the IMExample Application

To compile the IMExample sample application, enter the following at the command line, assuming you are in the directory where the Java source files are located:

```
javac *.java
```

To run the IMExample sample application, enter the following at the command line:

```
java IMExample
```

4.3 Description of the IMExample Application

The IMExample sample application when compiled creates the following class files:

- `IMExample` -- creates the sample application frame, maintains the only connection to the database, and allows compatibility with future releases of *inter*Media.
- `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, product description, retrieves and displays the product photo, audio, video, and testimonial data within the appropriate panel, and, supports retrieving, saving, deleting, and playing the media data. 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.
- `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 for the plug-in players for the operating system detected, 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 setProperties( ) 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.
- IMJOptionPane -- extends and puts into subclasses the JOptionPane class in order to add an accessible description message to the displayed dialog box.
- 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.

IMExample Class

This class makes a call to the *inter*Media OrdMediaUtil.imCompatibilityInit( ) method (see the bolded line in the code example) to allow compatibility with future releases of *inter*Media in case a database upgrade includes evolved object types, as follows:

```
protected static void setDBConnection(OracleConnection conn) throws Exception
{
   if (s_dbConn == null)
   {
      if (conn == null)
        throw new SQLException();
      s_dbConn = conn;

      // Allow compatibility with future versions.
      OrdMediaUtil.imCompatibilityInit(s_dbConn);
   }
   else
   {
       new IMMessage(IMConstants.ERROR, "ALREADY_CONNECTED");
   }
}
```

Calling the OrdMediaUtil.imCompatibilityInit( ) method is a recommended practice to help ensure client-side applications can maintain compatibility with the current release of the *inter*Media object types (OrdAudio, OrdImage, OrdVideo, OrdDoc, and OrdSource), in the event that there is a database upgrade that includes evolved object types.

IMProductDialog Class

This class defines the following methods followed by a description of what each 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 getORAData and getORADataFactory interfaces supplied by Oracle 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 *inter*Media and other Oracle object types and methods are used. See the IMImagePanel Class, IMAudioPanel Class, IMVideoPanel Class, and IMDocPanel Class sections for code examples of the corresponding m_j*Xxx*Panel.display( ) methods, where *Xxx* represents the particular media data type, Img, Aud, Vid, or Doc.

```
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.getORAData(1, OrdImage.getORADataFactory());
      m_imgThumb = (OrdImage)rs.getORAData(2, OrdImage.getORADataFactory());
      m_aud = (OrdAudio)rs.getORAData(3, OrdAudio.getORADataFactory());
      m_vid = (OrdVideo)rs.getORAData(4, OrdVideo.getORADataFactory());
      m_doc = (OrdDoc)rs.getORAData(5, OrdDoc.getORADataFactory());
    }
    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();
  if (!m_isProdIlluFocused)
    m_jButtonRevert.setNextFocusableComponent(jImgFocus);
}

/**
 * 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);
}
```

IMImagePanel Class

This class displays the image panel, the product photo and its attributes, and the thumbnail image. What follows is a more detailed description of each of the methods that are defined and what each does:

- The display( ) method, which first calls the insertProperty( ) method, which calls the *inter*Media image object type methods getMimeType( ), getHeight( ), getWidth( ), and getContentlength( ) to get the attributes of the image to display in a table.
- 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 *inter*Media importData( ) method first for the BFILE and returns the results of calling the *inter*Media 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 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 *inter*Media 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)) &&
                !("GIFF".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()}
      };

      IMAttrTableModel tm = new IMAttrTableModel(data, m_attrColNames);

      m_jAttrTbl = new IMTable(tm);
      m_jAttrScrollPane = new JScrollPane(m_jAttrTbl);
      m_jAttrScrollPane.setPreferredSize(new Dimension(300, 84));

      if (m_isGridLayout)
        addMediaComponent(m_jAttrScrollPane, 0, 1, 0.6, 6,
            GridBagConstraints.CENTER, true);
      else
      {
        m_jAttrPane.setLayout(new GridBagLayout());
        addMediaComponent(m_jAttrPane, m_jAttrScrollPane, 0, 1);
      }
    }

    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 IMMessage(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);
      }
    }
    else
    {
      if (m_jAttrTbl == null)
      {
        m_jEmpty.setText(s_sNotSupported);
      }
      else
      {
        m_jAttrTbl = null;
        m_jAttrPane.remove(m_jAttrScrollPane);
        m_jAttrPane.setLayout(new BorderLayout());
        m_jIconPane.remove(m_jIcon);
        layoutEmpty(s_sNotSupported);
      }
    }
    m_jAttrPane.validate();
    m_jIconPane.validate();
    validate();
  }
```

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 the IMImagePanel Class section 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 *inter*Media 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);

  if (m_isGridLayout)
    addMediaComponent(m_jIcon, 3, 0, 0.4, 3, GridBagConstraints.CENTER);
  else
    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()}
      };
      IMAttrTableModel tm = new IMAttrTableModel(data, m_attrColNames);

      m_jAttrTbl = new IMTable(tm);
      m_jAttrScrollPane = new JScrollPane(m_jAttrTbl);
      m_jAttrScrollPane.setPreferredSize(new Dimension(300, 100));
      if (m_isGridLayout)
        addMediaComponent(m_jAttrScrollPane, 0, 1, 0.6, 6, GridBagConstraints.CENTER, true);
      else
      {
        m_jAttrPane.setLayout(new GridBagLayout());
        addMediaComponent(m_jAttrPane, m_jAttrScrollPane, 0, 1);
      }
    }

    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 IMMessage(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);
      }
    }
    else
    {
      if (m_jAttrTbl == null)
      {
        m_jEmpty.setText(s_sNotSupported);
      }
      else
      {
        m_jAttrTbl = null;
        m_jAttrPane.remove(m_jAttrScrollPane);
        m_jAttrPane.setLayout(new BorderLayout());

        layoutEmpty(s_sNotSupported);
      }
    }
    m_jAttrPane.validate();
    validate();
  }
```

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 the IMImagePanel Class section 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 *inter*Media 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);

    if (m_isGridLayout)
      addMediaComponent(m_jIcon, 3, 0, 0.4, 3, GridBagConstraints.CENTER);
    else
      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",  sMimeType},
        {"Duration", new Integer(m_aud.getAudioDuration()).toString()},
        {"Content Length", new Integer(m_aud.getContentLength()).toString()}
      };

      IMAttrTableModel tm = new IMAttrTableModel(data, m_attrColNames);

      m_jAttrTbl = new IMTable(tm);
      m_jAttrScrollPane = new JScrollPane(m_jAttrTbl);
      m_jAttrScrollPane.setPreferredSize(new Dimension(300, 68));

      if (m_isGridLayout)
        addMediaComponent(m_jAttrScrollPane, 0, 1, 0.6, 6,
            GridBagConstraints.CENTER, true);
      else
      {
        m_jAttrPane.setLayout(new GridBagLayout());
        addMediaComponent(m_jAttrPane, m_jAttrScrollPane, 0, 1);
      }
    }

    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 IMMessage(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);
      }
    }
    else
    {
      if (m_jAttrTbl == null)
      {
        m_jEmpty.setText(s_sNotSupported);
      }
      else
      {
        m_jAttrTbl = null;
        m_jAttrPane.remove(m_jAttrScrollPane);
        m_jAttrPane.setLayout(new BorderLayout());

        layoutEmpty(s_sNotSupported);
      }
    }
    m_jAttrPane.validate();
    validate();
  }
```

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 the IMImagePanel Class section 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 *inter*Media 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);
    if (m_isGridLayout)
      addMediaComponent(m_jIcon, 3, 0, 0.4, 3, GridBagConstraints.CENTER);
    else
      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()}
      };

      IMAttrTableModel tm = new IMAttrTableModel(data, m_attrColNames);

      m_jAttrTbl = new IMTable(tm);
      m_jAttrScrollPane = new JScrollPane(m_jAttrTbl);
      m_jAttrScrollPane.setPreferredSize(new Dimension(300, 52));

      if (m_isGridLayout)
        addMediaComponent(m_jAttrScrollPane, 0, 1, 0.6, 6,
            GridBagConstraints.CENTER, true);
      else
      {
        m_jAttrPane.setLayout(new GridBagLayout());
        addMediaComponent(m_jAttrPane, m_jAttrScrollPane, 0, 1);
      }
    }

    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 IMMessage(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);
    }
  }
  else
  {
    if (m_jAttrTbl == null)
    {
      m_jEmpty.setText(s_sNotSupported);
    }
    else
    {
      m_jAttrTbl = null;
      m_jAttrPane.remove(m_jAttrScrollPane);
      m_jAttrPane.setLayout(new BorderLayout());
      layoutEmpty(s_sNotSupported);
    }
  }
  m_jAttrPane.validate();
  validate();
}
```

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 *inter*Media 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();
      }
    }

    // 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 IMMessage(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.getORAData(1, OrdImage.getORADataFactory());
      break;
    case AUD_TYPE:
      m_aud = (OrdAudio)rs.getORAData(1, OrdAudio.getORADataFactory());
      break;
    case VID_TYPE:
      m_vid = (OrdVideo)rs.getORAData(1, OrdVideo.getORADataFactory());
      break;
    case DOC_TYPE:
      m_doc = (OrdDoc)rs.getORAData(1, OrdDoc.getORADataFactory());
      break;
    default:
      new IMMessage(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.setORAData(1, m_img);
        break;
      case AUD_TYPE:
        m_aud.loadDataFromFile(m_jFileChooser.getText());
        isFormatSupported = IMUtil.setProperties(m_aud);
        m_aud.setLocal();
        pstmt.setORAData(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.setORAData(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.setORAData(1, m_doc);

        ((IMDocPanel)m_parent).setMedia(m_doc);
        ((IMDocPanel)m_parent).refreshPanel(isFormatSupported);
        break;
      default:
        new IMMessage(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);
  }
}
```

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 *inter*Media 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.getORAData(1, OrdImage.getORADataFactory());
        rs.close();
        pstmt.close();
      }

      // Generate the thumbnail image.
      img.processCopy("maxScale=64 64, fileFormat=GIFF", imgThumb);
      imgThumb.setProperties();
      imgThumb.setLocal();

      // 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.setORAData(1, imgThumb);
      pstmt.setInt(2, iProdId);
      pstmt.execute();
      pstmt.close();

      return imgThumb;
    }
    finally
    {
      IMUtil.cleanup(rs, pstmt);
    }
  }
```

5

Content-Based Retrieval Concepts

This chapter explains, at a high level, why and how to use content-based retrieval. It covers the following topics:

- Overview and benefits of content-based retrieval (see Section 5.1)
- How content-based retrieval works, including definitions and explanations of the visual attributes (color, texture, shape, location) and why you might emphasize specific attributes in certain situations (see Section 5.2)
- Image matching using a specified comparison image, including comparing how the weights of visual attributes determine the degree of similarity between images (see Section 5.3)
- Use of indexing to improve search and retrieval performance (see Section 5.4)
- Image preparation or selection to maximize the usefulness of comparisons (see Section 5.5)

5.1 Overview and Benefits

Inexpensive image-capture and storage technologies have allowed massive collections of digital images to be created. However, as an image database grows, the difficulty of finding relevant images increases. Two general approaches to this problem have been developed. Both use metadata for image retrieval.

- Using information manually entered or included in the table design, such as titles, descriptive keywords from a limited vocabulary, and predetermined classification schemes
- Using automated image feature extraction and object recognition to classify image content -- that is, using capabilities unique to content-based retrieval

With *inter*Media, you can combine both approaches in designing a table to accommodate images: use traditional text columns to describe the semantic significance of the image (for example, that the pictured automobile won a particular award, or that its engine has six or eight cylinders), and use the ORDImageSignature type to permit content-based queries based on intrinsic attributes of the image (for example, how closely its color and shape match a picture of a specific automobile).

As an alternative to defining image-related attributes in columns separate from the image, a database designer could create a specialized composite data type that combines an ORDImage object and the appropriate text, numeric, and date attributes.

The primary benefit of using content-based retrieval is reduced time and effort required to obtain image-based information. With frequent adding and updating of images in massive databases, it is often not practical to require manual entry of all attributes that might be needed for queries, and content-based retrieval provides increased flexibility and practical value. It is also useful in providing the ability to query on attributes such as texture or shape that are difficult to represent using keywords.

Examples of database applications where content-based retrieval is useful -- where the query is semantically of the form, "find objects that look like this one" -- include:

- Trademarks, copyrights, and logos
- Art galleries and museums
- Retailing
- Fashion and fabric design
- Interior design or decorating

For example, a Web-based interface to a retail clothing catalog might allow users to search by traditional categories (such as style or price range) and also by image properties (such as color or texture). Thus, a user might ask for formal shirts in a particular price range that are off-white with pin stripes. Similarly, fashion designers could use a database with images of fabric swatches, designs, concept sketches, and finished garments to facilitate their creative processes.

5.2 How Content-Based Retrieval Works

A content-based retrieval system processes the information contained in image data and creates an abstraction of its content in terms of visual attributes. Any query operations deal solely with this abstraction rather than with the image itself. Thus, every image inserted into the database is analyzed, and a compact representation of its content is stored in a feature vector, or **signature**.

The signature for the image in Figure 5–1 is extracted by segmenting the image into regions based on color as shown in Figure 5–2. Each region has associated with it color, texture, and shape information. The signature contains this region-based information along with global color, texture, and shape information to represent these attributes for the entire image. In Figure 5–2, there are a total of 55 shapes (patches of connected pixels with similar color) in this segmented image. In addition, there is also a "background" shape, which consists of small disjoint dark patches. These tiny patches (usually having distinct colors) do not belong to any of their adjacent shapes and are all classified into a single "background" shape. This background shape is also taken into consideration for image retrieval.

Figure 5–1 Unsegmented Image

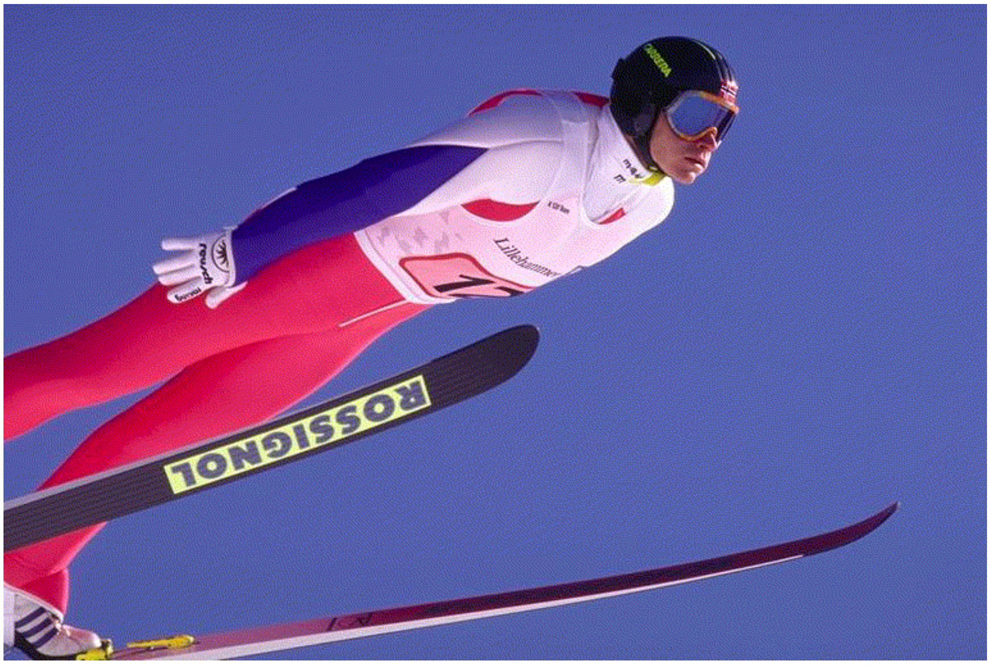
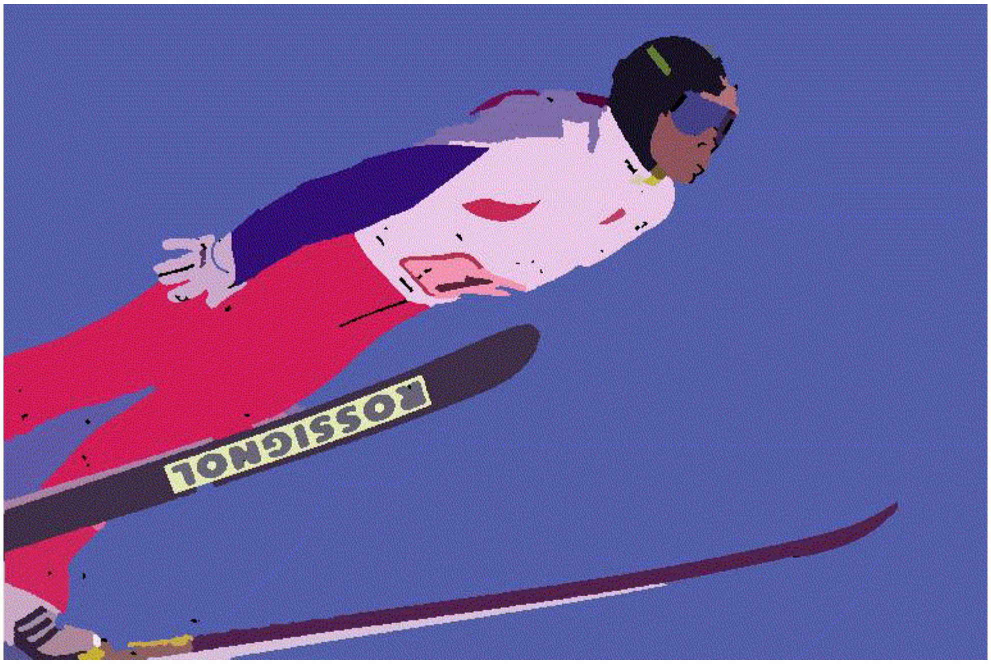


Figure 5–2 Segmented Image



Images are matched based on the color, texture, and shape attributes. The positions of these visual attributes in the image are represented by location. Location by itself is not a meaningful search parameter, but in conjunction with one of the three visual attributes represents a search where the visual attribute and the location of that visual attribute are both important.

The signature contains information about the following visual attributes:

- **Color** represents the distribution of colors within the entire image. This distribution includes the amounts of each color.
- **Texture** represents the low-level patterns and textures within the image, such as graininess or smoothness. Unlike shape, texture is very sensitive to features that appear with great frequency in the image.
- **Shape** represents the shapes that appear in the image, as determined by color-based segmentation techniques. A shape is characterized by a region of uniform color.

- **Location** represents the positions of the shape components, color, and texture components. For example, the color blue could be located in the top half of the image. A certain texture could be located in the bottom right corner of the image.

Feature data for all these visual attributes is stored in the signature, whose size typically ranges from 3000 to 4000 bytes. For better performance with large image databases, you can create an index based on the signatures of your images. See Section 5.4 for more information on indexing.

Images in the database can be retrieved by matching them with a comparison image. The comparison image can be any image inside or outside the current database, a sketch, an algorithmically generated image, and so forth.

The matching process requires that signatures be generated for the comparison image and each image to be compared with it. Images are seldom identical, and therefore matching is based on a similarity-measuring function for the visual attributes and a set of weights for each attribute. The **score** is the relative distance between two images being compared. The score for each attribute is used to determine the degree of similarity when images are compared, with a smaller distance reflecting a closer match, as explained in Section 5.3.3.

5.2.1 Color

Color reflects the distribution of colors within the entire image.

Color and location specified together reflect the color distributions *and* where they occur in the image. To illustrate the relationship between color and location, consider Figure 5–3.

Figure 5–3 Image Comparison: Color and Location

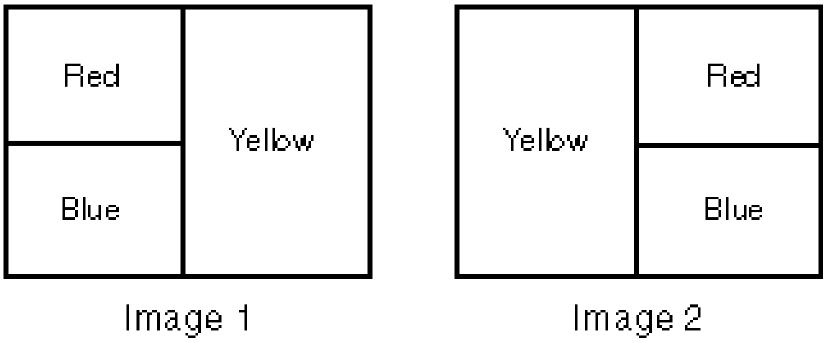


Image 1 on the left and Image 2 on the right are the same size and are filled with solid colors. In Image 1, the top left quarter (25%) is red, the bottom left quarter (25%) is blue, and the right half (50%) is yellow. In Image 2, the top right quarter (25%) is red, the bottom right quarter (25%) is blue, and the left half (50%) is yellow.

If the two images are compared first solely on color and then color and location, the following are the similarity results:

- Color: complete similarity (score = 0.0), because each color (red, blue, yellow) occupies the same percentage of the total image in each one
- Color and location: no similarity (score = 100), because there is no overlap in the placement of any of the colors between the two images

Thus, if you need to select images based on the dominant color or colors (for example, to find apartments with blue interiors), give greater relative weight to color. If you need to find images with common colors in common locations (for example, red dominant in the upper portion to find sunsets), give greater relative weight to location and color together.

Figure 5–4 shows two images very close in color. Figure 5–5 shows two images very close in both color and location.

Figure 5–4 Images Very Similar in Color

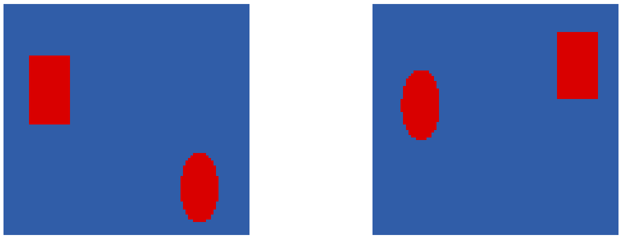
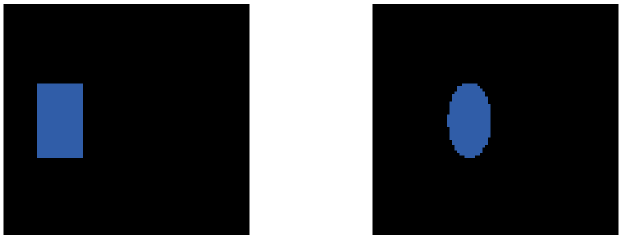


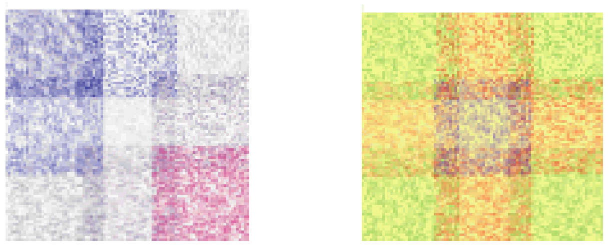
Figure 5–5 Images Very Similar in Color and Location



5.2.2 Texture

Texture reflects the texture of the entire image. Texture is most useful for full images of textures, such as catalogs of wood grains, marble, sand, or stones. These images are generally hard to categorize using keywords alone because our vocabulary for textures is limited. Texture can be used effectively alone (without color) for pure textures, but also with a little bit of color for some kinds of textures, like wood or fabrics. Figure 5–6 shows two similar fabric samples.

Figure 5–6 Fabric Images with Similar Texture



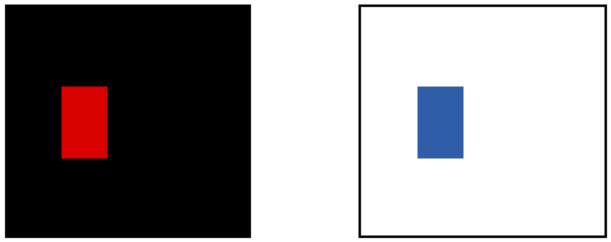
Texture and location specified together compare texture and location of the textured regions in the image.

5.2.3 Shape

Shape represents the shapes that appear in the image. Shapes are determined by identifying regions of uniform color.

Shape is useful to capture objects such as horizon lines in landscapes, rectangular shapes in buildings, and organic shapes such as trees. Shape is very useful for querying on simple shapes (like circles, polygons, or diagonal lines) especially when the query image is drawn by hand. Figure 5–7 shows two images very close in shape.

Figure 5–7 Images with Very Similar Shape



Shape and location specified together compare shapes and location of the shapes in the images.

5.3 How Matching Works

When you match images, you assign an importance measure, or weight, to each of the visual attributes, and *inter*Media calculates a similarity measure for each visual attribute.

5.3.1 Weight

Each **weight** value reflects how sensitive the matching process for a given attribute should be to the degree of similarity or dissimilarity between two images. For example, if you want color to be completely ignored in matching, assign a weight of 0.0 to color; in this case, any similarity or difference between the color of the two images is totally irrelevant in matching. On the other hand, if color is extremely important, assign it a weight greater than any of the other attributes; this will cause any similarity or dissimilarity between the two images with respect to color to contribute greatly to whether or not the two images match.

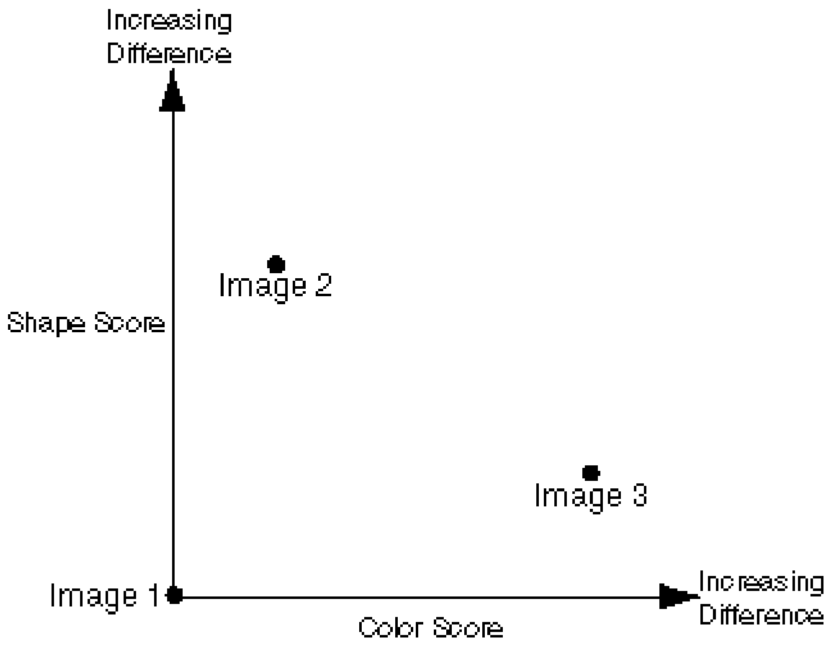
Weight values can be between 0.0 and 1.0. During processing, the values are normalized such that they total 1.0. The weight of at least one of the color, texture, or shape attributes must be set to greater than zero. See Section 5.3.3 for details of the calculation.

5.3.2 Score

The similarity measure for each visual attribute is calculated as the **score** or **distance** between the two images with respect to that attribute. The score can range from 0.00 (no difference) to 100.0 (maximum possible difference). Thus, the more similar two images are with respect to a visual attribute, the *smaller* the score will be for that attribute.

As an example of how distance is determined, assume that the dots in Figure 5–8 represent scores for three images with respect to two visual attributes, such as color and shape, plotted along the x-axis and y-axis of a graph.

Figure 5–8 Score and Distance Relationship



For matching, assume Image 1 is the comparison image, and Image 2 and Image 3 are each being compared with Image 1. With respect to the color attribute plotted on the x-axis, the distance between Image 1 and Image 2 is relatively small (for example, 15), whereas the distance between Image 1 and Image 3 is much greater (for example, 75). If the color attribute is given more weight, then the fact that the two distance values differ by a great deal will probably be very important in determining whether or not Image 2 and Image 3 match Image 1. However, if color is minimized and the shape attribute is emphasized instead, then Image 3 will match Image 1 better than Image 2 matches Image 1.

5.3.3 Similarity Calculation

In Section 5.3.2, Figure 5–8 showed a graph of only two of the attributes that *inter*Media can consider. In reality, when images are matched, the degree of similarity depends on a weighted sum reflecting the weight and distance of all three of the visual attributes in conjunction with location of the comparison image and the test image.

For example, assume that for the comparison image (Image 1) and one of the images being tested for matching (Image 2), Table 5–1 lists the relative distances between the two images for each attribute. Note that you would never see these individual numbers unless you computed three separate scores, each time highlighting one attribute and setting the others to zero. For simplicity, the three attributes are not considered in conjunction with location in this example.

Table 5–1 Distances for Visual Attributes Between Image1 and Image2

Visual Attribute	Distance
Color	15
Texture	5
Shape	50

In this example, the two images are most similar with respect to texture (distance = 5) and most different with respect to shape (distance = 50), as shown in Table 5–1.

Assume that for the matching process, the following weights have been assigned to each visual attribute:

- Color = 0.7
- Texture = 0.2
- Shape = 0.1

The weights are supplied in the range of 0.0 to 1.0. Within this range, a weight of 1 indicates the strongest emphasis, and a weight of 0 means the attribute should be ignored. The values you supply are automatically normalized such that the weights total 1.0, still maintaining the ratios you have supplied. In this example, the weights were specified such that normalization was not necessary.

The following formula is used to calculate the weighted sum of the distances, which is used to determine the degree of similarity between two images:

```
weighted_sum = color_weight    * color_distance +
               texture_weight  * texture_distance +
               shape_weight    * shape_distance+
```

The degree of similarity between two images in this case is computed as:

0.7*c_distance + 0.2*tex_distance + 0.1*shape_distance

Using the supplied values, this becomes:

(0.7*15 + 0.2*5 + 0.1*50) = (10.5 + 1.0 + 5.0) = 16.5

To illustrate the effect of different weights in this case, assume that the weights for color and shape were reversed. In this case, the degree of similarity between two images is computed as:

0.1*c_distance +0.2*tex_distance + 0.7*shape_distance

That is:

(0.1*15 + 0.2*5 + 0.7*50) = (1.5 + 1.0 + 35.0) = 37.5

In this second case, the images are considered to be less similar than in the first case, because the overall score (37.5) is greater than in the first case (16.5). Whether or not the two images are considered matching depends on the threshold value (explained in Section 5.3.4). If the weighted sum is less than or equal to the threshold, the images match; if the weighted sum is greater than the threshold, the images do not match.

In these two cases, the *correct* weight assignments depend on what you are looking for in the images. If color is extremely important, then the first set of weights is a better choice than the second set of weights, because the first set of weights grants greater significance to the disparity between these two specific images with respect to color. The two images differ greatly in shape (50) but that difference contributes less to the final score because the weight assigned to the attribute shape is low. With the second set of weights, the images have a higher score when shape is assigned a higher weight and the images are less similar with respect to shape than with respect to color.

5.3.4 Threshold Value

When you match images, you assign a **threshold** value. If the weighted sum of the distances for the visual attributes is less than or equal to the threshold, the images match; if the weighted sum is greater than the threshold, the images do not match.

Using the examples in Section 5.3.3, if you assign a threshold of 20, the images do not match when the weighted sum is 37.5, but they do match when the weighted sum is 16.5. If the threshold is 10, the images do not match in either case; and if the threshold is 37.5 or greater, the images match in both cases.

The following example shows a cursor (getphotos) that selects the `product_id` and `product_photo` columns from the `online_media` table where the threshold value is 20 for comparing photographs with a comparison image:

```
CURSOR getphotos IS
 SELECT product_id, product_photo FROM online_media WHERE
   ORDSYS.IMGSimilar(photo_sig, comparison_sig, 'color="0.4",
   texture="0.10", shape="0.3", location="0.2"', 20)=1;
```

Before the cursor executes, the generateSignature( ) method must be called to compute the signature of the comparison image (comparison_sig), and to compute signatures for each image in the table. See *Oracle interMedia Reference* for a description of all the operators, including IMGSimilar and IMGScore.

The number of matches returned generally increases as the threshold increases. Setting the threshold to 100 would return all images as matches. Such a result, of course, defeats the purpose of content-based retrieval. If your images are all very similar, you may find that even a threshold of 50 returns too many (or all) images as matches. Through trial and error, adjust the threshold to an appropriate value for your application.

You will probably want to experiment with different weights for the visual attributes and different threshold values, to see which combinations retrieve the kinds and approximate number of matches you want.

5.4 Using an Index to Compare Signatures

A domain index, or extensible index, is an approach for supporting complex data objects. Oracle Database and *inter*Media cooperate to define, build, and maintain an index for image data. This index is of type ORDImageIndex. Once it is created, the index automatically updates itself every time an image is inserted, updated, or removed from the database table. The index is created, managed, and accessed by the index type routines.

For better performance with large image databases, you should always create and use an index for searching through the image signatures. The default search model compares the signature of the query image to the signatures of all images stored in the database. This works well for simple queries against a few images such as, "Does this picture of an automobile match the image stored with the client's insurance records?" However, if you want to compare that image with thousands or millions of images to determine if the images match, then a linear search through the database would be impractical. In this case, an index based on the image signatures would greatly improve performance.

Assume you are using the `online_media` table from the `Product Media` schema.

Process each image using the generateSignature( ) method to generate the signatures.

```
DECLARE
 t_image   ORDSYS.ORDImage;
 image_sig ORDSYS.ORDImageSignature;
BEGIN
 SELECT p.product_photo, p.product_photo_signature INTO t_image, image_sig
  FROM pm.online_media p
  WHERE p.product_id = 1910 FOR UPDATE;
 -- Generate a signature:
 image_sig.generateSignature(t_image);
 UPDATE pm.online_media p SET p.product_photo_signature = image_sig
 WHERE    product_id = 1910;
 COMMIT;
END;
/
```

Note: Performance is greatly improved by loading the data tables prior to creating the index.

Once the signatures are created, the following command creates an index on this table, based on the data in the `product_photo_signature` column:

```
CREATE INDEX idx1 ON online_media(product_photo_signature) INDEXTYPE IS
ORDSYS.ORDIMAGEINDEX
    PARAMETERS ('ORDImage_Filter_Tablespace = <name>,ORDImage_Index_Tablespace = <name>');
```

The index name is limited to 24 or fewer characters. As with any Oracle table, do not use pound signs (#) or dollar signs ($) in the name. Also as usual, the tablespaces must be created before creating the index.

Note: The standard Oracle restriction is 30 characters for table or index names. However, *inter*Media requires an extra 6 characters for internal processing of the domain index.

The index data resides in two tablespaces, which must be created first. The first contains the actual index data, and the second is an internal index created on that data.

The following recommendations are good starting points for further index tuning:

- ORDIMAGE_FILTER_TABLESPACE -- Each signature requires approximately 350 bytes in this tablespace. The tablespace should be at least 350 times the number of signatures in the table.
- ORDIMAGE_INDEX_TABLESPACE -- The size of the tablespace should be 100 times the size of the initial and final extents specified. For example, if an extent is 10 KB, the tablespace size should be 1 MB. The initial and final extents should be equal to each other. The size of the tablespace should also be approximately equal to the size of ORDIMAGE_FILTER_TABLESPACE.
- Typically, it will be much faster if you create the index after the images are loaded into the database and signatures have been generated for them.
- When importing a large number of images, you should postpone index creation until after the import operation completes. Do this by specifying the following parameters to the IMPORT statement: INDEXES=N and INDEXNAME=<filename>. See *Oracle Database Utilities* for details.
- Rollback segments of an appropriate size are required. The size depends on the size of your transactions, such as, how many signatures are indexed at one time.
- Analyze the new index.

As with other Oracle indexes, you should analyze the new index as follows:

```
ANALYZE INDEX idx1 COMPUTE STATISTICS;
```

Two operators, IMGSimilar and IMGScore, support queries using the index. The operators automatically use the index if it is present. See *Oracle interMedia Reference* for syntax information and examples.

Queries for indexed and nonindexed comparisons are identical. The Oracle optimizer uses the domain index if it determines that the first argument passed to the IMGSimilar operator is a domain-indexed column. Otherwise, the optimizer invokes a functional implementation of the operator that compares the query signature with the stored signatures, one row at a time.

See *Oracle interMedia Reference* for examples of retrieving similar images. As in the example, be sure to specify the query signature as the second parameter.

5.5 Preparing or Selecting Images for Useful Matching

The human mind is infinitely smarter than a computer in matching images. If we are near a street and want to identify all red automobiles, we can easily do so because our minds rapidly adjust for the following factors:

- Whether the automobile is stopped or moving
- The distinction among red automobiles, red motorcycles, and red trailers
- The absolute size of the automobile, as well as its relative size in our field of vision (because of its distance from us)
- The location of the automobile in our field of vision (center, left, right, top, bottom)
- The direction in which the automobile is pointing or traveling (left or right, toward us, or away from us)

However, for a computer to find red automobiles (retrieving all red automobiles and no or very few images that are not red or not automobiles), it is helpful if all the automobile images have the automobile occupy almost the entire image, have no extraneous elements (people, plants, decorations, and so on), and have the automobiles pointing in the same direction. In this case, a match emphasizing color and shape would produce useful results. However, if the pictures show automobiles in different locations, with different relative sizes in the image, pointing in different directions, and with different backgrounds, it will be difficult to perform content-based retrieval with these images.

The following are some suggestions for selecting images or preparing images for comparison. The list is not exhaustive, but the basic principle to keep in mind is this: Know what you are looking for, and use common sense. If possible, crop and edit images in accordance with the following suggestions before performing content-based retrieval:

- Have what you expect to be looking for occupy almost all the image space, or at least occupy the same size and position on each image. For example, if you want to find all the red automobiles, each automobile image should show only the automobile and should have the automobile in approximately the same position within the overall image.
- Minimize any extraneous elements that might prevent desired matches or cause unwanted matches. For example, if you want to match red automobiles and if each automobile has a person standing in front of it, the color, shape, and position of the person (skin and clothing) will cause color and shape similarities to be detected, and might reduce the importance of color and shape similarities

between automobiles (because part of the automobile is behind the person and thus not visible). If you know that your images vary in this way, experiment with different thresholds and different weights for the various visual attributes until you find a combination that provides the best result set for your needs.

- During signature generation, images are temporarily scaled to a common size such that the resulting signatures are based on a common frame of reference. If you crop a section of an image, and then compare that piece back to the original, *inter*Media will likely find that the images are less similar than you would expect.

Note: *inter*Media has a fuzzy search engine, and is not designed to recognize objects. For example, *inter*Media cannot find a specific automobile in a parking lot. However, if you crop an individual automobile from a picture of a parking lot, you can then compare the automobile to known automobile images.

- When there are several objects in the image, *inter*Media matches them best when:
  - The colors in the image are distinct from each other. For example, an image of green and red as opposed to an image of dark green and light green.
  - The color in adjacent objects in the image contrast with each other.
  - The image consists of a few, simple shapes.

6 Custom DataSource and DataSink for JMF Versions 2.0 and 2.1

Sun Microsystems provides a Java Media Framework (JMF) API that enables audio, video and other time-based media to be added to Java applications and applets.

The Oracle *inter*Media Custom DataSource and DataSink feature is an extension to JMF versions 2.0 and 2.1. This feature allows a JMF application to upload time-based media data into or retrieve media data from a database that contains Oracle *inter*Media ("*inter*Media") video or audio objects. A protocol is defined that permits media data stored in *inter*Media video or audio objects in the database to be accessed through a URL that contains information necessary to select the audio or video data from the database.

6.1 Installing and Registering Custom DataSource and DataSink

To install Custom DataSource and DataSink, save the `ordjmf.jar` file to your hard disk. By default, the file will be stored in the following location:

`<ORACLE_HOME>\ord\jlib`

Then, enter the path name for the `ordjmf.jar` file into your CLASSPATH environment variable.

To register Custom DataSource and DataSink, select one of the methods described in the following subsections.

6.1.1 Registration Method 1

To register Custom DataSource and DataSink with Registration Method 1 follow these steps:

1. Start the `JMFRegistry.bat` file in your `%JMF_HOME%/bin` directory.
2. Add `oracle.ord` to both the **protocol Prefix List** and **Content Prefix List** on the **PackageManager** tab. Then, click **commit** for both lists.
3. Close the JMFRegistry.

6.1.2 Registration Method 2

To register Custom DataSource and DataSink with Registration Method 2:

Add `oracle.ord` to the **protocol Prefix List** and **Content Prefix List** in your application program through the JMF PackageManager.

For more information about JMF, see the JMF API documentation and specification available at the Sun Microsystems Web site at

`http://www.java.sun.com/jmf`

6.2 Using Custom DataSource and DataSink

Using Custom DataSource and DataSink includes the following procedures:

- Defining the `ordjmf.properties` file.
- Uploading media data into Oracle Database through Custom DataSink.
- Retrieving media data from Oracle Database through Custom DataSource.

You can access media data in the database through Sun Microsystems JMStudio or a JMF application.

The following subsections describe these procedures.

6.2.1 Defining the Property File

The property file, `ordjmf.properties`, is used to define the database connection and the PL/SQL procedures to retrieve or upload media data to the database. This file can be located in any directory, but the directory must be specified in your CLASSPATH variable. The `ordjmf.properties` file defines the following name-value pairs:

```
user=<database user>
password=<password>
url=<jdbc connection string>
<mediaSource1>=<procedure1>
<mediaSource2>=<procedure2>
```

```
<mediaSink1>=<procedure3>
...
```

The value `<jdbc connection string>` is used to set up a Java Database Connectivity (JDBC) connection with the database. The JDBC connection can be either an Oracle JDBC Thin Driver or an Oracle JDBC OCI Driver connection string.

For the Thin driver, the connection string should be similar to the following:

```
jdbc:oracle:thin:@<host>:<port>:<sid>
```

For the OCI driver, the connection string should be similar to the following:

```
jdbc:oracle.oci:@MyHostString
```

where `MyHostString` is a TNSNAMES entry in the file `tnsnames.ora` on the client computer from which you are connecting.

Multiple media data sources and data sinks can be defined in the `ordjmf.properties` file. Each source or sink is defined as a name and value pair. The name is used in the MediaLocator string (see Section 6.2.2 and Section 6.2.3). The value must be the name of an existing PL/SQL procedure that supports uploading or retrieving media data in the database.

6.2.2 Uploading Media Data

The format of the MediaLocator upload string is one continuous string, similar to the following:

```
im://upload/<sinkName>/<arg1>:<arg2>:...:<argn>
```

The parameter `upload` is the keyword that is required to indicate an upload operation of the media data.

The parameter `<sinkName>` is the name defined in the `ordjmf.properties` file. It refers to a predefined PL/SQL procedure that retrieves the media in a BLOB in a SQL SELECT ... FOR UPDATE statement. The first argument in the PL/SQL procedure must be an output parameter of BLOB data type. The second argument must be an input parameter of VARCHAR2 data type to represent the MIME type of the uploading media. The predefined PL/SQL procedure can use the input parameter of the media MIME type in any way necessary.

The following example shows that for a table named `videostore` with columns (OrdVideo `myvideo`, Integer `videoid`), the PL/SQL procedure could be defined as follows:

```
CREATE OR REPLACE PROCEDURE  getVideoForUpdate(
    myVideoLob   OUT BLOB,
    myMimetype   IN  VARCHAR2,
    myid         IN  INTEGER)
AS
    myvid ordsys.ordvideo;
BEGIN
    -- Update the MIME type first
    SELECT t.myvideo INTO myvid from videostore t
    WHERE videoid=myid FOR UPDATE;
    myvid.setMimeType(myMimetype);
    myvid.setLocal();
    myvid.setUpdateTime(SYSDATE);
    UPDATE videostore SET myvideo=myvid WHERE videoid=myid;
    -- Get the BLOB for uploading
    SELECT t.myvideo.getContent() INTO myVideoLob
    FROM videostore t
    WHERE videoid=myid FOR UPDATE;
END;
```

An example of the use of the MediaLocator upload string through the Custom DataSink is as follows:

`im://upload/sink1/1`

where `sink1` is the mediaSink name, and is defined in the `ordjmf.properties` file as `sink1=getVideoForUpdate`. Through the use of this URL, the JMF application can upload and update the media into the BLOB stored in the `videostore` table where the value of videoid is equal to 1.

The parameters `<arg1>, <arg2>, ..., <argn>` are the input parameters to the PL/SQL procedure, which can be used to locate the media data in the database. In the previous example, there is only one input parameter in the PL/SQL procedure getVideoForUpdate. Thus, the `<arg1>` parameter corresponds to the getVideoForUpdate input parameter `myid`, and the value of `<arg1>` is equal to `1`.

6.2.3 Retrieving Media Data

The format of the MediaLocator retrieval string is one continuous string, similar to the following:

`im://retrieval/<sourceName>/<arg1>:<arg2>:...:<argn>`

The parameter `retrieval` is the keyword that is required to indicate a retrieve operation of the media data.

The parameter <sourceName> is the name defined in the ordjmf.properties file. It refers to a predefined PL/SQL procedure that retrieves the media in a BLOB as well as the MIME type of the media. The first argument in the PL/SQL procedure must be an output parameter of BLOB data type. The second argument must be an output parameter of VARCHAR2 data type to represent the MIME type of the media.

The following example shows that, for a table named videostore with columns (OrdVideo myvideo, Integer videoid), the PL/SQL procedure could be defined as follows:

```
CREATE OR REPLACE PROCEDURE  getVideoForRetrieval(
           myVideoLob    OUT BLOB,
           myMimeType    OUT VARCHAR2,
           myid          IN  INTEGER)
       AS
       BEGIN
           SELECT t.myvideo.getContent(), t.myvideo.getMimeType()
           INTO myVideoLob, myMimeType
           FROM videostore t
           WHERE videoid=myid;
       END;
```

An example of the use of the MediaLocator retrieval string through the Custom DataSource feature is as follows:

```
im://retrieval/source2/1
```

where source2 is the data source name and is defined in the ordjmf.properties file as source2=getVideoForRetrieval. Through the use of this URL, the JMF application can retrieve media from the BLOB stored in the videostore table where the value of videoid is equal to 1.

The parameters <arg1>, <arg2>, ..., <argn> are input parameters to the PL/SQL procedure, which can be used to locate the media data in the database. In the previous example, there is only one input parameter in the PL/SQL procedure getVideoForRetrieval. Thus, the <arg1> parameter corresponds to the getVideoForRetrieval input parameter myid and the value of <arg1> is equal to 1.

6.2.4 Accessing Media Data Through JMStudio

To access media data in the database through Sun Microsystems JMStudio, follow these steps:

1. Start JMStudio.

2. Select **File**, then select **Open URL:**. In the **Open URL** dialog box, enter the MediaLocator string that points to the data in the database, then click **OK**.
3. JMStudio should play the media data stored in the database.

6.2.5 Accessing Media Data Through a JMF Application

To access media data in the database through a JMF application, use the MediaLocator strings defined in Section 6.2.2 and Section 6.2.3 as the input or output URL. This will enable the JMF application to access the media data in the database through Custom DataSource and DataSink for JMF 2.0 and 2.1.

For examples of JMF applications, refer to the JMF sample programs available at the Sun Microsystems Web site at

```
http://java.sun.com/jmf
```

7 Extending Oracle *inter*Media

Oracle *inter*Media 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 on 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.

Any method invoked from a source plug-in call has the first argument as obj (ORDSource) and the second argument as ctx (RAW).

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
processCommand	Not supported - raises exception: METHOD_NOT_SUPPORTED
import	Supported
import	Supported
importFrom	Supported
importFrom	Supported
export	Supported
getContentLength	Supported
getSourceAddress	Supported
open	Supported
close	Supported
trim	Not supported - raises exception: METHOD_NOT_SUPPORTED
read	Supported
write	Not supported - raises exception: METHOD_NOT_SUPPORTED

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  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_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
processCommand	Not supported - raises exception: METHOD_NOT_SUPPORTED
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 *inter*Media to Support a New Data Source

Extending *inter*Media 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,
                  dlob 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.

See Section A.1 and Section A.4 for more information on installing your own format plug-in and running the sample scripts provided.

See Section 7.2.1.2, Section 7.2.2.2, and Section 7.2.3.2 for examples and for more information on extending the supported audio and video, or other heterogeneous media data formats. See *Oracle interMedia Reference* for more examples.

7.2.1 Supporting Other ORDAudio Data Formats

Section 7.2.1.1, Section 7.2.1.2, and Section 7.2.1.3 describe support for other ORDAudio data formats.

7.2.1.1 Packages or PL/SQL Plug-ins

This section presents reference information on the packages or PL/SQL plug-ins provided for the ORDAudio type. Table 7–3 describes the PL/SQL plug-in packages provided in the `ORDPLUGINS` schema.

Table 7–3 ORDAudio PL/SQL Plug-ins Provided in the ORDPLUGINS Schema

PL/SQL Plug-in Packages	Audio Format	MIME Type
`ORDPLUGINS.ORDX_DEFAULT_AUDIO`	<format>	Dependent on file format
`ORDPLUGINS.ORDX_AUFF_AUDIO`	AUFF	audio/basic
`ORDPLUGINS.ORDX_AIFF_AUDIO`	AIFF	audio/x-aiff
`ORDPLUGINS.ORDX_AIFC_AUDIO`	AIFC	audio/x-aiff
`ORDPLUGINS.ORDX_WAVE_AUDIO`	WAVE	audio/x-wave
`ORDPLUGINS.ORDX_MPGA_AUDIO`	MPGA	audio/mpeg
`ORDPLUGINS.ORDX_ASF_AUDIO`	ASF	audio/x-ms-wma
`ORDPLUGINS.ORDX_MP4_AUDIO`	MP4	application/mpeg4

Section 7.2.1.2 describes the `ORDPLUGINS.ORDX_DEFAULT_AUDIO` package, the methods supported, and the level of support. Note that the methods supported and the level of support for the other PL/SQL plug-in packages described in Table 7–3 are identical for all plug-in packages, therefore, refer to Section 7.2.1.2.

7.2.1.2 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.

Note: The `ORDPLUGINS.ORDX_DEFAULT_AUDIO` package specification includes a number of methods deprecated beginning with release 8.1.6, which have been removed in the following listing. Do not implement these deprecated methods; they are there only for backward compatibility.

```
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 VARHAR2,
                        result    OUT RAW)
         RETURN RAW;

END;
/
```

Table 7–4 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 7–4 Methods Supported in the ORDPLUGINS.ORDX_DEFAULT_AUDIO 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.
checkProperties	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 NULL, raise an ORDSYS.ORDSourceExceptions.EMPTY_SOURCE exception; if the source is a BFILE, extract the attributes from the BFILE and compare them with the attribute values of the object; if the source is neither local nor a BFILE, 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.

Table 7–4 (Cont.) Methods Supported in the ORDPLUGINS.ORDX_DEFAULT_AUDIO Package

Name of Method	Level of Support
getAttribute	Not supported - raises exceptions: METHOD_NOT_SUPPORTED and AUDIO_PLUGIN_EXCEPTION.
getAllAttributes	Supported; if the source is local, get the attributes and return them, but if the source is `NULL`, raise an ORDSYS.ORDSourceExceptions.EMPTY_SOURCE exception; otherwise, if the source is external, raise an ORDSYS.ORDAudioExceptions.LOCAL_DATA_SOURCE_REQUIRED exception.
processCommand	Not supported - raises exceptions: METHOD_NOT_SUPPORTED and AUDIO_PLUGIN_EXCEPTION.

7.2.1.3 Extending *inter*Media to Support a New Audio Data Format

Extending *inter*Media 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 7.2.1.2 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.

   See Section A.1 for more information on installing your own format plug-in and running the sample scripts provided. See the `fplugins.sql` and `fpluginb.sql` files that are installed in the `$ORACLE_HOME/ord/aud/demo/` directory. These are demonstration (demo) plug-ins that you can use as a guideline to write any format plug-in that you want to support. See the `auddemo.sql` file in this same directory to learn how to install your own format plug-in.

4. Grant EXECUTE privileges on your new plug-in, for example, `ORDX_MY_AUDIO.SQL` 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".

See Section A.1 for more information on installing your own audio format plug-in and running the sample scripts provided.

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, Section 7.2.2.2, and Section 7.2.2.3 describe support for other ORDDoc data formats.

7.2.2.1 Packages or PL/SQL Plug-ins

This section presents reference information on the packages or PL/SQL plug-ins provided for the ORDDoc type. Table 7–5 describes the PL/SQL plug-in packages provided in the ORDPLUGINS schema.

Table 7–5 ORDDoc PL/SQL Plug-ins Provided in the ORDPLUGINS Schema

PL/SQL Plug-in Packages	Media Format	MIME Type
ORDPLUGINS.ORDX_DEFAULT_DOC	<format>	Dependent on file format

Section 7.2.2.2 describes the ORDPLUGINS.ORDX_DEFAULT_DOC package, the methods supported, and the level of support. The method supported and the level of support for the PL/SQL plug-in package is described in Table 7–6.

7.2.2.2 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. 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_DOC
authid current_user
AS

PROCEDURE setProperties(ctx IN OUT RAW,
                        obj IN OUT NOCOPY ORDSYS.ORDDoc,
                        setComments IN NUMBER := 0);

END;
/
```

Table 7–6 shows the method supported in the `ORDPLUGINS.ORDX_DEFAULT_DOC` package and the exception raised if the source is `NULL`.

Table 7–6 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.

7.2.2.3 Extending *inter*Media to Support a New Media Data Format

Extending *inter*Media to support a new media 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 7.2.2.2 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 media 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 `ORDPLUG-INS.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.SQL` 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".

See Section A.2 for more information on installing your own media format plug-in and running the sample scripts provided.

Example 7–3 Show the Package Body for Extending Support to a New Media 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 :=FALSE)
  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, Section 7.2.3.2, and Section 7.2.3.3 describe support for other video data formats.

7.2.3.1 Packages or PL/SQL Plug-ins

This section presents reference information on the packages or PL/SQL plug-ins provided with the ORDVideo object. Table 7–7 describes the PL/SQL plug-in packages provided in the `ORDPLUGINS` schema.

Table 7–7 ORDVideo PL/SQL Plug-ins Provided in the ORDPLUGINS Schema

PL/SQL Plug-in Packages	Video Format	MIME Type
`ORDPLUGINS.ORDX_DEFAULT_VIDEO`	<format>	Dependent on file format
`ORDPLUGINS.ORDX_AVI_VIDEO`	AVI	video/x-msvideo
`ORDPLUGINS.ORDX_MOOV_VIDEO`	MOOV	video/quicktime
`ORDPLUGINS.ORDX_RMFF_VIDEO`	RMFF	audio/x-pn-realaudio
`ORDPLUGINS.ORDX_MPEG_VIDEO`	MPG	video/mpeg
`ORDPLUGINS.ORDX_MP4_VIDEO`	MP4	Application/mpeg4
`ORDPLUGINS.ORDX_ASF_VIDEO`	ASF	video/x-ms-wvm

Section 7.2.3.2 describes the `ORDPLUGINS.ORDX_DEFAULT_VIDEO` package, the methods supported, and the level of support. Note that the methods supported and the level of support for the other PL/SQL plug-in packages described in Table 7–7 are identical for all plug-in packages, therefore, refer to Section 7.2.3.2.

7.2.3.2 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.

Note: The `ORDPLUGINS.ORDX_DEFAULT_VIDEO` package specification includes a number of methods deprecated beginning with release 8.1.6, which have been removed in the following listing. Do not implement these deprecated methods; they are there only for backward compatibility.

```
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 7–8 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 7–8 Methods Supported in the ORDPLUGINS.ORDX_DEFAULT_VIDEO Package

Name of Method	Level of Support
getAttribute	Not supported - raises exceptions: METHOD_NOT_SUPPORTED and VIDEO_PLUGIN_EXCEPTION
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.

Table 7–8 (Cont.) Methods Supported in the ORDPLUGINS.ORDX_DEFAULT_VIDEO

Name of Method	Level of Support
checkProperties	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 `NULL`, raise an ORDSYS.ORDSourceExceptions.EMPTY_SOURCE exception; if the source is a BFILE, then extract attributes from the BFILE data and compare them with the attribute values of the object; if the source is neither local nor a BFILE, get the media content into a temporary LOB, extract attributes from the media content and compare them with the attribute values of the object.
getAllAttributes	Supported; if the source is local, get the attributes and return them, but if the source is `NULL`, raise an ORDSYS.ORDSourceExceptions.EMPTY_SOURCE exception; otherwise, if the source is external, raise an ORDSYS.ORDVideoExceptions.LOCAL_DATA_SOURCE_ REQUIRED exception.
processCommand	Not supported - raises exceptions: METHOD_NOT_SUPPORTED and VIDEO_PLUGIN_EXCEPTION

7.2.3.3 Extending *inter*Media to Support a New Video Data Format

Extending *inter*Media 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 7.2.3.2 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.

   See Section A.4 for more information on installing your own format plug-in and running the sample scripts provided. See the `fplugins.sql` and fpluginb.sql files that are installed in the $ORACLE_HOME/ord/vid/demo/ directory. These are demonstration (demo) plug-ins that you can use as a guideline to write any format plug-in that you want to support. See the viddemo.sql file in this same directory to learn how to install your own format plug-in.

4. Grant EXECUTE privileges on your new plug-in, for example, ORDX_MY_VIDEO.SQL 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".

See Section A.4 for more information on installing your own video format plug-in and running the sample scripts provided.

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 *inter*Media 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 interMedia Reference* for more information and to determine the type of images that can are supported this way.

7.3 Extending *inter*Media with a New Type

You can use any of the *inter*Media 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 *inter*Media object type attribute and the *inter*Media object type is altered, which always occurs during an *inter*Media 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 interMedia 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 interMedia 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 interMedia Reference* for a description.

8 Tuning Tips for the DBA

This chapter provides tuning tips for the Oracle DBA who wants to achieve more efficient storage and management of multimedia data in the database when using *inter*Media.

The goals of your *inter*Media 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 *inter*Media application in a production environment.

Multimedia data consists of a variety of media types including images, audio clips, video clips, line drawings, and so forth. All these media types are typically stored in LOBs, in either internal BLOBs (stored in an internal database tablespace) or in BFILEs (external LOBs in operating system files outside of the database tablespaces). This chapter discusses only the management of audio, image, and video data stored in BLOBs.

Internal LOBs consist of: CLOBs, NCLOBs, and BLOBs and can be of unlimited size (8 terabytes (TB) to 128 TB depending on the database block size of 2 kilobytes (KB) to 32 KB), which are supported in these programming environments: Java using Java Database Connectivity (JDBC), PL/SQL using the DBMS_LOB Package, and C using Oracle Call Interface (OCI).

However, in these programming environments: COBOL using Pro*COBOL precompiler, C/C++ using Pro*C/C++ precompiler, Visual Basic using Oracle Objects for OLE (OO4O), and SQL, you can create and use LOB instances only up to 4 gigabytes (GB) in size. *inter*Media supports BLOBs up to 4 GB in size for Oracle Database 10*g* Release 1 (10.1).

BFILEs can be as large as the operating system will allow up to a maximum of 8 TB. *inter*Media supports BFILEs up to a maximum of 4 GB in size.

The following general topics will help you to better manage your *inter*Media LOB data:

- Setting database initialization parameters (see Section 8.1)
- Issues to consider in creating tables with *inter*Media objects containing LOBs (see Section 8.2)
- Improving multimedia data INSERT performance in *inter*Media objects containing LOBs (see Section 8.3)
- Getting the best performance results (see Section 8.7)
- Improving *inter*Media LOB data retrieval and update performance (see Section 8.8)

For more information about LOB partitioning, LOB tuning, and LOB buffering, see *Oracle Database Application Developer's Guide - Large Objects*, *Oracle Call Interface Programmer's Guide*, *Oracle Database Concepts*, and *Oracle Database Performance Tuning Guide*.

For information on restrictions to consider when using LOBs, see *Oracle Database Application Developer's Guide - Large Objects*.

For guidelines on using the DIRECTORY feature in Oracle, see *Oracle Database Application Developer's Guide - Large Objects*. This feature enables a simple, flexible, nonintrusive, and secure mechanism for the DBA to manage access to large files in the file system.

8.1 Setting Database Initialization Parameters

The information that follows is an excerpt from *Oracle Database Performance Tuning Guide* and *Oracle Database Reference*, and is presented as an overview of the topic. Refer to *Oracle Database Performance Tuning Guide* and *Oracle Database Reference* for more information.

Database tuning of the Oracle instance consists of tuning the system global area (SGA). The SGA is used to store data in memory for fast access. The SGA consumes a portion of your system's physical memory. The SGA must be sufficiently large to keep your data in memory but neither too small nor so large that performance begins to degrade. Degrading performance occurs when the operating system begins to page unused information to disk to make room for new information needed in memory, or begins to temporarily swap active processes to disk so other processes needing memory can use it. Excessive paging and swapping can bring a system to a standstill. The goal in sizing the SGA is to size it for the data that must be kept in main memory to keep performance optimal. With this in mind, you must size the SGA required for your *inter*Media application. This may mean increasing the physical memory of your system and monitoring your operating system behavior to ensure paging and swapping remain minimal.

The size of the SGA is determined by the values of the following database initialization parameters: DB_BLOCK_SIZE, DB_CACHE_SIZE, SHARED_POOL_SIZE, and LOG_BUFFER.

Beginning with Oracle9*i*, the SGA infrastructure is dynamic. This means that the following primary parameters used to size the SGA can be changed while the instance is running:

- Buffer cache (DB_CACHE_SIZE) -- the size in bytes of the cache of standard blocks
- Shared pool (SHARED _POOL_SIZE) -- the size in bytes of the area devoted to shared SQL and PL/SQL statements
- Large pool (LARGE_POOL_SIZE) (default is 0 bytes) -- the size in bytes of the large pool used in shared server systems for session memory, parallel execution for message buffers, and by backup and restore processes for disk I/O buffers

The LOG_BUFFER parameter is used when buffering redo entries to a redo log. It is a static parameter and represents a very small portion of the SGA and can be changed only by stopping and restarting the database to read the changed value for this parameter from the initialization parameter file (`init.ora`).

Note that even though you cannot change the MAX_SGA_SIZE parameter value dynamically, you do have the option of changing any of its three dependent primary parameters (DB_CACHE_SIZE, SHARED_POOL_SIZE, and LARGE_POOL_SIZE) to make memory tuning adjustments on the fly. To help you specify an optimal cache value, you can use the dynamic DB_CACHE_ADVICE parameter with statistics gathering enabled to predict behavior with different cache sizes through the V$DB_CACHE_ADVICE performance view. Use the ALTER SYSTEM...SET clause... statement to enable this parameter. See *Oracle Database Performance Tuning Guide* for more information about using this parameter.

Beginning with Oracle9*i*, there is a concept of creating tablespaces with multiple block sizes and specifying cache sizes corresponding with each block size. The SYSTEM tablespace uses a standard block size and additional tablespaces can use up to five non-standard block sizes.

The standard block size is specified by the DB_BLOCK_SIZE parameter. Its cache size is specified by the DB_CACHE_SIZE parameter. Non-standard block sizes are specified by the BLOCKSIZE clause of the CREATE TABLESPACE statement. The cache size for each corresponding non-standard block size is specified using the notation: DB_*n*K_CACHE_SIZE parameter, where the value *n* is 2, 4, 8, 16, or 32 KB.

The standard block size, known as the default block size, is usually set to the same size in bytes as the operating system block size, or a multiple of this size. The DB_CACHE_SIZE parameter, known as the DEFAULT cache size, specifies the size of the cache of standard block size (default is 48 megabytes (MB)). The system tablespace uses the standard block size and the DEFAULT cache size.

Either the standard block size or any of the non-standard block sizes and their associated cache sizes can be used for any of your other tablespaces. If you intend to use multiple block sizes in your database storage design, you must specify at least the DB_CACHE_SIZE and one DB_*n*K_CACHE_SIZE parameter value. You must specify all sub-caches for all the other non-standard block sizes that you intend to use. This block size/cache sizing scheme lets you use up to five different non-standard block sizes for your tablespaces and lets you specify respective cache sizes for each corresponding block size. For example, you can size your system tablespace to the normal 8 KB standard block size with a default DB_CACHE_SIZE of 48 MB or whatever size you want to specify. Then you can use any of the block sizes of 2 KB, 4 KB, 8 KB, 16 KB, or the maximum 32 KB for storing your *inter*Media LOB data in appropriate block-sized tablespaces and respective caches to achieve optimal LOB storage and retrieval performance.

Because the DB_BLOCK_SIZE parameter value can be changed only by re-creating the database, the value for this parameter must be chosen carefully and remain unchanged for the life of the database. See the next section "DB_BLOCK_SIZE" for more information about this parameter.

The following sections describe these and some related initialization parameters and their importance to *inter*Media performance.

DB_BLOCK_SIZE

The DB_BLOCK_SIZE parameter is the size in bytes of database blocks (2048-32768). Oracle manages the storage space in the data files of a database in units called data blocks. The data block is the smallest unit of I/O operation used by a database; this value should be a multiple of the operating system's block size within the maximum (port-specific) limit to avoid unnecessary I/O operations. This parameter value is set for each database from the DB_BLOCK_SIZE parameter value in the initialization parameter file when you create the database. This value cannot be changed unless you create the database again.

The size of a database block determines how many rows of data Oracle can store in a single database page. The size of an average row is one piece of data that a DBA can use to determine the correct database block size. *inter*Media objects with instantiated LOB locators range in size from 175 bytes for ORDImage to 260 bytes for ORDDoc, ORDAudio, and ORDVideo. This figure does *not* include the size of the media data. (The difference in row sizes between instantiated image and audio and video data is that audio and video data contain a Comments attribute that is about 85 bytes in size to hold the LOB locator.)

If LOB data is less than 4000 bytes, then it can be stored inline or on the same database page as the rest of the row data. LOB data can be stored inline only when the block size is large enough to accommodate it.

LOB data that is stored out of line, on database pages that are separate from the row data, is accessed (read and written) by Oracle in CHUNK size pieces where CHUNK is specified in the LOB storage clause (see Section 8.2.2 for more information about the CHUNK option). CHUNK must be an integer multiple of DB_BLOCK_SIZE and defaults to DB_BLOCK_SIZE if not specified. Generally, it is more efficient for Oracle to access LOB data in large chunks, up to 32 KB. However, when LOB data is updated, it may be versioned (for read consistency) and logged both to the rollback segments and the redo log in CHUNK size pieces. If updates to LOB data are frequent then it may be more efficient space wise to manipulate smaller chunks of LOB data, especially when the granularity of the update is much less than 32 KB.

The preceding discussion is meant to highlight the differences between the initialization parameter DB_BLOCK_SIZE and the LOB storage parameter CHUNK. Each parameter controls different aspects of the database design, and though related, they should not be automatically equated.

Tuning Memory Allocation

Allocating memory to database structures and proper sizing of these structures can greatly improve database performance when working with LOB data. See *Oracle Database Performance Tuning Guide* for a comprehensive, in-depth presentation of this subject, including understanding memory allocation issues as well as detecting and solving memory allocation problems. The following sections describe a few of the important initialization parameters specifically useful for optimizing LOB performance relative to tuning memory allocation.

DB_CACHE_SIZE

The DB_CACHE_SIZE parameter specifies the size of the DEFAULT buffer pool for buffers in bytes. This value is the database buffer value that is displayed when you issue a SQL SHOW SGA statement. Because you cannot change the value of the DB_BLOCK_SIZE parameter without re-creating the database, change the value of the DB_CACHE_SIZE parameter to control the size of the database buffer cache using the ALTER SYSTEM...SET clause... statement. The DB_CACHE_SIZE parameter is dynamic.

BUFFER_POOL_KEEP and BUFFER_POOL_RECYCLE - Tuning Multiple Buffer Pools Using the Standard Block Size

To greatly reduce I/O operations while reading and processing LOB data, tune the database instance by partitioning your buffer cache into multiple buffer pools for the tables containing the LOB columns.

Note: Multiple buffer pools are available only for the standard block size. Non-standard block size caches have a single DEFAULT pool. Therefore, the information presented in this section applies to only the scenario in which you are using only the standard block size.

By default, all tables are assigned to the DEFAULT pool. Tune this main cache buffer using the DB_CACHE_SIZE initialization parameter and assign the appropriate tables to the keep pool using the DB_KEEP_CACHE_SIZE initialization parameter and to the recycle pool using the DB_RECYCLE_CACHE_SIZE initialization parameter.

The keep pool contains buffers that always stay in memory and is intended for frequently accessed tables that contain important data. The recycle pool contains buffers that can always be recycled and is intended for infrequently accessed tables that contain much less important data. The size of the main buffer cache (DEFAULT) is calculated from the value specified for the DB_CACHE_SIZE parameter minus the values specified for the DB_KEEP_CACHE_SIZE and DB_RECYCLE_CACHE_SIZE parameters. Tables are assigned to respective buffer pools (KEEP, RECYCLE, DEFAULT) using the STORAGE (buffer_pool) clause of the CREATE or ALTER TABLE statement. Determine what tables you want allocated to which of these memory buffers and the ideal size of each buffer when you implement your memory allocation design. These parameter values can be changed only in the initialization parameter file and take effect only after stopping and restarting the database.

When working with very large images, set the DB_CACHE_SIZE parameter to a large number for your Oracle instance. For example, to cache a 40 MB image, set this parameter to a value of 48 MB. Some general guidelines to consider when working with LOB data are:

- You should have enough buffers to hold the object, regardless of table LOB logging and cache settings. See Section 8.2 for more information.
- When using log files you should make the log files larger, otherwise, more time is spent waiting for log switches. See Section 8.2 for more information.
- If the same BLOB is to be accessed frequently, set the table LOB CACHE parameter to TRUE. See Section 8.2 for more information.
- Use a large page size (DB_BLOCK_SIZE) if the database is going to contain primarily large objects.

See *Oracle Database Performance Tuning Guide* for more information about tuning multiple buffer pools.

SHARED_POOL_SIZE

The SHARED_POOL_SIZE parameter specifies the size in bytes of the shared pool that contains the library cache of shared SQL requests, shared cursors, stored procedures, the dictionary cache, and control structures, Parallel Execution message buffers, and other cache structures specific to a particular instance configuration. This parameter value is dynamic. This parameter represents most of the variable size value that is displayed when you issue a SQL SHOW SGA statement. Specifying a large value improves performance in multi-user systems. A large value for example, accommodates the loading and execution of *inter*Media PL/SQL scripts and stored procedures; otherwise, execution plans are more likely to be swapped out. A large value can also accommodate many clients connecting to the server with each client connection using some shared pool space. However, when the shared pool is full, the database is unable to accept additional client connections.

SHARED_POOL_RESERVED_SIZE

The SHARED_POOL_RESERVED_SIZE parameter specifies the shared pool space that is reserved for large contiguous requests for shared pool memory. This static parameter should be set high enough to avoid performance degradation in the shared pool from situations where pool fragmentation forces Oracle to search for free chunks of unused pool to satisfy the current request.

Ideally, this parameter should be large enough to satisfy any request scanning for memory on the reserved list without flushing objects from the shared pool.

The default value is 5% of the shared pool size, while the maximum value is 50% of the shared pool size. For *inter*Media applications, a value at or close to the maximum can provide performance benefits.

LOG_BUFFER

The LOG_BUFFER parameter specifies the amount of memory, in bytes, used for buffering redo entries to the redo log file. Redo entries are written to the on disk log file when a transaction commits or when the LOG_BUFFER is full and space must be made available for new redo entries. Large values for LOG_BUFFER can reduce the number of redo log file I/O operations by allowing more data to be flushed per write operation. Large values can also eliminate the waits that occur when redo entries are flushed to make space in the log buffer pool. *inter*Media applications that have buffering enabled for the LOB data can generate large amounts of redo data when media is inserted or updated. These applications would benefit from a larger LOG_BUFFER size. This is a static parameter.

8.2 Issues to Consider in Creating Tables with Column Objects Containing BLOBs

The following information provides some strategies to consider when you create tables with *inter*Media column objects containing BLOBs. You can explicitly indicate the tablespace and storage characteristics for each BLOB. These topics are discussed in more detail and with examples in *Oracle Database Application Developer's Guide - Large Objects*. The information that follows is excerpted from Chapter 2 and is briefly presented to give you an overview of the topic. Refer to *Oracle Database Application Developer's Guide - Large Objects* for more information.

8.2.1 Initializing Internal Column Objects Containing BLOBs to NULL or EMPTY

An *inter*Media column object containing a LOB value set to `NULL` has no locator. By contrast, an empty LOB stored in a table is a LOB of zero length that has a locator. So, if you select from an empty LOB column or attribute, you get back a locator, which you can use to fill the LOB with data using the OCI or DBMS_LOB routines or ORDxxx.import method.

Setting *inter*Media Column Objects Containing a BLOB to `NULL`

You may want to set the BLOB value to `NULL` upon inserting the row whenever you do not have the BLOB data at the time of the INSERT operation. In this case, you can issue a SELECT statement at some later time to obtain a count of the number of rows in which the value of the BLOB is `NULL`, and determine how many rows must be populated with BLOB data for that particular column object.

However, the drawback to this approach is that you must then issue a SQL UPDATE statement to reset the `NULL` BLOB column to EMPTY_BLOB( ). The point is that you cannot call the OCI or the PL/SQL DBMS_LOB functions on a BLOB that is `NULL`. These functions work only with a locator, and if the BLOB column is `NULL`, there is no locator in the row.

Setting an *inter*Media Column Object Containing a BLOB to `EMPTY`

If you do not want to set an *inter*Media column object containing a BLOB to `NULL`, another option is to set the BLOB value to `EMPTY` by using the EMPTY_BLOB( ) function in the INSERT statement. Even better, set the BLOB value to `EMPTY` by using the EMPTY_BLOB( ) function in the INSERT statement, and use the RETURNING clause (thereby eliminating a round-trip that is necessary for the subsequent SELECT statement). Then, immediately call OCI, the import method, or the PL/SQL DBMS_LOB functions to fill the LOB with data. See *Oracle Database Application Developer's Guide - Large Objects* for an example.

8.2.2 Specifying Tablespace and Storage Characteristics for Column Objects Containing BLOBs

When you create tables and define *inter*Media column objects containing BLOBs, you can explicitly indicate the tablespace and storage characteristics for each BLOB. The following guidelines can help you fine-tune BLOB storage.

Tablespace

The best performance for *inter*Media column objects containing BLOBs can often be achieved by specifying storage for BLOBs in a tablespace that is different from the one used for the table that contains the *inter*Media object with a BLOB. See the ENABLE | DISABLE STORAGE IN ROW clause near the end of this section for further considerations on storing BLOB data inline or out of line. If many different LOBs are to be accessed frequently, it may also be useful to specify a separate tablespace for each BLOB or attribute in order to reduce device contention. Preallocate the tablespace to the required allocation size to avoid allocation when inserting BLOB data. See *Oracle Database SQL Reference* for examples, specifically the CREATE TABLE statement and the LOB column example. See Example 8–1.

Example 8–1 assumes that you have already issued a CONNECT statement as a suitably privileged user. This example creates a separate tablespace, called MONTANA, that is used to store the *inter*Media column object containing BLOB data for the image column. Ideally, this tablespace would be located on its own high-speed storage device to reduce contention. Other image attributes and the imageID column are stored in the default tablespace. The initial allocation allows 100 MB of storage space. The images to be inserted are about 20 KB in size. To improve insert performance, NOCACHE and NOLOGGING options are specified along with a CHUNK size of 24 KB.

Example 8–1 Create a Separate Tablespace to Store an interMedia Column Object Containing LOB Data

```
SVRMGR> CREATE TABLESPACE MONTANA DATAFILE 'montana.tbs' SIZE 400M;
Statement processed.
SVRMGR> CREATE TABLE images (imageID INTEGER ,image ORDSYS.ORDImage)
     LOB (image.source.localData) STORE AS
            (
            TABLESPACE MONTANA
            STORAGE (
                   INITIAL 100M
                   NEXT 100M
                   )
            CHUNK 24K
            NOCACHE NOLOGGING
            );
```

LOB Index and LOB_index_clause

The LOB index is an internal structure that is strongly associated with the LOB storage.

Note: The LOB_index_clause in the CREATE TABLE statement is deprecated beginning with release 8.1.5. Oracle generates an index for each LOB column and beginning with release 8.1.5, LOB indexes are system named and system managed. For information on how Oracle manages LOB indexes in tables migrated from earlier releases, see *Oracle Database Upgrade Guide.*

PCTVERSION Option

When an *inter*Media column object containing a BLOB is modified, a new version of the BLOB page is made in order to support consistent reading of prior versions of the BLOB value.

PCTVERSION is the percent of all used LOB data space that can be occupied by old versions of LOB data pages. As soon as old versions of LOB data pages start to occupy more than the PCTVERSION amount of used LOB space, Oracle tries to reclaim the old versions and reuses them. In other words, PCTVERSION is the percentage of used LOB data blocks that is available for versions of old LOB data.

One way of approximating PCTVERSION is to set PCTVERSION = (% of LOBs updated at any given point in time) times (% of each LOB updated whenever a LOB is updated) times (% of LOBs being read at any given point in time). Allow for a percentage of LOB storage space to be used as old versions of LOB pages so users can get consistent read results of data that has been updated.

Setting PCTVERSION to twice the default allows more free pages to be used for old versions of data pages. Because large queries may require consistent reading of LOBs, it is useful to keep more old versions of LOB pages around. LOB storage may increase if you increase the PCTVERSION value because Oracle will not be reusing free pages aggressively.

The more infrequent and smaller the LOB updates are, the less space that needs to be reserved for old versions of LOB data. If existing LOBs are known to be read-only, you could safely set PCTVERSION to 0% because there would never be any pages needed for old versions of data.

CACHE or NOCACHE Option

Use the CACHE option on *inter*Media column objects containing BLOBs if the same BLOB data is to be accessed frequently. The CACHE option puts the data into the database buffer and makes it accessible for subsequent read operations. If you specify CACHE, then LOGGING is used; you cannot have CACHE and NOLOGGING.

Use the NOCACHE option (the default) if BLOB data is to be read only once or infrequently, or if you have too much BLOB data to cache, or if you are reading lots of images but none more frequently than others.

Use the CACHE READS option if the BLOB data is to be brought into the buffer cache only during frequent read operations and not during write operations.

See Example 8–1.

LOGGING or NOLOGGING Option

An example of when NOLOGGING is useful is with bulk loading or inserting of data. See Example 8–1. For instance, when loading data into the *inter*Media column objects containing BLOBs, if you do not care about redo logging and can just start the load over if it fails, set the BLOB data segment storage characteristics to NOCACHE NOLOGGING. This setting gives good performance for the initial loading of data. Once you have successfully completed loading the data, you can use the ALTER TABLE statement to modify the BLOB storage characteristics for the BLOB data segment to the desired storage characteristics for normal BLOB operations, such as CACHE or NOCACHE LOGGING.

CHUNK Option

Set the CHUNK option to the number of bytes of *inter*Media column objects containing BLOB data that are to be accessed at one time. That is, the number of bytes that are to be read or written using the object.readFromSource( ) or object.writeToSource( ) *inter*Media audio and video object methods or call, OCILobRead( ), OCILobWrite( ), DBMS_LOB.READ( ), or DBMS_LOB.WRITE( ) during one access of the BLOB value. Note that the default value for the CHUNK option is 1 Oracle block and does not vary across systems. If only 1 block of BLOB data is accessed at a time, set the CHUNK option to the size of 1 block. For example, if the database block size is 2 KB, then set the CHUNK option to 2 KB.

Set the CHUNK option to the next largest integer multiple of database block size that is slightly larger than the audio, image, or video data size being inserted. Specifying a slightly larger CHUNK option allows for some variation in the actual sizes of the multimedia data and ensures that the benefit is realized. For large-sized media data, a general rule is to set the CHUNK option as large as possible; the maximum is 32 KB. For example, if the database block size is 2 KB or 4 KB or 8 KB and the image data is mostly 21 KB in size, set the CHUNK option to 24 KB. See Example 8–1.

INITIAL and NEXT Parameters

If you explicitly specify the storage characteristics for the *inter*Media column object containing a BLOB, make sure that the INITIAL and NEXT parameters for the BLOB data segment storage are set to a size that is larger than the CHUNK size. For example, if the database block size is 2 KB and you specify a CHUNK value of 8 KB, make sure that the INITIAL and NEXT parameters are at least 8 KB, preferably higher (for example, at least 16 KB).

For LOB storage, Oracle automatically builds and maintains a LOB index that allows quick access to any chunk and thus any portion of a LOB. The LOB index gets the same storage extent parameter values as its LOBs. Consequently, to optimize LOB storage space, you should calculate the size of your LOB index size as well as the total storage space needed to store the media data including its overhead.

Assume that *N* files composed of *M* total bytes of media data are to be stored and that the value *C* represents the size of the LOB CHUNK storage parameter. To calculate the total number of bytes Y needed to store the media data:

$Y = M + (N*C)$

The expression (N*C) accounts for the worst case in which the last CHUNK of each LOB contains a single byte. Therefore, an extra CHUNK is allowed for each file that is stored. On average, the last CHUNK will be half full.

To calculate the total number of bytes X to store the LOB index:

X = CEIL(M/C) * 32

The value 32 indicates that the LOB index requires roughly 32 bytes for each CHUNK that is stored.

The total storage space needed for the media data plus its LOB index is then X + Y.

The following two examples describe these calculations in detail.

Example 1: Assume you have 500 video clips comprising a total size of 250 MB with an average size is 512 KB. Assume a LOB CHUNK size of 32768 bytes. The total space needed for the media data is 250 MB + (5000*32768) or 266 MB. The overhead is 16 MB or about 6.5% storage overhead. The total space needed to store the LOB index is CEIL(250 MB/32768) * 32 or 244 KB. The total space needed to store the media data plus its LOB index is then about 266.6 MB.

```
SQL> SELECT 250000000+(500*32768)+CEIL(250000000/32768)*32 FROM dual;

250000000+(500*32768)+CEIL(250000000/32768)*32
----------------------------------------------
                                     266628160
```

The following table definition could be used to store this amount of data:

```
CREATE TABLE video_items
(
  video_id         NUMBER,
  video_clip      ORDSYS.ORDVideo
)
-- storage parameters for table in general
TABLESPACE video1 STORAGE (INITIAL 1M NEXT 10M)
-- special storage parameters for the video content
LOB(video_clip.source.localdata) STORE AS
  (TABLESPACE video2 STORAGE (INITIAL 260K NEXT 270M)
   DISABLE STORAGE IN ROW NOCACHE NOLOGGING CHUNK 32768);
```

Example 2: Assume you have 5000 images comprising a total size of 274 MB with an average size of 56 KB. Because the average size of the images are smaller than the video clips in the preceding example, it is more space efficient to choose a smaller CHUNK size, for example 8192 bytes to store the data in the LOB. The total space needed for the media data is 274 MB + (5000*8192) or 314 MB. The overhead is about 40 MB or about 15% storage overhead. The total space needed to store the LOB index is CEIL(274 MB/8192) * 32 or 1.05 MB. The total space needed to store the media data plus its LOB index is then about 316 MB.

```
SQL> SELECT 274000000+(5000*8192)+CEIL(274000000/8192)*32 FROM dual;

274000000+(5000*8192)+CEIL(274000000/8192)*32
---------------------------------------------
                                    316030336
```

The following table definition could be used to store this amount of data:

```
CREATE TABLE image_items
(
  image_id         NUMBER,
  image            ORDSYS.ORDImage
)
-- storage parameters for table in general
TABLESPACE image1 STORAGE (INITIAL 1M NEXT 10M)
-- special storage parameters for the image content
LOB(image.source.localdata) STORE AS
  (TABLESPACE image2 STORAGE (INITIAL 1200K NEXT 320M)
   DISABLE STORAGE IN ROW NOCACHE NOLOGGING CHUNK 8192);
```

When working with very large BLOBs on the order of 1 gigabyte in size, choose a proportionately large INITIAL and NEXT extent parameter size, for example an INITIAL value slightly larger than your calculated LOB index size and a NEXT value of 100 MB, to reduce the frequency of extent creation, or commit the transaction more often to reuse the space in the rollback segment; otherwise, if the number of extents is large, the rollback segment can become saturated.

PCTINCREASE Parameter

Set the PCTINCREASE parameter value to 0 to make the growth of new extent sizes more manageable. When working with very large BLOBs and the BLOB is being filled up piece by piece in a tablespace, numerous new extents are created in the process. If the extent sizes keep increasing by the default value of 50% each time one is created, extents will become unmanageably big and eventually will waste space in the tablespace.

MAXEXTENTS Parameter

Set the MAXEXTENTS parameter value to suit the projected size of the BLOB or set it to UNLIMITED for safety. That is, when MAXEXTENTS is set to UNLIMITED, extents will be allocated automatically as needed and this minimizes fragmentation.

ENABLE | DISABLE STORAGE IN ROW Clause

You use the ENABLE | DISABLE STORAGE IN ROW clause to indicate whether the *inter*Media column objects containing a BLOB should be stored inline (that is, in the row) or out of line. You may not alter this specification once you have made it: if you ENABLE STORAGE IN ROW, you cannot alter it to DISABLE STORAGE IN ROW or the reverse. The default is ENABLE STORAGE IN ROW.

The maximum amount of LOB data that will be stored in the row is the maximum VARCHAR size (4000). Note that this includes the control information as well as the LOB value. If the user indicates that the LOB should be stored in the row, once the LOB value and control information are larger than 4000 bytes, the LOB value is automatically moved out of the row.

This suggests the following guideline: If the *inter*Media column object containing a BLOB is small (that is, less than 4000 bytes), then storing the BLOB data out of line will decrease performance. However, storing the BLOB in the row increases the size of the row. This has a detrimental impact on performance if you are doing a lot of base table processing, such as full table scans, multiple row accesses (range scans), or doing many UPDATE or SELECT statements to columns other than the *inter*Media column objects containing BLOBs. If you do not expect the BLOB data to be less than 4000 bytes, that is, if all BLOBs are big, then the default is the best choice because:

- The LOB data is automatically moved out of line once it gets bigger than 4000 bytes.
- Performance can be better if the BLOB data is small (less than 4000 bytes including control information) and is stored inline because the LOB locator and the BLOB data can be retrieved in the same buffer, thus reducing I/O operations.

8.2.3 Segment Attributes and Physical Attributes

The following physical attribute is important for optimum storage of BLOB data in the data block and consequently achieving optimum retrieval performance.

PCTFREE Parameter

The PCTFREE parameter specifies the percentage of space in each data block of the table or partition reserved for future updates to each row of the table. Setting this parameter to an appropriate value is useful for efficient inline storage of multimedia data. The default value is 10%.

Set this parameter to a high enough value to avoid row chaining or row migration. Because the INSERT statement for BLOBs requires an EMPTY_BLOB column object initialization followed by an UPDATE statement to load the BLOB data into the data block, you must set the PCTFREE parameter value to a proper value especially if the BLOB data will be stored inline. For example, row chaining can result after a row INSERT operation when insufficient space is reserved in the existing data block to store the entire row, including the inline BLOB data in the subsequent UPDATE operation. As a result, the row would be broken into multiple pieces and each piece stored in a separate data block. Consequently, more I/O operations would be needed to retrieve the entire row, including the BLOB data, resulting in poorer performance. Row migration can also result if there is insufficient space in the data block to store the entire row during the initial INSERT operation, and thus the row is stored in another data block.

To make best use of the PCTFREE parameter, determine the average size of the BLOB data being stored inline in each row, and then determine the entire row size, including the inline BLOB data. Set the PCTFREE parameter value to allow for sufficient free space to store an entire row of data in the data block. For example, if you have a large number of thumbnail images that are about 3 KB in size, and each row is about 3.8 KB in size, and the database block size is 8 KB, set the value of PCTFREE to a value that ensures that two complete rows can be stored in each data block in the initial INSERT operation. This approach initially uses 1.6 KB of space (0.8 KB/row *2 rows) leaving 6.4 KB of free space. Because two rows initially use 20% of the data block and 95% after an UPDATE operation and adding a third row would initially use 30% of the data block causing a chain to occur when the third row is updated, set the PCTRFEE parameter value to 75. This setting permits a maximum of two rows to be stored per data block and leaves sufficient space to update each row with its 3 KB thumbnail image leaving about 0.4 KB free space minus overhead per data block.

8.3 Improving Multimedia Data INSERT Performance in Objects Containing LOBs

There are a number of bulk loading methods available for loading FILE data into *inter*Media objects containing BLOBs. These include:

- *inter*Media import( ) method in a PL/SQL stored procedure
- SQL*Loader (conventional path load and direct path load)
- OCILobLoadFromFile( ) relational function
- DBMS_LOB.LOADFROMFILE( ) procedure in the DBMS_LOB package

- DBMS_LOB.LOADBLOBFROMFILE( ) procedure in the DBMS_LOB package
- Java loadDataFromFile( ) or loadDataFromInputStream( ) methods of Oracle *inter*Media Java Classes to load media data from a client file

Using *inter*Media import( ) Method in a PL/SQL Stored Procedure

Example 8–2 shows the contents of the `load1.bat` file, which invokes SQL*Plus and runs the `t1.sql` procedure (Example 8–3). The db_block_size for this schema is 8 KB bytes.

Example 8–2 Show the load1.bat File

```
sqlplus scott/tiger@intertcp @t1
```

Example 8–3 shows the contents of the `t1.sql` file. This procedure:

- Creates two tablespaces.
- Creates the `image_items` table and defines the physical properties of the table, specifically the physical attributes and LOB storage attributes.
- Partitions the table storage into each tablespace by range using the `image_id` value.
- Creates the `load_image` stored procedure that:
  - Declares a variable `nxtseq` defined as the ROWID data type.
  - Inserts a row into the `image_items` table and uses the INSERT RETURNING ROWID statement to return the ROWID value for fastest access to the row for loading the image BLOB data into the object columns of each row using the import( ) method.
  - Sets the image attribute properties automatically (by means of the import operation) for each loaded image (note that thumbnail images are stored inline, and regular images are stored out of line).
  - Commits the update operation.

Example 8–3 Show the t1.sql Procedure

```
spool t1.log
set echo on
connect sys/change_on_install as sysdba

create tablespace Image_h default storage (initial 30m next 400m pctincrease 0)
   datafile 'h:\IMPB\Image_h.DBF'
   size 2501M reuse;

create tablespace Image_i default storage (initial 30m next 400m pctincrease 0)
   datafile 'i:\IMPB\Image_i.DBF'
   size 2501M reuse;

connect scott/tiger

drop table image_items;

create table image_items(
  image_id              number,-- constraint pl_rm primary key,
  image_title           varchar2(128),
  image_artist          varchar2(128),
  image_publisher       varchar2(128),
  image_description varchar2(1000),
  image_price           number(6,2),
  image_file_path varchar2(128),
  image_thumb_path varchar2(128),
  image_thumb           ordsys.ordimage,
  image_clip            ordsys.ordimage
)
--
-- physical properties of table
--
  -- physical attributes clause
  pctfree 35 storage (initial 30M next 400M pctincrease 0)

  -- LOB storage clause (applies to LOB column)
  LOB (image_clip.source.localdata)
      store as (disable storage in row nocache nologging chunk 32768)
--
-- table properties (applies to whole table)
--
Partition by range (image_id)
(
Partition Part1 values less than (110001)
Tablespace image_h,
Partition Part2 values less than (maxvalue)
Tablespace image_i
);

connect scott/tiger;

create or replace procedure load_image
```

```
(
  image_id           number,
  image_title        varchar2,
  image_artist       varchar2,
  image_publisher    varchar2,
  image_description  varchar2,
  image_price        number,
  image_file_path    varchar2,
  image_thumb_path   varchar2,
  thumb_dir          varchar2,
  content_dir        varchar2,
  file_name1         varchar2,
  file_name2         varchar2)
as
  ctx raw(4000) := NULL;
  obj1           ORDSYS.ORDIMAGE;
  obj2           ORDSYS.ORDIMAGE;
  nxtseq         rowid;

Begin
  Insert into image_items(
    image_id,
    image_title,
    image_artist,
    image_publisher,
    image_description,
    image_price,
    image_file_path,
    image_thumb_path ,
    image_thumb,
    image_clip)
 values (
    image_id,
    image_title,
    image_artist,
    image_publisher,
    image_description,
    image_price,
    image_file_path,
    image_thumb_path ,
    ORDSYS.ORDIMAGE.init('FILE',upper(thumb_dir),file_name1),
    ORDSYS.ORDIMAGE.init('FILE',upper(content_dir),file_name2))
    returning rowid into nxtseq;

-- load up the thumbnail image
```

```
  select t.image_thumb,
t.image_clip
  into obj1, obj2
  from image_items t
    where t.rowid = nxtseq for update;
  obj1.import(ctx);        -- import sets properties
  obj2.import(ctx);
  Update image_items I
  set I.image_thumb = obj1,
      I.image_clip  = obj2
    where i.rowid = nxtseq;

  Commit;
End;
/
spool off
set echo off
```

Example 8–4 shows the contents of the `load1.sql` file. The image load directories are created and specified for each tablespace and user `scott` is granted read privilege on each load directory. The stored procedure named `load_image` is then executed, which loads values for each column row. By partitioning the data into different tablespaces, each partition can be loaded in a parallel data load operation.

Example 8–4 Show the load1.sql File that Executes the load_image Stored Procedure

```
connect sys/change_on_install as sysdba
drop directory IMAGE_H;
drop directory IMAGE_I;
create directory IMAGE_H as 'h:\image_files';
create directory IMAGE_I as 'i:\image_files';
grant read on directory IMAGE_H to scott;
grant read on directory IMAGE_I to scott;
EXEC Load_image(100001,'T_100001',1916,'Publisher','Visit our WEB page'
,8.71,'image_I\T_100001.jpg','image_I\T_100001_thumb1.jpg','image_I','image_
I','T_100001_thumb1.jpg','T_100001.jpg');
EXEC Load_image(100002,'T_100002',2050,'Publisher','Visit our WEB page'
,9.61,'image_I\T_100002.jpg','image_I\T_100002_thumb10.jpg','image_I','image_
I','T_100002_thumb10.jpg','T_100002.jpg');
exit
```

Using SQL*Loader

SQL*Loader provides two methods for loading data:

- Conventional Path Load

  A conventional path load (the default) uses the SQL INSERT statement and a bind array buffer to load data into database tables. When SQL*Loader performs a conventional path load, it competes equally with all other processes for buffer resources. This can slow the load significantly. Extra overhead is added as SQL statements are generated, passed to Oracle, and executed. Oracle looks for partially filled blocks and attempts to fill them on each insert operation. Although appropriate during normal use, this can slow bulk loads dramatically. Use conventional path load if you encounter certain restrictions on direct path loads.

- Direct Path Load

  A direct path load eliminates much of the database overhead by formatting Oracle data blocks and writing the data blocks directly to the database files. A direct load does not compete with other users for database resources, so it can usually load data at near disk speed. In addition, if the asynchronous I/O operations feature is available on your host platform, multiple buffers are used for the formatted data blocks to further increase load performance.

See *Oracle Database Utilities* for a complete list of restrictions for using either the conventional path load or direct path load method for loading data using SQL*Loader. See *Oracle Database Application Developer's Guide - Fundamentals* for more information on LOBs.

Using SQL*Loader to Load Multimedia Data into Oracle Database Using *inter*Media Column Objects

Example 8–5 shows the use of the control file to load one ORDVideo object per file into a table named JUKE that has three columns, with the last one being a column object. Each LOB file is the source of a single LOB and follows the column object name with the LOBFILE data type specifications. Two LOB files are loaded in this example.

Example 8–5 Show the Control File for Loading Video Data

```
LOAD DATA
INFILE *
INTO TABLE JUKE
REPLACE
FIELDS TERMINATED BY ','
( id integer external,
 file_name char(1000),
  mediacontent column object
```

```
    (
        source column object
        (
1)       localData_fname FILLER CHAR(128),
2)      localData LOBFILE (mediacontent.source.localData_fname) terminated by EOF
        )

    )
)

BEGINDATA
1,slynne,slynne.rm
2,Commodores,Commodores - Brick House.rm
```

Notes:

1. The FILLER field is mapped to the 128-byte long data field which is read using the SQL*Loader CHAR data type.
2. SQL*Loader gets the LOB file name from the localData_fname FILLER field. It then loads the data from the LOB file (using the BLOB data type) from its beginning to the EOF character, whichever is reached first. Note that if no existing LOB file is specified, the `localData` field is initialized to empty.

Using the OCILobLoadFromFile( ) Relational Function

Oracle Call Interface (OCI) is an application programming interface (API) that allows you to manipulate data and schemas in a database using a host programming language, such as C.

The OCI relational function, OCILobLoadFromFile( ), loads or copies all or a portion of a file into an *inter*Media column object containing a specified BLOB. The data is copied from the source file to the destination *inter*Media column objects containing a BLOB. When binary data is loaded into an *inter*Media column object containing a BLOB, no character set conversions are performed. Therefore, the file data must already be in the same character set as the BLOB in the database. No error checking is performed to verify this.

See *Oracle Call Interface Programmer's Guide* for more information.

Using the DBMS_LOB.LOADFROMFILE( ) Procedure in the DBMS_LOB Package

The DBMS_LOB package provides subprograms to operate on BLOBs, CLOBs, NCLOBs, BFILEs, and temporary LOBs. You can use the DBMS_LOB package for access and manipulation of specific parts of an *inter*Media column object containing a BLOB, as well as complete BLOBs. DBMS_LOB can read as well as modify BLOBs, CLOBs, and NCLOBs, and provides read-only operations for BFILEs. The majority of the LOB operations are provided by this package.

The DBMS_LOB.LOADFROMFILE( ) procedure copies all, or part of, a source-external LOB (BFILE) to a destination internal LOB.

You can specify the offsets for both the source LOB (BFILE) and destination *inter*Media column object containing the BLOB and the number of bytes to copy from the source BFILE. The amount and src_offset, because they refer to the BFILE, are in terms of bytes, and the destination offset is either in bytes or characters for BLOBs and CLOBs respectively.

The input BFILE must have been opened prior to using this procedure. No character-set conversions are performed implicitly when binary BFILE data is loaded into a CLOB. The BFILE data must already be in the same character set as the CLOB in the database. No error checking is performed to verify this. See *PL/SQL Packages and Types Reference* for more information.

Using the DBMS_LOB.LOADBLOBFROMFILE( ) Procedure in the DBMS_ LOB Package

The DBMS_LOB.LOADBLOBFROMFILE( ) procedure loads a persistent or temporary BLOB instance with data from a BFILE. This procedure achieves the same result as using DBMS_LOB.LOADFROMFILE, but returns the new offset in bytes in the destination BLOB right after the end of the write operation, which is also where the next write operation should begin and the offset in bytes in the source BFILE right after the end of the read operation, which is also where the next read operation should begin.

To use this procedure, you can specify the offsets for both the source and destination LOBs, and the number of bytes to copy from the source BFILE. The value you pass for the `amount` parameter to the DBMS_ LOB.LOADBLOBFROMFILE function must be either an amount less than or equal to the actual size (in bytes) of the BFILE you are loading or the maximum allowable LOB size: DBMS_LOB.LOBMAXSIZE. Passing this latter value causes the function to load the entire BFILE, which is a useful technique for loading the entire BFILE without introspecting the size of the BFILE.

To use this procedure, the target BLOB instance and the source BFILE must both exist and the BFILE must be opened and later closed after calling this procedure.

See *PL/SQL Packages and Types Reference* for more information.

Using Java loadDataFrom...( ) Methods to Load Media Data from a Client File

From the Java client, you can use the Java loadDataFromByteArray( ), loadDataFromFile( ), or loadDataFromInputStream( ) methods of Oracle *inter*Media Java Classes to load media data from a given file into a server-side media object designated by the corresponding media locator parameters. You must specify the name of the file from which to load the data and the method returns true if loading is successful, false otherwise. See *Oracle interMedia Java Classes Reference* for more information.

8.4 Loading Multimedia Data Using a WebDAV-Compliant Client Application

You can also use a WebDAV-compliant client application, such as Adobe GoLive, Microsoft's Web Folders, and other available software products to individually store and retrieve multimedia objects, such as audio, video, and image data, in a database. Using WebDAV, client applications have read/write access to multimedia content in the database.

To configure WebDAV access to multimedia data, using Oracle HTTP Server and the mod_oradav component (OraDAV), the necessary software components can be downloaded from the Oracle *inter*Media Software section of the Oracle Technology Network Web site

```
http://otn.oracle.com/products/intermedia
```

8.5 Transferring Multimedia Data Using Oracle Data Pump

Oracle Data Pump ("Data Pump") enables very high-speed movement of data and metadata from one database to another using the Data Pump Export and Data Pump Import utilities. Data Pump enables you to specify whether or not a job should move a subset of the data and metadata. This is done using data filters and metadata filters, which are implemented through Export and Import parameters using the Metadata API and the Data Pump API. The Metadata API uses the procedures provided in the DBMS_METADATA PL/SQL package and the Data Pump API uses the procedures provided in the DBMS_DATAPUMP PL/SQL package. See *Oracle Database Concepts*, *Oracle Database Utilities*, and *PL/SQL Packages and Types Reference* for more information.

8.6 Reading Data from an ORDVideo Object Using the readFromSource( ) Method in a PL/SQL Script

Example 8–6 shows the contents of the `readvideo1.sql` file. This procedure reads data from an ORDVideo object with the video stored in a BLOB in the database using the readFromSource( ) method in a PL/SQL script until no more data is found. The procedure then returns a NO_DATA_FOUND exception when the read operation is complete and displays an "End of data" message.

Note: This example can be modified to work with the ORDAudio, ORDDoc, and ORDImage objects too.

Example 8–6 Read Data from an ORDVideo Column Object Using the interMedia readFromSource( ) Method in a PL/SQL Stored Procedure

```
create or replace procedure readVideo1(i integer)  as

   obj ORDSYS.ORDVideo;
   buffer RAW (32767);
   numbytes BINARY_INTEGER := 32767;
   startpos integer := 1;
   read_cnt integer := 1;
   ctx RAW(4000) := NULL;

BEGIN

   Select  mediacontent into obj from juke where id = 100001;

   LOOP
           obj.readFromSource(ctx,startpos,numbytes,buffer);
           startpos := startpos + numBytes;
           read_cnt := read_cnt + 1;

   END LOOP;

EXCEPTION

   WHEN NO_DATA_FOUND THEN
   DBMS_OUTPUT.PUT_LINE('End of data ');
   DBMS_OUTPUT.PUT_LINE('doing read  '|| read_cnt);
   DBMS_OUTPUT.PUT_LINE('start position :'|| startpos);

END;
```

```
/
show errors
```

8.7 Getting the Best Performance Results

The following guidelines can be used to help you achieve the best performance when working with *inter*Media objects:

- Because *inter*Media objects are big, attain the best performance by reading and writing large CHUNKS of an *inter*Media object value at a time. This helps in several respects:
  - If you are accessing the *inter*Media object from the client side and the client is on a different node than the server, large read/write operations reduce network overhead.
  - If you are using the NOCACHE option, each small read/write operation incurs an I/O impact. Reading and writing large quantities of data reduces the I/O impact.
  - Writing to the *inter*Media object creates a new version of the *inter*Media object CHUNK. Therefore, writing small amounts at a time will incur the cost of a new version for each small write operation. If logging is on, the CHUNK is also stored in the redo log.
- If you need to read or write small pieces of *inter*Media object data on the client, use LOB buffering (see OCILobEnableBuffering( ), OCILobDisableBuffering( ), OCILobFlushBuffer( ), OCILobWrite( ), OCILobRead( ) in *Oracle Call Interface Programmer's Guide* for more information.). Turn on LOB buffering before reading or writing small pieces of *inter*Media object data. For more information about LOB buffering, its advantages, guidelines for use, and usage, see *Oracle Database Application Developer's Guide - Large Objects*.
- Use *inter*Media methods (readFromSource( ) and writeToSource( )) for audio and video data or OCILobWrite( ) and OCILobRead( ) with a callback for image data so media data is streamed to and from the BLOB. Ensure that the length of the entire write operation is set in the `numBytes` parameter using *inter*Media methods or in the `amount` parameter using OCI calls on input. Whenever possible, read and write in multiples of the LOB CHUNK size.
- Use a checkout/checkin model for LOBs. LOBs are optimized for the following:
  - Updating *inter*Media object data: SQL UPDATE operations, which replaces the entire BLOB value.

  - Copying the entire LOB data to the client, modifying the LOB data on the client side, and copying the entire LOB data back to the database. This can be done using OCILobRead( ) and OCILobWrite( ) with streaming.

- Commit changes frequently.
- Follow temporary LOB performance guidelines.

  See *Oracle Database Application Developer's Guide - Large Objects* for information and guidelines about using temporary LOBs.
- Use *inter*Media column objects containing BLOBs in table partitions.

  See the information about LOBs in partitioned tables in *Oracle Database Application Developer's Guide - Large Objects* and see *Oracle Database SQL Reference* for examples, specifically the CREATE TABLE statement and the Partitioned Table with LOB Columns example.

See *Oracle Database Application Developer's Guide - Large Objects* for more information.

8.8 Improving Multimedia LOB Data Retrieval and Update Performance

Once the LOB data is stored in the database, a modified strategy must be used to improve the performance of retrieving and updating the LOB data compared to the insertion strategy described in Section 8.3. The following guidelines should be considered:

- Use the CACHE option on LOBs if the same LOB data is to be accessed frequently by other users.
- Increase the number of buffers if you are going to use the CACHE option.
- Have enough buffers to hold the object. Using a small number of buffers for large objects is not good. Set the DB_CACHE_SIZE parameter to a value that you know will hold the object.
- Ensure that your redo log files are much larger than they usually are; otherwise, you may be waiting for log switches, especially if you are making many updates to your LOB data.
- Ensure that you use a larger page size (DB_BLOCK_SIZE), especially if the majority of the data in the database is LOB data.

9

*inter*Media Examples

This chapter provides examples that show common operations with *inter*Media. Examples are presented by audio (Section 9.1), media (Section 9.2), image (Section 9.3), and video (Section 9.4) data groups. In addition, Section 9.5 describes handling exceptions in PL/SQL and Java for some of the more common *inter*Media errors and other types of errors.

For more examples, see the Oracle Technology Network (OTN) Web site

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

Select the **Sample Code** icon, then under Oracle Database, select **Oracle *inter*Media** to go to the Oracle *inter*Media Sample Code Web page.

9.1 Audio Data Examples

Audio data examples using *inter*Media include the following common operations:

- Using *inter*Media with object views
- Using a set of scripts for creating and populating an audio table with BLOB data stored in the database

Reference information on the methods used in these examples is presented in *Oracle interMedia 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 from data --

of either built-in or user-defined types -- 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,
);
```

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 on defining, using, and updating object views.

9.1.2 Scripts for Populating ORDAudio Objects with BLOB Data

The following scripts can be found on the Oracle Technology Network (OTN) Web site

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

These scripts are end-to-end scripts that show you how to populate an *inter*Media ORDAudio object from a BLOB stored in the database. You can get to this site by selecting the **Sample Code** icon, then under Oracle Database, select **Oracle *inter*Media** to go to the Oracle *inter*Media Sample Code Web page.

The following set of scripts:

- Creates a tablespace for the audio data, creates a user and grants certain privileges to this new user, creates an audio data load directory (`create_auduser.sql`).
- Creates the `soundtable` table with two columns (`id`, `sound`), inserts a row into the table and initializes the BLOB column with an empty BLOB, and loads the audio data clip with a SELECT FOR UPDATE operation using a DBMS_LOB loadfromfile call to load the data from a BFILE, (`create_soundtable.sql`).
- Creates the `audtable` table with two columns (`id`, `audio`), inserts three rows and initializes the object column to empty with a locator and initializes the

object attributes in the object, loads the audio data with a SELECT FOR UPDATE operation using an import( ) method to import the data from a BFILE for ID=1, and sets the properties of the object (`create_audtable.sql`).

- Copies the BLOB audio data clip stored in the BLOB column of the `soundtable` table to the ORDAudio object column of the `audtable` table using an UPDATE statement for ID=3 and updates the properties of the object and the time stamp with another UPDATE statement.
- Checks the properties of each audio data clip object, one that was imported from a BFILE into the ORDAudio object type for ID=1, and the other that was copied from a BLOB into the ORDAudio object type for ID=3. The attributes for each audio data clip should be identical.
- A sixth script (`setup_audschema2.sql`) automates this entire process by running each script in the required order.

To successfully load audio data, you must have an `auddir` directory created on your system. This directory contains your sample audio clip file, `chimes.wav`. Actually, you can copy any supported audio data clip file to the `auddir` directory to run this script. Be sure to change the data file names in the script to correspond with the name of the data file you use. This directory path and disk drive must be specified in the CREATE DIRECTORY statement in the `creat_auduser.sql` file.

Script 1: Create a Tablespace and an Audio User, Grant Privileges to the Audio User, and Create an Audio Data Load Directory (create_auduser.sql)

This script creates the `auddemo` tablespace. It contains a data file named `auddemo.dbf` of 200 MB in size, an initial extent of 64 KB, and a next extent of 128 KB, and turns on table logging. Next, the auddemo user is created and given connect, resource, create library, and create directory privileges followed by creating the audio data load directory. Before running this script, you must change the create directory line to point to your data load directory location.

Note: You must edit the `create_auduser.sql` file and either enter the SYS password in the CONNECT statement or comment out the CONNECT statement and run this file as SYS AS SYSDBA. You must specify the disk drive in the CREATE DIRECTORY statement. Also, create the `temp` temporary tablespace if you have not already created it, otherwise this file will not run.

```
-- create_auduser.sql

-- Connect as admin.
CONNECT SYS AS SYSDBA/<SYS password>;

-- Edit this script and either enter your sys password here
-- to replace <SYS password> or comment out this CONNECT
-- statement and connect as SYS AS SYSDBA before running this script.

SET SERVEROUTPUT ON;
SET ECHO ON;

-- You need SYSDBA privileges to delete a user.
-- Note: No need to delete auddemo user if you don’t delete the
-- auddemo tablespace, therefore comment out the next line.

-- DROP USER auddemo CASCADE;

-- You need SYSDBA privileges to delete a directory. If there is no need
-- to really delete it, then comment out the next line.

-- DROP DIRECTORY auddir;

-- Delete, then create, a tablespace.

-- Note: It is better to not delete and create tablespaces,
-- so comment this next line out. The CREATE TABLESPACE statement
-- will fail if it already exists.

-- DROP TABLESPACE auddemo INCLUDING CONTENTS;

-- If you uncomment the line above and really want to delete the
-- auddemo tablespace, remember to manually delete the auddemo.dbf
-- file to complete the operation. Otherwise, you cannot create
-- the auddemo tablespace again because the auddemo.dbf file already
-- exists. Therefore, it might be best to create this tablespace
-- once and not delete it.

-- Create a tablespace.
CREATE TABLESPACE auddemo
       DATAFILE ’auddemo.dbf’ SIZE 25M
       MINIMIM EXTENT 64K
       DEFAULT STORAGE (INITIAL 64K NEXT 128K)
       LOGGING;
```

```
-- Create the auddemo user.
CREATE USER auddemo IDENTIFIED BY auddemo
DEFAULT TABLESPACE auddemo
TEMPORARY TABLESPACE temp;

-- Note: If you do not have a temp tablespace already defined,
--       you will have to create it first for this script to work.

GRANT CONNECT, RESOURCE, CREATE LIBRARY TO auddemo;
GRANT CREATE ANY DIRECTORY TO auddemo;

-- Note: If this user already exists, you will get an error message
-- when you try to create this user again.

-- Connect as auddemo.
CONNECT auddemo/auddemo

-- Create the auddemo load directory, the directory where the audio
-- clips are residing. Replace directory specification with your own.

CREATE OR REPLACE DIRECTORY auddir
       AS 'e:\auddir';
GRANT READ ON DIRECTORY auddir TO PUBLIC WITH GRANT OPTION;
```

Script 2: Create the Sound Table, Insert a Row with an Empty BLOB, Load the Row with BLOB Data, and Check the Length of the BLOB Data

This script creates the `soundtable` table, performs an insert operation inserting a row with an empty BLOB, loads the row with BLOB data, then checks the length of the BLOB data to ensure that the BLOB data was loaded.

```
--create_soundtable.sql
--
-- Create the soundtable table.
-- 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.
-- Open the file and use the locator to insert the file.
-- Close the files and commit the transaction.
-- Check the length of the BLOB loaded. Is the length
--  what you are expecting?

CONNECT auddemo/auddemo;
```

```
SET SERVEROUTPUT ON;
SET ECHO ON;

DROP TABLE soundtable;
CREATE TABLE soundtable (id number,
                         sound BLOB
       default EMPTY_BLOB());

INSERT INTO soundtable(id, sound) VALUES (1, EMPTY_BLOB());
COMMIT;

DECLARE
   f_lob  BFILE := BFILENAME('AUDDIR','chimes.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);
  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.
   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;
/
```

Script 3: Create the Audtable Table, Insert Three Rows with Empty BLOBs, Initialize Object Attributes, and Load One Row (ID=1) with Audio Data (create_audtable.sql)

This script creates the `audtable` table, and then performs an insert operation to initialize the column object to empty for three rows. Initializing the column object creates the BLOB locator that is required for populating each row with BLOB data in a subsequent data load operation. Next the script performs a SELECT FOR UPDATE operation to load the audio data by first setting the source for loading the audio data from a file for ID=1, importing the data, setting the properties for the BLOB data, updating the row for ID=1, and committing the transaction. To successfully run this script, you must copy one audio clip to your `auddir` directory using the names specified in this script, or modify this script to match the file names of your audio clips.

```
-- create_audtable.sql
--
-- Create the audtable table.
-- Insert three rows with empty BLOBs and initialize object attributes.
-- Import a BFILE into the ORDAudio object for ID=1.

CONNECT auddemo/auddemo;
SET SERVEROUTPUT ON;
SET ECHO ON;

  DROP TABLE audtable;
  CREATE TABLE audtable (id    NUMBER,
                         audio ORDSYS.ORDAudio);

-- Insert rows with an empty BLOB and initialize the object attributes.

  INSERT INTO audtable VALUES(1,ORDSYS.ORDAudio.init());
  INSERT INTO audtable VALUES(2,ORDSYS.ORDAudio.init());
  INSERT INTO audtable VALUES(3,ORDSYS.ORDAudio.init());

  COMMIT;

DECLARE
  obj ORDSYS.ORDAUDIO;
  ctx RAW(64) := NULL;

BEGIN
```

```
-- This performs a SELECT FOR UPDATE from table audtable for ID=1,
-- imports the audio file chimes.wav from the AUDDIR directory
-- as a BFILE on a local file system (srcType=FILE), sets the properties,
-- updates the row in table audtable for ID=1, then commits the transaction.

  SELECT audio INTO obj FROM audtable WHERE id = 1 FOR UPDATE;
  obj.setSource('FILE','AUDDIR','chimes.wav');
  obj.import(ctx);
  obj.setProperties(ctx);

  UPDATE audtable SET audio = obj WHERE id = 1;
  COMMIT;

END;
/
```

Script 4: Copy the BLOB Data to the ORDAudio Object

This script copies the BLOB audio data in the sound column of the `soundtable` table for a row (ID=1) to the ORDAudio object column of the `audtable` table for a row (ID=3). The script uses a SQL UPDATE statement to set the contents of T.audio.source.localData in the `audtable` table to be the same as the contents of the `sound` column of the `soundtable` table, which performs the copy operation. The script then sets the properties and updates the time stamp for the new BLOB stored in the ORDAudio object.

```
--copyblob3.sql
--
CONNECT auddemo/auddemo;
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 audtable table containing an audio column defined as
-- an ORDSYS.ORDAudio object type in which the ID=3. 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 ORDSYS.ORDAudio;
   ctx RAW(64) :=NULL;

BEGIN
   UPDATE audtable T SET T.audio.source.localData = (SELECT sound FROM
          soundtable S WHERE S.id = 1) WHERE T.id=3;
   COMMIT;

   SELECT audio INTO obj FROM audtable WHERE id = 3 FOR UPDATE;
   obj.setProperties(ctx);
   obj.setUpdateTime(SYSDATE);
   UPDATE audtable SET audio = obj WHERE id = 3;
   COMMIT;

EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Operation failed');
END;
/
EXECUTE UPDATE_PROC;
```

Script 5: Check the Properties of the Loaded Data in Each Row (chkprop.sql)

This script performs a SELECT operation of two rows (ID=1 and ID=3) of the `audtable` table and gets the audio characteristics of the BLOB data to check that the BLOB data is in fact loaded. The properties of each stored audio clip should be identical.

```
-- chkprop.sql
SET SERVEROUTPUT ON;
--connect auddemo/auddemo
--Query audtable for ORDSYS.ORDAudio content.

DECLARE
  audio ORDSYS.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=3.

  SELECT id, audio INTO idnum, audio FROM audtable WHERE id=1;
  dbms_output.put_line('audio id:          '|| idnum);

  properties_match := audio.checkProperties(ctx);
  IF properties_match THEN DBMS_OUTPUT.PUT_LINE('Check Properties Succeeded');
  END IF;

   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=3.

   SELECT id, audio INTO idnum, audio FROM audtable WHERE id=3;
   dbms_output.put_line('audio id:          '|| idnum);

  properties_match := audio.checkProperties(ctx);
  IF properties_match THEN DBMS_OUTPUT.PUT_LINE('Check Properties Succeeded');
  END IF;

   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 `chkprop.sql` show that the properties are identical for each stored audio clip.

```
audio id:          1
Check Properties Succeeded
audio encoding:        MS_PCM
audio number of channels:  1
audio MIME type:       audio/x-wav
audio file format:     WAVE
BLOB Length:           15932
----------------------------------------------
audio id:          3
Check Properties Succeeded
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.
```

Automated Script (setup_audschema2.sql)

This script runs each of the previous five scripts in the correct order to automate this entire process.

```
-- setup_audschema2.sql
-- Create the auddemo user, a tablespace, and a load directory to
-- hold the BFILE:
@create_auduser.sql

-- Create a soundtable table and populate it with
-- an audio clip:
@create_soundtable.sql

-- Create an audtable table and import an audio clip:
@create_audtable.sql

-- Copy a BLOB into an ORDAudio object, set the properties,
-- and update the time:
@copyblob3.sql

-- Check the properties of the audio clips. The properties
-- should be identical:
```

```
@chkprop.sql

--exit;
```

9.2 Media Data Examples

Media data examples using *inter*Media include the following common operations:

- Using the ORDDoc object type as a repository
- Using a set of scripts for creating and populating a media table from a BFILE data source

Reference information on the methods used in these examples is presented in *Oracle interMedia Reference*.

9.2.1 Using the ORDDoc Object Type as a Repository

The ORDDoc object type is most useful for applications that require the storage of different types of media, such as audio, image, video, and any other type of document in the same column so you can build a common metadata index on all the different types of media and perform searches across different types of media using this index.

Note: You cannot use this same search technique if the different types of media are stored in different types of objects in different columns of relational tables.

Example 9–3 shows how to create a repository of media using the `tdoc` table by running the `createschema.sql` script followed by the `createindex.sql` script.

A requirement for creating the metadata index is to create a primary key constraint on column `n`. After initializing each row, load each row with different media, in this case, rows 1 and 2 with an audio clip, rows 3 and 4 with a video clip, and rows 5 and 6 with an image. For each media file, call the setProperties( ) method after each row is loaded and specify the setComments = `TRUE` value for this parameter to populate the comments attribute of the object with a set of format and application properties in XML form. Because the format of each media type is natively supported by *inter*Media, the setProperties( ) method is used to extract the properties from the media source and the comments field of the object is populated in XML form. If the format of the media type is not known, then the setProperties( ) method raises a DOC_PLUGIN_EXCEPTION exception. *inter*Media does not support any document media file type (html, pdf, doc, and so forth), therefore you must create your own format plug-in in order to extract the media attributes from the media data. After loading the media data, display the MIME type, format, and content length of the `doc` column for each row.

Next, use Oracle Text and create a metadata index on the comments attribute of the `ORDDOC` column. Then, search for the format `MPGA` in the comments attribute of each row; only one row, row 2, returns a match. Finally, perform a substring search of the CLOB comments attribute for row 2 to locate the value `MPGA` (the value is bolded here for contrast). At this point, you can begin to search for other interesting media formats, such as MOOV; or mimeTypes, such as audio/mpeg, and so forth, in the stored rows using the Oracle Text index.

Example 9–3 Build a Repository of Media

```
-- createschema.sql
-- Connect as SYSDBA to create a tablespace and a user.
-- May need to create a temp tablespace for this to work.

CONNECT SYS AS SYSDBA
SET SERVEROUTPUT ON;
SET ECHO ON;

--Create tablespace docrepository.

CREATE TABLESPACE docrepository
     DATAFILE 'docrepos.dbf' SIZE 200M
     MINIMUM EXTENT 64K
     DEFAULT STORAGE (INITIAL 64K NEXT 128K)
     LOGGING;

-- Create a docuser user.
-- Create a temp tablespace if you do not have one.

CREATE USER DOCUSER IDENTIFIED BY DOCUSER
DEFAULT TABLESPACE docrepository
TEMPORARY TABLESPACE temp;

GRANT CONNECT, RESOURCE, CREATE LIBRARY, CTXAPP to docuser;
GRANT CREATE ANY DIRECTORY TO docuser;

GRANT EXECUTE ON CTX_CLS TO docuser;
GRANT EXECUTE ON CTX_DDL TO docuser;
GRANT EXECUTE ON CTX_DOC TO docuser;
GRANT EXECUTE ON CTX_OUTPUT TO docuser;
GRANT EXECUTE ON CTX_QUERY TO docuser;
GRANT EXECUTE ON CTX_REPORT TO docuser;
GRANT EXECUTE ON CTX_THES TO docuser;

-- End of SYSDBA tasks.

-- Begin user tasks.

CONNECT docuser/docuser
SET SERVEROUTPUT ON;
SET ECHO ON;

DROP TABLE tdoc;

-- Create the docdir directory. Replace directory specification with your own.
CREATE OR REPLACE DIRECTORY docdir
              as 'c:\media';
GRANT READ ON DIRECTORY docdir TO PUBLIC WITH GRANT OPTION;
-- Create the tdoc table.
CREATE TABLE tdoc (n NUMBER CONSTRAINT n_pk PRIMARY KEY, doc ORDSYS.ORDDoc)
   STORAGE (INITIAL 100K NEXT 100K PCTINCREASE 0);
INSERT INTO tdoc VALUES(1, ORDSYS.ORDDoc.init());
INSERT INTO tdoc VALUES(2, ORDSYS.ORDDoc.init());
INSERT INTO tdoc VALUES(3, ORDSYS.ORDDOC.init());
INSERT INTO tdoc VALUES(4, ORDSYS.ORDDOC.init());
INSERT INTO tdoc VALUES(5, ORDSYS.ORDDOC.init());
INSERT INTO tdoc VALUES(6, ORDSYS.ORDDOC.init());

DECLARE
  obj ORDSYS.ORDDoc;
  ctx RAW(64) := NULL;
BEGIN
-- This imports the audio file aud1.wav from the docdir directory
-- on a local file system (srcType=file) and sets the properties.
SELECT doc INTO obj FROM tdoc WHERE n = 1 FOR UPDATE;
  obj.setSource('file','DOCDIR','aud1.wav');
  obj.import(ctx,FALSE);
  obj.setProperties(ctx,TRUE);
UPDATE tdoc SET doc = obj WHERE n = 1;
COMMIT;
-- This imports the audio file aud2.mp3 from the docdir directory
-- on a local file system (srcType=file) and sets the properties.
SELECT doc INTO obj FROM tdoc WHERE n = 2 FOR UPDATE;
  obj.setSource('file','DOCDIR','aud2.mp3');
  obj.import(ctx,FALSE);
  obj.setProperties(ctx, TRUE);
UPDATE tdoc SET doc = obj WHERE n = 2;
COMMIT;
-- This imports the video file vid1.mov from the docdir directory
-- on a local file system (srcType=file) and sets the properties.
SELECT doc INTO obj FROM tdoc WHERE n = 3 FOR UPDATE;
  obj.setSource('file','DOCDIR','vid1.mov');
  obj.import(ctx,FALSE);
  obj.setProperties(ctx,TRUE);
UPDATE tdoc SET doc = obj WHERE n = 3;
COMMIT;
-- This imports the video file vid2.mov from the docdir directory
-- on a local file system (srcType=file) and sets the properties.
SELECT doc INTO obj FROM tdoc WHERE n = 4 FOR UPDATE;
  obj.setSource('file','DOCDIR','vid2.mov');
  obj.import(ctx,FALSE);
  obj.setProperties(ctx, TRUE);
UPDATE tdoc SET doc = obj WHERE n = 4;
COMMIT;
-- This imports the image file img71.gif from the docdir directory
-- on a local file system (srcType=file) and sets the properties.
SELECT doc INTO obj FROM tdoc WHERE n = 5 FOR UPDATE;
  obj.setSource('file','DOCDIR','img71.gif');
  obj.import(ctx,FALSE);
  obj.setProperties(ctx, TRUE);
UPDATE tdoc SET doc = obj WHERE n = 5;
COMMIT;
-- This imports the image file img50.gif from the docdir directory
-- on a local file system (srcType=file) and sets the properties.
SELECT doc INTO obj FROM tdoc WHERE n = 6 FOR UPDATE;
  obj.setSource('file','DOCDIR','img50.gif');
  obj.import(ctx,FALSE);
  obj.setProperties(ctx, TRUE);
UPDATE tdoc SET doc = obj WHERE n = 6;
COMMIT;
END;
/
--Display the MIME type, format, and content length of the media.
DECLARE
  doc ORDSYS.ORDDOC;
  idnum integer;
BEGIN
 FOR I IN 1..6 LOOP
  SELECT n, doc into idnum, doc from tdoc where n=I;
   dbms_output.put_line('media n:           '|| idnum);
   dbms_output.put_line('media MIME type:   '|| doc.getMimeType);
   dbms_output.put_line('media file format: '|| doc.getFormat);
   dbms_output.put_line('BLOB length:      '|| TO_CHAR(doc.getContentLength()));
   dbms_output.put_line('-------------------------------------------');
 END loop;
END;
/

-- Display the output.
media n:           1
media MIME type:   audio/x-wav
media file format: WAVE
BLOB length:       93594
-------------------------------------------
media n:           2
media MIME type:   audio/mpeg
media file format: MPGA
BLOB length:       51537
-------------------------------------------
media n:           3
media MIME type:   video/quicktime
media file format: MOOV
BLOB length:       4958415
-------------------------------------------
media n:           4
media MIME type:   video/quicktime
media file format: MOOV
BLOB length:       2891247
-------------------------------------------
media n:           5
media MIME type:   image/gif
media file format: GIFF
BLOB length:       1124
-------------------------------------------
media n:           6
media MIME type:   image/gif
media file format: GIFF
BLOB length:       686
-------------------------------------------
PL/SQL procedure successfully completed.

-- createindex.sql
-- Connect as DOCUSER.
-- Create the index using Oracle Text.
--
```

```
CONNECT DOCUSER/DOCUSER;
SET SERVEROUTPUT ON;
SET ECHO ON;
--
-- Next, you can create a preference, and
-- create media attribute sections for each media attribute,
-- that is, format, mimeType, and contentLength.
--
-- Create a preference.
EXECUTE ctx_ddl.drop_preference('ANNOT_WORDLIST');
EXECUTE ctx_ddl.create_preference('ANNOT_WORDLIST', 'BASIC_WORDLIST');
EXECUTE ctx_ddl.set_attribute('ANNOT_WORDLIST', 'stemmer', 'ENGLISH');
EXECUTE ctx_ddl.set_attribute('ANNOT_WORDLIST', 'fuzzy_match', 'ENGLISH');
-- Create a section group.
-- Define Media Attribute sections, that is, the XML tags for the attributes
-- or samples.
EXECUTE CTX_DDL.DROP_SECTION_GROUP('MEDIAANN_TAGS');
EXECUTE CTX_DDL.CREATE_SECTION_GROUP('MEDIAANN_TAGS','xml_section_group');
EXECUTE CTX_DDL.ADD_ZONE_SECTION('MEDIAANN_TAGS',
'MEDIAFORMATENCODINGTAG','MEDIA_FORMAT_ENCODING_CODE');
EXECUTE CTX_DDL.ADD_ZONE_SECTION('MEDIAANN_TAGS','MEDIASOURCEMIMETYPETAG',
'MEDIA_SOURCE_MIME_TYPE');
EXECUTE CTX_DDL.ADD_ZONE_SECTION('MEDIAANN_TAGS', 'MEDIASIZETAG','MEDIA_SIZE');
--
-- Add the following PARAMETERS clause to the end of the CREATE INDEX statement:
-- PARAMETERS ('section group MEDIAANN_TAGS'), so the statement appears
-- as follows:
DROP INDEX mediaidx FORCE;
--
CREATE INDEX mediaidx ON tdoc(doc.comments) INDEXTYPE IS
     CTXSYS.CONTEXT PARAMETERS('stoplist CTXSYS.EMPTY_STOPLIST wordlist
     ANNOT_WORDLIST filter CTXSYS.NULL_FILTER section group MEDIAANN_TAGS');
COMMIT;
--
-- Now, perform a SELECT statement on the attributes in the doc.comments column.
--
SELECT score(1), n from tdoc t WHERE CONTAINS(t.doc.comments, 'MPGA',1)>0;
-- Should find one row, representing the aud2.mp3 audio file.
-- Display the results, MPGA is found in row 2.

  SCORE(1)          N
---------- ----------
         5          2

-- Look for MPGA in the comments attribute of row 2 (bolded value MPGA).
```

```
SELECT DBMS_LOB.SUBSTR(t.doc.comments, 2000,1) FROM tdoc t WHERE n=2;
-- Display the output.

DBMS_LOB.SUBSTR(T.DOC.COMMENTS,2000,1)
--------------------------------------------------------------------------------
<?xml version="1.0"?>
<!-- Generated by Oracle interMedia Annotator 1.0 -->
<AudioCDTrackAnn dt="oracle.ord.media.annotator.annotations.AudioCDTrackAnn">
  <Attributes>
    <MEDIA_FORMAT_ENCODING desc="Format of the media" dt="java.lang.String"><![C
DATA[Layer III]]></MEDIA_FORMAT_ENCODING>
    <AUDIO_CD_TRACK_ALBUM desc="Audio CD Title" dt="java.lang.String"><![CDATA[N
one]]></AUDIO_CD_TRACK_ALBUM>
    <MEDIA_DURATION desc="Duration in seconds of the media" dt="java.lang.Long">
<![CDATA[4]]></MEDIA_DURATION>
    <MEDIA_BITRATE desc="Bitrate of the media in bits per second" dt="java.lang.
DBMS_LOB.SUBSTR(T.DOC.COMMENTS,2000,1)
--------------------------------------------------------------------------------
Integer"><![CDATA[96000]]></MEDIA_BITRATE>
    <MEDIA_FORMAT_ENCODING_CODE desc="Format of the media in the form of a verbo
se code" dt="java.lang.String"><![CDATA[LAYER3]]></MEDIA_FORMAT_ENCODING_CODE>
    <MEDIA_SOURCE_FILE_FORMAT_CODE desc="Media file format code" dt="java.lang.S
tring"><![CDATA[MPGA]]></MEDIA_SOURCE_FILE_FORMAT_CODE>
    <MEDIA_SOURCE_FILE_FORMAT desc="Media file format" dt="java.lang.String"><![
CDATA[MPEG1 Audio (ISO/IEC 11172-3)]]></MEDIA_SOURCE_FILE_FORMAT>
    <MEDIA_SOURCE_MIME_TYPE desc="MIME Type of the media/its samples" dt="java.l
ang.String"><![CDATA[audio/mpeg]]></MEDIA_SOURCE_MIME_TYPE>
    <AUDIO_ARTIST desc="Main artist for the audio clip" dt="java.lang.String"><!
[CDATA[Oracle]]></AUDIO_ARTIST>
DBMS_LOB.SUBSTR(T.DOC.COMMENTS,2000,1)
--------------------------------------------------------------------------------
    <AUDIO_NUM_CHANNELS desc="The number of audio channels" dt="java.lang.Intege
r"><![CDATA[1]]></AUDIO_NUM_CHANNELS>
    <AUDIO_SAMPLE_RATE desc="Audio sample rate (samples/sec)" dt="java.lang.Inte
ger"><![CDATA[44100]]></AUDIO_SAMPLE_RATE>
    <MEDIA_CONTENT_DATE desc="Creation date for the media content" dt="java.lang
.String"><![CDATA[1999]]></MEDIA_CONTENT_DATE>
    <MEDIA_USER_DATA desc="String containing all user data" dt="java.lang.String
"><![CDATA[Welcome to Oracle 8i64]]></MEDIA_USER_DATA>
    <MEDIA_TITLE desc="Title of the media" dt="java.lang.String"><![CDATA[welcom
e3.mp3]]></MEDIA_TITLE>
  </Attributes>
DBMS_LOB.SUBSTR(T.DOC.COMMENTS,2000,1)
--------------------------------------------------------------------------------
```

```
  <Samples>
  </Samples>
</AudioCDTrackAnn>
```

9.2.2 Scripts for Creating and Populating a Media Table from a BFILE Data Source

The following scripts can be found on the Oracle Technology Network (OTN) Web site

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

These scripts are end-to-end scripts that create and populate a media table from a BFILE data source. You can get to this site by selecting the **Sample Code** icon, then under Oracle Database, select **Oracle *inter*Media** to go to the Oracle *inter*Media Sample Code Web page.

The following set of scripts:

1. Creates a tablespace for the media data, creates a user and grants certain privileges to this new user, and creates a media data load directory (`create_docuser.sql`).
2. Creates a table with two columns, inserts two rows into the table and initializes the object column to empty with a locator (`create_doctable.sql`).
3. Loads the media data with a SELECT FOR UPDATE operation using an import method to import the data from a BFILE (`importdoc.sql`).
4. Performs a check of the properties for the loaded data to ensure that it is really there (`chkprop.sql`).

The fifth script (`setup_docschema.sql`) automates this entire process by running each script in the required order. The last script (`readdoc.sql`) creates a stored procedure that performs a SELECT operation to read a specified amount of media data from the BLOB, beginning at a particular offset, until all the media data is read. To successfully load the media data, you must have a `docdir` directory created on your system. This directory contains the `aud1.wav` and `aud2.mp3` files, which are installed in the <*ORACLE_HOME*>`/ord/aud/demo` directory; this directory path and disk drive must be specified in the CREATE DIRECTORY statement in the `create_docuser.sql` file.

Script 1: Create a Tablespace, Create a Media User, Grant Privileges to the Media User, and Create a Media Data Load Directory (create_ docuser.sql)

This script creates the docdemo tablespace. It contains a data file named `docdemo.dbf` of 200 MB in size, an initial extent of 64 KB, and a next extent of 128 KB, and turns on table logging. Next, the docdemo user is created and given connect, resource, create library, and create directory privileges followed by creating the media data load directory. Before running this script, you must change the create directory line to point to your data load directory location.

Note: You must edit the `create_docuser.sql` file and either enter the SYS password in the CONNECT statement or comment out the CONNECT statement and run this file as SYS AS SYSDBA. You must specify the disk drive in the CREATE DIRECTORY statement. Also, create the temp temporary tablespace if you have not already created it, otherwise this file will not run.

```
-- create_docuser.sql
-- Connect as admin.
connect SYS AS SYSDBA/<SYS password>;

-- Edit this script and either enter your sys password here
-- to replace <SYS password> or comment out this CONNECT
-- statement and connect as SYS AS SYSDBA before running this script.

set serveroutput on
set echo on

-- You need SYSDBA privileges to delete a user.
-- Note: There is no need to delete docdemo user if you do not delete
-- the docdemo tablespace, therefore comment out the next line.

-- DROP USER docdemo CASCADE;

-- You need SYSDBA privileges to delete a directory. If there is
-- no need to delete it, then comment out the next line.

-- DROP DIRECTORY docdir;

-- Delete and then create a tablespace.
-- Note: It is better to not delete and create tablespaces,
-- so comment this next line out. The CREATE TABLESPACE statement
-- will fail if it already exists.

-- DROP TABLESPACE docdemo INCLUDING CONTENTS;

-- If you uncomment the preceding line and really want to delete the
-- docdemo tablespace, remember to manually delete the docdemo.dbf
-- file to complete this operation. Otherwise, you cannot create
-- the docdemo tablespace again because the docdemo.dbf file
-- already exists. Therefore, it might be best to create this tablespace
-- once and not delete it.

create tablespace docdemo
       datafile 'docdemo.dbf' size 200M
       minimum extent 64K
       default storage (initial 64K next 128K)
       logging;

-- Create the docdemo user.
create user docdemo identified by docdemo
default tablespace docdemo
temporary tablespace temp;

-- Note: If you do not have a temp tablespace already defined, you will have to
-- create it first for this script to work.

grant connect, resource, create library to docdemo;
grant create any directory to docdemo;

-- Note: If this user already exists, you will get an error message
-- when you try to create this user again.

-- Connect as docdemo.
connect docdemo/docdemo

-- Create the docdir load directory; this is the directory where the media
-- files are residing. Replace directory specification with your own.

create or replace directory docdir
       as 'e:\oracle\ord\aud\demo';
grant read on directory docdir to public with grant option;
-- Note for Solaris, the directory specification could be '/user/local'
```

Script 2: Create the Media Table and Initialize the Column Object (create_doctable.sql)

This script creates the media table and then performs an INSERT operation to initialize the column object to empty for two rows. Initializing the column object creates the BLOB locator that is required for populating each row with BLOB data in a subsequent data load operation.

```
--create_doctable.sql

connect docdemo/docdemo;
set serveroutput on
set echo on

drop table doctable;
create table doctable (id number,
       Document ordsys.ordDoc);

-- Insert a row with an empty BLOB.
insert into doctable values(1,ORDSYS.ORDDoc.init());

-- Insert a row with an empty BLOB.
insert into doctable values(2,ORDSYS.ORDDoc.init());
commit;
```

Script 3: Load the Media Data (importdoc.sql)

This script performs a SELECT FOR UPDATE operation to load the media data by first setting the source for loading the media data from a file, importing the data, setting the properties for the BLOB data, updating the row, and committing the transaction. To successfully run this script, you must copy two media files to your `docdir` directory using the names specified in this script, or modify this script to match the file names of your media.

```
-- importdoc.sql

set serveroutput on
set echo on
-- Import two files into the database.

DECLARE
  obj ORDSYS.ORDDOC;
  ctx RAW(64) := NULL;

BEGIN
-- This imports the audio file aud1.wav from the DOCDIR directory
-- on a local file system (srcType=file) and sets the properties.

  select Document into obj from doctable where id = 1 for update;
  obj.setSource('file','DOCDIR','aud1.wav');
  obj.import(ctx,TRUE);
  update doctable set document = obj where id = 1;
  commit;

-- This imports the audio file aud2.mp3 from the DOCDIR directory
-- on a local file system (srcType=file) and sets the properties.

  select Document into obj from doctable where id = 2 for update;
  obj.setSource('file','DOCDIR','aud2.mp3');
  obj.import(ctx,TRUE);
  update doctable set document = obj where id = 2;
  commit;
END;
/
```

Script 4: Check the Properties of the Loaded Data (chkprop.sql)

This script performs a SELECT operation of the rows of the media table, then gets the media characteristics of the BLOB data to check that the BLOB data is in fact loaded.

```
--chkprop.sql
connect docdemo/docdemo
set serveroutput on;
--Query doctable for ORDSYS.ORDDoc.
DECLARE
  document ORDSYS.ORDDoc;
  idnum integer;
  properties_match BOOLEAN;
  ctx RAW(64) := NULL;

BEGIN
 FOR I IN 1..2 LOOP
  SELECT id, document into idnum, document from doctable 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;
/
```

Results from running the script `chkprop.sql` are the following:

```
SQL> @chkprop.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.
```

Automated Script (setup_docschema.sql)

This script runs each of the previous four scripts in the correct order to automate this entire process.

```
--setup_docschema.sql
-- Create the docdemo user, tablespace, and load directory to
-- hold the media files:
@create_docuser.sql

-- Create the media table:
@create_doctable.sql

--Import 2 media clips and set properties:
@importdoc.sql

--Check the properties of the media clips:
@chkprop.sql

--exit;
```

Read Data from the BLOB (readdoc.sql)

This script creates a stored procedure that performs a SELECT operation to read a specified amount of media data from the BLOB, beginning at a particular offset, until all the media data is read.

```
--readdoc.sql
```

```
set serveroutput on
set echo on

create or replace procedure readdocument as

   obj ORDSYS.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 doctable 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.
```

9.3 Image Data Examples

Image data examples using *inter*Media include the following common operations:

- Using a set of scripts for creating and populating an image table from a BFILE data source
- Using a set of scripts for creating and populating an image table from an HTTP data source
- Addressing globalization support issues

9.3.1 Scripts for Creating and Populating an Image Table from a BFILE Data Source

The following scripts can be found on the Oracle Technology Network (OTN) Web site

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

These scripts are as end-to-end scripts that create and populate an image table from a BFILE data source. You can get to this site by selecting the **Sample Code** icon, then under Oracle Database, select **Oracle *inter*Media** to go to the Oracle *inter*Media Sample Code Web page.

The following set of scripts:

1. Creates a tablespace for the image data, creates a user and grants certain privileges to this new user, creates an image data load directory (`create_ imguser.sql`).

2. Creates a table with two columns, inserts two rows into the table and initializes the object column to empty with a locator (`create_imgtable.sql`).
3. Loads the image data with a SELECT FOR UPDATE operation using an import method to import the data from a BFILE (`importimg.sql`).
4. Performs a check of the properties for the loaded data to ensure that it is really there (`chkprop.sql`).

The fifth script (`setup_imgschema.sql`) automates this entire process by running each script in the required order. The last script (`readimage.sql`) creates a stored procedure that performs a SELECT operation to read a specified amount of image data from the BLOB beginning at a particular offset until all the image data is read. To successfully load the image data, you must have an `imgdir` directory created on your system containing the `img71.gif` and `img50.gif` files, which are installed in the <*ORACLE_HOME*>`/ord/img/demo` directory; this directory path and disk drive must be specified in the CREATE DIRECTORY statement in the `create_imguser.sql` file.

Script 1: Create a Tablespace, Create an Image User, Grant Privileges to the Image User, and Create an Image Data Load Directory (create_imguser.sql)

This script creates the imgdemo tablespace with a data file named `imgdemo.dbf` of 200 MB in size, with an initial extent of 64 KB, a next extent of 128 KB, and turns on table logging. Next, the imgdemo user is created and given connect, resource, create library, and create directory privileges, followed by creating the image data load directory.

Note: You must edit the `create_imguser.sql` file and either enter the SYS password in the CONNECT statement or comment out the CONNECT statement and run this file as SYS AS SYSDBA. You must specify the disk drive in the CREATE DIRECTORY statement. Also, create the temp temporary tablespace if you have not already created it, otherwise this file will not run.

```
-- create_imguser.sql
-- Connect as admin.
connect SYS AS SYSDBA/<SYS password>;
-- Edit this script and either enter your SYS password here
-- to replace <SYS password> or comment out this CONNECT
-- statement and connect as SYS AS SYSDBA before running this script.
```

```
set serveroutput on
set echo on

-- You need SYSDBA privileges to delete a user.
-- Note: There is no need to delete imgdemo user if you do not delete the
-- imgdemo tablespace, therefore comment out the next line.

-- DROP USER imgdemo CASCADE;

-- You need SYSDBA privileges to delete a directory. If there is
-- no need to really delete it, then comment out the next line.

-- DROP DIRECTORY imgdir;

-- Delete, then create the tablespace.

-- Note: It is better to not delete and create tablespaces,
-- so comment this next line out. The CREATE TABLESPACE statement
-- will fail if it already exists.

-- DROP TABLESPACE imgdemo INCLUDING CONTENTS;

-- If you uncomment the preceding line and really want to delete the
-- imgdemo tablespace, remember to manually delete the imgdemo.dbf
-- file to complete the operation. Otherwise, you cannot create
-- the imgdemo tablespace again because the imgdemo.dbf file
-- already exists. Therefore, it might be best to create this
-- tablespace once and not delete it.

-- Create the tablespace.
create tablespace imgdemo
       datafile 'imgdemo.dbf' size 200M
       minimum extent 64K
       default storage (initial 64K next 128K)
       logging;

-- Create the imgdemo user.
create user imgdemo identified by imgdemo
default tablespace imgdemo
temporary tablespace temp;

-- Note: If you do not have a temp tablespace already defined, you will
-- have to create it first for this script to work.
```

```
grant connect, resource, create library to imgdemo;
grant create any directory to imgdemo;

-- Note: If this user already exists, you will get an error message when you
-- try to create this user again.

-- Connect as imgdemo.
connect imgdemo/imgdemo

-- Create the imgdir load directory; this is the directory where the image
-- files are residing. Replace directory specification with your own.

create or replace directory imgdir
       as 'e:\oracle\ord\img\demo';
grant read on directory imgdir to public with grant option;
```

Script 2: Create the Image Table and Initialize the Column Object (create_imgtable.sql)

This script creates the image table and then performs an INSERT operation to initialize the column object to empty for two rows. Initializing the column object creates the BLOB locator that is required for populating each row with BLOB data in a subsequent data load operation.

```
-- create_imgtable.sql
connect imgdemo/imgdemo;
set serveroutput on
set echo on

drop table imgtable;
create table imgtable (id number,
       Image ordsys.ordImage);

-- Insert a row with an empty BLOB.
insert into imgtable values(1,ORDSYS.ORDImage.init());

-- Insert a row with an empty BLOB.
insert into imgtable values(2,ORDSYS.ORDImage.init());
commit;
```

Script 3: Load the Image Data (importimg.sql)

This script performs a SELECT FOR UPDATE operation to load the image data by first setting the source for loading the image data from a file, importing the data, setting the properties for the BLOB data, updating the row, and committing the transaction. To successfully run this script, you must copy two image files to your `imgdir` directory using the names specified in this script, or modify this script to match the file names of your image files.

```
--importimg.sql
set serveroutput on
set echo on
-- Import the two files into the database.

DECLARE
  obj ORDSYS.ORDIMAGE;
  ctx RAW(64) := NULL;
BEGIN
-- This imports the image file img71.gif from the IMGDIR directory
-- on a local file system (srcType=file) and sets the properties.

  select Image into obj from imgtable where id = 1 for update;
  obj.setSource('file','IMGDIR','img71.gif');
  obj.import(ctx);

  update imgtable set image = obj where id = 1;
  commit;

-- This imports the image file img50.gif from the IMGDIR directory
-- on a local file system (srcType=file) and sets the properties.

  select Image into obj from imgtable where id = 2 for update;
  obj.setSource('file','IMGDIR','img50.gif');
  obj.import(ctx);

  update imgtable set image = obj where id = 2;
  commit;
END;
/
```

Script 4: Check the Properties of the Loaded Data (chkprop.sql)

This script performs a SELECT operation of the rows of the image table, then gets the image characteristics of the BLOB data to check that the BLOB data is in fact loaded.

```
-- chkprop.sql
connect imgdemo/imgdemo
set serveroutput on;
--Query imgtable for ORDSYS.ORDImage.
DECLARE
  image ORDSYS.ORDImage;
  idnum integer;
  properties_match BOOLEAN;

BEGIN
 FOR I IN 1..2 LOOP
  SELECT id, image into idnum, image from imgtable where id=I;
   dbms_output.put_line('image id:          '|| idnum);
  properties_match := image.checkProperties();
  IF properties_match THEN DBMS_OUTPUT.PUT_LINE('Check Properties Succeeded');
  END IF;

   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;
/
```

Results from running the script chkprop.sql are the following:

```
SQL> @chkprop.sql
image id:          1
Check Properties Succeeded
image height:      15
image width:       43
image MIME type:   image/gif
image file format: GIFF
BLOB Length:       1124
-------------------------------------------
image id:          2
Check Properties Succeeded
image height:      32
image width:       110
image MIME type:   image/gif
image file format: GIFF
BLOB Length:       686
```

```
-------------------------------------------

PL/SQL procedure successfully completed.
```

Automated Script (setup_imgschema.sql)

This script runs each of the previous four scripts in the correct order to automate this entire process.

```
-- setup_imgschema.sql
-- Create imgdemo user, tablespace, and load directory to
-- hold image files:
@create_imguser.sql

-- Create image table:
@create_imgtable.sql

--Import 2 images and set properties:
@importimg.sql

--Check the properties of the images:
@chkprop.sql

--exit;
```

Read Data from the BLOB (readimage.sql)

This script performs a SELECT operation to read a specified amount of image data from the BLOB, beginning at a particular offset until all the image data is read.

```
-- readimage.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 ORDSYS.ORDImage;
   amt BINARY_INTEGER := 32767;
   pos integer := 1;
   read_cnt integer := 1;

BEGIN

   Select  t.image.getcontent() into src from imgtable t where t.id = 1;
   Select image into obj from imgtable 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 readimage;
Content length is: 1124
start position: 1
doing read  1
----------------
End of data

PL/SQL procedure successfully completed.
```

9.3.2 Scripts for Populating an Image Table from an HTTP Data Source

The following scripts can be found on the Oracle Technology Network (OTN) Web site

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

These scripts are end-to-end scripts that create and populate an image table from an HTTP data source. You can get to this site by selecting the **Sample Code** icon, then under Oracle Database, select **Oracle *inter*Media** to go to the Oracle *inter*Media Sample Code Web page.

> **Note:** Before you run the `importimg.sql` script described in this section to load image data from an HTTP data source, check to ensure you have already run the `create_imguser.sql` and `create_imgtable.sql` scripts described in Section 9.3.1.

The following set of scripts performs a row insert operation and an import operation, then checks the properties of the loaded images to ensure that the images are really loaded.

Initialize the Column Object and Import the Image Data (importimghttp.sql)

This script inserts two rows into the `imgtable` table, initializing the object column for each row to empty with a locator, and indicating the HTTP source information (source type (HTTP), URL location, and HTTP object name). Within a SELECT FOR UPDATE statement, an import operation loads each image object into the database followed by an UPDATE statement to update the object attributes for each image, and finally a COMMIT statement to commit the transaction.

To successfully run this script, you must modify this script to point to two images located on your own Web site.

```
--importimghttp.sql
-- Import the two HTTP images from a Web site into the database.
-- Running this script assumes you have already run the
-- create_imguser.sql and create_imgtable.sql scripts.
-- 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.

connect imgdemo/imgdemo;

-- Insert two rows with an empty BLOB.

insert into imgtable values (7,ORDSYS.ORDImage.init(
             'http','your.web.site.com/intermedia','image1.gif'));

insert into imgtable values (8,ORDSYS.ORDImage.init(
             'http','your.web.site.com/intermedia','image2.gif'));

DECLARE
  obj ORDSYS.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 imgtable where id = 7 for update;
    obj.import(ctx);

  update imgtable 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 imgtable where id = 8 for update;
    obj.import(ctx);

  update imgtable set image = obj where id = 8;
  commit;
END;
/
```

Check the Properties of the Loaded Data

This script performs a SELECT operation of the rows of the image table, then gets the image characteristics of the BLOB data to check that the BLOB data is in fact loaded.

```
--chkprop.sql
set serveroutput on;
--connect imgdemo/imgdemo
--Query imgtable for ORDSYS.ORDImage.
```

```
DECLARE
    image ORDSYS.ORDImage;
    idnum integer;
    properties_match BOOLEAN;
BEGIN
  FOR I IN 7..8 LOOP
    SELECT id , image into idnum, image from imgtable where id=I;
     dbms_output.put_line('image id: '|| idnum);
    properties_match := image.checkProperties();
    IF properties_match THEN DBMS_OUTPUT.PUT_LINE('Check Properties Succeeded');
    END IF;
     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;
/
```

9.3.3 Addressing Globalization Support Issues

Example 9–4 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 application addresses globalization support issues.

Example 9–4 Address a Globalization Support Issue

```
ALTER SESSION SET NLS_LANGUAGE = FRENCH;
ALTER SESSION SET NLS_TERRITORY = FRANCE;
DECLARE
    myimage ORDSYS.ORDImage;
    mylargeimage ORDSYS.ORDImage;
BEGIN
    SELECT photo, large_photo INTO myimage, mylargeimage
      FROM emp FOR UPDATE;
    myimage.setProperties();
    myimage.ProcessCopy('scale=",75"', mylargeimage);
    UPDATE emp SET photo = myimage, large_photo = mylargeimage;
    COMMIT;
END;
/
```

9.4 Video Data Examples

See *Oracle interMedia Reference* for video data examples.

9.5 Handling Exceptions

Possible errors that can arise during runtime should always be handled in your application in order for the program to continue to operate despite the presence of these errors. In other words, end users should always be able to recover from an error, whenever possible, while running an application and also know what went wrong. This section describes how you can accomplish this task of properly handling errors by showing examples for handling some of the more common *inter*Media and other types of errors in PL/SQL and Java programs. These examples come from the sample applications described in Chapter 3. Also, see *Oracle interMedia Reference* for more examples.

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

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

See Section 9.5.1 for examples of handling exceptions in PL/SQL.

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
}
```

See Section 9.5.2 for examples of handling exceptions in Java.

When you design, code, and debug your application, you will know the places in your program where it is possible for your program to stop processing because it failed to anticipate an error. These are the places where you must add exception handling blocks to handle these instances.

For more information about handling PL/SQL exceptions, see *PL/SQL User's Guide and Reference*. For more information about handling Java exceptions, see *Oracle Database Java Developer's Guide* and *Oracle Database JDBC Developer's Guide and Reference*.

9.5.1 Handling *inter*Media Exceptions in PL/SQL

This section shows examples and describes handling exceptions in the *inter*Media PL/SQL Web Toolkit Photo Album application.

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;
```

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.

Handling Image Processing for Unknown Image Formats

If your program tries to proces 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 *inter*Media 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.

9.5.2 Handling *inter*Media Exceptions in Java

This section shows examples and describes handling exceptions using the `try/catch` block that are in either the *inter*Media Java Servlet Photo Album application or the *inter*Media JavaServer Pages Photo Album application, or are in both applications. In addition, there are several examples of throwing exceptions.

Handling *inter*Media Version Compatibility Initialization

In the getConnection( ) method in both the `PhotoAlbumServlet` class of the *inter*Media Java Servlet Photo Album application and in the `PhotoAlbumBean` class of the *inter*Media JavaServer Pages Photo Album application, when trying to get a free connection from the stack, if the stack is empty, a new connection object is created. Within the `try/catch` block a call is made to the version compatibility initialization method. Making this call on the client-side is recommended to ensure that the application will always work, without upgrading, with any potential future release of *inter*Media, which may have evolved object types. See *Oracle interMedia Java Classes Reference* for more information about the OrdMediaUtil.imCompatibilityInit( ) method. A `catch` block catches any SQL exception and throws a new SQLException, returning a string representation of the object thrown with the toString( ) 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 );
                try
                {
                    OrdMediaUtil.imCompatibilityInit( conn );
                }
                catch ( Exception e )
                {
                    throw new SQLException( e.toString() );
                }
            }
            else
            {
                conn = (OracleConnection)connStack.pop();
            }
        }

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

        return conn;
    }
```

Handling Image Processing for Unknown Image Formats

In the insertNewPhoto( ) method in both the `PhotoAlbumServlet` class of the *inter*Media Java Servlet Photo Album application and in the `PhotoAlbumBean` class of the *inter*Media JavaServer Pages Photo Album application, a new photo 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.

A Sample Programs

Oracle *inter*Media includes a number of scripts and sample programs that you can use. These consist of SQL, OCI, Java, PL/SQL, and ASP/VBScript sample applications (demos).

Sample *inter*Media SQL, Java, and OCI applications are available in the following directories after you install *inter*Media:

```
On UNIX
<ORACLE_HOME>/ord/aud/demo/
<ORACLE_HOME>/ord/doc/demo/
<ORACLE_HOME>/ord/img/demo/
<ORACLE_HOME>/ord/vid/demo/
<ORACLE_HOME>/ord/http/demo/
<ORACLE_HOME>/ord/im/demo/java/

On Windows
<ORACLE_HOME>\ord\aud\demo\
<ORACLE_HOME>\ord\doc\demo\
<ORACLE_HOME>\ord\img\demo\
<ORACLE_HOME>\ord\vid\demo\
<ORACLE_HOME>\ord\http\demo\
<ORACLE_HOME>\ord\im\demo\java\
```

A.1 Sample Audio SQL Scripts

The audio SQL scripts consist of the following files:

- `auddemo.sql` - audio demo that shows features of the audio object including:
  - Checking *inter*Media objects
  - Creating a sample table with audio in it

  - Inserting NULL rows into the audio table
  - Selecting the rows
  - Checking all the audio attributes directly
  - Checking all the audio attributes by calling methods
  - Installing your own format plug-in using the two files, `fplugins.sql` and `fpluginb.sql` described in the next two list items and in Section 7.2.1.3 on how to extend *inter*Media audio features to support a new audio data format

- `fplugins.sql` - demo format plug-in specification that you can use as a guideline to write any audio format plug-in you want to support
- `fpluginb.sql` - demo format plug-in body that you can use as a guideline to write any audio format plug-in you want to support

See the `README.txt` file in the `<`*`ORACLE_HOME`*`>/ord/aud/demo` directory on UNIX and `<`*`ORACLE_HOME`*`>\ord\aud\demo` directory on Windows for requirements and instructions on running this SQL demo.

See Section A.5 for a description of the Java sample application that is provided to help you learn to use the multimedia client-side Java classes so you can build your own applications.

A.2 Sample ORDDoc SQL Scripts

The ORDDoc SQL scripts consist of the following files:

- `docdemo.sql` - ORDDoc demo that shows features of the ORDDoc object.

See the `README.txt` file in the `<`*`ORACLE_HOME`*`>/ord/doc/demo/` directory on UNIX and `<`*`ORACLE_HOME`*`>\ord\doc\demo\` directory on Windows for requirements and instructions on running this SQL demo.

See Section A.5 for a description of the Java sample application that is provided to help you learn to use the multimedia client-side Java classes so you can build your own applications.

A.3 Sample OCI C Program for Modifying Images or Testing Image Installation

Once you have installed *inter*Media, you may choose to run the *inter*Media image OCI C program. This program can also be used as a test to confirm successful installation.

This section describes how to run the *inter*Media image sample program.

The *inter*Media image sample files are located in <*ORACLE_HOME*>`/ord/img/demo` on UNIX and <*ORACLE_HOME*>`\ord\img\demo` on Windows where <*ORACLE_HOME*> is the Oracle home directory.

A.3.1 Sample Program Installation Steps

For *inter*Media image features, see the `README.txt` file at <*ORACLE_HOME*>`/ord/img/demo` (on UNIX), and <*ORACLE_HOME*>`\ord\img\demo` (on Windows), where <*ORACLE_HOME*> is the Oracle home.

A.3.2 Running the Program

The file `imgdemo.c` is a sample program that shows how *inter*Media image features can be used from within a program. The program is written in C and uses OCI, Oracle Call Interface, to access the database and use *inter*Media image features.

The program operates on `imgdemo.dat`, 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 program. In either case, once the image has been manipulated by *inter*Media, the resulting image is written to the file `imgdemo.out` and can then be viewed with common rendering tools that you supply.

When the program is run, it deletes and re-creates a table named `IMGDEMOTAB` in the `SCOTT/TIGER` schema of the default database. This table is used to hold the program 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.

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 program 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`.

Upon completion, the program leaves the `imgdemo.out` file in the current directory. It also leaves the table `IMGDEMOTAB` in the `SCOTT/TIGER` schema of the database.

Execute the program by typing `imgdemo` on the command line.

Use the command shown in Example A–1.

Example A–1 Execute the Sample Program from the Command Line

```
$ imgdemo <optional-image-filename>
```

The program 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 program encounters any errors, it is likely that either *inter*Media software has not been installed correctly, or the database has not been started. If the program 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.

See Section A.5 for a description of the Java sample application that is provided to help you learn to use the multimedia client-side Java classes so you can build your own applications.

A.4 Sample Video SQL Scripts

The video SQL scripts consist of the following files:

- `viddemo.sql` - video demo that shows features of the video object including:
  - Checking *inter*Media objects
  - Creating a sample table with video in it
  - Inserting NULL rows into the video table
  - Selecting the rows
  - Checking all the video attributes directly
  - Checking all the video attributes by calling methods
  - Installing your own format plug-in using the two files, `fplugins.sql` and `fpluginb.sql` described in the next two list items and in Section 7.2.3.3 on how to extend *inter*Media video features to support a new video data format
- `fplugins.sql` - demo format plug-in specification that you can use as a guideline to write any video format plug-in you want to support
- `fpluginb.sql` - demo format plug-in body that you can use as a guideline to write any video format plug-in you want to support

See the `README.txt` file in the <*ORACLE_HOME*>`/ord/vid/demo` directory on UNIX and <*ORACLE_HOME*>`\ord\vid\demo` directory on Windows for requirements and instructions on how to run this SQL demo.

See Section A.5 for a description of the Java sample application that is provided to help you learn to use the multimedia client-side Java classes so you can build your own applications.

A.5 Java Sample Applications

An IMExample Java sample 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 applications. In this sample application, the sample schema is used to demonstrate the use of the OrdAudio, OrdVideo, OrdImage, and OrdDoc Java objects.

See the `README.txt` file in the <*ORACLE_HOME*>`/ord/im/demo/java` directory on UNIX and <*ORACLE_HOME*>`\ord\im\demo\java` directory on Windows for requirements and instructions on how to run this Java sample application. See Chapter 4 for a description of this Java sample application. See *Oracle interMedia Java Classes Reference* for information about using Oracle *inter*Media Java Classes.

The IMExample Java sample application files are located in:

`<ORACLE_HOME>/ord/im/demo/java` (on UNIX)

`<ORACLE_HOME>\ord\im\demo\java` (on Windows)

The *inter*Media Java Servlet Photo Album application shows how to use *inter*Media Java Classes for servlets and JSP to upload and retrieve multimedia data. See the `README.txt` file at:

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

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

The *inter*Media JavaServer Pages Photo Album application shows how to use *inter*Media Java Classes for servlets and JSP to upload and retrieve multimedia data. See the `README.txt` file at:

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

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

A.6 Additional PL/SQL Sample Packages

Two additional PL/SQL sample application packages are available after installing *inter*Media. These packages include:

- Oracle *inter*Media PL/SQL Web Toolkit Photo Album application

  The *inter*Media 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 at:

  `<ORACLE_HOME>/ord/http/demo/plsqlwtk` (on UNIX)

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

  See Section 3.1.1 for more information about installing and using this application.

- Oracle *inter*Media Code Wizard for the PL/SQL Gateway

  The *inter*Media Code Wizard 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 *inter*Media object types. The SQL scripts and `README.txt` file are at:

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

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

See Section 3.2 for more information about installing and using this application.

A.7 Additional ASP/VBScript Sample Application

The *inter*Media ASP/VBScript Photo Album application illustrates how to upload and retrieve multimedia data with an ASP/VBScript application. See the `README.txt` file at:

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

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

A.8 Other Sample Programs

See the program examples available from the *inter*Media Web page on the Oracle Technology Network at

`http://otn.oracle.com/sample_code/products/intermedia/content.html`

Sample SQL scripts that demonstrate how to set up a schema on your database are also included on the Oracle Technology Network Web site.

B

Installing and Upgrading Oracle *inter*Media

This appendix describes the manual installation of Oracle *inter*Media (see Section B.1) as well as the manual upgrading of an installed version of Oracle *inter*Media (see Section B.2).

Note: See the *inter*Media `README.txt` file located in *<ORACLE_HOME>*`/ord/im/admin` on UNIX systems or *<ORACLE_HOME>*`\ord\im\admin` on Windows systems for the latest information.

B.1 Installing Oracle *inter*Media

Oracle *inter*Media is installed and configured with Oracle Database 10*g*. If, for some reason, you need to install *inter*Media manually, you can follow the instructions in this section, according to the following topics:

- Installation decisions
- Preinstallation steps
- Installation steps

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 *inter*Media users (`ORDSYS`, `ORDPLUGINS`, and `SI_INFORMTN_SCHEMA`), and which tablespace to use for the

Spatial/*inter*Media Location Services user (MDSYS). Oracle Corporation 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 into 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 *inter*Media packages and objects installed in the ORDSYS, ORDPLUGINS, and SI_INFORMTN_SCHEMA schemas.

B.1.2 Preinstallation Steps

Perform the following preinstallation tasks prior to manually installing and configuring *inter*Media. For instructions, see *Oracle Installation Guide* for your operating system:

1. Install Oracle Database 10*g* Release 1 (10.1), including PL/SQL and Oracle JVM.
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 configuration steps. Remember, you need to do this only if you are configuring *inter*Media manually. You do *not* need to do this 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 *inter*Media 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 UNIX)
        @<ORACLE_HOME>\ord\admin\ordinst.sql SYSAUX SYSAUX (on Windows)
   ```

3. Install *inter*Media types and packages.

   ```
   SQL> @<ORACLE_HOME>/ord/im/admin/iminst.sql (on UNIX)
        @<ORACLE_HOME>\ord\im\admin\iminst.sql (on Windows)
   ```

4. Start the listener.

   The listener must be configured to use external procedure calls. Check your `tnsnames.ora` file for an entry called `extproc_connection_data` and in the `listener.ora` file for an entry called `extproc`.

   See your network documentation for details. If this is not done properly, *inter*Media will *not* work for all supported formats.

Once these mandatory installation steps have been completed, *inter*Media is ready for use.

B.2 Upgrading an Installed Version of Oracle *inter*Media

If you upgrade a database from an earlier release of Oracle Database to Oracle Database 10*g*, *inter*Media 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 *inter*Media

After installing or upgrading *inter*Media, you can verify the *inter*Media installation by invoking the *inter*Media check script.

To run the *inter*Media check script, connect as SYSDBA and invoke `imchk.sql` as follows:

1. Start SQL*Plus.

   ```
   % sqlplus
   ```

2. Connect as SYSDBA.

   ```
   SQL> connect / as SYSDBA
   ```

3. Invoke `imchk.sql`.

   ```
   On UNIX
   SQL> @<ORACLE_HOME>/ord/im/admin/imchk.sql

   On Windows
   SQL> @<ORACLE_HOME>\ord\im/admin\imchk.sql
   ```

The check script will produce the list of *inter*Media components and their status values, and a summary line indicating whether or not the *inter*Media installation is valid. All *inter*Media components are expected to have the status `VALID`.

B.4 Downgrading an Installed Version of Oracle *inter*Media

Oracle *inter*Media is automatically downgraded when you downgrade a database with the *inter*Media feature installed. See *Oracle Database Upgrade Guide* for detailed instructions.

Index