# Oracle® *inter*Media Java Classes

User's Guide and Reference

Release 9.2

March 2002

Part No.  A96121-01

Oracle *inter*Media Java Classes is a component of Oracle *inter*Media, a product designed to manage multimedia Web content within an Oracle database.

Oracle *inter*Media Java Classes User’s Guide and Reference, Release 9.2

Part No. A96121-01

Primary Author: Sue Pelski

Contributors: Sue Mavris, Simon Oxbury, Manjari Yalavarthy, Susan Shepard, Rod Ward

# Contents

## List of Examples

# Send Us Your Comments

**Oracle *inter*Media Java Classes User's Guide and Reference, Release 9.2**

**Part No. A96121-01**

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 03062-2804
  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 Java Classes.

## Audience

This guide is for developers or database administrators who are interested in storing, retrieving, and manipulating media data in an Oracle database, including developers of multimedia specialization applications. Users of this guide should have experience with Java and JDBC.

## 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:

| Book Element | Description |
|---|---|
| Chapter 1 | Contains a general introduction. |
| Chapter 2 | Contains information on the examples included with the Java Classes installation. |
| Chapter 3 | Contains reference information on the OrdAudio class. |
| Chapter 4 | Contains reference information on the OrdDoc class. |
| Chapter 5 | Contains reference information on the OrdImage class. |
| Chapter 6 | Contains reference information on the OrdImageSignature class. |
| Chapter 7 | Contains reference information on input and output streams designed to work with JAI. |
| Chapter 8 | Contains reference information on the OrdVideo class. |
| Chapter 9 | Contains reference information on Java classes for servlets and JSPs. |
| Appendix A | Contains information on running the sample files included with the Java Classes. |
| Appendix B | Contains information on possible exceptions and errors. |
| Appendix C | Contains information on methods that have been deprecated. |

## Related Documentation

This guide is not intended as a standalone document.  It is a supplement to *Oracle interMedia User’s Guide and Reference*. You need both guides to successfully perform operations on *inter*Media objects using the Java interface.

For more information about using *inter*Media in a development environment, see the following documents in the Oracle documentation set:

- *Oracle Call Interface Programmer's Guide*
- *Oracle9i Database Concepts*
- *PL/SQL User's Guide and Reference*

For more information on using JDBC, see *Oracle9i JDBC Developer's Guide and Reference*.

For reference information in Javadoc format, see the Oracle API documentation (also known as Javadoc). The following directory includes ZIP files that contain reference information for Oracle *inter*Media Java Classes and Oracle *inter*Media Java Classes for Servlets and JSPs in Javadoc format:

`<ORACLE_HOME>/ord/im/javadoc (on UNIX)`

`<ORACLE_HOME>\ord\im\javadoc (on Windows NT)`

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`

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 the following location:

`<`*`Oracle_Home`*`>/ord/im/admin/README.txt`

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

For the latest documentation, see the Oracle Technology Network Web site at

`http://otn.oracle.com`

In North America, printed documentation is available for sale in the Oracle Store at

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

Customers in Europe, the Middle East, and Africa (EMEA) can purchase documentation from

`http://www.oraclebookshop.com/`

Other customers can contact their Oracle representative to purchase printed documentation.

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

`http://otn.oracle.com/admin/account/membership.html`

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/docs/index.htm`

To access the database documentation search engine directly, go to

`http://tahiti.oracle.com`

## Conventions

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.

The java.lang.String object is sometimes abbreviated as String.

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 |
|---|---|
| .<br>.<br>. | 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.<br><br>In code examples, a boldface number in brackets (for example, **`[1]`**) indicates that particular code will be described in more detail in the subsequent numbered list. |
| *italic text* | Italic text is used for emphasis and for book titles. |
| < > | Angle brackets enclose user-supplied names. |

| Convention | Meaning |
|---|---|
| [ ] | Brackets enclose optional clauses from which you can choose one or none. |

# 1 Introduction

Oracle *inter*Media provides Java Classes to enable users to write Java applications using *inter*Media objects.

## 1.1 Oracle *inter*Media Objects

The capabilities of *inter*Media objects include the storage, retrieval, management, and manipulation of media data managed by Oracle9*i*. Oracle *inter*Media supports multimedia storage, retrieval, and management of the following:

- Binary large objects (BLOBs) stored locally in an Oracle9*i* database and containing media data
- File-based large objects, or BFILEs, stored locally in operating system-specific file systems and containing media data
- URLs containing media data stored on any HTTP server such as Oracle9*i* Application Server, Netscape Application Server, Microsoft Internet Information Server, Apache HTTPD server, and Spyglass servers
- Data stored on specialized media servers

*inter*Media is a building block for various multimedia applications rather than an end-user application. It consists of object types along with related methods for managing and processing media data. Some example applications that could use *inter*Media objects are the following:

- 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, archiving, and cataloging
- Photograph collections (for example, for professional photographers)
- Internet video stores and digital video-clip previews
- Digital video sources for streaming video delivery systems
- Libraries of digital video training programs

## 1.2 Audio Concepts

This section contains information about digitized audio concepts and using the *inter*Media OrdAudio object to build audio applications or specialized *inter*Media audio objects.

### 1.2.1 Digitized Audio

*inter*Media audio services integrate the storage, retrieval, and management of digitized audio data in Oracle databases using Oracle9*i*.

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 characteristics include format, encoding type, number of channels, sampling rate, sample size, compression type, and audio duration.

### 1.2.2 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. *inter*Media audio services can store and retrieve audio data of any data format, and automatically extract metadata from audio data of a variety of popular audio formats. *inter*Media audio services can also extract application attributes and store them in the comments field of the object in XML form, identical to what is provided by Oracle *inter*Media Annotator. Supported audio attributes depend upon available hardware capabilities or processing power for any user-defined formats. See *Oracle interMedia User's Guide and Reference* for a list of supported data formats from which *inter*Media audio services can extract and store attributes and other audio features.

*inter*Media audio services are extensible and can 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.

## 1.3 Media Concepts

This section contains information about digitized media concepts and using the *inter*Media OrdDoc object to build media applications or specialized *inter*Media objects.

### 1.3.1 Digitized Media Data

*inter*Media media services integrate the storage, retrieval, and management of media files in Oracle databases using Oracle9*i*.

An *inter*Media OrdDoc object can store audio, image, and video data in a single database column. Instead of having separate columns for audio, image, and video objects, you can use one column of *inter*Media OrdDoc objects to represent all types of media.

### 1.3.2 Media Components

Media files consist of the media data (digitized bits) and attributes that describe and characterize the data.

*inter*Media OrdDoc objects can store and retrieve media data of any data format. The OrdDoc object type can be used in applications that require you to store different types of media data (such as audio, image, video, and any other kind of media file) in the same column, so you can build a common metadata index on all the different types of media files. Using this index, you can search across all the different types of files. Note that you cannot use this same search technique if the different types of media data are stored in different types of objects, in separate columns of relational tables.

*inter*Media can automatically extract metadata from a variety of popular audio, image, and video data formats. *inter*Media can also extract application attributes and store them in the comments attribute of the object, in XML form. See Appendix A of *Oracle interMedia User's Guide and Reference* for a list of supported data formats from which *inter*Media can extract and store attributes and other features. *inter*Media is extensible and can be made to recognize and support additional formats.

## 1.4 Image Concepts

This section contains information about digitized image concepts and using the *inter*Media OrdImage object to build image applications or specialized *inter*Media image objects.

### 1.4.1 Digitized Images

*inter*Media image services integrate the storage, retrieval, and management of digitized images in Oracle databases using Oracle9*i*.

*inter*Media image services support 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.

### 1.4.2 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 including 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**.

*inter*Media image services can store or retrieve image data of any data format. *inter*Media image services can process and automatically extract properties of images of a variety of popular formats. See *Oracle interMedia User's Guide and Reference* for a list of supported data formats for which *inter*Media image services can process and extract metadata. In addition, certain foreign images (formats not natively understood by *inter*Media image services) have limited support for image processing.

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 in or 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 use digitized images within these. *inter*Media image services support storage and retrieval of all formats, as well as processing and attribute extraction of many of those formats.

### 1.4.3 Content-Based Retrieval Concepts

Inexpensive image-capture and storage technologies have allowed massive collections of digital images to be created. However, as a database grows, the difficulty of finding relevant images increases. Two general approaches to this problem have been developed, both of which 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 image services, 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 OrdImage object type for the image, 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 *inter*Media image services 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 the following:

- 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 help them in their work.

### 1.4.4 How Content-Based Retrieval Works

A content-based retrieval system processes the information contained in image data and creates an abstraction of that 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 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, but not the locations of colors.
- 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 shapes, 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.

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 a signature be generated for the comparison image. Image matching is based on comparing the signatures. A **score**, which is a weighted sum of the value of each attribute, is used to determine the degree of similarity when images are compared, with a smaller difference reflecting a closer match. The **weights** are positive real numbers whose values reflect how sensitive the matching process for a given attribute should be to the degree of similarity or dissimilarity between two images. If the weighted sum of the differences of the visual attributes is less than or equal to the **threshold** value, the images match; if the weighted sum is greater than the threshold, the images do not match.

Oracle *inter*Media Java Classes provides APIs for the OrdImageSignature object, which is used to store and compare image signature information in a Java object.

For more information on the visual attributes, weights, similarity calculation, threshold values, and indexing, see *Oracle interMedia User's Guide and Reference*.

### 1.4.5 Input and Output Streams

As an extension to Java, Sun has provided the Java Advanced Imaging (JAI) API. JAI lets you introduce advanced image processing operations in your Java applications. With Oracle *inter*Media, you can read and write images stored in the database from your JAI applications.

Oracle *inter*Media Java Classes provides APIs for three types of streams, which let you read data from BLOBs and BFILEs and write to BLOBs in your JAI applications. These stream objects are not meant to replace the input and output stream objects provided by Sun; these objects are included to provide an interface to image data stored in BLOBs and BFILEs in OrdImage objects that can be used by JAI without loss in performance.

See Chapter 7 for more information on these stream objects.

## 1.5 Video Concepts

This section contains information about digitized video concepts and using the *inter*Media OrdVideo object to build video applications or specialized *inter*Media video objects.

### 1.5.1 Digitized Video

*inter*Media video services integrate the storage, retrieval, and management of digitized video data in Oracle databases using Oracle9*i*.

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 that picked up by a video camera or video recorded on magnetic media, and convert it into digital values with specific video characteristics. Such characteristics include format, encoding type, frame rate, frame size (width and height), frame resolution, video length, compression type, number of colors, and bit rate.

### 1.5.2 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, with video data by storing descriptive text in an attribute or column in the database table.

The video data can have different formats, compression types, frame rates, frame sizes, frame resolutions, playing times, compression types, numbers of colors, and bit rates, depending upon how the video data was digitally recorded. *inter*Media video services can store and retrieve video data of any data format, and automatically extract metadata from video data of a variety of popular video formats. *inter*Media video services can also extract application attributes and store them in the comments field of the object, in XML form. Supported video attributes depend upon available hardware capabilities or processing power for any user-defined formats. See *Oracle interMedia User's Guide and Reference* for a list of supported data formats from which *inter*Media video services can extract and store attributes and other video features.

*inter*Media video services are extensible and can support additional video formats.

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.

## 1.6 Java Application Support

Oracle *inter*Media lets you store your multimedia information in a database table, retrieve it from the table, and manipulate it. *inter*Media Java Classes lets you write your own Java applications to use, manipulate, and modify media data stored in an Oracle database.

*inter*Media Java Classes lets an application retrieve an object from a result set and manipulate the contents of the object.

## 1.7 Writing a Java Application to Access *inter*Media Objects

Perform the following operations to write an application that will access *inter*Media objects in an Oracle database table:

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

   Call a getConnection( ) method to obtain an OracleConnection object. See Example 2–3 for an example of a method that makes a connection to the database.

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.

   See Example 2–3 for an example of a method that disables auto-commit mode.

   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. See steps 1 and 2 of Example 2–4.

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

   See Example 2–6.

4. Retrieve the *inter*Media object from the result set.

   See step 5 of Example 2–4 for an example of how to retrieve the *inter*Media object from the result set as an instance of one of the *inter*Media Java classes.

5. Perform operations on the Java application object. See Chapter 2 for examples of the operations you can perform.

   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. See step 6 of Example 2–4 for an example of how to load new data into an 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. See step 9 of Example 2–4.

   Commit the transaction by calling the commit( ) method. See step 4 of Example 2–2.

7. Close the connection to the database table.

   See step 5 of Example 2–2.

   For more information on using JDBC, see *Oracle9i JDBC Developer's Guide and Reference.*

## 1.8 Compatibility with Previous Releases of *inter*Media

Oracle Corporation may improve the *inter*Media object types by adding new object attributes in a future release of *inter*Media. Client-side applications may be able to maintain compatibility with the 9.0.1 release of the *inter*Media object types (OrdAudio, OrdDoc, OrdImage, and OrdVideo), even after a server upgrade that changes the object types, if they make a call to the compatibility initialization function at the beginning of the application.

Client-side applications written in Java using *inter*Media Java Classes should call the OrdMediaUtil.imCompatibilityInit( ) method after connecting to the Oracle database.

```
public static void imCompatibilityInit(OracleConnection con)
    throws Exception
```

This Java function takes an OracleConnection object as an argument.

See step 2 of Example 2–2 for an example of the imCompatibilityInit( ) method.

# 2

# Program Examples Using Java Classes

This chapter provides full-length examples of user-defined classes using *inter*Media Java Classes. Sample SQL scripts that demonstrate how to set up a schema on your database server are also included.

This code will not necessarily match the code shipped as AudioExample.java, DocumentExample.java, ImageExample.java, or VideoExample.java with the *inter*Media Java Classes installation. If you want to run an example on your system, use the files provided with the *inter*Media Java Classes installation; do not attempt to compile and run the code presented in this chapter.

**Note:** This chapter contains examples of Java and SQL code. Some of the code examples display boldface numbers enclosed in brackets; these indicate that further explanation of that code will be in the numbered list immediately following the example.

## 2.1 OrdAudio Example

The audio example (including AudioExample.sql and AudioExample.java) contains user-defined methods that use SQL, JDBC, and *inter*Media Java Classes APIs to perform the following operations:

- Create a database server table that contains test content.
- Load data into both application and database OrdAudio objects from a local file.
- Load data into both application and database OrdAudio objects from a local stream.

- Load data into both application and database OrdAudio objects from a local byte array.
- Extract and print properties from the application OrdAudio object.
- Demonstrate error handling through a failed call to a database method.

### 2.1.1 AudioExample.sql

Example 2–1 shows the complete contents of the AudioExample.sql sample file.

***Example 2–1 Contents of AudioExample.sql***

```
set echo on

-- PLEASE change system password
connect system/manager
drop user  AUDIOUSER cascade;

[1] create user AUDIOUSER identified by AUDIOUSER;
grant connect,resource to AUDIOUSER identified by AUDIOUSER;

[2] connect AUDIOUSER/AUDIOUSER

[3] CREATE TABLE TAUD(n NUMBER, aud ORDSYS.ORDAUDIO);

--
-- Note - the OrdAudio.init method was added in interMedia 8.1.7.
-- If you are running against an older release of interMedia and the
-- Oracle database, you will have to modify the following INSERT statements
-- to use the OrdAudio default constructor.
--
[4] INSERT INTO TAUD VALUES(1, ORDSYS.ORDAudio.init( ));
INSERT INTO TAUD VALUES(2, ORDSYS.ORDAudio.init( ));
INSERT INTO TAUD VALUES(3, ORDSYS.ORDAudio.init( ));
commit;
```

The SQL statements in AudioExample.sql perform the following operations:

1. Create a user named AUDIOUSER and grant the appropriate permissions to the user.
2. Connect to the database server as AUDIOUSER.
3. Create a table named TAUD with two columns: a column of numbers and a column of OrdAudio objects.

**4.** Add three rows to the table, each containing an empty OrdAudio object.

See *Oracle interMedia User's Guide and Reference* for more information on the init method.

## 2.1.2 AudioExample.java

Section 2.1.2.1 through Section 2.1.2.8 show the methods contained in the AudioExample.java sample file.

### 2.1.2.1 main( ) Method

Example 2–2 shows the main( ) method.

***Example 2–2 main( ) Method (Audio)***

```
public static void main (String args[ ]){
     byte[ ] ctx = new byte[4000];
     OracleConnection con = null;
     try {
          AudioExample tk = new AudioExample( );
          [1] con = tk.connect( );
          //Include the following line only if you are running
          //an Oracle 8.1.7 database or later.
          //If you are running a database server prior to 8.1.7,
          //the call will fail.
          [2] OrdMediaUtil.imCompatibilityInit(con);
          [3] tk.loadDataFromFile(con);
          tk.extractProperties(con);
          tk.printProperties(con);
          tk.otherMethods(con);
          tk.loadDataFromStream(con);
          tk.loadDataFromByteArray(con);
          [4] con.commit( );
          [5] con.close( );
          System.out.println("Done.");
     }
     [6] catch (Exception e) {
          try {
               System.out.println("Exception : " + e);
               con.close( );
          }
          catch(Exception ex) {
               System.out.println("Close Connection Exception : " + ex);
          }
     }
}
```

The code in the main( ) method performs the following operations:

1. Uses the connect( ) method to make a connection to a database table.
2. Ensures the compatibility of your application with later releases of the Oracle database. See Section 1.8 for more information.
3. Calls several methods (also defined in AudioExample.java) that manipulate objects on the database server and the local machine.
4. Commits any changes made to the database table.
5. Closes the connection to the database.
6. Handles any errors or exceptions raised by the code.

Section 2.1.2.2 through Section 2.1.2.8 will provide information on the methods called from the main( ) method in the order in which they are called, not in the order they appear in AudioExample.java.

#### 2.1.2.2 connect( ) Method

Example 2–3 shows a user-defined method named connect( ), which makes a connection from the application to the database.

***Example 2–3 connect( ) Method (Audio)***

```
public OracleConnection connect( ) throws Exception {
     String connectString;
     [1] Class.forName ("oracle.jdbc.driver.OracleDriver");
     [2] connectString = "jdbc:oracle:oci8:@";
     [3] OracleConnection con = (OracleConnection)DriverManager.getConnection
          (connectString,"AUDIOUSER","AUDIOUSER");
     [4] con.setAutoCommit(false);
     return con;
}
```

The connect( ) method performs the following operations:

1. Loads the JDBC drivers directly, because the Oracle database uses a JDK-compliant Java Virtual Machine.
2. Defines a string that contains the URL of the database to which you will connect. You may need to change this string to match your database.

3. Sets the connection to the database, using the URL contained in connectString, the user name AUDIOUSER, and the password AUDIOUSER. The user name and password were created by AudioExample.sql.
4. Disables the auto-commit mode. This means that you must commit or roll back manually with the commit( ) or rollback( ) methods, respectively.

#### 2.1.2.3 loadDataFromFile( ) Method

Example 2–4 shows a user-defined method named loadDataFromFile( ), which uses the *inter*Media loadDataFromFile( ) method to populate the application object with media data.

***Example 2–4 loadDataFromFile( ) Method (Audio)***

```
public void loadDataFromFile(OracleConnection con) {
     try {
          [1] Statement s = con.createStatement( );
          [2] OracleResultSet rs = (OracleResultSet) s.executeQuery
               ("select * from TAUD where n = 1 for update ");
          int index = 0;
          [3] while(rs.next( )){
               [4] index = rs.getInt(1);
               [5] OrdAudio audObj = (OrdAudio) rs.getCustomDatum
                    (2, OrdAudio.getFactory( ));
               [6] audObj.loadDataFromFile("testaud.dat");
               [7] audObj.getDataInFile("output1.dat");
               System.out.println("************AFTER getDataInFile ");
               [8] System.out.println(" getContentLength output : " +
                    audObj.getContentLength( ));
               [9] OraclePreparedStatement stmt1 = (OraclePreparedStatement)
                    con.prepareCall("update taud set aud = ? where
                    n = " + index);
               stmt1.setCustomDatum(1,audObj);
               stmt1.execute( );
               stmt1.close( ) ;
          }
          System.out.println("loading successful");
     }
     [10] catch(Exception e) {
          System.out.println("exception raised " + e);
          System.out.println("loading unsuccessful");
     }
}
```

The loadDataFromFile( ) method performs the following operations:

1. Creates an OracleStatement object.
2. Executes the given SQL query and puts the results into a local OracleResultSet object. In this case, the SQL query selects the data in the database row where n=1.
3. Performs the operations in the loop while there are results in the OracleResultSet that have not been processed. However, in this case, there is only one row included in the OracleResultSet, so the operations in the loop will run once.
4. Sets an index variable to the value of the integer in the first column of the first row in the OracleResultSet (in this case, the value is 1).
5. Creates a local OrdAudio object named audObj. Populates audObj with the contents of the OrdAudio object in the second column of the current row in the OracleResultSet.
6. Uses the OrdAudio loadDataFromFile( ) method to load the media data in testaud.dat into the database OrdAudio object and into audObj. This also sets the local field on audObj, but not the database object.
7. Uses the getDataInFile( ) method to get the media data from audObj and load it into a file on the local system named output1.dat.
8. Gets the content length of audObj and prints it to the screen to verify the success of the loading.
9. Creates and executes a SQL statement that will update the database OrdAudio object with the contents of audObj.
10. Handles any errors or exceptions raised by the code.

#### 2.1.2.4 extractProperties( ) Method

Example 2–5 shows a user-defined method named extractProperties( ), which sets the properties in the application object.

***Example 2–5 extractProperties( ) Method (Audio)***

```
public void extractProperties(OracleConnection con){
     byte[ ] ctx[ ] = new byte [4000][1];
     try {
          [1] Statement s = con.createStatement( );
          OracleResultSet rs = (OracleResultSet) s.executeQuery
               ("select * from TAUD where n = 1 for update");
          int index = 0;
          while(rs.next( )){
               index = rs.getInt(1);
               OrdAudio audObj = (OrdAudio) rs.getCustomDatum
                    (2, OrdAudio.getFactory( ));
               [2] audObj.setProperties(ctx);
               System.out.println("set Properties called");
               [3] if(audObj.checkProperties(ctx)){
                    System.out.println("checkProperties called");
                    System.out.println("setProperties successful");
                    System.out.println("checkProperties successful");
                    System.out.println("extraction  successful");
               }
               else{
                    System.out.println("checkProperties called");
                    System.out.println("extraction not successful");
                    System.out.println("checkProperties successful");
               }
               [4] OraclePreparedStatement stmt1 = (OraclePreparedStatement)
                    con.prepareCall("update taud set aud = ?
                    where n = " + index);
               stmt1.setCustomDatum(1,audObj);
               stmt1.execute( );
               stmt1.close( ) ;
          }
          rs.close( );
          s.close( );
     }
     [5] catch(Exception e) {
          System.out.println("exception raised " + e);
          System.out.println("extract properties unsuccessful");
     }
}
```

The extractProperties( ) method performs the following operations:

1. Creates a statement, a local OracleResultSet, and a local OrdAudio object named audObj, and populates audObj with media data through the same process described in steps 1 through 5 of Example 2–4. In this method, you will be operating on the contents of the second column of the row in the database table where n=1.
2. Calls setProperties( ) to extract properties values from the media data and set them in the application OrdAudio object. See "setProperties(byte[ ][ ])" in Chapter 3 for a list of the properties values extracted and set.

3. Calls checkProperties( ) to compare the properties values in the application object with the values in the media data. If all values are the same, checkProperties( ) returns true and the appropriate messages are printed to the screen. If any values differ, checkProperties( ) returns false and the appropriate messages are printed to the screen.
4. Creates and executes a SQL statement that will update the database OrdAudio object with the contents of audObj (including the properties extracted by setProperties( )).
5. Handles any errors or exceptions raised by the code.

#### 2.1.2.5 printProperties( ) Method

Example 2–6 shows a user-defined method named printProperties( ), which prints the attributes of the application object to the screen.

***Example 2–6 printProperties( ) Method (Audio)***

```
public void printProperties(OracleConnection con){
     try {
         [1] Statement s = con.createStatement( );
          OracleResultSet rs = (OracleResultSet)
               s.executeQuery("select * from TAUD where n = 1 ");
          int index = 0;
          while(rs.next( )) {
               index = rs.getInt(1);
               OrdAudio audObj = (OrdAudio) rs.getCustomDatum
                    (2, OrdAudio.getFactory( ));
               [2] System.out.println("format: " + audObj.getFormat( ));
               System.out.println("mimeType: " + audObj.getMimeType( ));
               System.out.println("encoding: " + audObj.getEncoding( ));
               System.out.println("numberOfChannels: " +
                    audObj.getNumberOfChannels( ));
               System.out.println("samplingRate: " +
                    audObj.getSamplingRate( ));
               System.out.println("sampleSize: " + audObj.getSampleSize( ));
               System.out.println("compressionType : " +
                    audObj.getCompressionType( ));
               System.out.println("audioDuration: " +
                    audObj.getAudioDuration( ));
               System.out.println("description: " + audObj.getDescription( ));
          }
     }
     [3] catch(Exception e){
          System.out.println("exception raised " + e);
          System.out.println("print proerties unsuccessful");
     }
}
```

The printProperties( ) method performs the following operations:

1. Creates a statement, a local OracleResultSet, and a local OrdAudio object named audObj, and populates audObj with media data through the same process described in steps 1 through 5 of Example 2–4. In this method, you will be operating on the contents of the second column of the row in the database table where n=1.
2. Gets the values of the properties in audObj and prints them to the screen.
3. Handles any errors or exceptions raised by the code.

#### 2.1.2.6 otherMethods( ) Method

Example 2–7 shows a user-defined method named otherMethods( ), which attempts to use the processSourceCommand( ) method.

***Example 2–7 otherMethods( ) Method (Audio)***

```
public void otherMethods(OracleConnection con){
     byte[ ] ctx[ ] = {new byte[4000]};
     byte[ ] res[ ] = {new byte[20]};
     [1] int suc = 1;
     try {
          [2] Statement s1 = con.createStatement( );
          OracleResultSet rs1 = (OracleResultSet) s1.executeQuery
               ("select * from TAUD where n = 1 for update ");
          int index1 = 0;
          while(rs1.next( )) {
               index1 = rs1.getInt(1);
               OrdAudio audObj = (OrdAudio) rs1.getCustomDatum
                    (2, OrdAudio.getFactory( ));
               [3] try {
                    byte[ ] pSRes = audObj.processSourceCommand(ctx,
                         "", "", res);
                    suc = 0;
               }
               [4] catch (Exception e) {
                    System.out.println("Expected Exception raised in
                         processSourceCommand(...)" );
               }
```

```
                [5] OraclePreparedStatement stmt1 = (OraclePreparedStatement)
                     con.prepareCall("update taud set aud = ? where
                     n = " + index1);
                stmt1.setCustomDatum(1,audObj);
                stmt1.execute( );
                stmt1.close( ) ;
           }
           rs1.close( );
           s1.close( );
      }
      [6] catch(Exception e){
           System.out.println("Exception raised " );
      }
      [7] if(suc == 1)
           System.out.println("other methods successful");
      else
           System.out.println("other methods unsuccessful");
}
```

The otherMethods( ) method performs the following operations:

1. Creates an integer that will be used to indicate the success or failure of the method and sets it initially to 1 (for success).
2. Creates a statement, a local OracleResultSet, and a local OrdAudio object named audObj, and populates audObj with media data through the same process described in steps 1 through 5 of Example 2–4. In this method, you will be operating on the contents of the second column of the row in the database table where n=1.
3. Tries to call processSourceCommand( ) with no value specified for the command to be called on the server side. This should raise an exception, which means the code following the processSourceCommand( ) call will not be run and the code in the catch loop will. If an exception is not raised, then the method has failed and the success indicator is set to 0 (for failure).
4. Prints the expected exception that was raised in step 3.
5. Creates and executes a SQL statement that will update the database OrdAudio object with the contents of audObj.
6. Handles any unexpected errors or exceptions raised by the code.
7. Prints the appropriate message to the screen based on the success or failure of the method.

#### 2.1.2.7 loadDataFromStream( ) Method

Example 2–8 shows a user-defined method named loadDataFromStream( ), which uses the *inter*Media loadDataFromInputStream( ) method to load media data into the application object.

***Example 2–8 loadDataFromStream( ) Method (Audio)***

```
public void loadDataFromStream(OracleConnection con){
     try {
          [1] Statement s = con.createStatement( );
          OracleResultSet rs = (OracleResultSet) s.executeQuery
               ("select * from TAUD where n = 2 for update ");
          int index = 0;
          while(rs.next( )){
               index = rs.getInt(1);
               OrdAudio audObj = (OrdAudio) rs.getCustomDatum
                    (2, OrdAudio.getFactory( ));
               [2] FileInputStream fStream = new
                    FileInputStream("testaud.dat");
               [3] audObj.loadDataFromInputStream(fStream);
               [4] audObj.getDataInFile("output2.dat");
               [5] fStream.close( );
               System.out.println("************AFTER getDataInFile ");
               [6] System.out.println(" getContentLength output : " +
                    audObj.getContentLength( ));
               [7] OraclePreparedStatement stmt1 = (OraclePreparedStatement)
                    con.prepareCall("update taud set aud = ? where
                    n = " + index);
               stmt1.setCustomDatum(1,audObj);
               stmt1.execute( );
               stmt1.close( );
          }
          System.out.println("load data from stream successful");
     }
     [8] catch(Exception e) {
          System.out.println("exception raised " + e);
          System.out.println("load data from stream unsuccessful");
     }
}
```

The loadDataFromStream( ) method performs the following operations:

1. Creates a statement, a local OracleResultSet, and a local OrdAudio object named audObj, and populates audObj with media data through the same

process described in steps 1 through 5 of Example 2–4. In this method, you will be operating on the contents of the second column of the row in the database table where n=2.

2. Creates a new FileInputStream object. This input stream contains the contents of the local file testaud.dat.
3. Uses the loadDataFromInputStream( ) method to load the media data in the input stream into the database OrdAudio object and into audObj. This also sets the local field on audObj, but not the database object.
4. Uses the getDataInFile( ) method to get the media data from the application OrdAudio object and load it into a file on the local system named output2.dat.
5. Closes the local input stream.
6. Gets the content length of audObj and prints it to the screen to verify the success of the loading.
7. Creates and executes a SQL statement that will update the database OrdAudio object with the contents of audObj. This update will set the attributes on the database object to match the application object.
8. Handles any errors or exceptions raised by the code.

### 2.1.2.8 loadDataFromByteArray( ) Method

Example 2–9 shows a user-defined method named loadDataFromByteArray( ), which uses the *inter*Media loadDataFromByteArray( ) method to load media data into the application object.

***Example 2–9 loadDataFromByteArray( ) Method (Audio)***

```
public void loadDataFromByteArray(OracleConnection con){
     try {
          [1] Statement s = con.createStatement( );
          OracleResultSet rs = (OracleResultSet) s.executeQuery
               ("select * from TAUD where n = 3 for update ");
          int index = 0;
          while(rs.next( )) {
               index = rs.getInt(1);
               OrdAudio audObj = (OrdAudio) rs.getCustomDatum
                    (2, OrdAudio.getFactory( ));
               [2] File ff = new File("testaud.dat");
               int fileLength = (int) ff.length( );
               byte[ ] data = new byte[fileLength];
               [3] FileInputStream fStream = new
                    FileInputStream("testaud.dat");
               [4] fStream.read(data,0,fileLength);
               [5] audObj.loadDataFromByteArray(data);
               [6] fStream.close( );
               [7] audObj.getDataInFile("output3.dat");
               [8] byte[ ] resArr = audObj.getDataInByteArray( );
               [9] System.out.println("byte array length : " +
                    resArr.length);
               [10] FileOutputStream outStream = new FileOutputStream
                    ("output4.dat");
               [11] outStream.write(resArr);
               [12] outStream.close( );
               [13] InputStream inpStream = audObj.getDataInStream( );
               int length = 32768;
               byte[ ] tempBuffer = new byte[32768];
               [14] int numRead = inpStream.read(tempBuffer,0,length);
               try {
                    [15] outStream = new FileOutputStream("output5.dat");
                    [16] while (numRead != -1) {
                         [17] if (numRead < 32768) {
                              length = numRead;
                              outStream.write(tempBuffer,0,length);
                              break;
                         }
                         [18] else
                              outStream.write(tempBuffer,0,length);
                         [19] numRead = inpStream.read(tempBuffer,0,length);
                    }
               }
               [20] finally {
                    outStream.close( );
                    inpStream.close( );
               }
               System.out.println("************AFTER getDataInFile ");
               [21] System.out.println("getContentLength output : " +
                    audObj.getContentLength( ));
               [22] OraclePreparedStatement stmt1 = (OraclePreparedStatement)
                    con.prepareCall("update taud set aud = ? where
                    n = " + index);
               stmt1.setCustomDatum(1,audObj);
               stmt1.execute( );
               stmt1.close( ) ;
          }
     }
     [23] catch(Exception e) {
```

```
            System.out.println("exception raised " + e);
            System.out.println("load data from byte array unsuccessful");
        }
    }
```

The loadDataFromByteArray( ) method performs the following operations:

1. Creates a statement, a local OracleResultSet, and a local OrdAudio object named audObj, and populates audObj with media data through the same process described in steps 1 through 5 of Example 2–4. In this method, you will be operating on the contents of the second column of the row in the database table where n=3.
2. Determines the size (in bytes) of the local file testaud.dat and creates a byte array of the same size.
3. Creates a new FileInputStream object. This input stream contains the contents of testaud.dat.
4. Reads the contents of the input stream into the byte array.
5. Uses the loadDataFromByteArray( ) method to load the media data in the byte array into the database OrdAudio object and into audObj. This also sets the local field on audObj, but not the database object.
6. Closes the input stream.
7. Uses the getDataInFile( ) method to get the media data from the application OrdAudio object and load it into a file on the local system named output3.dat.
8. Uses the getDataInByteArray( ) method to get the media data from the application OrdAudio object and load it into a local byte array named resArr.
9. Gets the length of resArr and prints it to the screen to verify the success of the loading.
10. Creates a new FileOutputStream object named outStream. This output stream will write data to a local file named output4.dat.
11. Writes the contents of resArr to output4.dat.
12. Closes the output stream.
13. Creates a new input stream named inpStream. Uses the getDataInStream( ) method to get the media data from the application OrdAudio object and store it in inpStream.
14. Reads 32768 bytes from the beginning (that is, at an offset of 0) of inpStream into the byte array tempBuffer. The integer numRead will be set to the total

number of bytes read, or -1 if the end of the input stream has been reached. In this case, if loading is successful, numRead should be equal to 32768.

15. Re-opens OutStream. In this case, it will write data to a local file named output5.dat.
16. Runs the operations in the while loop if numRead is not equal to -1. The program should enter this loop.
17. Enters the if loop if numRead is less than 32768 (that is, if all the data was read). The if loop will write the number of bytes read into tempBuffer into outStream, and then break out of the loop.
18. Writes 32768 bytes into outStream if numRead is 32768.
19. Attempts to read more data from the input stream into the byte array. If all data has been read successfully, then numRead will be set to -1 and the program will exit the loop. If there is still unread data in the input stream, then it will be read into the byte array and steps 17 and 18 will be repeated.
20. Closes both the input stream and the output stream after exiting the while loop.
21. Gets the content length of audObj and prints it to the screen to verify the success of the loading.
22. Creates and executes a SQL statement that will update the database OrdAudio object with the contents of audObj. This update will set the attributes on the database object to match the application object.
23. Handles any errors or exceptions raised by the code.

## 2.2 OrdDoc Example

The OrdDoc example (including DocumentExample.sql and DocumentExample.java) contains user-defined methods that use SQL, JDBC, and *inter*Media Java Classes APIs to perform the following operations:

- Create a database server table that contains test content.
- Load data into both application and database OrdDoc objects from a local file.
- Load data into both application and database OrdDoc objects from a local stream.
- Load data into both application and database OrdDoc objects from a local byte array.
- Extract and print properties from the application OrdDoc object.

- Demonstrate error handling through a failed call to a database method.

## 2.2.1 DocumentExample.sql

Example 2–10 shows the complete contents of the DocumentExample.sql sample file.

***Example 2–10 Contents of DocumentExample.sql***

```
set echo on

--PLEASE change system password
connect system/manager
drop user DOCUSER cascade;

[1] create user DOCUSER identified by DOCUSER ;
grant connect,resource to DOCUSER identified by DOCUSER;

[2] connect DOCUSER/DOCUSER

[3] CREATE TABLE TDOC(n NUMBER, doc ORDSYS.ORDDOC);

[4] INSERT INTO TDOC VALUES(1, ORDSYS.ORDDoc.init( ));
INSERT INTO TDOC VALUES(2, ORDSYS.ORDDoc.init( ));
INSERT INTO TDOC VALUES(3, ORDSYS.ORDDoc.init( ));

commit;
```

The SQL statements in DocumentExample.sql perform the following operations:

1. Create a user named DOCUSER and grant the appropriate permissions to the user.
2. Connect to the database server as DOCUSER.
3. Create a table named TDOC with two columns: a column of numbers and a column of OrdDoc objects.
4. Add three rows to the table, each containing an empty OrdDoc object.

## 2.2.2 DocumentExample.java

Section 2.2.2.1 through Section 2.2.2.8 show the methods defined in the DocumentExample.java sample file.

#### 2.2.2.1 main( ) Method

Example 2–11 shows the contents of the main( ) method.

***Example 2–11 main( ) Method (Doc)***

```
public static void main (String args[ ]){
     byte[ ] ctx = new byte[4000];
     OracleConnection con = null;
     try {
          DocumentExample tk = new DocumentExample( );
          [1] con = tk.connect( );
          [2] OrdMediaUtil.imCompatibilityInit(con);
          [3] tk.loadDataFromFile(con);
          tk.extractProperties(con);
          tk.printProperties(con);
          tk.loadDataFromStream(con);
          tk.otherMethods(con);
          tk.loadDataFromByteArray(con);
          [4] con.commit( );
          [5] con.close( );
          System.out.println("Done.");
     }
     [6] catch (Exception e) {
          try {
               System.out.println("Exception : " + e);
               con.close( );
          }
          catch(Exception ex) {
          System.out.println("Close Connection Exception : " + ex);
          }
     }
}
```

The code in the main( ) method performs the following operations:

1. Uses the connect( ) method to make a connection to a database table.
2. Ensures the compatibility of your application with later releases of the Oracle database. See Section 1.8 for more information.
3. Calls several methods (also defined in DocumentExample.java) that manipulate objects on the database server and the local machine.
4. Commits any changes made to the database table.
5. Closes the connection to the database.

6. Handles any errors or exceptions raised by the code.

Section 2.2.2.2 through Section 2.2.2.8 will provide information on the methods called from the main( ) method in the order in which they are called, not in the order they appear in DocumentExample.java.

#### 2.2.2.2 connect( ) Method

Example 2–12 shows a user-defined method named connect( ), which makes a connection from an application to a database.

***Example 2–12 connect( ) Method (Doc)***

```
public OracleConnection connect( ) throws Exception{
     String connectString;
     [1] Class.forName ("oracle.jdbc.driver.OracleDriver");
     [2] connectString = "jdbc:oracle:oci8:@";
     [3] OracleConnection con = (OracleConnection)
          DriverManager.getConnection(connectString,"DOCUSER","DOCUSER");
     [4] con.setAutoCommit(false);
     return con;
}
```

The connect( ) method performs the following operations:

1. Loads the JDBC drivers directly, because the Oracle database uses a JDK-compliant Java virtual machine.
2. Defines a string that contains the URL of the database to which you will connect. You may need to change this string to match your database.
3. Sets the connection to the database, using the URL contained in connectString, the user name DOCUSER, and the password DOCUSER. The user name and password were created by DocumentExample.sql.
4. Disables the auto-commit mode. This means that you must commit or roll back manually with the commit( ) or rollback( ) methods, respectively.

#### 2.2.2.3 loadDataFromFile( ) Method

Example 2–13 shows a user-defined method named loadDataFromFile( ), which uses the *inter*Media loadDataFromFile( ) method to populate the application object with media data.

***Example 2–13 loadDataFromFile( ) Method (Doc)***

```
public void loadDataFromFile(OracleConnection con){
     try {
          [1] Statement s = con.createStatement( );
          [2] OracleResultSet rs = (OracleResultSet) s.executeQuery
               ("select * from TDOC where n = 1 for update");
          int index = 0;
          [3] while(rs.next( )){
               [4] index = rs.getInt(1);
               [5] OrdDoc docObj = (OrdDoc) rs.getCustomDatum(2,
                    OrdDoc.getFactory( ));
               [6] docObj.loadDataFromFile("testaud.dat");
               [7] docObj.getDataInFile("output1.dat");
               System.out.println("************AFTER getDataInFile ");
               [8] System.out.println("getContentLength output: " +
                    docObj.getContent( ).length( ));
               [9] OraclePreparedStatement stmt1 = (OraclePreparedStatement)
                    con.prepareCall("update tdoc set doc = ? where n = " +
                    index);
               stmt1.setCustomDatum(1,docObj);
               stmt1.execute( );
               stmt1.close( );
          }
     System.out.println("loading successful");
     }
     [10] catch(Exception e) {
          System.out.println("exception raised " + e);
          System.out.println("loading unsuccessful");
     }
}
```

The loadDataFromFile( ) method performs the following operations:

1. Creates an OracleStatement object.
2. Executes the given SQL query and puts the results into a local OracleResultSet object. In this case, the SQL query selects the data in the database row where n=1.
3. Performs the operations in the loop while there are results in the OracleResultSet that have not been processed. However, in this case, there is only one row included in the OracleResultSet, so the operations in the loop will run once.

4. Sets an index variable to the value of the integer in the first column of the first row in the OracleResultSet (in this case, the value is 1).
5. Creates a local OrdDoc object named docObj. Populates docObj with the contents of the OrdDoc object in the second column of the current row in the OracleResultSet.
6. Uses the OrdDoc loadDataFromFile( ) method to load the media data in testaud.dat into the database OrdDoc object and into docObj. This also sets the local field on docObj, but not the database object.
7. Uses the getDataInFile( ) method to get the media data from docObj and load it into a file on the local system named output1.dat.
8. Gets the content length of docObj and prints it to the screen to verify the success of the loading.
9. Creates and executes a SQL statement that will update the database OrdDoc object with the contents of docObj.
10. Handles any errors or exceptions raised by the code.

### 2.2.2.4 extractProperties( ) Method

Example 2–14 shows a user-defined method named extractProperties( ), which sets the properties in the application object.

***Example 2–14 extractProperties( ) Method (Doc)***

```
public void extractProperties(OracleConnection con){
     byte[ ] ctx[ ] = new byte [4000][1];
     try {
          [1] Statement s = con.createStatement( );
          OracleResultSet rs = (OracleResultSet)
          s.executeQuery("select * from TDOC where n = 1 for update");
          int index = 0;
          while(rs.next( )){
               index = rs.getInt(1);
               OrdDoc docObj = (OrdDoc)rs.getCustomDatum(2,
                    OrdDoc.getFactory( ));
               [2] docObj.setProperties(ctx,false);
               System.out.println("setProperties successful");
               [3] OraclePreparedStatement stmt1 = (OraclePreparedStatement)
                    con.prepareCall("update tdoc set doc = ? where n = " +
                    index);
               stmt1.setCustomDatum(1,docObj);
               stmt1.execute( );
```

```
                stmt1.close( ) ;
           }
           rs.close( );
           s.close( );
      }
      [4] catch(Exception e) {
           System.out.println("exception raised " + e);
           System.out.println("extract prop unsuccessful");
      }
}
```

The extractProperties( ) method performs the following operations:

1. Creates a statement, a local OracleResultSet, and a local OrdDoc object named docObj, and populates docObj with media data through the same process described in steps 1 through 5 of Example 2–13. In this method, you will be operating on the contents of the row where n=1.
2. Calls setProperties to extract properties values from the media data and set them in the application OrdDoc object. See "setProperties( )" in Chapter 4 for a list of the properties values extracted and set.
3. Creates and executes a SQL statement that will update the database OrdDoc object with the contents of docObj (including the properties extracted by setProperties( )).
4. Handles any exceptions or errors raised by the code.

#### 2.2.2.5 printProperties( ) Method

Example 2–15 shows a user-defined method named printProperties( ), which prints the attributes of the application object to the screen.

***Example 2–15 printProperties( ) Method (Doc)***

```
public void printProperties(OracleConnection con){
     try {
          [1] Statement s = con.createStatement( );
          OracleResultSet rs = (OracleResultSet)s.executeQuery("select * from
               TDOC where n = 1 ");
          int index = 0;
          while(rs.next( )){
               index = rs.getInt(1);
               OrdDoc docObj = (OrdDoc) rs.getCustomDatum(2,
                    OrdDoc.getFactory( ));
               [2] System.out.println("format: " + docObj.getFormat( ));
```

```
                System.out.println("mimetype: " + docObj.getMimeType( ));
                System.out.println("contentlength: " +
                     docObj.getContentLength( ));
           }
      }
      [3] catch(Exception e) {
           System.out.println("exception raised " + e);
           System.out.println("print properties unsuccessful");
      }
}
```

The printProperties( ) method performs the following operations:

1. Creates a statement, a local OracleResultSet, and a local OrdDoc object named docObj, and populates docObj with media data through the same process described in steps 1 through 5 of Example 2–13. In this method, you will be operating on the contents of the row where n=1.
2. Gets the values of the properties of docObj and prints them to the screen.
3. Handles any exceptions raised by the code.

#### 2.2.2.6 loadDataFromStream( ) Method

Example 2–16 shows a user-defined method named loadDataFromStream( ), which uses the *inter*Media loadDataFromInputStream( ) method to load media data into the application object.

***Example 2–16 loadDataFromStream( ) Method (Doc)***

```
public void loadDataFromStream(OracleConnection con){
     try {
          [1] Statement s = con.createStatement( );
          OracleResultSet rs = (OracleResultSet)s.executeQuery("select * from
               TDOC where n = 2 for update ");
          int index = 0;
          while(rs.next( )){
               index = rs.getInt(1);
               OrdDoc docObj = (OrdDoc) rs.getCustomDatum(2,
                    OrdDoc.getFactory( ));
               [2] FileInputStream  fStream  = new FileInputStream
                    ("testaud.dat");
               [3] docObj.loadDataFromInputStream(fStream);
               [4] docObj.getDataInFile("output2.dat");
               [5] fStream.close( );
               System.out.println("************AFTER getDataInFile ");
```

```
                [6] System.out.println("getContentLength output: " +
                docObj.getContent( ).length( ));
                [7] OraclePreparedStatement stmt1 = (OraclePreparedStatement)
                     con.prepareCall("update tdoc set doc = ? where n = " +
                     index);
                stmt1.setCustomDatum(1,docObj);
                stmt1.execute( );
                stmt1.close( ) ;
           }
           System.out.println("load data from stream successful");
      }
      [8] catch(Exception e) {
           System.out.println("exception raised " + e);
           System.out.println("load data from stream unsuccessful");
      }
}
```

The loadDataFromStream( ) method performs the following operations:

1. Creates a statement, a local OracleResultSet, and a local OrdDoc object named docObj, and populates docObj with media data through the same process described in steps 1 through 5 of Example 2–13. In this method, you will be operating on the contents of the row where n=2.
2. Creates a new FileInputStream object. This input stream contains the contents of the local file testaud.dat.
3. Uses the loadDataFromInputStream( ) method to load the media data in the input stream into the database OrdDoc object and into docObj. This also sets the local field on docObj, but not the database object.
4. Uses the getDataInFile( ) method to get the media data from the application OrdDoc object and load it into a file on the local system named output2.dat.
5. Closes the local input stream.
6. Gets the content length of docObj and prints it to the screen to verify the success of the loading.
7. Creates and executes a SQL statement that will update the database OrdDoc object with the contents of docObj. This update will set the attributes on the database object to match the application object.
8. Handles any errors or exceptions raised by the code.

### 2.2.2.7 otherMethods( ) Method

Example 2–17 shows a user-defined method named otherMethods( ), which attempts to use the processSourceCommand( ) method.

***Example 2–17 otherMethods( ) Method (Doc)***

```
public void otherMethods(OracleConnection con){
     byte[ ] ctx[ ] = {new byte[4000]};
     byte[ ] res[ ] = {new byte[20]};
     [1] int suc = 1;
     try {
          [2] Statement s1 = con.createStatement( );
          OracleResultSet rs1 = (OracleResultSet)
          s1.executeQuery("select * from TDOC where n = 1 for update ");
          int index1 = 0;
          while(rs1.next( )){
               index1 = rs1.getInt(1);
               OrdDoc docObj = (OrdDoc) rs1.getCustomDatum(2,
                    OrdDoc.getFactory( ));
               [3] try {
                    byte[ ] pSRes = docObj.processSourceCommand(ctx, "", "",
                         res);
                    suc = 0;
               }
               [4] catch (Exception e) {
                    System.out.println("Expected Exception raised in
                         processSourceCommand(...)" );
               }
               [5] OraclePreparedStatement stmt1 = (OraclePreparedStatement)
                    con.prepareCall("update tdoc set doc = ? where n = " +
                    index1);
               stmt1.setCustomDatum(1,docObj);
               stmt1.execute( );
               stmt1.close( ) ;
          }
          rs1.close( );
          s1.close( );
     }
     [6] catch(Exception e){
          System.out.println("Exception raised " );
     }
     [7] if(suc ==1)
          System.out.println("other methods successful");
     else
          System.out.println("other methods unsuccessful");
}
```

The otherMethods( ) method performs the following operations:

1. Creates an integer that will be used to indicate the success or failure of the method and sets it initially to 1 (for success).
2. Creates a statement, a local OracleResultSet, and a local OrdDoc object named docObj, and populates docObj with media data through the same process described in steps 1 through 5 of Example 2–13. In this method, you will be operating on the contents of the row where n=1.
3. Tries to call processSourceCommand( ) with no value specified for the command to be called on the server side. This should raise an exception, which means the code following the processSourceCommand( ) call will not be run and the code in the catch loop will. If an exception is not raised, then the method has failed and the success indicator is set to 0 (for failure).
4. Prints the expected exception that was raised in step 3.
5. Creates and executes a SQL statement that will update the database OrdDoc object with the contents of docObj.
6. Handles any unexpected errors or exceptions raised by the code.
7. Prints the appropriate message to the screen based on the success or failure of the method.

### 2.2.2.8 loadDataFromByteArray( ) Method

Example 2–18 shows a user-defined method named loadDataFromByteArray( ), which uses the *inter*Media loadDataFromByteArray( ) method to load media data into the application object.

***Example 2–18 loadDataFromByteArray( ) Method (Doc)***

```
public void loadDataFromByteArray(OracleConnection con){
     try {
          [1] Statement s = con.createStatement( );
          OracleResultSet rs = (OracleResultSet) s.executeQuery("select * from
               TDOC where n = 3 for update ");
          int index = 0;
          while(rs.next( )){
               index = rs.getInt(1);
               OrdDoc docObj = (OrdDoc) rs.getCustomDatum(2,
                    OrdDoc.getFactory( ));
```

```
[2] File ff = new File("testaud.dat");
int fileLength = (int) ff.length( );
byte[ ] data = new byte[fileLength];
[3] FileInputStream  fStream  = new FileInputStream
     ("testaud.dat");
[4] fStream.read(data,0,fileLength);
[5] docObj.loadDataFromByteArray(data);
[6] fStream.close( );
[7] docObj.getDataInFile("output3.dat");
[8] byte[ ] resArr = docObj.getDataInByteArray( );
[9] System.out.println("byte array length: " + resArr.length);
[10] FileOutputStream outStream = new FileOutputStream
     ("output4.dat");
[11] outStream.write(resArr);
[12] outStream.close( );
[13] InputStream inpStream = docObj.getDataInStream( );
int length = 32768;
byte[ ] tempBuffer = new byte[32768];
[14] int numRead = inpStream.read(tempBuffer,0,length);
try {
     [15] outStream = new FileOutputStream("output5.dat");
     [16] while(numRead != -1) {
          [17] if (numRead < 32768) {
               length = numRead;
               outStream.write(tempBuffer,0,length);
               break;
          }
          [18] else
               outStream.write(tempBuffer,0,length);
          [19] numRead = inpStream.read(tempBuffer,0,length);
     }
}
[20] finally {
     outStream.close( );
     inpStream.close( );
}
System.out.println("************AFTER getDataInFile ");
[21] System.out.println("getContentLength output: " +
     docObj.getContent( ).length( ));
[22] OraclePreparedStatement stmt1 = (OraclePreparedStatement)
     con.prepareCall("update tdoc set doc = ? where n = " +
     index);
stmt1.setCustomDatum(1,docObj);
stmt1.execute( );
stmt1.close( ) ;
```

```
            }
        }
        [23] catch(Exception e) {
            System.out.println("exception raised " + e);
            System.out.println("loadData from byte array unsuccessful");
        }
}
```

The loadDataFromByteArray( ) method performs the following operations:

1. Creates a statement, a local OracleResultSet, and a local OrdDoc object named docObj, and populates docObj with media data through the same process described in steps 1 through 5 of Example 2–13. In this method, you will be operating on the contents of the row where n=3.
2. Determines the size (in bytes) of the local file testaud.dat and creates a byte array of the same size.
3. Creates a new FileInputStream object. This input stream contains the contents of testaud.dat.
4. Reads the contents of the input stream into the byte array.
5. Uses the loadDataFromByteArray( ) method to load the media data in the byte array into the database OrdDoc object and into docObj. This also sets the local field on docObj, but not the database object.
6. Closes the input stream.
7. Uses the getDataInFile( ) method to get the media data from the application OrdDoc object and load it into a file on the local system named output3.dat.
8. Uses the getDataInByteArray( ) method to get the media data from the application OrdDoc object and load it into a local byte array named resArr.
9. Gets the length of resArr and prints it to the screen to verify the success of the loading.
10. Creates a new FileOutputStream object named outStream. This output stream will write data to a local file named output4.dat.
11. Writes the contents of resArr to output4.dat.
12. Closes the output stream.
13. Creates a new input stream named inpStream. Uses the getDataInStream( ) method to get the media data from the application OrdDoc object and store it in inpStream.

14. Reads 32768 bytes from the beginning (that is, at an offset of 0) of inpStream into the byte array tempBuffer. The integer numRead will be set to the total number of bytes read, or -1 if the end of the input stream has been reached. In this case, if loading is successful, numRead should be equal to 32768.
15. Re-opens OutStream. In this case, it will write data to a local file named output5.dat.
16. Runs the operations in the while loop if numRead is not equal to -1. The program should enter this loop.
17. Enters the if loop if numRead is less than 32768 (that is, if all the data was read). The if loop will write the number of bytes read into tempBuffer into outStream, and then break out of the loop.
18. Writes 32768 bytes into outStream if numRead is 32768.
19. Attempts to read more data from the input stream into the byte array. If all data has been read successfully, then numRead will be set to -1 and the program will exit the loop. If there is still unread data in the input stream, then it will be read into the byte array and steps 17 and 18 will be repeated.
20. Closes both the input stream and the output stream after exiting the while loop.
21. Gets the content length of docObj and prints it to the screen to verify the success of the loading.
22. Creates and executes a SQL statement that will update the database OrdDoc object with the contents of docObj. This update will set the attributes on the database object to match the application object.
23. Handles any errors or exceptions raised by the code.

## 2.3 OrdImage Example

The image example (including ImageExample.sql and ImageExample.java) contains user-defined methods that use SQL, JDBC, and *inter*Media Java Classes APIs to perform the following operations:

- Create a database server table that contains test content.
- Load data into both application and database OrdImage objects from a local file.
- Load data into both application and database OrdImage objects from a local stream.

- Load data into both application and database OrdImage objects from a local byte array.
- Extract and print properties from the application OrdImage object.
- Show an example of the process( ) and processCopy( ) methods.
- Generate an image signature.
- Compare two image signatures based on different criteria.
- Compare two image signatures and determine if they match.

## 2.3.1 ImageExample.sql

Example 2–19 shows the contents of ImageExample.sql.

***Example 2–19 Contents of ImageExample.sql***

```
set echo on

-- Please Change system password.
connect / as sysdba;
drop user IMAGEUSER cascade;

[1] grant connect,resource to IMAGEUSER identified by IMAGEUSER;

-- Replace C:\Oracle\Ora’ with your ORACLE HOME
[2] create or replace directory ORDIMAGEDIR as ’C:\Oracle\Ora\ord\img\demo’;
grant read on directory ORDIMAGEDIR to public with grant option;

[3] connect IMAGEUSER/IMAGEUSER;

[4] create table ordimagetab(id number, image ORDSYS.ORDImage, image2
ORDSYS.ORDImage);

-- Note - the OrdImage.init method was added in interMedia 8.1.7.
-- If you are running against an older release of interMedia and the
-- Oracle database, you will have to modify the following INSERT statements
-- to use the OrdImage default constructor.
--
[5] insert into ordimagetab values
(1, ORDSYS.ORDImage.init( ),
    ORDSYS.ORDImage.init( ));

insert into ordimagetab values
```

```
(2, ORDSYS.ORDImage.init( ),
    ORDSYS.ORDImage.init( ));

insert into ordimagetab values
(3, ORDSYS.ORDImage.init( ),
    ORDSYS.ORDImage.init( ));

insert into ordimagetab values
(4, ORDSYS.ORDImage.init( ),
    ORDSYS.ORDImage.init( ));

[6] insert into ordimagetab values
(5, ORDSYS.ORDImage.init('file','ORDIMAGEDIR','imgdemo.dat'),
    ORDSYS.ORDImage.init( ));

insert into ordimagetab values
(6, ORDSYS.ORDImage.init('file','ORDIMAGEDIR','imgdemo.dat'),
    ORDSYS.ORDImage.init( ));

[7] insert into ordimagetab values
(10, ORDSYS.ORDImage.init('file','ORDIMAGEDIR','cbrdemo1.dat'),
     ORDSYS.ORDImage.init( ));
[8] insert into ordimagetab values
(11, ORDSYS.ORDImage.init('file','ORDIMAGEDIR','cbrdemo2.dat'),
     ORDSYS.ORDImage.init( ));

[9] create table sigtable(id number, sig ORDSYS.ORDImageSignature);

[10] insert into sigtable values
(10, ORDSYS.ORDImageSignature.init( ));
insert into sigtable values
(11, ORDSYS.ORDImageSignature.init( ));

commit;
set echo off
exit;
```

The SQL statements in ImageExample.sql perform the following operations:

1. Create a user named IMAGEUSER and grant the appropriate permissions to the user.
2. Create a directory named ORDIMAGEDIR and set the appropriate permissions. You will need to change the directory to match your Oracle home.
3. Connect to the database server as IMAGEUSER.

4. Create a table named ordimagetab, which contains one column of numbers and two columns of OrdImage objects.
5. Using the init method, add four rows with two empty objects each.
6. Using the init method, add two rows with one object based on imgdemo.dat and one empty object.
7. Using the init method, add a row with one object based on cbrdemo1.dat and one empty object.
8. Using the init method, add a row with one object based on cbrdemo2.dat and one empty object.
9. Create a table named sigtable, which contains one column of numbers and one column of OrdImageSignature numbers.
10. Using the init method, add two rows with an empty OrdImageSignature object.

See *Oracle interMedia User's Guide and Reference* for more information on the init method.

## 2.3.2 ImageExample.java

Section 2.3.2.1 through Section 2.3.2.12 show the methods contained in the ImageExample.java sample file.

### 2.3.2.1 main( ) Method

Example 2–20 shows the main( ) method.

***Example 2–20 main( ) Method (Image)***

```
public static void main (String args[ ]){
     byte[ ] ctx = new byte[4000];
     OracleConnection con = null;
     try{
          ImageExample ie = new ImageExample( );
          [1] con = ie.connect( );
          //Include the following line only if you are running
          //an Oracle 8.1.7 database or later.
          //If you are running a database server prior to 8.1.7,
          //the call will fail.
          [2] OrdMediaUtil.imCompatibilityInit(con);
          [3] ie.setPropertiesExample(con);
          ie.displayPropertiesExample(con);
          ie.fileBasedExample(con);
```

```
            ie.streamBasedExample(con);
            ie.byteArrayBasedExample(con);
            ie.processExample(con);
            [4] ie.sigGeneration(con);
            ie.sigSimilarity(con);
            ie.imageMatchingScore(con);
            [5] con.commit( );
            [6] con.close( );
            System.out.println("Done.");
        }
        [7] catch (Exception e){
            try{
                System.out.println("Exception : " + e);
                con.close( );
            }
            catch(Exception ex){
                System.out.println("Close Connection Exception : " + ex);
            }
        }
    }
```

The code in the main( ) method performs the following operations:

1. Uses the connect( ) method to make a connection to a database table.
2. Ensures the compatibility of your application with later releases of the Oracle database. See Section 1.8 for more information.
3. Calls several methods (also defined in ImageExample.java) that manipulate objects on the database server and the local machine.
4. Calls several methods (also defined in ImageExample.java) that test the OrdImageSignature object.
5. Commits any changes made to the database table.
6. Closes the connection to the database.
7. Handles any errors or exceptions raised by the code.

Section 2.3.2.2 through Section 2.3.2.12 will provide information on the methods called from the main( ) method.

### 2.3.2.2 connect( )Method

Example 2–21 shows a user-defined method named connect( ), which makes a connection from the application to the database.

***Example 2–21 connect( ) Method (Image)***

```
public OracleConnection connect( ) throws Exception{
     String connectString;
     [1] Class.forName ("oracle.jdbc.driver.OracleDriver");
     [2] connectString = "jdbc:oracle:oci8:@";
     [3] OracleConnection con = (OracleConnection)DriverManager.getConnection
          (connectString,"IMAGEUSER","IMAGEUSER");
     [4] con.setAutoCommit(false);
     return con;
}
```

The connect( ) method performs the following operations:

1. Loads the JDBC drivers directly, because the Oracle database uses a JDK-compliant Java virtual machine.
2. Defines a string that contains the URL of the database to which you will connect. You may need to change this string to match your database.
3. Sets the connection to the database, using the URL contained in connectString, the user name IMAGEUSER, and the password IMAGEUSER. The user name and password were created by ImageExample.sql.
4. Disables the auto-commit mode. This means that you must commit or roll back manually with the commit( ) or rollback( ) methods, respectively.

### 2.3.2.3 setPropertiesExample( ) Method

Example 2–22 shows a user-defined method named setPropertiesExample( ), which sets the properties in the application object.

***Example 2–22 setPropertiesExample( ) Method (Image)***

```
public void setPropertiesExample(OracleConnection con){
     try{
          int index = 0;
          [1] Statement s = con.createStatement( );
          [2] OracleResultSet rs = (OracleResultSet)s.executeQuery
               ("select * from ordimagetab where id = 5 for update");
          [3] while(rs.next( )){
               [4] index = rs.getInt(1);
               [5] OrdImage imgObj = (OrdImage)rs.getCustomDatum
                    (2, OrdImage.getFactory( ));
               [6] imgObj.setProperties( );
               System.out.println("set Properties called");
               [7] if(imgObj.checkProperties( )){
                    System.out.println("checkProperties called");
                    System.out.println("setProperties successful");
                    System.out.println("checkProperties successful");
                    System.out.println("successful");
               }
               else{
                    System.out.println("checkProperties called");
                    System.out.println("setProperties not successful");
                    System.out.println("checkProperties successful");
               }
               [8] OraclePreparedStatement stmt1 = (OraclePreparedStatement)
                    con.prepareCall("update ordimagetab set
                    image = ? where id = " + index);
               stmt1.setCustomDatum(1,imgObj);
               stmt1.execute( );
               stmt1.close( ) ;
          }
          rs.close( );
          s.close( );
     }
     [9] catch(Exception e){
          System.out.println("exception raised " + e);
     }
}
```

The setPropertiesExample( ) method performs the following operations:

1. Creates an OracleStatement object.
2. Executes the given SQL query and puts the results into a local OracleResultSet object. In this case, the SQL query selects the data in the database row where id=5.
3. Performs the operations in the loop while there are results in the OracleResultSet that have not been processed. However, in this case, there is only one row included in the OracleResultSet, so the operations in the loop will run once.
4. Sets an index variable to the value of the integer in the first column of the first row in the OracleResultSet (in this case, the value is 5).
5. Creates a local OrdImage object named imgObj. Populates imgObj with the contents of the OrdImage object in the second column of the current row in the OracleResultSet.

6. Calls setProperties( ) to extract properties values from the media data and set them in the application OrdImage object. See "setProperties( )" in Chapter 5 for a list of the properties values extracted and set.
7. Calls checkProperties( ) to compare the properties values in the application object with the values in the media data. If all values are the same, checkProperties( ) returns true and the appropriate messages are printed to the screen. If any values differ, checkProperties( ) returns false and the appropriate messages are printed to the screen.
8. Creates and executes a SQL statement that will update the database OrdImage object with the contents of imgObj.
9. Handles any errors or exceptions raised by the code.

#### 2.3.2.4 displayPropertiesExample( ) Method

Example 2–23 shows a user-defined method named displayPropertiesExample( ), which prints the attributes of the application object to the screen.

***Example 2–23 displayPropertiesExample( ) Method (Image)***

```
public void displayPropertiesExample(OracleConnection con){
     try{
          int index = 0;
          [1] Statement s = con.createStatement( );
          OracleResultSet rs = (OracleResultSet)s.executeQuery(
               "select * from ordimagetab where id = 5 ");
          while(rs.next( )){
               index = rs.getInt(1);
               OrdImage imgObj = (OrdImage) rs.getCustomDatum(2,
                    OrdImage.getFactory( ));
               [2] System.out.println("format : " + imgObj.getFormat( ));
               System.out.println("mimeType: " + imgObj.getMimeType( ));
               System.out.println("height: " + imgObj.getHeight( ));
               System.out.println("width: " + imgObj.getWidth( ));
               System.out.println("contentLength: " +
                    imgObj.getContentLength( ));
               System.out.println("contentFormat: " +
                    imgObj.getContentFormat( ));
               System.out.println("compressionFormat: " +
                    imgObj.getCompressionFormat( ));
               System.out.println("source type: " +
                    imgObj.getSourceType( ));
               System.out.println("source loc: " +
                    imgObj.getSourceLocation( ));
                System.out.println("source name: " + imgObj.getSourceName( ));
                System.out.println("source : " + imgObj.getSource( ));
                [3] try{
                    String attrString = getAllAttributesAsString(imgObj);
                    System.out.println(attrString);
                }
                [4] catch (Exception e){
                    System.out.println("Exception raised in
                        getAllAttributesAsString:");
                }
                System.out.println("successful");
            }
        }
        [5] catch(Exception e) {
            System.out.println("exception raised " + e);
        }
}
```

The displayPropertiesExample( ) method performs the following operations:

1. Creates a statement, a local OracleResultSet, and a local OrdImage object named imgObj, and populates imgObj with media data through the same process described in steps 1 through 5 of Example 2–22. In this method, you will be operating on the contents of the row where id=5. This is the same row you operated on in Example 2–22.
2. Gets the values of the properties in imgObj and prints them to the screen.
3. Gets the attributes of imgObj and stores them in a string by using the getAllAttributesAsString( ) method, and prints the contents of the string to the screen. See Section 2.3.2.5 for more information on getAllAttributesAsString( ).
4. Handles any errors or exceptions raised by the call to getAllAttributesAsString( ).
5. Handles any errors or exceptions raised by the code in general.

### 2.3.2.5 getAllAttributesAsString( ) Method

Example 2–24 shows a user-defined method named getAllAttributesAsString( ), which creates a String object that contains the values of the application object attributes.

***Example 2–24 getAllAttributesAsString( ) Method (Image)***

```
public String getAllAttributesAsString (OrdImage imgObj) throws Exception{
     [1] String attStr = imgObj.getSource( ) + " mimeType = " +
          imgObj.getMimeType( ) + ", fileFormat = " +
          imgObj.getFormat( ) + ", height = " + imgObj.getHeight( )
          + ", width = " + imgObj.getWidth( ) + ", contentLength = "
          + imgObj.getContentLength( ) + ", contentFormat = " +
          imgObj.getContentFormat( ) + ", compressionFormat = " +
          imgObj.getCompressionFormat( );
     [2] return attStr;
}
```

The getAllAttributesAsString( ) method performs the following operations:

1. Creates a String object named attStr. Gets the values of several attributes from the application image object and stores their values in attStr.
2. Returns attStr to the method that called this method.

### 2.3.2.6 fileBasedExample( ) Method

Example 2–25 shows a user-defined method named fileBasedExample( ), which uses the loadDataFromFile( ) method to load media data into the application object.

***Example 2–25 fileBasedExample( ) Method (Image)***

```
public void fileBasedExample(OracleConnection con){
     try{
          int index = 0;
          [1] Statement s = con.createStatement( );
          OracleResultSet rs = (OracleResultSet)s.executeQuery(
               "select * from ORDIMAGETAB where id = 2 for update ");
          while(rs.next( )){
               index = rs.getInt(1);
               OrdImage imgObj = (OrdImage) rs.getCustomDatum(2,
                    OrdImage.getFactory( ));
               [2] imgObj.loadDataFromFile("imgdemo.dat");
               [3] imgObj.setProperties( );
               [4] imgObj.getDataInFile("fileexample.dat");
               [5] OraclePreparedStatement stmt1 = (OraclePreparedStatement)
                    con.prepareCall("update ordimagetab set image =
                    ? where id = " + index);
               stmt1.setCustomDatum(1,imgObj);
               stmt1.execute( );
               stmt1.close( ) ;
          }
          System.out.println("successful");
     }
     [6] catch(Exception e){
          System.out.println("exception raised " + e);
     }
}
```

The fileBasedExample( ) method performs the following operations:

1. Creates a statement, a local OracleResultSet, and a local OrdImage object named imgObj, and populates imgObj with media data through the same process described in steps 1 through 5 of Example 2–22. In this method, you will be operating on the contents of the row where id=2.
2. Uses the loadDataFromFile( ) method to load the media data from the local file imgdemo.dat into the database OrdImage object and into imgObj. This also sets the local field on imgObj, but not the database object.
3. Calls setProperties( ) to extract properties values from the media data and set them in the application OrdImage object. See "setProperties( )" in Chapter 5 for a list of the properties values extracted and set.
4. Uses the getDataInFile( ) method to get the media data from the application OrdImage object and load it into a file on the local system named fileexample.dat.
5. Creates and executes a SQL statement that will update the database OrdImage object with the contents of imgObj. This update will set the attributes on the database object, to match the application object.
6. Handles any errors or exceptions raised by the code.

#### 2.3.2.7 streamBasedExample( ) Method

Example 2–26 shows a user-defined method named streamBasedExample( ), which uses the loadDataFromInputStream( ) method to load media data into the application object.

***Example 2–26 streamBasedExample( ) Method (Image)***

```
public void streamBasedExample(OracleConnection con){
     try{
          int index = 0;
          [1] Statement s = con.createStatement( );
          OracleResultSet rs = (OracleResultSet)s.executeQuery(
               "select * from ORDIMAGETAB where id = 3 for update ");
          while(rs.next( )){
               index = rs.getInt(1);
               OrdImage imgObj = (OrdImage) rs.getCustomDatum(2,
                    OrdImage.getFactory( ));
               [2] FileInputStream fStream = new FileInputStream
                    ("imgdemo.dat");
               [3] imgObj.loadDataFromInputStream(fStream);
               [4] fStream.close( );
               [5] imgObj.setProperties( );
               [6] InputStream inpStream = imgObj.getDataInStream( );
               int length = 32300;
               byte[ ] tempBuffer = new byte[length];
               [7] int numRead = inpStream.read(tempBuffer,0,length);
               FileOutputStream outStream=null;
               try{
                    [8] outStream = new FileOutputStream
                         ("streamexample.dat");
                    [9] while(numRead != -1){
                         [10] if (numRead < length){
                              length = numRead;
                              outStream.write(tempBuffer,0,length);
                              break;
                         }
                         [11] else
                              outStream.write(tempBuffer,0,length);
                         [12] numRead = inpStream.read(tempBuffer,0,
                              length);
                    }
               }
               [13] finally{
                    if (outStream != null)
                         outStream.close( );
                    inpStream.close( );
               }
               [14] OraclePreparedStatement stmt1 = (OraclePreparedStatement)
                    con.prepareCall("update ordimagetab set
                    image = ? where id = " + index);
               stmt1.setCustomDatum(1,imgObj);
               stmt1.execute( );
               stmt1.close( ) ;
          }
          System.out.println("successful");
     }
     [15] catch(Exception e){
```

```
              System.out.println("exception raised " + e);
          }
      }
```

The streamBasedExample( ) method performs the following operations:

1. Creates a statement, a local OracleResultSet, and a local OrdImage object named imgObj, and populates imgObj with media data through the same process described in steps 1 through 5 of Example 2–22. In this method, you will be operating on the contents of the row where id=3.
2. Creates a new FileInputStream object. This input stream contains the contents of the local file imgdemo.dat.
3. Uses the loadDataFromInputStream( ) method to load the media data in the input stream into the database OrdImage object and into imgObj. This also sets the local field on imgObj, but not the database object.
4. Closes the input stream.
5. Calls setProperties( ) to extract properties values from the media data and set them in the application OrdImage object. See "setProperties( )" in Chapter 5 for a list of the properties values extracted and set.
6. Creates a new InputStream named inpStream. Calls getDataInStream( ) to get the media data from the application OrdImage object and stores it in inpStream.
7. Reads 32300 bytes from the beginning (that is, at an offset of 0) of inpStream into the byte array tempBuffer. The integer numRead will be set to the total number of bytes read, or -1 if the end of the input stream has been reached. In this case, if loading is successful, numRead should be equal to 32300.
8. Creates a new FileOutputStream object named outStream. This output stream will write data to a local file named streamexample.dat.
9. Runs the operations in the while loop if numRead is not equal to -1. The program should enter this loop.
10. Writes the number of bytes read into tempBuffer into outStream if numRead is less than 32300 (that is, if not all the data was read).
11. Writes 32300 bytes into outStream if numRead is 32300.
12. Attempts to read more data from the input stream into the byte array. If all data has been read successfully, then numRead will be set to -1 and the program will exit the loop. If there is still unread data in the input stream, then it will be read into the byte array and steps 10 and 11 will be repeated.

13. Closes both the input stream and the output stream after exiting the while loop.
14. Creates and executes a SQL statement that will update the database OrdImage object with the contents of imgObj. This update will set the attributes on the database object to match the application object.
15. Handles any errors or exceptions raised by the code.

### 2.3.2.8 byteArrayBasedExample( ) Method

Example 2–27 shows a user-defined method named byteArrayBasedExample( ), which uses the loadDataFromByteArray( ) method to load media data into the application object.

***Example 2–27 byteArrayBasedExample( ) Method (Image)***

```
public void byteArrayBasedExample(OracleConnection con){
     try{
          int index = 0;
          [1] Statement s = con.createStatement( );
          OracleResultSet rs = (OracleResultSet)s.executeQuery
               ("select * from ORDIMAGETAB where id = 4 for update ");
          while(rs.next( )){
               index = rs.getInt(1);
               OrdImage imgObj = (OrdImage) rs.getCustomDatum(2,
                    OrdImage.getFactory( ));
               [2] File ff = new File("imgdemo.dat");
               int fileLength = (int) ff.length( );
               byte[ ] data = new byte[fileLength];
               [3] FileInputStream fStream = new
                    FileInputStream("imgdemo.dat");
               [4] fStream.read(data,0,fileLength);
               [5] imgObj.loadDataFromByteArray(data);
               [6] fStream.close( );
               [7] imgObj.setProperties( );
               [8] byte[ ] resArr = imgObj.getDataInByteArray( );
               [9] System.out.println("byte array length : " +
                    resArr.length);
               [10] OraclePreparedStatement stmt1 = (OraclePreparedStatement)
                    con.prepareCall("update ordimagetab set image =
                    ? where id = " + index);
               stmt1.setCustomDatum(1,imgObj);
               stmt1.execute( );
               stmt1.close( ) ;
          }
```

```
            System.out.println("successful");
        }
        [11] catch(Exception e){
            System.out.println("exception raised " + e);
        }
}
```

The byteArrayBasedExample( ) method performs the following operations:

1. Creates a statement, a local OracleResultSet, and a local OrdImage object named imgObj, and populates imgObj with media data through the same process described in steps 1 through 5 of Example 2–22. In this method, you will be operating on the contents of the row where id=4.
2. Determines the size (in bytes) of the local file imgdemo.dat and creates a byte array of the same size.
3. Creates a new FileInputStream object. This input stream contains the contents of imgdemo.dat.
4. Reads the contents of the input stream into the byte array.
5. Uses the loadDataFromByteArray( ) method to load the media data in the byte array into the database OrdImage object and into imgObj. This also sets the local field on imgObj, but not the database object.
6. Closes the input stream.
7. Calls setProperties( ) to extract properties values from the media data and set them in the application OrdImage object. See "setProperties( )" in Chapter 5 for a list of the properties values extracted and set.
8. Uses the getDataInByteArray( ) method to get the media data from the application OrdImage object and load it into a local byte array named resArr.
9. Gets the length of resArr and prints it to the screen to verify the success of the loading.
10. Creates and executes a SQL statement that will update the database OrdImage object with the contents of imgObj. This update will set the attributes on the database object to match the application object.
11. Handles any errors or exceptions raised by the code.

### 2.3.2.9 processExample( ) Method

Example 2–28 shows a user-defined method named processExample( ), which uses the process( ) and processCopy( ) methods to manipulate the media data in the application object.

***Example 2–28 processExample( ) Method (Image)***

```
public void processExample(OracleConnection con){
     try{
          int index1 = 0;
          [1] Statement s1 = con.createStatement( );
          OracleResultSet rs1 = (OracleResultSet)s1.executeQuery
               ("select * from ORDIMAGETAB where id = 2 for update ");
          while(rs1.next( )){
               index1 = rs1.getInt(1);
               OrdImage imgObj = (OrdImage) rs1.getCustomDatum(2,
                    OrdImage.getFactory( ));
               [2] OrdImage imgObj2 = (OrdImage) rs1.getCustomDatum(3,
                    OrdImage.getFactory( ));
               try{
                    [3] imgObj.processCopy("maxScale=32 32, fileFormat=
                         GIFF", imgObj2);
                    [4] imgObj.process("fileFormat=JFIF");
                    [5] System.out.println(getAllAttributesAsString
                         (imgObj));
                    [6] System.out.println(getAllAttributesAsString(imgObj2));
               }
               [7] catch (Exception e){
                    System.out.println("Exception raised in process"
                         + e );
               }
               [8] OraclePreparedStatement stmt1 = (OraclePreparedStatement)
                    con.prepareCall("update ordimagetab set image =
                    ?, image2 = ? where id = " + index1);
               stmt1.setCustomDatum(1,imgObj);
               stmt1.setCustomDatum(2,imgObj2);
               stmt1.execute( );
               stmt1.close( ) ;
          }
          rs1.close( );
          s1.close( );
     }
     [9] catch(Exception e){
          System.out.println("Exception raised: " + e);
     }
     System.out.println("successful");
}
```

The processExample( ) method performs the following operations:

1. Creates a statement, a local OracleResultSet, and a local OrdImage object named imgObj, and populates imgObj with media data through the same process described in steps 1 through 5 of Example 2–22. In this method, you will be operating on the contents of the second column of the row where id=2. The database OrdImage object is named image.
2. Creates a local OrdImage object named imgObj2. Populates imgObj2 with the contents of the OrdImage object in the third column of the current row in the OracleResultSet. This database OrdImage column is named image2.
3. Populates the image data in imgObj2 with a 32 x 32 GIF thumbnail image generated from the image data in imgObj. imgObj is unchanged by this operation.
4. Uses the process( ) method to convert the image in imgObj to a JPEG (JFIF) image.
5. Gets the attributes of imgObj by using the getAllAttributesAsString( ) method, and prints the attributes to the screen. See Section 2.3.2.5 for more information on getAllAttributesAsString( ).
6. Gets the attributes of imgObj2 by using the getAllAttributesAsString( ) method, and prints the attributes to the screen. See Section 2.3.2.5 for more information on getAllAttributesAsString( ).
7. Handles any errors or exceptions raised by the code in steps 3 through 6.
8. Creates and executes a SQL statement that will update the appropriate database OrdImage objects with the contents of imgObj and imgObj2.
9. Handles any errors or exceptions raised by the code.

#### 2.3.2.10 sigGeneration( ) Method

Example 2–29 shows a user-defined method named sigGeneration( ), which generates a signature for a given OrdImage object and stores it in the database.

***Example 2–29 sigGeneration( ) Method (Image)***

```
public void sigGeneration(OracleConnection con){
     byte[ ] ctx[ ] = new byte [4000][1];
```

```
        try{
             [1] int IMG_ID=10;
             int SIG_ID=11;
             int IMG_COL=2;
             int SIG_COL=2;
             [2] Statement get_image_obj = con.createStatement( );
             Statement get_sig_obj = con.createStatement( );
             [3] OracleResultSet get_image_obj_result =
                  (OracleResultSet)get_image_obj.executeQuery ("select * from
                  ordimagetab where id =" + IMG_ID + " for update");
             OracleResultSet get_sig_result = (OracleResultSet)
                  get_sig_obj.executeQuery("select * from sigtable where id =" +
                  SIG_ID + " for update");
             [4] get_image_obj_result.next( );
             get_sig_result.next( );
             [5] OrdImage img_obj = (OrdImage)get_image_obj_result.getCustomDatum
                  (IMG_COL, OrdImage.getFactory( ));
             [6] OrdImageSignature img_sig = (OrdImageSignature)
                  get_sig_result.getCustomDatum(SIG_COL,
                  OrdImageSignature.getFactory( ));
             [7] img_obj.setProperties( );
             [8] img_obj.importData(ctx);
             [9] img_sig.generateSignature(img_obj);
             [10] OraclePreparedStatement update_sig = (OraclePreparedStatement)
                  con.prepareCall("update sigtable set sig = ?
                  where id =" + SIG_ID);
             update_sig.setCustomDatum(1,img_sig);
             update_sig.execute( );
             update_sig.close( ) ;
             [11] get_sig_result.close( );
             get_image_obj_result.close( );
             get_sig_obj.close( );
             get_image_obj.close( );
             System.out.println("image signature generation complete.");
        }
        [12] catch(Exception e){
             System.out.println("exception raised " + e);
             System.out.println("image signature generation failed");
        }
   }
```

The code in sigGeneration( ) performs the following operations:

1. Creates four integers that have the following values:

- IMG_ID is the primary key of the OrdImage object in ORDIMAGETAB for which the signature will be generated.
- SIG_ID is the primary key of the OrdImageSignature object in SIGTABLE where the signature will be stored.
- IMG_COL is the column number of the OrdImage objects in ORDIMAGETAB.
- SIG_COL is the column number of the OrdImageSignature objects in SIGTABLE.

2. Creates two Statement objects.
3. Executes the given SQL queries and puts the results in two local OracleResultSet objects. In this case, the SQL queries select the data in the following locations:
   - In ORDIMAGETAB, the database row where the primary key is 10.
   - In SIGTABLE, the database row where the primary key is 11.
4. Advances to the proper row of the result sets.
5. Creates a local OrdImage object named img_obj. Populates img_obj with the contents of the OrdImage object in the given column of the current row of the result set.
6. Creates a local OrdImageSignature object named img_sig. Populates img_sig with the contents of the OrdImageSignature object in the given column of the current row of the result set.
7. Calls setProperties( ) to extract properties values from the media data and set them in the application OrdImage object. See "setProperties( )" in Chapter 5 for a list of the properties values extracted and set.
8. Imports data into the local OrdImage object.
9. Generates a signature for img_obj and stores it in img_sig.
10. Creates and executes a SQL statement that will update the database OrdImageSignature object with the contents of img_sig.
11. Explicitly closes the OracleResultSet and Statement objects.
12. Catches any exceptions or errors raised by the code.

#### 2.3.2.11 imageMatchingScore( ) Method

Example 2–30 shows the imageMatchingScore( ) method, which generates signatures for two OrdImage objects and compares them based on different criteria.

***Example 2–30 imageMatchingScore( ) Method (Image)***

```
public void imageMatchingScore(OracleConnection con){
     byte[ ] ctx[ ] = new byte [4000][1];
     try {
          [1] int IMG_ID1=10;
          int SIG_ID1=10;
          int IMG_ID2=11;
          int SIG_ID2=11;
          int IMG_COL=2;
          int SIG_COL=2;
          [2] Statement get_image_obj = con.createStatement( );
          Statement get_sig_obj = con.createStatement( );
          [3] OracleResultSet get_image_obj_result1 = (OracleResultSet)
               get_image_obj.executeQuery("select * from ordimagetab
               where id ="+ IMG_ID1 +" for update");
          OracleResultSet get_sig_result1 = (OracleResultSet)
               get_sig_obj.executeQuery("select * from sigtable
               where id =" +SIG_ID1 + " for update");
          OracleResultSet get_image_obj_result2 = (OracleResultSet)
               get_image_obj.executeQuery("select * from ordimagetab
               where id ="+ IMG_ID2 +" for update");
          OracleResultSet get_sig_result2 = (OracleResultSet)
               get_sig_obj.executeQuery("select * from sigtable
               where id =" +SIG_ID2 + " for update");
          [4] get_image_obj_result1.next( );
          get_sig_result1.next( );
          get_image_obj_result2.next( );
          get_sig_result2.next( );
          [5] OrdImage img_obj1 = (OrdImage)
               get_image_obj_result1.getCustomDatum(IMG_COL,
               OrdImage.getFactory( ));
          [6] OrdImageSignature img_sig1 = (OrdImageSignature)
               get_sig_result1.getCustomDatum(
               SIG_COL,OrdImageSignature.getFactory( ));
          [7] img_obj1.setProperties( );
          [8] img_obj1.importData(ctx);
          [9] img_sig1.generateSignature(img_obj1);
          [10] OrdImage img_obj2 = (OrdImage)get_image_obj_result2.
               getCustomDatum(IMG_COL, OrdImage.getFactory( ));
          [11] OrdImageSignature img_sig2 = (OrdImageSignature)
               get_sig_result2.getCustomDatum(
               SIG_COL,OrdImageSignature.getFactory( ));
          [12] img_obj2.setProperties( );
          [13] img_obj2.importData(ctx);
          [14] img_sig2.generateSignature(img_obj2);
          [15] float gscore = OrdImageSignature.evaluateScore(img_sig1,
               img_sig2,"color=1", con);
          System.out.println("score value (global color comparison):" +
               gscore);
          [16] float lscore = OrdImageSignature.evaluateScore(img_sig1,
               img_sig2,"color=1 location=1", con);
          System.out.println("Score value (local color comparison):" + lscore);
          [17] get_sig_result2.close( );
          get_image_obj_result2.close( );
          get_sig_result1.close( );
          get_image_obj_result1.close( );
          get_sig_obj.close( );
          get_image_obj.close( );
     }
     [18] catch(Exception e){
          System.out.println("exception raised " + e);
          System.out.println("Image matching score computation failed");
     }
}
```

The code in the imageMatchingScore( ) method performs the following operations:

1. Creates six integers that have the following values:
   - IMG_ID1 is the primary key of the OrdImage object in ORDIMAGETAB for which the first signature will be generated.
   - SIG_ID1 is the primary key of the OrdImageSignature object in SIGTABLE where the first signature will be stored.
   - IMG_ID2 is the primary key of the OrdImage object in ORDIMAGETAB for which the second signature will be generated.
   - SIG_ID2 is the primary key of the OrdImageSignature object in SIGTABLE where the second signature will be stored.
   - IMG_COL is the column number of the OrdImage objects in ORDIMAGETAB.
   - SIG_COL is the column number of the OrdImageSignature objects in SIGTABLE.

2. Creates two Statement objects.
3. Executes the given SQL queries and puts the results in four local OracleResultSet objects. In this case, the SQL queries select the data in the following locations:
   - In ORDIMAGETAB, the database row where the primary key is 10.
   - In SIGTABLE, the database row where the primary key is 10.
   - In ORDIMAGETAB, the database row where the primary key is 11.
   - In SIGTABLE, the database row where the primary key is 11.
4. Advances to the proper row of the result sets.
5. Creates a local OrdImage object named img_obj1. Populates img_obj1 with the contents of the OrdImage object in the second column of the row in ORDIMAGETAB where the primary key is 10.
6. Creates a local OrdImageSignature object named img_sig1. Populates img_sig1 with the contents of the OrdImageSignature object in the second column of the row in SIGTABLE where the primary key is 10.
7. Calls setProperties( ) to extract properties values from the media data and set them in img_obj1. See "setProperties( )" in Chapter 5 for a list of the properties values extracted and set.
8. Imports data into img_obj1.
9. Generates a signature for img_obj1 and stores it in img_sig1.
10. Creates a local OrdImage object named img_obj2. Populates img_obj2 with the contents of the OrdImage object in the second column of the row in ORDIMAGETAB where the primary key is 11.
11. Creates a local OrdImageSignature object named img_sig2. Populates img_sig2 with the contents of the OrdImageSignature object in the second column of the row in SIGTABLE where the primary key is 11.
12. Calls setProperties( ) to extract properties values from the media data and set them in img_obj2. See "setProperties( )" in Chapter 5 for a list of the properties values extracted and set.
13. Imports data into img_obj2.
14. Generates a signature for img_obj2 and stores it in img_sig2.
15. Compares the two image signatures based on the color attribute and print the newly generated score to the screen.

16. Compares the two image signatures based on the color and location attributes and print the newly generated score to the screen.
17. Explicitly closes the Statement and OracleResultSet objects.
18. Handles any exceptions or errors that were generated by the method.

### 2.3.2.12 sigSimilarity( ) Method

Example 2–31 shows the sigSimilarity( ) method, which compares the image signatures of two objects and determines if they match.

***Example 2–31 sigSimilarity( ) Method (Image)***

```
public void sigSimilarity(OracleConnection con){
     byte[ ] ctx[ ] = new byte [4000][1];
     try {
          [1] int IMG_ID1=10;
          int SIG_ID1=10;
          int IMG_ID2=11;
          int SIG_ID2=11;
          int IMG_COL=2;
          int SIG_COL=2;
          [2] double similarity_threshold= 20.0;
          [3] Statement get_image_obj = con.createStatement( );
          Statement get_sig_obj = con.createStatement( );
          [4] OracleResultSet get_image_obj_result1 = (OracleResultSet)
               get_image_obj.executeQuery("select * from ordimagetab
               where id ="+ IMG_ID1 +" for update");
          OracleResultSet get_sig_result1 = (OracleResultSet)
               get_sig_obj.executeQuery("select * from sigtable
               where id =" +SIG_ID1 + " for update");
          OracleResultSet get_image_obj_result2 = (OracleResultSet)
               get_image_obj.executeQuery("select * from ordimagetab
               where id ="+ IMG_ID2 +" for update");
          OracleResultSet get_sig_result2 = (OracleResultSet)
               get_sig_obj.executeQuery("select * from sigtable
               where id =" +SIG_ID2 + " for update");
          [5] get_image_obj_result1.next( );
          get_sig_result1.next( );
          get_image_obj_result2.next( );
          get_sig_result2.next( );
          [6] OrdImage img_obj1 = (OrdImage)get_image_obj_result1.
               getCustomDatum(IMG_COL, OrdImage.getFactory( ));
          [7] OrdImageSignature img_sig1 = (OrdImageSignature)get_sig_result1.
               getCustomDatum(SIG_COL,OrdImageSignature.getFactory( ));
```

```
            [8] img_obj1.setProperties( );
            [9] img_obj1.importData(ctx);
            [10] img_sig1.generateSignature(img_obj1);
            [11] OrdImage img_obj2 = (OrdImage) get_image_obj_result2.
                 getCustomDatum(IMG_COL, OrdImage.getFactory( ));
            [12] OrdImageSignature img_sig2 = (OrdImageSignature)get_sig_result2.
                 getCustomDatum(SIG_COL,OrdImageSignature.getFactory( ));
            [13] img_obj2.setProperties( );
            [14] img_obj2.importData(ctx);
            [15] img_sig2.generateSignature(img_obj2);
            [16] int result = OrdImageSignature.isSimilar(img_sig1, img_sig2,
                 "color=1 texture=1 shape=1 location=1",
                 (float)similarity_threshold,con);
            if (result==1){
                 System.out.println("Signatures are similar ");
            }
            else{
                 System.out.println("Signatures are different ");
            }
            [17] get_sig_result2.close( );
            get_image_obj_result2.close( );
            get_sig_result1.close( );
            get_image_obj_result1.close( );
            get_sig_obj.close( );
            get_image_obj.close( );
       }
       [18] catch(Exception e){
            System.out.println("exception raised " + e);
            System.out.println("Signature similarity computation failed");
       }
}
```

1. Creates six integers that have the following values:
   - IMG_ID1 is the primary key of the OrdImage object in ORDIMAGETAB for which the first signature will be generated.
   - SIG_ID1 is the primary key of the OrdImageSignature object in SIGTABLE where the first signature will be stored.
   - IMG_ID2 is the primary key of the OrdImage object in ORDIMAGETAB for which the second signature will be generated.
   - SIG_ID2 is the primary key of the OrdImageSignature object in SIGTABLE where the second signature will be stored.

- IMG_COL is the column number of the OrdImage objects in ORDIMAGETAB.
- SIG_COL is the column number of the OrdImageSignature objects in SIGTABLE.

2. Sets the threshold score at which the images will be considered a match.
3. Creates two Statement objects.
4. Executes the given SQL queries and puts the results in four local OracleResultSet objects. In this case, the SQL queries select the data in the following locations:
   - In ORDIMAGETAB, the database row where the primary key is 10.
   - In SIGTABLE, the database row where the primary key is 10.
   - In ORDIMAGETAB, the database row where the primary key is 11.
   - In SIGTABLE, the database row where the primary key is 11.
5. Advances to the proper row of the result sets.
6. Creates a local OrdImage object named img_obj1. Populates img_obj1 with the contents of the OrdImage object in the second column of the row in ORDIMAGETAB where the primary key is 10.
7. Creates a local OrdImageSignature object named img_sig1. Populates img_sig1 with the contents of the OrdImageSignature object in the second column of the row in SIGTABLE where the primary key is 10.
8. Calls setProperties( ) to extract properties values from the media data and set them in img_obj1. See "setProperties( )" in Chapter 5 for a list of the properties values extracted and set.
9. Imports data into img_obj1.
10. Generates a signature for img_obj1 and stores it in img_sig1.
11. Creates a local OrdImage object named img_obj2. Populates img_obj2 with the contents of the OrdImage object in the second column of the row in ORDIMAGETAB where the primary key is 11.
12. Creates a local OrdImageSignature object named img_sig2. Populates img_sig2 with the contents of the OrdImageSignature object in the second column of the row in SIGTABLE where the primary key is 11.

13. Calls setProperties( ) to extract properties values from the media data and set them in img_obj2. See "setProperties( )" in Chapter 5 for a list of the properties values extracted and set.
14. Imports data into img_obj2.
15. Generates a signature for img_obj2 and stores it in img_sig2.
16. Determines if the two image signatures match, based on the color, texture, shape, and location attributes. If the score is less than or equal to 20, the result is 1 and the appropriate message is printed to the screen. If not, the result is 0 and the appropriate message is printed to the screen.
17. Explicitly closes the Statement and OracleResultSet objects.
18. Handles any exceptions or errors that were generated by the method.

## 2.4 OrdVideo Example

The video example (including VideoExample.sql and VideoExample.java) contains user-defined methods that use SQL, JDBC, and *inter*Media Java Classes APIs to perform the following operations:

- Create a database server table that contains test content.
- Load data into both application and database OrdVideo objects from a local file.
- Load data into both application and database OrdVideo objects from a local stream.
- Load data into both application and database OrdVideo objects from a local byte array.
- Extract and print properties from the application OrdVideo object.
- Demonstrate error handling through a failed call to a database method.

### 2.4.1 VideoExample.sql

Example 2–32 shows the contents of VideoExample.sql.

***Example 2–32 Contents of VideoExample.sql***

```
set echo on

--PLEASE change system password
connect system/manager
```

```
drop user VIDEOUSER cascade;
[1] create user VIDEOUSER identified by VIDEOUSER ;
grant connect,resource to VIDEOUSER identified by VIDEOUSER;

[2] connect VIDEOUSER/VIDEOUSER

[3] CREATE TABLE TVID(n NUMBER, vid ORDSYS.ORDVIDEO);

-- Note - the OrdVideo.init method was added in interMedia 8.1.7.
-- If you are running against an older release of interMedia and the
-- Oracle database, you will have to modify the following INSERT statements
-- to use the OrdVideo default constructor.

[4] INSERT INTO TVID VALUES(1, ORDSYS.ORDVideo.init( ));
INSERT INTO TVID VALUES(2, ORDSYS.ORDVideo.init( ));
INSERT INTO TVID VALUES(3, ORDSYS.ORDVideo.init( ));
commit;
/
```

The SQL statements in VideoExample.sql perform the following operations:

1. Create a user named VIDEOUSER amd grant the appropriate permissions to the user.
2. Connect to the database server as VIDEOUSER.
3. Create a table named TVID with two columns: a column of numbers and a column of OrdVideo objects.
4. Add three rows to the table, each containing an empty OrdVideo object.

See *Oracle interMedia User's Guide and Reference* for more information on the init method.

## 2.4.2 VideoExample.java

Section 2.4.2.1 through Section 2.4.2.8 show the methods contained in the VideoExample.java sample file.

### 2.4.2.1 main( ) Method

Example 2–33 shows the main( ) method.

***Example 2–33 main( ) Method (Video)***

```
public static void main (String args[ ]){
```

```
          byte[ ] ctx = new byte[4000];
          OracleConnection con = null;
          try {
               VideoExample tk = new VideoExample( );
               [1] con = tk.connect( );
               //Include the following line only if you are running
               //an Oracle 8.1.7 database or later.
               //If you are running a database server prior to 8.1.7,
               //the call will fail.
               [2] OrdMediaUtil.imCompatibilityInit(con);
               [3] tk.loadDataFromFile(con);
               tk.extractProperties(con);
               tk.printProperties(con);
               tk.loadDataFromStream(con);
               tk.otherMethods(con);
               tk.loadDataFromByteArray(con);
               [4] con.commit( );
               [5] con.close( );
               System.out.println("Done.");
          }
          [6] catch (Exception e) {
               try {
                    System.out.println("Exception : " + e);
                    con.close( );
               }
               catch(Exception ex) {
                    System.out.println("Close Connection Exception : " + ex);
               }
          }
     }
```

The code in the main( ) method performs the following operations:

1. Uses the connect( ) method to make a connection to a database table.
2. Ensures the compatibility of your application with later releases of the Oracle database. See Section 1.8 for more information.
3. Calls several methods (also defined in VideoExample.java) that manipulate objects on the database server and the local machine.
4. Commits any changes made to the database table.
5. Closes the connection to the database.
6. Handles any errors or exceptions raised by the code.

Section 2.4.2.2 through Section 2.4.2.8 will provide information on the methods called from the main( ) method in the order in which they are called, not in the order they appear in VideoExample.java.

### 2.4.2.2 connect( ) Method

Example 2–34 shows a user-defined method named connect( ), which makes a connection from the application to the database.

***Example 2–34 connect( ) Method (Video)***

```
public OracleConnection connect( ) throws Exception{
     String connectString;
     [1] Class.forName ("oracle.jdbc.driver.OracleDriver");
     [2] connectString = "jdbc:oracle:oci8:@";
     [3] OracleConnection con = (OracleConnection)
          DriverManager.getConnection(connectString,"VIDEOUSER","VIDEOUSER");
     [4] con.setAutoCommit(false);
     return con;
}
```

The connect( ) method performs the following operations:

1. Loads the JDBC drivers directly, because the Oracle database uses a JDK-compliant Java virtual machine.
2. Defines a string that contains the URL of the database to which you will connect. You may need to change this string to match your database.
3. Sets the connection to the database, using the URL contained in connectString, the user name VIDEOUSER, and the password VIDEOUSER. The user name and password were created by VideoExample.sql.
4. Disables the auto-commit mode. This means that you must commit or roll back manually with the commit( ) or rollback( ) methods, respectively.

### 2.4.2.3 loadDataFromFile( ) Method

Example 2–35 shows a user-defined method named loadDataFromFile( ), which uses the *inter*Media loadDataFromFile( ) method to load media data into the application object.

***Example 2–35 loadDataFromFile( ) Method (Video)***

```
public void loadDataFromFile(OracleConnection con){
     try {
          [1] Statement s = con.createStatement( );
          [2] OracleResultSet rs = (OracleResultSet)s.executeQuery
               ("select * from TVID where n = 1 for update ");
          int index = 0;
          [3] while(rs.next( )){
               [4] index = rs.getInt(1);
               [5] OrdVideo vidObj = (OrdVideo) rs.getCustomDatum(2,
                    OrdVideo.getFactory( ));
               [6] vidObj.loadDataFromFile("testvid.dat");
               [7] vidObj.getDataInFile("output1.dat");
               System.out.println("************AFTER getDataInFile ");
               [8] System.out.println("getContentLength output : " +
                    vidObj.getContentLength( ));
               [9] OraclePreparedStatement stmt1 = (OraclePreparedStatement)
                    con.prepareCall("update tvid set vid = ?
                    where n = " + index);
               stmt1.setCustomDatum(1,vidObj);
               stmt1.execute( );
               stmt1.close( ) ;
          }
          System.out.println("loading successful");
     }
     [10] catch(Exception e) {
          System.out.println("exception raised " + e);
          System.out.println("loading unsuccessful");
     }
}
```

The loadDataFromFile( ) method performs the following operations:

1. Creates an OracleStatement object.
2. Executes the given SQL query and puts the results into a local OracleResultSet object. In this case, the SQL query selects the data in the database row where n=1.
3. Performs the operations in the loop while there are results in the OracleResultSet that have not been processed. However, in this case, there is only one row included in the OracleResultSet, so the operations in the loop will run once.
4. Sets an index variable to the value of the integer in the first column of the first row in the OracleResultSet (in this case, the value is 1).

5. Creates a local OrdVideo object named vidObj. Populates vidObj with the contents of the OrdVideo object in the second column of the current row in the OracleResultSet.
6. Uses the loadDataFromFile( ) method to load the media data in testvid.dat into the database OrdVideo object and into vidObj. This also sets the local field on vidObj, but not the database object.
7. Uses the getDataInFile( ) method to get the media data from the application OrdVideo object and load it into a file on the local system named output1.dat.
8. Gets the content length of vidObj and prints it to the screen to verify the success of the loading.
9. Creates and executes a SQL statement that will update the database OrdVideo object with the contents of vidObj. This update will set the local attribute on the database object to match the application object.
10. Handles any errors or exceptions raised by the code.

### 2.4.2.4 extractProperties( ) Method

Example 2–36 shows a user-defined method named extractProperties( ), which sets the properties in the application object.

***Example 2–36 extractProperties( ) Method (Video)***

```
public void extractProperties(OracleConnection con){
     byte[ ] ctx[ ] = new byte [4000][1];
     try {
          [1] Statement s = con.createStatement( );
          OracleResultSet rs = (OracleResultSet)s.executeQuery
               ("select * from TVID where n = 1 for update");
          int index = 0;
          while(rs.next( )){
               index = rs.getInt(1);
               OrdVideo vidObj = (OrdVideo) rs.getCustomDatum(2,
                    OrdVideo.getFactory( ));
               [2] vidObj.setProperties(ctx);
               System.out.println("set Properties called");
               [3] if(vidObj.checkProperties(ctx)) {
                    System.out.println("checkProperties called");
                    System.out.println("setBindParams successful");
                    System.out.println("setProperties successful");
                    System.out.println("checkProperties successful");
                    System.out.println("extraction  successful");
```

```
                }
                else {
                     System.out.println("checkProperties called");
                     System.out.println("extraction not successful");
                     System.out.println("checkProperties successful");
                }
                [4] OraclePreparedStatement stmt1 = (OraclePreparedStatement)
                     con.prepareCall("update tvid set vid = ? where
                     n = " + index);
                stmt1.setCustomDatum(1,vidObj);
                stmt1.execute( );
                stmt1.close( ) ;
           }
           rs.close( );
           s.close( );
      }
      [5] catch(Exception e) {
           System.out.println("exception raised " + e);
           System.out.println("extract prop unsuccessful");
      }
}
```

The extractProperties( ) method performs the following operations:

1. Creates a statement, a local OracleResultSet, and a local OrdVideo object named vidObj, and populates vidObj with media data through the same process described in steps 1 through 5 of Example 2–35. In this method, you will be operating on the contents of the row where n=1.
2. Calls setProperties( ) to extract properties values from the media data and set them in the application OrdVideo object. See "setProperties(byte[ ][ ])" in Chapter 8 for a list of the properties values extracted and set.
3. Calls checkProperties( ) to compare the properties values in the application object with the values in the media data. If all values are the same, checkProperties( ) returns true and the appropriate messages are printed to the screen. If any values differ, checkProperties( ) returns false and the appropriate messages are printed to the screen.
4. Creates and executes a SQL statement that will update the database OrdVideo object with the contents of vidObj.
5. Handles any errors or exceptions raised by the code.

### 2.4.2.5 printProperties( ) Method

Example 2–37 shows a user-defined method named printProperties( ), which prints the attributes of the application object to the screen.

***Example 2–37 printProperties( ) Method (Video)***

```
public void printProperties(OracleConnection con){
     try {
          [1] Statement s = con.createStatement( );
          OracleResultSet rs = (OracleResultSet)s.executeQuery
               ("select * from TVID where n = 1 ");
          int index = 0;
          while(rs.next( )){
               index = rs.getInt(1);
               OrdVideo vidObj = (OrdVideo) rs.getCustomDatum(2,
                    OrdVideo.getFactory( ));
               [2] System.out.println("format: " + vidObj.getFormat( ));
               System.out.println("mimetype: " + vidObj.getMimeType( ));
               System.out.println("width: " + vidObj.getWidth( ));
               System.out.println("height: " + vidObj.getHeight( ));
               System.out.println("frame resolution: " +
                    vidObj.getFrameResolution( ));
               System.out.println("frame rate: " + vidObj.getFrameRate( ));
               System.out.println("video duration: " +
                    vidObj.getVideoDuration( ));
               System.out.println("number of frames: " +
                    vidObj.getNumberOfFrames( ));
               System.out.println("description : " +
                    vidObj.getDescription( ));
               System.out.println("compression type: " +
                    vidObj.getCompressionType( ));
               System.out.println("bit rate: " + vidObj.getBitRate( ));
               System.out.println("num of colors: " +
                    vidObj.getNumberOfColors( ));
          }
     }
     [3] catch(Exception e) {
          System.out.println("exception raised " + e);
          System.out.println("print proerties unsuccessful");
     }
}
```

The printProperties( ) method performs the following operations:

1. Creates a statement, a local OracleResultSet, and a local OrdVideo object named vidObj, and populates vidObj with media data through the same process described in steps 1 through 5 of Example 2–35. In this method, you will be operating on the contents of the row where n=1.
2. Gets the values of the properties in vidObj and prints them to the screen.
3. Handles any errors or exceptions raised by the code.

#### 2.4.2.6 loadDataFromStream( ) Method

Example 2–38 shows a user-defined method named loadDataFromStream( ), which uses the *inter*Media loadDataFromInputStream( ) method to load media data into the application object.

***Example 2–38 loadDataFromStream( ) Method (Video)***

```
public void loadDataFromStream(OracleConnection con){
     try {
          [1] Statement s = con.createStatement( );
          OracleResultSet rs = (OracleResultSet) s.executeQuery
               ("select * from TVID where n = 2 for update ");
          int index = 0;
          while(rs.next( )){
               index = rs.getInt(1);
               OrdVideo vidObj = (OrdVideo) rs.getCustomDatum(2,
                    OrdVideo.getFactory( ));
               [2] FileInputStream  fStream  = new FileInputStream
                    ("testvid.dat");
               [3] vidObj.loadDataFromInputStream(fStream);
               [4] vidObj.getDataInFile("output2.dat");
               [5] fStream.close( );
               System.out.println("************AFTER getDataInFile ");
               [6] System.out.println("getContentLength output : " +
                    vidObj.getContentLength( ));
               [7] OraclePreparedStatement stmt1 = (OraclePreparedStatement)
                    con.prepareCall("update tvid set vid = ?
                    where n = " + index);
               stmt1.setCustomDatum(1,vidObj);
               stmt1.execute( );
               stmt1.close( ) ;
          }
          System.out.println("load data from stream successful");
     }
     [8] catch(Exception e) {
          System.out.println("exception raised " + e);
          System.out.println("load data from stream unsuccessful");
     }
}
```

The loadDataFromStream( ) method performs the following operations:

1. Creates a statement, a local OracleResultSet, and a local OrdVideo object named vidObj, and populates vidObj with media data through the same process described in steps 1 through 5 of Example 2–35. In this method, you will be operating on the contents of the row where n=2.
2. Creates a new FileInputStream object. This input stream contains the contents of the local file testvid.dat.
3. Uses the loadDataFromInputStream( ) method to load the media data in the input stream into the database OrdVideo object and into vidObj. This also sets the local field on vidObj, but not the database object.
4. Uses the getDataInFile( ) method to get the media data from the application OrdVideo object and load it into a file on the local system named output2.dat.
5. Closes the local input stream.
6. Gets the content length of vidObj and prints it to the screen to verify the success of the loading.
7. Creates and executes a SQL statement that will update the database OrdVideo object with the contents of vidObj. This update will set the attributes on the database object to match the application object.
8. Handles any errors or exceptions raised by the code.

#### 2.4.2.7 otherMethods( ) Method

Example 2–39 shows a user-defined method named otherMethods( ), which attempts to use the processSourceCommand( ) method.

***Example 2–39 otherMethods( ) Method (Video)***

```
public void otherMethods(OracleConnection con){
     byte[ ] ctx[ ] = {new byte[4000]};
     byte[ ] res[ ] = {new byte[20]};
     [1] int suc = 1;
     try {
          [2] Statement s1 = con.createStatement( );
          OracleResultSet rs1 = (OracleResultSet)
          s1.executeQuery("select * from TVID where n = 1 for
               update ");
          int index1 = 0;
          while(rs1.next( )) {
               index1 = rs1.getInt(1);
               OrdVideo vidObj = (OrdVideo) rs1.getCustomDatum(2,
                    OrdVideo.getFactory( ));
               [3] try {
                    byte[ ] pSRes = vidObj.processSourceCommand(ctx,
                         "", "", res);
                    suc = 0;
               }
               [4] catch (Exception e) {
                    System.out.println("Expected Exception raised in
                         processSourceCommand(...)" );
               }
               [5] OraclePreparedStatement stmt1 = (OraclePreparedStatement)
                    con.prepareCall("update tvid set vid = ? where
                         n = " + index1);
               stmt1.setCustomDatum(1,vidObj);
               stmt1.execute( );
               stmt1.close( ) ;
          }
          rs1.close( );
          s1.close( );
     }
     [6] catch(Exception e){
          System.out.println("Exception raised " );
     }
     [7] if(suc == 1)
          System.out.println("other methods successful");
     else
          System.out.println("other methods unsuccessful");
}
```

The otherMethods( ) method performs the following operations:

1. Creates an integer that will be used to indicate the success or failure of the method and sets it initially to 1 (for success).
2. Creates a statement, a local OracleResultSet, and a local OrdVideo object named vidObj, and populates vidObj with media data through the same process described in steps 1 through 5 of Example 2–35. In this method, you will be operating on the contents of the row where n=1.

3. Tries to call processSourceCommand( ) with no value specified for the command to be called on the server side. This should raise an exception, which means the code following the processSourceCommand( ) call will not be run and the code in the catch loop will. If an exception is not raised, then the method has failed and the success indicator is set to 0 (for failure).
4. Prints the expected exception that was raised in step 3.
5. Creates and executes a SQL statement that will update the database OrdVideo object with the contents of vidObj.
6. Handles any unexpected errors or exceptions raised by the code.
7. Prints the appropriate message to the screen based on the success or failure of the method.

### 2.4.2.8 loadDataFromByteArray( ) Method

Example 2–40 shows a user-defined method named loadDataFromByteArray( ), which uses the *inter*Media loadDataFromByteArray( ) method to load media data into the application object.

***Example 2–40 loadDataFromByteArray( ) Method (Video)***

```
public void loadDataFromByteArray(OracleConnection con){
     try {
          [1] Statement s = con.createStatement( );
          OracleResultSet rs = (OracleResultSet) s.executeQuery
               ("select * from TVID where n = 3 for update ");
          int index = 0;
          while(rs.next( )){
               index = rs.getInt(1);
               OrdVideo vidObj = (OrdVideo) rs.getCustomDatum(2,
                    OrdVideo.getFactory( ));
               [2] File ff = new File("testvid.dat");
               int fileLength = (int) ff.length( );
               byte[ ] data = new byte[fileLength];
               [3] FileInputStream  fStream  = new
                    FileInputStream("testvid.dat");
               [4] fStream.read(data,0,fileLength);
               [5] vidObj.loadDataFromByteArray(data);
               [6] fStream.close( );
               [7] vidObj.getDataInFile("output3.dat");
               [8] byte[ ] resArr = vidObj.getDataInByteArray( );
               [9] System.out.println("byte array length : " +
                    resArr.length);
```

```
                [10] FileOutputStream outStream = new FileOutputStream
                     ("output4.dat");
                [11] outStream.write(resArr);
                [12] outStream.close( );
                [13] InputStream inpStream = vidObj.getDataInStream( );
                int length = 32768;
                byte[ ] tempBuffer = new byte[32768];
                [14] int numRead = inpStream.read(tempBuffer,0,length);
                try {
                     [15] outStream = new FileOutputStream("output5.dat");
                     [16] while(numRead != -1) {
                          [17] if (numRead < 32768) {
                               length = numRead;
                               outStream.write(tempBuffer,0,length);
                               break;
                          }
                          [18] else
                               outStream.write(tempBuffer,0,length);
                          [19] numRead = inpStream.read(tempBuffer,0,length);
                     }
                }
                [20] finally {
                     outStream.close( );
                     inpStream.close( );
                }
                System.out.println("************AFTER getDataInFile ");
                [21] System.out.println(" getContentLength output : " +
                     vidObj.getContentLength( ));
                [22] OraclePreparedStatement stmt1 = (OraclePreparedStatement)
                     con.prepareCall("update tvid set vid = ? where
                     n = " + index);
                stmt1.setCustomDatum(1,vidObj);
                stmt1.execute( );
                stmt1.close( ) ;
           }
      }
      [23] catch(Exception e) {
           System.out.println("exception raised " + e);
           System.out.println("loadData from byte array unsuccessful");
      }
}
```

The loadDataFromByteArray( ) method performs the following operations:

1. Creates a statement, a local OracleResultSet, and a local OrdVideo object named vidObj, and populates vidObj with media data through the same process described in steps 1 through 5 of Example 2–35. In this method, you will be operating on the contents of the row where n=3.
2. Determines the size (in bytes) of the local file testvid.dat and creates a byte array of the same size.
3. Creates a new FileInputStream object. This input stream contains the contents of testvid.dat.
4. Reads the contents of the input stream into the byte array.
5. Uses the loadDataFromByteArray( ) method to load the media data in the byte array into the database OrdVideo object and into vidObj. This also sets the local field on vidObj, but not the database object.
6. Closes the input stream.
7. Uses the getDataInFile( ) method to get the media data from the application OrdVideo object and load it into a file on the local system named output3.dat.
8. Uses the getDataInByteArray( ) method to get the media data from the application OrdVideo object and load it into a local byte array named resArr.
9. Gets the length of resArr and prints it to the screen to verify the success of the loading.
10. Creates a new FileOutputStream object named outStream. This output stream will write data to a local file named output4.dat.
11. Writes the contents of resArr to output4.dat.
12. Closes the output stream.
13. Creates a new input stream named inpStream. Uses the getDataInStream( ) method to get the media data from the application OrdVideo object and store it in inpStream.
14. Reads 32768 bytes from the beginning (that is, at an offset of 0) of inpStream into the byte array tempBuffer. The integer numRead will be set to the total number of bytes read, or -1 if the end of the input stream has been reached. In this case, if loading is successful, numRead should be equal to 32768.
15. Re-opens OutStream. In this case, it will write data to a local file named output5.dat.
16. Runs the operations in the while loop if numRead is not equal to -1. The program should enter this loop.

17. Writes the number of bytes read into tempBuffer into outStream if numRead is less than 32768 (that is, if all the data was read).
18. If numRead is 32768, writes 32768 bytes into outStream.
19. Attempts to read more data from the input stream into the byte array. If all data has been read, then numRead will be set to -1 and the program will exit the loop. If there is still unread data in the input stream, then it will be read into the byte array and steps 17 and 18 will be repeated.
20. Closes both the input stream and the output stream after exiting the while loop.
21. Gets the content length of vidObj and prints it to the screen to verify the success of the loading.
22. Creates and executes a SQL statement that will update the database OrdVideo object with the contents of vidObj. This update will set the attributes on the database object to match the application object.
23. Handles any errors or exceptions raised by the code.

# 3 OrdAudio Reference Information

The OrdAudio class is used to represent an instance of the ORDSYS.ORDAudio database type in a Java application. The OrdAudio class includes a set of methods to get and set various object attributes, as well as a set of methods that perform various operations on an OrdAudio Java object.

Almost all methods operate on the attributes of the OrdAudio Java object in the application. The exceptions are those methods that access the audio data for read or write purposes, which are described in the following list:

- Methods that operate on the database BLOB specified by the localData attribute, read and write data stored in the database BLOB.
- Methods that operate on the database BFILE specified by the srcLocation and srcName attributes when the srcType attribute is "file," read data from the specified file in the database server.
- Methods that operate on the URL specified by the srcType, srcLocation, and srcName attributes when the srcType attribute is "http," read data from the resource at the specified URL.

If your application modifies the OrdAudio Java object or the audio data in the database, you must update the ORDAudio SQL object in the database to make those changes permanent.

Some methods in the OrdAudio Java class are handed off to a database source plug-in or database format plug-in for processing; these methods have `byte [ ] [ ] ctx` as a context parameter. Applications should allocate a 64-byte array to hold any context information that may be required by a source plug-in or a format plug-in. For example, a plug-in may initialize the context information in one call and use that information in a subsequent call. The source plug-in context requires one array; the format plug-in context requires another array. For most plug-ins, 64 bytes should be sufficient. Some user-defined plug-ins may need additional space. The following example shows how to allocate a plug-in context information array:

```
byte [] [] ctx = new byte[1][64];
```

**Note:** In the current release, no Oracle-supplied source plug-ins or format plug-ins maintain context. Also, not all user-written source plug-ins or format plug-ins maintain context. However, if you include the context parameter as described, your application should work with any current or future source plug-ins or format plug-ins.

See *Oracle interMedia User's Guide and Reference* for more information about plug-ins.

## 3.1 Prerequisites

You will need to include the following import statements in your Java file in order to run *inter*Media methods:

```
import java.sql.*;
import java.io.*;
import oracle.jdbc.driver.*;
import oracle.sql.*;
import oracle.ord.im.*;
```

The examples in this reference chapter are based on the assumption that the following operations have already been performed:

- A connection has been made to a table that contains a column of type OrdAudio.
- A local OrdAudio object named audObj has been created and populated with data.

For examples of making a connection and populating a local object, see Section 2.1.2.

## 3.2 Reference Information

This section presents reference information on the methods that operate on OrdAudio objects.

## checkProperties( )

### Format

public boolean checkProperties(byte[ ] [ ] ctx )

### Description

Checks if the properties of the audio data are consistent with the attributes of the OrdAudio Java object.

### Parameters

**ctx**
The format plug-in context information.

### Return Value

This method returns true if the properties of the audio data are consistent with the attributes of the OrdAudio Java object; false otherwise.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding checkProperties( ) method in the database.

### Examples

```
byte [ ] [ ] ctx = new byte[1][64];
if(audObj.checkProperties(ctx))
     System.out.println("checkProperties successful");
```

where:

- ctx: contains the format plug-in context information.

## clearLocal( )

### Format

public void clearLocal( )

### Description

Clears the local attribute to indicate that the audio data is stored externally.

### Parameters

None.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the local attribute.

### Examples

```
audObj.clearLocal( )
```

## closeSource( )

### Format

public int closeSource(byte[ ] [ ] ctx)

### Description

Closes a data source.

### Parameters

**ctx**
The source plug-in context information.

### Return Value

This method returns the status as an integer, where zero indicates success and a non-zero value indicates a failure code specific to the source plug-in.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding closeSource( ) method in the database.

### Examples

```
byte [ ] [ ] ctx = new byte[1][64];
int i = audObj.closeSource(ctx);
if(i == 0)
     System.out.println("Source close successful");
else
     System.out.println("Source close unsuccessful");
```

where:

- ctx: contains the source plug-in context information.

## deleteContent( )

### Format

public void deleteContent( )

### Description

Deletes any data stored in the database BLOB specified by the localData attribute.

### Parameters

None.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding deleteContent( ) method in the database.

### Examples

```
audObj.deleteContent( );
```

## export( )

### Format

public void export (byte[ ] [ ] ctx, String srcType, String srcLocation, String srcName)

### Description

Exports the data from the BLOB specified by the localData attribute. This method calls the corresponding export( ) method in the database to export the audio data to a location specified by the srcType, srcLocation, and srcName parameters.

Not all source plug-ins support this method. Only the "file" source type is natively supported.

This method will work only if you are running Oracle database server release 8.1.7 or later.

The remainder of this description describes the use of the export( ) method and the Oracle-supplied "file" source plug-in. User-written plug-ins will behave differently.

The export( ) method implemented by the "file" source plug-in copies, but does not modify, the audio data stored in the database BLOB specified by the localData attribute.

After exporting the audio data, all the audio property attributes remain unchanged. However, the srcType, srcLocation, and srcName attributes are updated with values of the srcType, srcLocation, and srcName parameters passed to the export( ) method. After calling the export( ) method, if you no longer intend to manage the audio data within the database, call the clearLocal( ) method to indicate that the audio data is stored outside the database, and call the deleteContent( ) method to delete the audio data stored in the database BLOB.

The export( ) method in the database writes only to a database directory object that the user has privileges to access. That is, you can access a directory that you have created using the SQL CREATE DIRECTORY statement, or one to which you have been granted READ access. To execute the CREATE DIRECTORY statement, you must have the CREATE ANY DIRECTORY privilege. In addition, you must use the DBMS_JAVA.GRANT_PERMISSION method to specify which files can be written.

For example, the following SQL*Plus command grants the user, MEDIAUSER, the permission to write to the file named filmtrack1.au:

```
CALL DBMS_JAVA.GRANT.PERMISSION(
    'MEDIAUSER',
    'java.io.FilePermission',
    '/audio/movies/filmtrack1.au',
    'write');
```

The previous example shows how to authorize access to write to a single file. In addition, there are various wildcard path specifications that authorize write access to multiple directories and file names. For example, a path specification that ends in a slash and asterisk (/*), where the slash is the file-separator character that is operating-system dependent, indicates all the files contained in the specified directory. A path specification that ends with a slash and hyphen (/-) indicates all files contained in the specified directory and all its subdirectories. A path name consisting of the special token <<ALL FILES>> authorizes access to any file.

See *Oracle9i Java Developer's Guide* and the java.io.FilePermission class in the Java API for more information about security and performance. See *Oracle interMedia User's Guide and Reference* for more information about the required privileges.

## Parameters

**ctx**
The source plug-in context information.

**srcType**
The source type to which the content will be exported.

**srcLocation**
The source location to which the content will be exported.

**srcName**
The source name to which the content will be exported.

## Return Value

None.

## Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding export( ) method in the database.

## Examples

```
byte [ ] [ ] ctx = new byte[1][64];
audObj.export(ctx,"file","AUDIODIR","complete.wav");
```

where:

- ctx: contains the source plug-in context information.
- file: is the source plug-in used to export the content.
- AUDIODIR: is the location to which the content will be exported.
- complete.wav: is the file to which the content will be exported.

## getAllAttributes( )

### Format

public CLOB getAllAttributes(byte[ ] [ ] ctx)

### Description

Returns the values of the audio properties in a temporary CLOB in a form defined by the format plug-in. For natively supported formats, the information is presented as a comma-separated list of attributes. The list of attributes is of the form attributeName=attributeValue, where the list contains the following attributes: format, mimeType, encoding, numberOfChannels, samplingRate, sampleSize, compressionType, and audioDuration. For user-defined formats, the information is presented in a form defined by the format plug-in.

**Note:** The application must free the temporary CLOB after reading the information it contains.

### Parameters

**ctx**
The format plug-in context information.

### Return Value

This method returns the values of the attributes as a temporary oracle.sql.CLOB.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding getAllAttributes( ) method in the database.

### Examples

```
byte [ ] ctx[ ] = new byte[1][64];
CLOB attributes = audObj.getAllAttributes(ctx);
```

where:

- ctx: contains the format plug-in context information.

## getAttribute( )

### Format

public String getAttribute(byte[ ] [ ] ctx, String name)

### Description

Returns the value of the requested audio property. This method is used by user-defined format plug-ins to return the value of an audio property that is not available as an attribute of the OrdAudio Java object. This method is not implemented by any Oracle-supplied format plug-ins.

### Parameters

**ctx**
The format plug-in context information.

**name**
The property or attribute name.

### Return Value

This method returns the value of the attribute, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding getAttribute( ) method in the database.

### Examples

```
byte [ ] [ ] ctx = new byte[1][64];
int attribute = audObj.getAttribute(ctx,"numberOfChannels")
```

where:

- ctx: contains the format plug-in context information.
- numberOfChannels: is the value of the attribute to get from the object.

## getAudioDuration( )

### Format

public int getAudioDuration( )

### Description

Returns the value of the audioDuration attribute.

### Parameters

None.

### Return Value

This method returns the value of the audioDuration attribute, as an integer.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the audioDuration attribute.

### Examples

```
int audioDuration = audObj.getAudioDuration( );
```

## getBFILE( )

### Format

public oracle.sql.BFILE getBFILE( )

### Description

Returns a BFILE locator from the database when the srcType attribute is "file." This method calls the corresponding getBFILE( ) method in the database, which creates the BFILE using the srcLocation and srcName attributes.

### Parameters

None.

### Return Value

This method returns an oracle.sql.BFILE locator.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding getBFILE( ) method in the database.

### Examples

```
BFILE audioBFILE = audObj.getBFILE( );
```

## getComments( )

### Format

public oracle.sql.CLOB getComments( )

### Description

Returns the CLOB locator from the comments attribute.

### Parameters

None.

### Return Value

This method returns the value of the comments attribute as an oracle.sql.CLOB locator.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the comments attribute.

### Examples

```
CLOB comments = audObj.getComments( )
```

## getCompressionType( )

### Format

public String getCompressionType( )

### Description

Returns the value of the compressionType attribute.

### Parameters

None.

### Return Value

This method returns the value of the compressionType attribute, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the compressionType attribute.

### Examples

```
String compressionType = audObj.getCompressionType( );
```

## getContent( )

### Format

public oracle.sql.BLOB getContent( )

### Description

Returns the BLOB locator from the localData attribute.

### Parameters

None.

### Return Value

This method returns an oracle.sql.BLOB locator.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the localData attribute.

### Examples

```
BLOB localContent = audObj.getContent( );
```

## getContentInLob( )

### Format

public oracle.sql.BLOB getContentInLob(byte[ ] [ ] ctx, String mimetype[ ], String format[ ])

### Description

Returns the data from the BLOB specified by the localData attribute in a temporary BLOB in the database. This method creates a temporary BLOB in the database, reads the data from the BLOB specified by the localData attribute, writes the data to the temporary BLOB, then returns the temporary BLOB locator to the caller.

**Note:** The application must free the temporary BLOB after accessing the data it contains.

### Parameters

**ctx**
The format plug-in context information.

**mimetype**
A String array, 1 element in length, into which the mimeType attribute is written as element 0.

**format**
A String array, 1 element in length, into which the format attribute is written as element 0.

### Return Value

This method returns the audio data in a temporary oracle.sql.BLOB locator.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs creating the temporary BLOB or executing the corresponding getContentInLob( ) method in the database.

## Examples

```
byte [ ] [ ] ctx = new byte[1][64];
String mimetype[ ] = new String[1];
String format[ ] = new String[1];
BLOB localContent = audObj.getContentInLob(ctx,mimetype,format);
```

where:

- ctx: contains the format plug-in context information.
- mimetype: is an array of Strings whose first value contains the MIME type. This value is generated by the server.
- format: is an array of Strings whose first value contains the format. This value is generated by the server.

## getContentLength( )

### Format

public int getContentLength( )

### Description

Returns the length of the audio data. This method calls the corresponding getContentLength( ) method in the database.

This method is not supported for all source types. For example, the "http" source type does not support this method.

### Parameters

None.

### Return Value

This method returns the value of the contentLength attribute, as an integer.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding getContentLength( ) method in the database.

### Examples

```
int contentLength = audObj.getContentLength( );
```

## getContentLength(byte[ ][ ])

### Format

public int getContentLength(byte[ ] [ ] ctx)

### Description

Returns the length of the audio data using source plug-in context information. This method calls the corresponding getContentLength( ) method in the database.

This method is not supported for all source types. For example, the "http" source type does not support this method.

### Parameters

**ctx**
The source plug-in context information.

### Return Value

This method returns the value of the contentLength attribute, as an integer.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding getContentLength( ) method in the database.

### Examples

```
byte [ ] [ ] ctx = new byte[1][64];
int contentLength = audObj.getContentLength(ctx);
```

where:

- ctx: contains the source plug-in context information.

## getDataInByteArray( )

### Format

public byte[ ] getDataInByteArray( )

### Description

Returns a byte array containing the data from the database BLOB specified by the localData attribute.

### Parameters

None.

### Return Value

This method returns the byte array containing the data.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing an object attribute.

java.io.IOException

This exception is thrown if an error occurs reading the data from the BLOB.

java.lang.OutOfMemoryError

This exception is thrown if sufficient memory cannot be allocated to hold the data.

### Examples

```
byte[ ] byteArr = audObj.getDataInByteArray( );
```

## getDataInFile( )

### Format

public boolean getDataInFile(String filename)

### Description

Writes the data from the database BLOB specified by the localData attribute to a local file.

### Parameters

**filename**
The name of the file to which the data will be written.

### Return Value

This method returns true if the data is written to the file successfully; otherwise, an exception is raised if an error occurs. This method never returns false.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing an object attribute.

java.io.IOException

This exception is thrown if an error occurs reading the data from the BLOB or writing the data to the output file.

### Examples

```
boolean load = audObj.getDataInFile("output1.dat");
if(load)
     System.out.println("getDataInFile completed successfully");
else
     System.out.println("Error in getDataInFile");
```

where:

- output1.dat: is the file to which the data will be written.

## getDataInStream( )

### Format

public InputStream getDataInStream( )

### Description

Returns an InputStream object from which the data in the database BLOB specified by the localData attribute can be read.

### Parameters

None.

### Return Value

This method returns an InputStream object from which the data will be read.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing an object attribute.

### Examples

```
InputStream inpStream = audObj.getDataInStream( );
```

## getDescription( )

### Format

public String getDescription( )

### Description

Returns the value of the description attribute.

### Parameters

None.

### Return Value

This method returns the value of the description attribute, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the description attribute.

### Examples

```
String desc = audObj.getDescription( );
```

## getEncoding( )

### Format

public String getEncoding( )

### Description

Returns the value of the encoding attribute.

### Parameters

None.

### Return Value

This method returns the value of the encoding attribute, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the encoding attribute.

### Examples

```
String encoding = audObj.getEncoding( );
```

## getFactory( )

### Format

public static oracle.sql.CustomDatumFactory getFactory( )

### Description

Returns the OrdAudio CustomDatumFactory interface for use by the getCustomDatum( ) method. Specify the getFactory( ) method as the factory parameter of the getCustomDatum( ) method when retrieving an OrdAudio object from an OracleResultSet or OracleCallableStatement object.

### Parameters

None.

### Return Value

This method returns the OrdAudio implementation of the CustomDatumFactory interface.

### Exceptions

None.

### Examples

```
OrdAudio aud = (OrdAudio)rset.getCustomDatum( 1, OrdAudio.getFactory() );
```

## getFormat( )

### Format

public String getFormat( )

### Description

Returns the value of the format attribute.

### Parameters

None.

### Return Value

This method returns the value of the format attribute, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the format attribute.

### Examples

```
String format = audObj.getFormat( );
```

## getMimeType( )

### Format

public String getMimeType( )

### Description

Returns the value of the mimeType attribute.

### Parameters

None.

### Return Value

This method returns the value of the mimeType attribute, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the mimeType attribute.

### Examples

```
String mimeType = audObj.getMimeType( );
```

## getNumberOfChannels( )

### Format

public int getNumberOfChannels( )

### Description

Returns the value of the numberOfChannels attribute.

### Parameters

None.

### Return Value

This method returns the value of the numberOfChannels attribute, as an integer.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the numberOfChannels attribute.

### Examples

```
int channels = audObj.getNumberOfChannels( );
```

## getSampleSize( )

### Format

public int getSampleSize( )

### Description

Returns the value of the sampleSize attribute.

### Parameters

None.

### Return Value

This method returns the value of the sampleSize attribute, as an integer.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the sampleSize attribute.

### Examples

```
int sampleSize = audObj.getSampleSize( );
```

## getSamplingRate( )

### Format

public int getSamplingRate( )

### Description

Returns the value of the samplingRate attribute.

### Parameters

None.

### Return Value

This method returns the value of the samplingRate attribute, as an integer.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the samplingRate attribute.

### Examples

```
int samplingRate = audObj.getSamplingRate( );
```

## getSource( )

### Format

public String getSource( )

### Description

Returns the source information in the form: srcType://srcLocation/srcName.

### Parameters

None.

### Return Value

This method returns the source information, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding getSource( ) method in the database.

### Examples

```
String source = audObj.getSource( );
```

## getSourceLocation( )

### Format

public String getSourceLocation( )

### Description

Returns the value of the srcLocation attribute.

### Parameters

None.

### Return Value

This method returns the value of the srcLocation attribute, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the srcLocation attribute.

### Examples

```
String location = audObj.getSourceLocation( );
```

## getSourceName( )

### Format

public String getSourceName( )

### Description

Returns the value of the srcName attribute.

### Parameters

None.

### Return Value

This method returns the value of the srcName attribute, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the srcName attribute.

### Examples

```
String name = audObj.getSourceName( );
```

## getSourceType( )

### Format

public String getSourceType( )

### Description

Returns the value of the srcType attribute.

### Parameters

None.

### Return Value

This method returns the value of the srcType attribute, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the srcType attribute.

### Examples

```
String type = audObj.getSourceType( );
```

## getUpdateTime( )

### Format

public java.sql.Timestamp getUpdateTime( )

### Description

Returns the value of the updateTime attribute.

### Parameters

None.

### Return Value

This method returns the value of the updateTime attribute as a java.sql.Timestamp object.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the updateTime attribute.

### Examples

```
Timestamp time = audObj.getUpdateTime( );
```

## importData( )

### Format

public void importData(byte[ ] [ ] ctx)

### Description

Imports data from an external source into the database BLOB specified by the localData attribute. The external data source is specified by the srcType, srcLocation, and srcName attributes.

### Parameters

**ctx**
The source plug-in context information.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding import( ) method in the database.

### Examples

```
byte [ ] [ ] ctx = new byte[1][64];
audObj.importData(ctx);
```

where:

- ctx: contains the source plug-in information.

## importFrom( )

### Format

public void importFrom(byte[ ] [ ] ctx, String srcType, String srcLocation, String srcName)

### Description

Imports data from an external source into the database BLOB specified by the localData attribute. The external data source is specified by the srcType, srcLocation, and srcName parameters. The srcType, srcLocation, and srcName attributes are updated with values of the srcType, srcLocation, and srcName parameters passed to the importFrom( ) method.

### Parameters

**ctx**
The source plug-in context information. See *Oracle interMedia User's Guide and Reference* for more information.

**srcType**
The source type from which the data will be imported.

**srcLocation**
The source location from which the data will be imported.

**srcName**
The source name from which the data will be imported.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding importFrom( ) method in the database.

## Examples

```
byte [ ] [ ] ctx = new byte[1][64];
audObj.importFrom("file","AUDIODIR","testaud.dat");
```

where:

- ctx: contains the source plug-in context information.
- file: is the source plug-in used to import the data.
- AUDIODIR: is the location of the file on the database server from which the data will be imported.
- testaud.dat: is the file from which the data will be imported.

## isLocal( )

### Format

public boolean isLocal( )

### Description

Indicates if the audio data is stored locally in the database in a BLOB specified by the localData attribute.

### Parameters

None.

### Return Value

This method returns true if the data is stored locally in the database in a BLOB; false otherwise.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the localData attribute.

### Examples

```
if(audObj.isLocal( ))
     System.out.println("local attribute is set to true");
else
     System.out.println("local attribute is set to false");
```

## loadDataFromByteArray( )

### Format

public boolean loadDataFromByteArray(byte[ ] byteArr)

### Description

Loads data from a byte array into the database BLOB specified by the localData attribute. Before loading the data, this method calls the deleteContent( ) method to delete any existing data in the BLOB. It also calls the setLocal( ) method to set the local flag. In addition, this method calls the setUpdateTime( ) method to set the updateTime attribute to the database server's current SYSDATE time.

### Parameters

**byteArr**
A byte array from which the data will be loaded.

### Return Value

This method returns true if the data is loaded successfully; otherwise, an exception is raised if an error occurs. This method never returns false.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing an object attribute or executing a method in the database.

java.io.IOException

This exception is thrown if an error occurs reading the byte array.

### Examples

```
byte[ ] data = new byte[32000];
FileInputStream fStream = new FileInputStream("testaud.dat");
fStream.read(data,0,32000);
boolean success = audObj.loadDataFromByteArray(data);
if(success)
     System.out.println("loadDataFromByteArray was successful");
else
```

```
     System.out.println("loadDataFromByteArray was unsuccessful");
```

where:

- data: is the local byte array from which the data will be loaded.
- testaud.dat: is a local file that contains 32,000 bytes of data.

## loadDataFromFile( )

### Format

public boolean loadDataFromFile(String filename)

### Description

Loads data from a file into the database BLOB specified by the localData attribute. Before loading the data, this method calls the deleteContent( ) method to delete any existing data in the BLOB. It also calls the setLocal( ) method to set the local flag. In addition, this method calls the setUpdateTime( ) method to set the updateTime attribute to the database server's current SYSDATE time.

### Parameters

**filename**
The name of the file from which the data will be loaded.

### Return Value

This method returns true if the data is loaded successfully; otherwise, an exception is raised if an error occurs. This method never returns false.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing an object attribute or executing a method in the database.

java.io.IOException

This exception is thrown if an error occurs reading the data file.

### Examples

```
audObj.loadDataFromFile("testaud.dat");
```

where:

- testaud.dat: is a local file that contains audio data.

## loadDataFromInputStream( )

### Format

public boolean loadDataFromInputStream(InputStream inpStream)

### Description

Loads data from an InputStream object into the database BLOB specified by the localData attribute. Before loading the data, this method calls the deleteContent( ) method to delete any existing data in the BLOB. It also calls the setLocal( ) method to set the local flag. In addition, this method calls the setUpdateTime( ) method to set the updateTime attribute to the database server's current SYSDATE time.

### Parameters

**inpStream**
The InputStream object from which the data will be loaded.

### Return Value

This method returns true if the data is loaded successfully; otherwise, an exception is raised if an error occurs. This method never returns false.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing an object attribute or executing a method in the database.

java.io.IOException

This exception is thrown if an error occurs reading the InputStream object.

### Examples

```
FileInputStream fStream = new FileInputStream("testaud.dat");
audObj.loadDataFromInputStream(fStream);
```

where:

- testaud.dat: is a local file that contains audio data.

- fStream: is the local InputStream object that will load audio data into the OrdAudio object.

## openSource( )

### Format

public int openSource(byte[ ] userarg, byte[ ] [ ] ctx)

### Description

Opens a data source.

### Parameters

**userarg**
Additional source plug-in information that may be required by user-defined source plug-ins.

**ctx**
The source plug-in context information. See *Oracle interMedia User's Guide and Reference* for more information.

### Return Value

This method returns the status as an integer, where zero indicates success and a non-zero value indicates a failure code specific to the source plug-in.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding openSource( ) method in the database.

### Examples

```
byte[ ] userarg = new byte[64];
byte [ ] [ ] ctx = new byte[1][64];
int i = audObj.openSource(userarg,ctx);
if(i == 0)
     System.out.println("openSource successful");
else
     System.out.println("openSource unsuccessful");
```

where:

- userarg: contains permission-related parameters.
- ctx: contains the source plug-in context information.

## processAudioCommand( )

### Format

public byte[ ] processAudioCommand(byte[ ] [ ] ctx, String cmd, String args, byte[ ] [ ] result)

### Description

Calls the format plug-in in the database to execute a command. This method is used with user-written format plug-ins only; it raises an exception if used with the format plug-ins supplied by Oracle.

### Parameters

**ctx**
The format plug-in context information.

**cmd**
The command to be executed by the format plug-in.

**args**
The arguments of the command.

**result**
A byte array of the form [1][*n*] into which the result of the command execution is written.

### Return Value

This method returns the results of executing the command.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding processAudioCommand( ) method in the database.

### Examples

```
byte [ ] [ ] ctx = new byte[1][64]
String cmd;
String args;
```

```
byte [ ] [ ] result;
//assign a command value to cmd
//assign any arguments to args
byte[ ] commandResults = audObj.processAudioCommand(ctx,cmd,
     args,result);
```

where:

- ctx: contains the format plug-in information.
- cmd: is the command to be run.
- args: contains any arguments required by the command.
- result: is the results of executing the command.

## processSourceCommand( )

### Format

public byte[ ] processSourceCommand(byte[ ] [ ] ctx, String cmd, String args, byte[ ] [ ] result)

### Description

Calls the source plug-in in the database to execute a command. This method is used with user-written plug-ins only; it raises an exception if used with the source plug-ins supplied by Oracle.

### Parameters

**ctx**
The source plug-in context information. See *Oracle interMedia User's Guide and Reference* for more information.

**cmd**
The command to be executed by the source plug-in.

**args**
The arguments of the command.

**result**
A byte array of the form [1][*n*] into which the result of the command execution is written.

### Return Value

This method returns the results of executing the command.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding processSourceCommand( ) method in the database.

### Examples

```
byte [ ] [ ] ctx = new byte[1][64];
```

```
String cmd;
String args;
byte [ ] [ ] result;
//assign a command value to cmd
//assign any arguments to args
byte[ ] commandResults = audObj.processSourceCommand(ctx,cmd,
     args,result);
```

where:

- ctx: contains the source plug-in information.
- cmd: is the command to be run.
- args: contains any arguments required by the command.
- result: is the results of executing the command.

## readFromSource( )

### Format

public int readFromSource(byte[ ] [ ] ctx, int startpos, int numbytes, byte[ ] [ ] buffer)

### Description

Reads data from the data source. This method reads the specified number of bytes into the application buffer from the data source, starting at the specified position in the data source.

Not all source plug-ins require that the data source be opened before it can be read. However, to ensure that an application will work with any current or future source plug-ins, call the openSource( ) method before calling this method.

### Parameters

**ctx**
The source plug-in context information. See *Oracle interMedia User's Guide and Reference* for more information.

**startpos**
The start position in the data source.

**numbytes**
The number of bytes to be read from the data source.

**buffer**
A byte array of the form [1][*n*], where *n* is greater than or equal to numbytes.

### Return Value

This method returns the number of bytes read, as an integer.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding readFromSource( ) method in the database.

## Examples

```
byte [ ] [ ] ctx = new byte[1][64];
byte [ ] [ ] commentBuffer = new byte[12];
int i = audObj.readFromSource(ctx,0,12,commentBuffer);
```

where:

- ctx: contains the source plug-in context information.
- 0: is the position to begin reading from the comments field.
- 12: is the number of bytes to be read.
- commentBuffer: is the location to which the data will be read.

## setAudioDuration( )

### Format

public void setAudioDuration(int audioDuration)

### Description

Sets the value of the audioDuration attribute.

The setProperties( ) method sets this attribute automatically for certain audio formats; use this method only if you are not using the setProperties( ) method. This method sets only the attribute value; it does not modify the audio data itself.

### Parameters

**audioDuration**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the audioDuration attribute.

### Examples

```
audObj.setAudioDuration(16);
```

where:

- 16: is the value to be set, in seconds, in the audioDuration attribute.

## setComments( )

### Format

public void setComments(oracle.sql.CLOB comments)

### Description

Sets the value of the comments attribute.

The comments attribute is reserved for use by *inter*Media. You can set your own value, but it could be overwritten by Oracle *inter*Media Annotator or by the setProperties( ) method.

### Parameters

**comments**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the comments attribute.

### Examples

```
audObj.setComments(commentsData);
```

where:

- commentsData: is a CLOB that contains data to be set in the comments attribute.

## setCompressionType( )

### Format

public void setCompressionType(String compressionType)

### Description

Sets the value of the compressionType attribute.

The setProperties( ) method sets this attribute automatically for certain audio formats; use this method only if you are not using the setProperties( ) method. This method sets only the attribute value; it does not modify the audio data itself.

### Parameters

**compressionType**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the compressionType attribute.

### Examples

```
audObj.setCompressionType("8BITMONOAUDIO");
```

where:

- 8BITMONOAUDIO: is the value to be set in the compressionType attribute.

## setDescription( )

### Format

public void setDescription(String description)

### Description

Sets the value of the description attribute.

### Parameters

**description**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the description attribute.

### Examples

```
audObj.setDescription("My audio file");
```

where:

- My audio file: is the value to be set in the description attribute.

## setEncoding( )

### Format

public void setEncoding(String encoding)

### Description

Sets the value of the encoding attribute.

The setProperties( ) method sets this attribute automatically for certain audio formats; use this method only if you are not using the setProperties( ) method. This method sets only the attribute value; it does not modify the audio data itself.

### Parameters

**encoding**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the encoding attribute.

### Examples

```
audObj.setEncoding("MULAW");
```

where:

- MULAW: is the value to be set in the encoding attribute.

## setFormat( )

### Format

public void setFormat(String format)

### Description

Sets the value of the format attribute.

The format attribute determines which format plug-in is used to handle calls to methods that operate on the audio data. In particular, the setProperties( ) method uses the format attribute to determine which format plug-in to call to parse the audio data properties. See the setProperties( ) method for more information on how to initialize the format attribute before calling the setProperties( ) method, and for information on how the setProperties( ) method in the default, Oracle-supplied plug-in, sets the value of the format attribute. Calling the setFormat( ) method sets only the attribute value; it does not modify the audio data itself.

### Parameters

**format**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the format attribute.

### Examples

```
audObj.setFormat("AUFF");
```

where:

- AUFF: is the value to be set in the format attribute.

## setKnownAttributes( )

### Format

```
public void setKnownAttributes(String format, String encoding,
                               int numberOfChannels, int samplingRate,
                               int sampleSize, String compressionType,
                                int audioDuration)
```

### Description

Sets the values of the known attributes of the OrdAudio Java object.

The setProperties( ) method sets the following attributes automatically for certain audio formats: format, encoding, numberOfChannels, samplingRate, sampleSize, compressionType, and audioDuration. Use this method only if you are not using the setProperties( ) method. This method sets only the specified attribute values; it does not modify the audio data itself.

### Parameters

**format**
The new attribute value, as a String.

**encoding**
The new attribute value, as a String.

**numberOfChannels**
The new attribute value, as an integer.

**samplingRate**
The new attribute value, as an integer.

**sampleSize**
The new attribute value, as an integer.

**compressionType**
The new attribute value, as a String.

**audioDuration**
The new attribute value, as an integer.

## Return Value

None.

## Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding setKnownAttributes( ) method in the database.

## Examples

```
audObj.setKnownAttributes("AUFF","MULAW",1,8,8,"8BITMONOAUDIO",16);
```

where:

- AUFF: is the value to be set in the format attribute.
- MULAW: is the value to be set in the encoding attribute.
- 1: is the value to be set in the numberOfChannels attribute.
- 8: is the value to be set, in samples per second, in the samplingRate attribute.
- 8: is the value to be set in the sampleSize attribute.
- 8BITMONOAUDIO: is the value to be set in the compressionType attribute.
- 16: is the value to be set, in seconds, in the audioDuration attribute.

## setLocal( )

### Format

public void setLocal( )

### Description

Sets the value of the local attribute to indicate that the audio data is stored locally in the database in a BLOB specified by the localData attribute.

### Parameters

None.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the localData attribute.

### Examples

```
audObj.setLocal( );
```

## setMimeType( )

### Format

public void setMimeType(String mimeType)

### Description

Sets the value of the mimeType attribute.

The setProperties( ) method sets this attribute automatically for certain audio formats; use this method only if you are not using the setProperties( ) method. This method sets only the attribute value; it does not modify the audio data itself.

### Parameters

**mimeType**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the mimeType attribute.

### Examples

```
audObj.setMimeType("audio/basic");
```

where:

- audio/basic: is the MIME type to be set.

## setNumberOfChannels( )

### Format

public void setNumberOfChannels(int numberOfChannels)

### Description

Sets the value of the numberOfChannels attribute.

The setProperties( ) method sets this attribute automatically for certain audio formats; use this method only if you are not using the setProperties( ) method. This method sets only the attribute value; it does not modify the audio data itself.

### Parameters

**numberOfChannels**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the numberOfChannels attribute.

### Examples

```
audObj.setNumberOfChannels(1);
```

where:

- 1: is the value to be set in the numberOfChannels attribute.

## setProperties(byte[ ][ ])

### Format

public void setProperties(byte[ ] [ ] ctx)

### Description

Parses the audio data properties and sets the values of the attributes in the OrdAudio Java object. This method sets the values of the format, mimeType, encoding, numberOfChannels, samplingRate, sampleSize, compressionType, and audioDuration attributes. An attribute is set to null if the corresponding property cannot be extracted for a specific audio format. This method throws a SQLException error if the audio format is not recognized.

The format attribute determines which format plug-in is used to parse the audio data properties. If the format attribute is null when the setProperties( ) method is called, then the default, Oracle-supplied, format plug-in is used to parse the audio data properties and fill in various attributes, including the actual audio data format, for supported audio formats. See *Oracle interMedia User's Guide and Reference* for information on the audio formats supported by the Oracle-supplied format plug-ins. Note that the ORDAudio.init methods in the database always set the value of the format attribute to null. If the format attribute is not null, then the format plug-in specified by the format attribute will be called when the setProperties( ) method is called.

### Parameters

**ctx**
The format plug-in context information.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding setProperties( ) method in the database.

## Examples

```
byte [ ] [ ] ctx = new byte[1][64];
audObj.setProperties(ctx);
```

where:

- ctx: contains the format plug-in context information.

## setProperties(byte[ ][ ], boolean)

### Format

public void setProperties(byte[ ] [ ] ctx, boolean setComments)

### Description

Parses the audio data properties, sets the values of the attributes in the OrdAudio Java object, and optionally populates the CLOB specified by the comments attribute. This method sets the values of the format, mimeType, encoding, numberOfChannels, samplingRate, sampleSize, compressionType, and audioDuration attributes. An attribute is set to null if the corresponding property cannot be extracted for a specific audio format. If the setComments parameter is true, this method also populates the CLOB specified by the comments attribute with all extracted properties in XML form. If the setComments parameter is false, the comments attribute is not modified. This method throws a SQLException error if the audio format is not recognized.

The format attribute determines which format plug-in is used to parse the audio data properties. If the format attribute is null when the setProperties( ) method is called, then the default, Oracle-supplied, format plug-in is used to parse the audio data properties and fill in various attributes, including the actual audio data format, for supported audio formats. See *Oracle interMedia User's Guide and Reference* for information on the audio formats supported by the Oracle-supplied format plug-ins. Note that the ORDAudio.init methods in the database always set the value of the format attribute to null. If the format attribute is not null, then the format plug-in specified by the format attribute will be called when the setProperties( ) method is called.

### Parameters

**ctx**
The format plug-in context information.

**setComments**
A Boolean value that specifies whether or not to populate the CLOB specified by the comments attribute.

## Return Value

None.

## Exceptions

java.sql.SQLException

This exception is thrown  if an error occurs executing the corresponding setProperties( ) method in the database.

## Examples

```
byte [ ] [ ] ctx = new byte[1][64];
audObj.setProperties(ctx,true);
```

where:

- ctx: contains the format plug-in context information.
- true: indicates that the comments field will be populated.

## setSampleSize( )

### Format

public void setSampleSize(int sampleSize)

### Description

Sets the value of the sampleSize attribute.

The setProperties( ) method sets this attribute automatically for certain audio formats; use this method only if you are not using the setProperties( ) method. This method sets only the attribute value; it does not modify the audio data itself.

### Parameters

**sampleSize**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the sampleSize attribute.

### Examples

```
audObj.setSampleSize(8);
```

where:

- 8: is the value to be set in the sampleSize attribute.

## setSamplingRate( )

### Format

public void setSamplingRate(int samplingRate)

### Description

Sets the value of the samplingRate attribute.

The setProperties( ) method sets this attribute automatically for certain audio formats; use this method only if you are not using the setProperties( ) method. This method sets only the attribute value; it does not modify the audio data itself.

### Parameters

**samplingRate**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the samplingRate attribute.

### Examples

```
audObj.setSamplingRate(8);
```

where:

- 8: is the value to be set, in samples per second, in the samplingRate attribute.

## setSource( )

### Format

public void setSource(String srcType, String srcLocation, String srcName)

### Description

Sets the values of the srcType, srcLocation, and srcName attributes.

### Parameters

**srcType**
The type of the source.

**srcLocation**
The location of the source.

**srcName**
The name of the source.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the srcType, srcLocation, or srcName attributes.

### Examples

```
audObj.setSource("file","AUDIODIR","audio.au");
```

where:

- file: is the source type.
- AUDIODIR: is the source location.
- audio.au: is the source name.

## setUpdateTime( )

### Format

public void setUpdateTime(java.sql.Timestamp currentTime)

### Description

Sets the value of the updateTime attribute. This method sets the value of the updateTime attribute to the specified time, or to the database server's current SYSDATE time if the currentTime attribute is specified as null.

### Parameters

**currentTime**
The update time, or the null value, used to set the value of the updateTime attribute to the database server's current SYSDATE time.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding setUpdateTime( ) method in the database.

### Examples

```
audObj.setUpdateTime(null);
```

## trimSource( )

### Format

public int trimSource(byte[ ] [ ] ctx, int newLen)

### Description

Trims the data to the specified length.

Not all source plug-ins support trim operations. For example, applications can trim the data stored in a BLOB specified by the localData attribute; however, the "file" and "http" data source types do not support write access, and so do not support this method. Furthermore, those source plug-ins that do support write access may not support the trim operation.

Not all source plug-ins require that the data source be opened before it can be modified. However, to ensure that an application will work with any current or future source plug-ins, call the openSource( ) method before calling this method.

### Parameters

**ctx**
The source plug-in context information. See *Oracle interMedia User's Guide and Reference* for more information.

**newLen**
The length to which the data is to be trimmed.

### Return Value

This method returns the status as an integer, where zero indicates success and a non-zero value indicates a failure code specific to the source plug-in.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding trimSource( ) method in the database.

## Examples

```
byte [ ] [ ] ctx = new byte[1][64];
int i = audObj.trimSource(ctx,10);
if (i == 0)
     System.out.println("trimSource successful");
else
     System.out.println("trimSource unsuccessful");
```

where:

- ctx: contains the source plug-in context information.
- 10: is the new length of the source.

## writeToSource( )

### Format

public int writeToSource(byte[ ] [ ] ctx, int startpos, int numbytes, byte[ ] buffer)

### Description

Writes data to the data source. This method writes the specified number of bytes from the application buffer to the data source, starting at the specified position in the data source.

Not all source plug-ins support write operations. For example, applications can write to a BLOB specified by the localData attribute; however, the "file" and "http" data source types do not support write access, and so do not support this method. Furthermore, those source plug-ins that do support write access may support only sequential write access, and may not support write access to arbitrary starting positions within the data source.

Not all source plug-ins require that the data source be opened before it can be written. However, to ensure that an application will work with any current or future source plug-ins, call the openSource( ) method before calling this method.

### Parameters

**ctx**
The source plug-in context information. See *Oracle interMedia User's Guide and Reference* for more information.

**startpos**
The start position in the data source.

**numbytes**
The number of bytes to be written to the data source.

**buffer**
A byte array containing the data to be written.

### Return Value

This method returns the number of bytes written, as an integer.

## Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding writeToSource( ) method in the database.

## Examples

```
byte [ ] [ ] ctx = new byte[1][64];
byte[ ] data = new byte[20];
//populate data with 20 bytes of content
int i = audObj.writeToSource(ctx,1,20,data);
```

where:

- ctx: contains the source plug-in context information.
- 1: is the position in the comments field where writing will begin.
- 20: is the number of bytes to be written.
- data: contains the content to be written.

# 4 OrdDoc Reference Information

The OrdDoc class is used to represent an instance of the ORDSYS.ORDDoc database type in a Java application. The OrdDoc class includes a set of methods to get and set various object attributes, as well as a set of methods that perform various operations on an OrdDoc Java object.

Almost all methods operate on the attributes of the OrdDoc Java object in the application. The exceptions are those methods that access the media data for read or write purposes, which are described in the following list:

- Methods that operate on the database BLOB specified by the localData attribute, read and write data stored in the database BLOB.
- Methods that operate on the database BFILE specified by the srcLocation and srcName attributes when the srcType attribute is "file," read data from the specified file in the database server.
- Methods that operate on the URL specified by the srcType, srcLocation, and srcName attributes when the srcType attribute is "http," read data from the resource at the specified URL.

If your application modifies the OrdDoc Java object, or the media data in the database, you must update the ORDDoc SQL object in the database to make those changes permanent.

Some methods in the OrdDoc Java class are handed off to a database source plug-in or database format plug-in for processing; these methods have `byte [ ] [ ] ctx` as a context parameter. Applications should allocate a 64-byte array to hold any context information that may be required by a source plug-in or a format plug-in. For example, a plug-in may initialize the context information in one call and use that information in a subsequent call. The source plug-in context requires one array; the format plug-in context requires another array. For most plug-ins, 64 bytes should be sufficient. Some user-defined plug-ins may need additional space.

The following example illustrates how to allocate a plug-in context information array:

```
byte [] [] ctx = new byte[1][64];
```

**Note:** In the current release, no Oracle-supplied source plug-ins or format plug-ins maintain context. Also, not all user-written source plug-ins or format plug-ins maintain context. However, if you include the context parameter as described, your application should work with any current or future source plug-ins or format plug-ins.

See *Oracle interMedia User's Guide and Reference* for more information about plug-ins.

## 4.1 Prerequisites

You will need to include the following import statements in your Java file in order to run *inter*Media methods:

```
import java.sql.*;
import java.io.*;
import oracle.jdbc.driver.*;
import oracle.sql.*;
import oracle.ord.im.*;
```

The examples in this reference chapter are based on the assumption that the following operations have already been performed:

- A connection has been made to a table that contains a column of type OrdDoc.
- A local OrdDoc object named docObj has been created and populated with data.

For examples of making a connection and populating a local object, see Section 2.2.2.

## 4.2 Reference Information

This section presents reference information on the methods that operate on OrdDoc objects.

## clearLocal( )

### Format

public void clearLocal( )

### Description

Clears the local attribute to indicate that the media data is stored externally.

### Parameters

None.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the local attribute.

### Examples

```
docObj.clearLocal( )
```

## closeSource( )

### Format

public int closeSource(byte[ ] [ ] ctx)

### Description

Closes a data source.

### Parameters

**ctx**
The source plug-in context information.

### Return Value

This method returns the status as an integer, where zero indicates success and a non-zero value indicates a failure code specific to the source plug-in.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding closeSource( ) method in the database.

### Examples

```
byte [ ] [ ] ctx = new byte[1][64];
int i = docObj.closeSource(ctx);
if(i == 0)
     System.out.println("Source close successful");
else
     System.out.println("Source close unsuccessful");
```

where:

- ctx: contains the source plug-in context information.

## deleteContent( )

### Format

public void deleteContent( )

### Description

Deletes any data stored in the database BLOB specified by the localData attribute.

### Parameters

None.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding deleteContent( ) method in the database.

### Examples

```
docObj.deleteContent( );
```

## export( )

### Format

public void export (byte[ ] [ ] ctx, String srcType, String srcLocation, String srcName)

### Description

Exports the data from the BLOB specified by the localData attribute. This method calls the corresponding export( ) method in the database to export the media data to a location specified by the srcType, srcLocation, and srcName parameters.

Not all source plug-ins support the export method. Only the "file" source type is natively supported.

This method will work only if you are running Oracle database server release 8.1.7 or later.

The remainder of this description describes the use of the export( ) method and the Oracle-supplied "file" source plug-in. User-written plug-ins will behave differently.

The export( ) method implemented by the Oracle-supplied "file" source plug-in copies, but does not modify, the media data stored in the database BLOB specified by the localData attribute.

After exporting the media data, all the media property attributes remain unchanged. However, the srcType, srcLocation, and srcName attributes are updated with values of the srcType, srcLocation, and srcName parameters passed to the export( ) method. After calling the export( ) method, if you no longer intend to manage the media data within the database, call the clearLocal( ) method to indicate that the media data is stored outside the database, and call the deleteContent( ) method to delete the media data stored in the database BLOB.

The export( ) method in the database writes only to a database directory object that the user has privileges to access. That is, you can access a directory that you have created using the SQL CREATE DIRECTORY statement, or one to which you have been granted READ access. To execute the CREATE DIRECTORY statement, you must have the CREATE ANY DIRECTORY privilege. In addition, you must use the DBMS_JAVA.GRANT_PERMISSION method to specify which files can be written.

For example, the following SQL*Plus command grants the user, MEDIAUSER, the permission to write to the file named filmclip1.mov:

```
CALL DBMS_JAVA.GRANT_PERMISSION(
    'MEDIAUSER',
     'java.io.FilePermission',
     '/media/movies/filmclip1.mov',
     'write');
```

The previous example shows how to authorize access to write to a single file. In addition, there are various wildcard path specifications that authorize write access to multiple directories and file names. For example, a path specification that ends in a slash and asterisk (/*), where the slash is the file-separator character that is operating-system dependent, indicates all the files contained in the specified directory. A path specification that ends with a slash and hyphen (/-) indicates all files contained in the specified directory and all its subdirectories. A path name consisting of the special token <<ALL FILES>> authorizes access to any file.

See *Oracle9i Java Developer's Guide* and the java.io.FilePermission class in the Java API for more information about security and performance. See *Oracle interMedia User's Guide and Reference* for more information about the required privileges.

## Parameters

**ctx**
The source plug-in context information.

**srcType**
The source type to which the content will be exported.

**srcLocation**
The source location to which the content will be exported.

**srcName**
The source name to which the content will be exported.

## Return Value

None.

## Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding export( ) method in the database.

## Examples

```
byte [ ] [ ] ctx = new byte[1][64];
```

```
docObj.export(ctx,"file","DOCDIR","complete.xml");
```

where:

- ctx: contains the source plug-in context information.
- file: is the source plug-in used to export the content.
- DOCDIR: is the location to which the content will be exported.
- complete.xml: is the file to which the content will be exported.

## getBFILE( )

### Format

public oracle.sql.BFILE getBFILE( )

### Description

Returns a BFILE locator from the database when the srcType attribute is "file". This method calls the corresponding getBFILE( ) method in the database, which creates the BFILE using the srcLocation and srcName attributes.

### Parameters

None.

### Return Value

This method returns an oracle.sql.BFILE locator.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding getBFILE( ) method in the database.

### Examples

```
BFILE documentBFILE = docObj.getBFILE( );
```

## getComments( )

### Format

public oracle.sql.CLOB getComments( )

### Description

Returns the CLOB locator from the comments attribute.

### Parameters

None.

### Return Value

This method returns the value of the comments attribute as an oracle.sql.CLOB locator.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the comments attribute.

### Examples

```
CLOB comments = docObj.getComments( )
```

## getContent( )

### Format

public oracle.sql.BLOB getContent( )

### Description

Returns the BLOB locator from the localData attribute.

### Parameters

None.

### Return Value

This method returns an oracle.sql.BLOB locator.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the localData attribute.

### Examples

```
BLOB localContent = docObj.getContent( );
```

## getContentInLob( )

### Format

public oracle.sql.BLOB getContentInLob(byte[ ] [ ] ctx, String mimetype[ ], String format[ ])

### Description

Returns the data from the BLOB specified by the localData attribute in a temporary BLOB in the database. This method creates a temporary BLOB in the database, reads the data from the BLOB specified by the localData attribute, writes the data to the temporary BLOB, then returns the temporary BLOB locator to the caller.

**Note:** The application must free the temporary BLOB after accessing the data it contains.

### Parameters

**ctx**
The source plug-in context information.

**mimetype**
A String array, 1 element in length, into which the mimeType attribute is written as element 0.

**format**
A String array, 1 element in length, into which the format attribute is written as element 0.

### Return Value

This method returns the media data in a temporary oracle.sql.BLOB locator.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs creating the temporary BLOB or executing the corresponding getContentInLob( ) method in the database.

## Examples

```
byte [ ] [ ] ctx = new byte[1][64];
String mimetype[ ] = new String[1];
String format[ ] = new String[1];
BLOB localContent = docObj.getContentInLob(ctx,mimetype,format);
```

where:

- ctx: contains the source plug-in context information.
- mimetype: is an array of Strings whose first value contains the MIME type of the LOB data. This value is generated by the server.
- format: is an array of Strings whose first value contains the format of the LOB data. This value is generated by the server.

## getContentLength( )

### Format

public int getContentLength( )

### Description

Returns the value of the contentLength attribute.

### Parameters

None.

### Return Value

This method returns the value of the contentLength attribute, as an integer.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the contentLength attribute.

### Examples

```
int contentLength = docObj.getContentLength( );
```

## getDataInByteArray( )

### Format

public byte[ ] getDataInByteArray( )

### Description

Returns a byte array containing the data from the database BLOB specified by the localData attribute.

### Parameters

None.

### Return Value

This method returns a byte array containing the data.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing an object attribute.

java.io.IOException

This exception is thrown if an error occurs reading the data from the BLOB.

java.lang.OutOfMemoryError

This exception is thrown if sufficient memory cannot be allocated to hold the data.

### Examples

```
byte[ ] byteArr = docObj.getDataInByteArray( );
```

## getDataInFile( )

### Format

public boolean getDataInFile(String filename)

### Description

Writes the data from the database BLOB specified by the localData attribute to a local file.

### Parameters

**filename**
The name of the file to which the data will be written.

### Return Value

This method returns true if the data is written to the file successfully; otherwise, an exception is raised if an error occurs. This method never returns false.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing an object attribute.

java.io.IOException

This exception is thrown if an error occurs reading the data from the BLOB or writing the data to the output file.

### Examples

```
boolean load = docObj.getDataInFile("output1.dat");
if(load)
     System.out.println("getDataInFile completed successfully");
else
     System.out.println("Error in getDataInFile");
```

where:

- output1.dat: is the file into which the data will be loaded.

## getDataInStream( )

### Format

public InputStream getDataInStream( )

### Description

Returns an InputStream object from which the data in the database BLOB specified by the localData attribute can be read.

### Parameters

None.

### Return Value

This method returns an InputStream object from which the data will be read.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing an object attribute.

### Examples

```
InputStream inpStream = docObj.getDataInStream( );
```

## getFactory( )

### Format

public static oracle.sql.CustomDatumFactory getFactory( )

### Description

Returns the OrdDoc CustomDatumFactory interface for use by the getCustomDatum( ) method. Specify the getFactory( ) method as the factory parameter of the getCustomDatum( ) method when retrieving an OrdDoc object from an OracleResultSet or OracleCallableStatement object.

### Parameters

None.

### Return Value

This method returns the OrdDoc implementation of the CustomDatumFactory interface.

### Exceptions

None.

### Examples

```
OrdDoc media = (OrdDoc)rset.getCustomDatum( 1, OrdDoc.getFactory() );
```

## getFormat( )

### Format

public String getFormat( )

### Description

Returns the value of the format attribute.

### Parameters

None.

### Return Value

This method returns the value of the format attribute, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the format attribute.

### Examples

```
String format = docObj.getFormat( );
```

## getMimeType( )

### Format

public String getMimeType( )

### Description

Returns the value of the mimeType attribute.

### Parameters

None.

### Return Value

This method returns the value of the mimeType attribute, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the mimeType attribute.

### Examples

```
String mimeType = docObj.getMimeType( );
```

## getSource( )

### Format

public String getSource( )

### Description

Returns the source information in the form: srcType://srcLocation/srcName.

### Parameters

None.

### Return Value

This method returns the source information, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding getSource( ) method in the database.

### Examples

```
String source = docObj.getSource( );
```

## getSourceLocation( )

### Format

public String getSourceLocation( )

### Description

Returns the value of the srcLocation attribute.

### Parameters

None.

### Return Value

This method returns the value of the srcLocation attribute, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the srcLocation attribute.

### Examples

```
String location = docObj.getSourceLocation( );
```

## getSourceName( )

### Format

public String getSourceName( )

### Description

Returns the value of the srcName attribute.

### Parameters

None.

### Return Value

This method returns the value of the srcName attribute, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the srcName attribute.

### Examples

```
String name = docObj.getSourceName( );
```

## getSourceType( )

### Format

public String getSourceType( )

### Description

Returns the value of the srcType attribute.

### Parameters

None.

### Return Value

This method returns the value of the srcType attribute, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the srcType attribute.

### Examples

```
String type = docObj.getSourceType( );
```

## getUpdateTime( )

### Format

public java.sql.Timestamp getUpdateTime( )

### Description

Returns the value of the updateTime attribute.

### Parameters

None.

### Return Value

This method returns the value of the updateTime attribute as a java.sql.Timestamp object.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the updateTime attribute.

### Examples

```
Timestamp time = docObj.getUpdateTime( );
```

## importData( )

### Format

public void importData(byte[ ] [ ] ctx, boolean setProp)

### Description

Imports data from an external source into the database BLOB specified by the localData attribute, and optionally calls the setProperties( ) method. If the setProp parameter is true, this method calls the setProperties( ) method in the database to set the property attributes and populate the CLOB specified by the comments attribute. The external data source is specified by the srcType, srcLocation, and srcName attributes.

### Parameters

**ctx**
The source plug-in context information.

**setProp**
A Boolean value that specifies whether or not to call the setProperties( ) method.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding import( ) method in the database.

### Examples

```
byte [ ] [ ] ctx = new byte[1][64];
docObj.importData(ctx);
```

where:

- ctx: contains the source plug-in information.

## importFrom( )

### Format

public void importFrom(byte[ ] [ ] ctx, String srcType, String srcLocation, String srcName, boolean setProp)

### Description

Imports data from an external source into the database BLOB specified by the localData attribute, and optionally calls the setProperties( ) method. If the setProp parameter is true, this method calls the setProperties( ) method in the database to set the property attributes and populate the CLOB specified by the comments attribute. The external data source is specified by the srcType, srcLocation, and srcName parameters. The srcType, srcLocation, and srcName attributes are updated with values of the srcType, srcLocation, and srcName parameters passed to the importFrom( ) method.

### Parameters

**ctx**
The source plug-in context information. See *Oracle interMedia User's Guide and Reference* for more information.

**srcType**
The source type from which the data will be imported.

**srcLocation**
The source location from which the data will be imported.

**srcName**
The source name from which the data will be imported.

**setProp**
A Boolean value that specifies whether or not to call the setProperties( ) method.

### Return Value

None.

## Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding importFrom( ) method in the database.

## Examples

```
byte [ ] [ ] ctx = new byte[1][64];
docObj.importFrom("file","DOCDIR","testdoc.dat");
```

where:

- ctx: contains the source plug-in context information.
- file: is the source plug-in used to import the data.
- DOCDIR: is the location of the file on the database server from which the data will be imported.
- testdoc.dat: is the file from which the data will be imported.

## isLocal( )

### Format

public boolean isLocal( )

### Description

Indicates if the media data is stored locally in the database in a BLOB specified by the localData attribute.

### Parameters

None.

### Return Value

This method returns true if the data is stored locally in the database in a BLOB; false otherwise.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the localData attribute.

### Examples

```
if(docObj.isLocal( ))
     System.out.println("local attribute is set to true");
else
     System.out.println("local attribute is set to false");
```

## loadDataFromByteArray( )

### Format

public boolean loadDataFromByteArray(byte[ ] byteArr)

### Description

Loads data from a byte array into the database BLOB specified by the localData attribute. Before loading the data, this method calls the deleteContent( ) method to delete any existing data in the BLOB. It also calls the setLocal( ) method to set the local flag. In addition, this method calls the setUpdateTime( ) method to set the updateTime attribute to the database server's current SYSDATE time.

### Parameters

**byteArr**
The name of the local byte array from which the data will be loaded.

### Return Value

This method returns true if the data is loaded successfully; otherwise, an exception is raised if an error occurs. This method never returns false.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing an object attribute or executing a method in the database.

java.io.IOException

This exception is thrown if an error occurs reading the byte array.

### Examples

```
byte[ ] data = new byte[32000];
FileInputStream fStream = new FileInputStream("testdoc.dat");
fStream.read(data,0,32000);
boolean success = docObj.loadDataFromByteArray(data);
if(success)
     System.out.println("loadDataFromByteArray was successful");
else
```

```
     System.out.println("loadDataFromByteArray was unsuccessful");
```

where:

- data: is the local byte array from which the data will be loaded.
- testdoc.dat: is a local file that contains 32,000 bytes of data.

## loadDataFromFile( )

### Format

public boolean loadDataFromFile(String filename)

### Description

Loads data from a file into the database BLOB specified by the localData attribute. Before loading the data, this method calls the deleteContent( ) method to delete any existing data in the BLOB. It also calls the setLocal( ) method to set the local flag. In addition, this method calls the setUpdateTime( ) method to set the updateTime attribute to the database server's current SYSDATE time.

### Parameters

**filename**
The name of the local file from which the data will be loaded.

### Return Value

This method returns true if the data is loaded successfully; otherwise, an exception is raised if an error occurs. This method never returns false.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing an object attribute or executing a method in the database.

java.io.IOException

This exception is thrown if an error occurs reading the data file.

### Examples

```
docObj.loadDataFromFile("testdoc.dat");
```

where:

- testdoc.dat: is a local file that contains media data.

## loadDataFromInputStream( )

### Format

public boolean loadDataFromInputStream(InputStream inpStream)

### Description

Loads data from an InputStream object into the database BLOB specified by the localData attribute. Before loading the data, this method calls the deleteContent( ) method to delete any existing data in the BLOB. It also calls the setLocal( ) method to set the local flag. In addition, this method calls the setUpdateTime( ) method to set the updateTime attribute to the database server's current SYSDATE time.

### Parameters

**inpStream**
The name of the InputStream object from which the data will be loaded.

### Return Value

This method returns true if the data is loaded successfully; otherwise, an exception is raised if an error occurs. This method never returns false.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing an object attribute or executing a method in the database.

java.io.IOException

This exception is thrown if an error occurs reading the InputStream object.

### Examples

```
FileInputStream fStream = new FileInputStream("testdoc.dat");
docObj.loadDataFromInputStream(fStream);
```

where:

- testdoc.dat: is a local file that contains media data.

- fStream: is the local input stream that will load data into the OrdDoc object.

## openSource( )

### Format

public int openSource(byte[ ] userarg, byte[ ] [ ] ctx)

### Description

Opens a data source.

### Parameters

**userarg**
Additional source plug-in information that may be required by user-defined source plug-ins.

**ctx**
The source plug-in context information. See *Oracle interMedia User's Guide and Reference* for more information.

### Return Value

This method returns the status as an integer, where zero indicates success and a non-zero value indicates a failure code specific to the source plug-in.

### Exceptions

java.lang.Exception

This exception is thrown if an error occurs executing the corresponding openSource( ) method in the database.

### Examples

```
byte[ ] userarg = new byte[64];
byte [ ] [ ] ctx = new byte[1][64];
int i = docObj.openSource(userarg,ctx);
if(i == 0)
     System.out.println("openSource successful");
else
     System.out.println("openSource unsuccessful");
```

where:

- userarg: contains permission-related parameters.
- ctx: contains the source plug-in context information.

## processSourceCommand( )

### Format

public byte[ ] processSourceCommand(byte[ ] [ ] ctx, String cmd, String args, byte[ ] [ ] result)

### Description

Calls the source plug-in in the database to execute a command. This method is used with user-written plug-ins only; it raises an exception if used with the source plug-ins supplied by Oracle.

### Parameters

**ctx**
The source plug-in context information. See *Oracle interMedia User's Guide and Reference* for more information.

**cmd**
The command to be executed by the source plug-in.

**args**
The arguments of the command.

**result**
A byte array of the form [1][*n*] into which the result of the command execution is written.

### Return Value

This method returns the results of executing the command.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding processSourceCommand( ) method in the database.

### Examples

```
byte [ ] [ ] ctx = new byte[1][64];
```

```
String cmd;
String args;
byte [ ] [ ] result;
//assign a command value to cmd
//assign any arguments to args
byte[ ] commandResults = docObj.processSourceCommand(ctx,cmd,
     args,result);
```

where:

- ctx: contains the source plug-in information.
- cmd: is the command to be run.
- args: contains any arguments required by the command.
- result: is the results of executing the command.

## readFromSource( )

### Format

public int readFromSource(byte[ ] [ ] ctx, int startpos, int numbytes, byte[ ] [ ] buffer)

### Description

Reads data from the data source. This method reads the specified number of bytes into the application buffer from the data source starting at the specified position in the data source.

Not all source plug-ins require that the data source be opened before it can be read. However, to ensure that an application will work with any current or future source plug-ins, call the openSource( ) method before calling this method.

### Parameters

**ctx**
The source plug-in context information. See *Oracle interMedia User's Guide and Reference* for more information.

**startpos**
The start position in the data source.

**numbytes**
The number of bytes to be read from the data source.

**buffer**
A byte array of the form [1][*n*], where *n* is greater than or equal to numbytes.

### Return Value

This method returns the number of bytes read, as an integer.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding readFromSource( ) method in the database.

## Examples

```
byte [ ] [ ] ctx = new byte[1][64];
byte [ ] [ ] buffer = new byte[12][1];
int i = docObj.readFromSource(ctx,0,12,buffer);
```

where:

- ctx: contains the source plug-in context information.
- 0: is the position to begin reading from the localData field.
- 12: is the number of bytes to be read.
- buffer: is the buffer to which the data will be read. The data will be stored in buffer[0].

## setComments( )

### Format

public void setComments(oracle.sql.CLOB comments)

### Description

Sets the value of the comments attribute.

The comments attribute is reserved for use by *inter*Media. You can set your own value, but it could be overwritten by Oracle *inter*Media Annotator or by the setProperties( ) method.

### Parameters

**comments**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the comments attribute.

### Examples

```
docObj.setComments(commentsData);
```

where:

- commentsData: is a CLOB that contains data to be set in the comments attribute.

## setContentLength( )

### Format

public void setContentLength(int contentLength)

### Description

Sets the value of the contentLength attribute.

The setProperties( ) method sets this attribute automatically for certain media formats; use this method only if you are not using the setProperties( ) method. This method sets only the attribute value; it does not modify the media data itself.

### Parameters

**contentLength**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the contentLength attribute.

### Examples

None.

## setFormat( )

### Format

public void setFormat(String format)

### Description

Sets the value of the format attribute.

The format attribute determines which format plug-in is used to handle calls to methods that operate on the media data. In particular, the setProperties( ) method uses the format attribute to determine which format plug-in to call to parse the media data properties. See the setProperties( ) method for more information on how to initialize the format attribute before calling the setProperties( ) method, and for information on how the setProperties( ) method in the default, Oracle-supplied plug-in, sets the value of the format attribute. Calling this method sets only the attribute value; it does not modify the media data itself.

### Parameters

**format**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the format attribute.

### Examples

```
docObj.setFormat("AUFF");
```

where:

- AUFF: is the value to be set in the format attribute.

## setLocal( )

### Format

public void setLocal( )

### Description

Sets the value of the local attribute to indicate that the media data is stored locally in the database in a BLOB specified by the localData attribute.

### Parameters

None.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the localData attribute.

### Examples

```
docObj.setLocal( );
```

## setMimeType( )

### Format

public void setMimeType(String mimeType)

### Description

Sets the value of the mimeType attribute.

The setProperties( ) method sets this attribute automatically for certain media formats; use this method only if you are not using the setProperties( ) method. This method sets only the attribute value; it does not modify the media data itself.

### Parameters

**mimeType**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the mimeType attribute.

### Examples

```
docObj.setMimeType("application/pdf");
```

where:

- application/pdf: is the MIME type to be set.

## setProperties( )

### Format

public void setProperties(byte[ ] [ ] ctx, boolean setComments)

### Description

Parses the media data properties, sets the values of the attributes in the OrdDoc Java object, and optionally populates the CLOB specified by the comments attribute. This method sets the values of the format, mimeType, and contentLength attributes. An attribute is set to null if the corresponding property cannot be extracted for a specific media format. If the setComments parameter is true, this method also populates the CLOB specified by the comments attribute with all extracted properties in XML form. If the setComments parameter is false, the comments attribute is not modified. This method throws a SQLException error if the media format is not recognized.

The format attribute determines which format plug-in is used to parse the media data properties. If the format attribute is null when the setProperties( ) method is called, then the default, Oracle-supplied, format plug-in is used to parse the media data properties and fill in various attributes, including the actual media data format, for supported media formats. See *Oracle interMedia User's Guide and Reference* for information on the media formats supported by the Oracle-supplied format plug-ins. Note that the ORDDoc.init methods in the database always set the value of the format attribute to null. If the format attribute is not null, then the format plug-in specified by the format attribute will be called when the setProperties( ) method is called.

### Parameters

**ctx**
The format plug-in context information.

**setComments**
A Boolean value that specifies whether or not to populate the CLOB specified by the comments attribute.

### Return Value

None.

## Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding setProperties( ) method in the database.

## Examples

```
byte [ ] [ ] ctx = new byte[1][64];
docObj.setProperties(ctx,true);
```

where:

- ctx: contains the format plug-in context information.
- true: indicates that the comments field will be populated.

## setSource( )

### Format

public void setSource(String srcType, String srcLocation, String srcName)

### Description

Sets the values of the srcType, srcLocation, and srcName attributes.

### Parameters

**srcType**
The type of the source.

**srcLocation**
The location of the source.

**srcName**
The name of the source.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the srcType, srcLocation, or srcName attributes.

### Examples

```
docObj.setSource("file","DOCDIR","sample.pdf");
```

where:

- file: is the source type.
- DOCDIR: is the source location.
- sample.pdf: is the source name.

## setUpdateTime( )

### Format

public void setUpdateTime(java.sql.Timestamp currentTime)

### Description

Sets the value of the updateTime attribute. This method sets the value of the updateTime attribute to the specified time, or to the database server's current SYSDATE time if the currentTime attribute is specified as null.

### Parameters

**currentTime**
The update time, or the null value, used to set the value of the updateTime attribute to the database server's current SYSDATE time.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding setUpdateTime( ) method in the database.

### Examples

```
docObj.setUpdateTime(null);
```

## trimSource( )

### Format

public int trimSource(byte[ ] [ ] ctx, int newLen)

### Description

Trims the data to the specified length.

Not all source plug-ins support trim operations. For example, applications can trim the data stored in a BLOB specified by the localData attribute; however, the "file" and "http" data source types do not support write access, and so do not support this method. Furthermore, those source plug-ins that do support write access may not support the trim operation.

Not all source plug-ins require that the data source be opened before it can be modified. However, to ensure that an application will work with any current or future source plug-ins, call the openSource( ) method before calling this method.

### Parameters

**ctx**
The source plug-in context information. See *Oracle interMedia User's Guide and Reference* for more information.

**newLen**
The length to which the data is to be trimmed.

### Return Value

This method returns the status as an integer, where zero indicates success and a non-zero value indicates a failure code specific to the source plug-in.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding trimSource( ) method in the database.

## Examples

```
byte [ ] [ ] ctx = new byte[1][64];
int i = docObj.trimSource(ctx,10);
if (i == 0)
     System.out.println("trimSource successful");
else
     System.out.println("trimSource unsuccessful");
```

where:

- ctx: contains the source plug-in context information.
- 10: is the new length of the source.

## writeToSource( )

### Format

public int writeToSource(byte[ ] [ ] ctx, int startpos, int numbytes, byte[ ] buffer)

### Description

Writes data to the data source. This method writes the specified number of bytes from the application buffer to the data source, starting at the specified position in the data source.

Not all source plug-ins support write operations. For example, applications can write to a BLOB specified by the localData attribute; however, the "file" and "http" data source types do not support write access, and so do not support this method. Furthermore, those source plug-ins that do support write access may support only sequential write access, and may not support write access to arbitrary starting positions within the data source.

Not all source plug-ins require that the data source be opened before it can be written. However, to ensure that an application will work with any current or future source plug-ins, call the openSource( ) method before calling this method.

### Parameters

**ctx**
The source plug-in context information. See *Oracle interMedia User's Guide and Reference* for more information.

**startpos**
The start position in the data source.

**numbytes**
The number of bytes to be written to the data source.

**buffer**
A byte array containing the data to be written.

### Return Value

This method returns the number of bytes written, as an integer.

## Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding writeToSource( ) method in the database.

## Examples

```
byte [ ] [ ] ctx = new byte[1][64];
byte[ ] data = new byte[20];
//populate data with 20 bytes of content
int i = docObj.writeToSource(ctx,1,20,data);
```

where:

- ctx: contains the source plug-in context information.
- 1: is the position in the localData field where writing will begin.
- 20: is the number of bytes to be written.
- data: contains the content to be written.

# 5 OrdImage Reference Information

The OrdImage class is used to represent an instance of the ORDSYS.ORDImage database type in a Java application. The OrdImage class includes a set of methods to get and set various object attributes, as well as a set of methods that perform various operations on an OrdImage Java object.

Almost all methods operate on the attributes of the OrdImage Java object in the application. The exceptions are those methods that access the image data for read or write purposes, which are described in the following list:

- Methods that operate on the database BLOB specified by the localData attribute, read and write data stored in the database BLOB.
- Methods that operate on the database BFILE specified by the srcLocation and srcName attributes when the srcType attribute is "file," read data from the specified file in the database server.
- Methods that operate on the URL specified by the srcType, srcLocation, and srcName attributes when the srcType attribute is "http," read data from the resource at the specified URL.

If your application modifies the OrdImage Java object, or the image data in the database, you must update the ORDImage SQL object in the database to make those changes permanent.

Some methods invoked at the OrdImage level are handed off to the database source plug-in for processing; these methods have `byte [ ] [ ] ctx` as a context parameter. In cases where a client system is connecting to a database server, the space for the parameter is created by the client (in the reference examples, 64 bytes of space), but the content of the context parameter is generated by the server. The context parameter is passed from the client to the server for the processing of context information.

Some methods in the OrdImage Java class are handed off to a database source plug-in or database format plug-in for processing; these methods have `byte [ ] [ ] ctx` as a context parameter. Applications should allocate a 64-byte array to hold any context information that may be required by a source plug-in or a format plug-in. For example, a plug-in may initialize the context information in one call and use that information in a subsequent call. The source plug-in context requires one array; the format plug-in context requires another array. For most plug-ins, 64 bytes should be sufficient. Some user-defined plug-ins may need additional space. The following example shows how to allocate a plug-in context information array:

```
byte [] [] ctx = new byte[1][64];
```

**Note:** In the current release, no Oracle-supplied source plug-ins or format plug-ins maintain context. Also, not all user-written source plug-ins or format plug-ins maintain context. However, if you include the context parameter as described, your application should work with any current or future source plug-ins or format plug-ins.

See *Oracle interMedia User's Guide and Reference* for more information about plug-ins.

## 5.1 Prerequisites

You will need to include the following import statements in your Java file in order to run *inter*Media methods:

```
import java.sql.*;
import java.io.*;
import oracle.jdbc.driver.*;
import oracle.sql.*;
import oracle.ord.im.*;
```

The examples in this reference chapter are based on the assumption that the following operations have already been performed:

- A connection has been made to a table that contains a column of type OrdImage.
- A local OrdImage object named imgObj has been created and populated with data.

For examples of making a connection and populating a local object, see Section 2.3.2.

## 5.2 Reference Information

This section presents reference information on the methods that operate on OrdImage objects.

## checkProperties( )

### Format

public boolean checkProperties( )

### Description

Checks if the properties of the image data are consistent with the attributes of the OrdImage Java object.

### Parameters

None.

### Return Value

This method returns true if the properties of the image data are consistent with the attributes of the OrdImage Java object; false otherwise.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding checkProperties( ) method in the database.

### Examples

```
if(imgObj.checkProperties( ))
     System.out.println("checkProperties successful");
```

## clearLocal( )

### Format

public void clearLocal( )

### Description

Clears the local attribute to indicate that the image data is stored externally.

### Parameters

None.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the local attribute.

### Examples

```
imgObj.clearLocal( );
```

## copy( )

### Format

public void copy(OrdImage dest)

### Description

Copies an OrdImage Java object. This method calls the corresponding copy( ) method in the database. The copy( ) method copies all the attributes of the current OrdImage object to the destination OrdImage object with the exception of the BLOB specified by the localData attribute. If the image data is stored locally in the database, then the data is copied from the BLOB specified by the localData attribute in the current OrdImage object to the BLOB specified by the localData attribute in the destination object.

### Parameters

**dest**
The destination OrdImage object to which the data will be copied.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs calling the corresponding copy( ) method in the database.

### Examples

```
//create and populate an object named imgObj2
imgObj.copy(imgObj2);
```

where

- imgObj2: is an OrdImage object that will receive the image data from imgObj.

## deleteContent( )

### Format

public void deleteContent( )

### Description

Deletes any data stored in the database BLOB specified by the localData attribute.

### Parameters

None.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding deleteContent( ) method in the database.

### Examples

```
imgObj.deleteContent( );
```

## export( )

### Format

public void export (byte[ ] [ ], ctx String srcType, String srcLocation, String srcName)

### Description

Exports the data from the BLOB specified by the localData attribute. This method calls the corresponding export( ) method in the database to export the image data to a location specified by the srcType, srcLocation, and srcName parameters.

Not all source plug-ins support the export( ) method. For example, the "file" source type is the only Oracle-supplied source type that supports the export( ) method.

This method will work only if you are running Oracle database server release 8.1.7 or later.

The remainder of this description describes the use of the export( ) method and the Oracle-supplied "file" source plug-in. User-written plug-ins will behave differently.

The export( ) method implemented by the Oracle-supplied "file" source plug-in copies, but does not modify, the media data stored in the database BLOB specified by the localData attribute.

After exporting the image data, all the image property attributes remain unchanged, however, the srcType, srcLocation, and srcName attributes are updated with values of the srcType, srcLocation, and srcName parameters passed to the export( ) method. After calling the export( ) method, if you no longer intend to manage the image data within the database, call the clearLocal( ) method to indicate the image data is stored outside the database and call the deleteContent( ) method to delete the image data stored in the database BLOB.

The export( ) method in the database writes only to a database directory object that the user has privileges to access. That is, you can access a directory that you have created using the SQL CREATE DIRECTORY statement, or one to which you have been granted READ access. To execute the CREATE DIRECTORY statement, you must have the CREATE ANY DIRECTORY privilege. In addition, you must use the DBMS_JAVA.GRANT_PERMISSION method to specify which files can be written.

For example, the following SQL*Plus command grants the user, MEDIAUSER, the permission to write to the file named scenery1.jpg:

```
CALL DBMS_JAVA.GRANT_PERMISSION(
    'MEDIAUSER',
'java.io.FilePermission',
'/images/outdoors/scenery1.jpg',
'write');
```

The previous example shows how to authorize access to write to a single file. In addition, there are various wildcard path specifications that authorize write access to multiple directories and file names. For example, a path specification that ends in a slash and asterisk (/*), where the slash is the file-separator character that is operating-system dependent, indicates all the files contained in the specified directory. A path specification that ends with a slash and hyphen (/-) indicates all files contained in the specified directory and all its subdirectories. A path name consisting of the special token <<ALL FILES>> authorizes access to any file.

See *Oracle9i Java Developer's Guide* and the java.io.FilePermission class in the Java API for more information about security and performance. See *Oracle interMedia User's Guide and Reference* for more information about the required privileges.

## Parameters

**ctx**
The source plug-in context information.

**srcType**
The source type to which the content will be exported.

**srcLocation**
The source location to which the content will be exported.

**srcName**
The source name to which the content will be exported.

## Return Value

None.

## Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding export( ) method in the database.

## Examples

```
byte [ ] [ ] ctx = new byte[1][64];
```

```
imgObj.export(ctx,"file","IMAGEDIR","image.gif");
```

where:

- ctx: contains the source plug-in context information.
- file: is the source plug-in used to export the content.
- IMAGEDIR: is the location on the database server to which the content will be exported.
- image.gif: is the file to which the content will be exported.

## getBFILE( )

### Format

public oracle.sql.BFILE getBFILE( )

### Description

Returns a BFILE locator from the database when the srcType attribute is "file." This method calls the corresponding getBFILE( ) method in the database, which creates the BFILE using the srcLocation and srcName attributes.

### Parameters

None.

### Return Value

This method returns an oracle.sql.BFILE locator.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding getBFILE( ) method in the database.

### Examples

```
BFILE imageBFILE = imgObj.getBFILE( );
```

## getCompressionFormat( )

### Format

public String getCompressionFormat( )

### Description

Returns the value of the compressionFormat attribute.

### Parameters

None.

### Return Value

This method returns the value of the compressionFormat attribute, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the compressionFormat attribute.

### Examples

```
String compression = imgObj.getCompressionFormat( );
```

## getContent( )

### Format

public oracle.sql.BLOB getContent( )

### Description

Returns the BLOB locator from the localData attribute.

### Parameters

None.

### Return Value

This method returns an oracle.sql.BLOB locator.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the localData attribute.

### Examples

```
BLOB localContent = imgObj.getContent( );
```

## getContentFormat( )

### Format

public String getContentFormat( )

### Description

Returns the value of the contentFormat attribute.

### Parameters

None.

### Return Value

This method returns the value of the contentFormat attribute, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the contentFormat attribute.

### Examples

```
String format = imgObj.getContentFormat( );
```

## getContentLength( )

### Format

public int getContentLength( )

### Description

Returns the value of the contentLength attribute.

### Parameters

None.

### Return Value

This method returns the value of the contentLength attribute, as an integer.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the contentLength attribute.

### Examples

```
int length = imgObj.getContentLength( );
```

## getDataInByteArray( )

### Format

public byte[ ] getDataInByteArray( )

### Description

Returns a byte array containing the data from the database BLOB specified by the localData attribute.

### Parameters

None.

### Return Value

This method returns the byte array containing the data.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing an object attribute.

java.io.IOException

This exception is thrown if an error occurs reading the data from the BLOB.

java.lang.OutOfMemoryError

This exception is thrown if sufficient memory cannot be allocated to hold the data.

### Examples

See Section 2.3.2.8 for an example of this method.

## getDataInFile( )

### Format

public boolean getDataInFile(String filename)

### Description

Writes the data from the database BLOB specified by the localData attribute to a local file.

### Parameters

**filename**
The name of the file to which the data will be written.

### Return Value

This method returns true if the data is written to the file successfully; otherwise, an exception is raised if an error occurs. This method never returns false.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing an object attribute.

java.io.IOException

This exception is thrown if an error occurs reading the data from the BLOB or writing the data to the output file.

### Examples

See Section 2.3.2.6 for an example of this method.

## getDataInStream( )

### Format

public InputStream getDataInStream( )

### Description

Returns an InputStream object from which the data in the database BLOB specified by the localData attribute can be read.

### Parameters

None.

### Return Value

This method returns an InputStream object from which the data will be read.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing an object attribute.

### Examples

See Section 2.3.2.7 for an example of this method.

## getFactory( )

### Format

public static oracle.sql.CustomDatumFactory getFactory( )

### Description

Returns the OrdImage CustomDatumFactory interface for use by the getCustomDatum( ) method. Specify the getFactory( ) method as the factory parameter of the getCustomDatum( ) method when retrieving an OrdImage object from an OracleResultSet or OracleCallableStatement object.

### Parameters

None.

### Return Value

This method returns the OrdImage implementation of the CustomDatumFactory interface.

### Exceptions

None.

### Examples

```
OrdImage img = (OrdImage)rset.getCustomDatum( 1, OrdImage.getFactory() );
```

## getFormat( )

### Format

public String getFormat( )

### Description

Returns the value of the fileFormat attribute.

### Parameters

None.

### Return Value

This method returns the value of the fileFormat attribute, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the fileFormat attribute.

### Examples

```
String format = imgObj.getFormat( );
```

## getHeight( )

### Format

public int getHeight( )

### Description

Returns the value of the height attribute.

### Parameters

None.

### Return Value

This method returns the value of the height attribute, as an integer.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the height attribute.

### Examples

```
int height = imgObj.getHeight( );
```

## getMimeType( )

### Format

public String getMimeType( )

### Description

Returns the value of the mimeType attribute.

### Parameters

None.

### Return Value

This method returns the value of the mimeType attribute, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the mimeType attribute.

### Examples

```
String mime = imgObj.getMimeType( );
```

## getSource( )

### Format

public String getSource( )

### Description

Returns the source information in the form: srcType://srcLocation/srcName.

### Parameters

None.

### Return Value

This method returns the source information, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding getSource( ) method in the database.

### Examples

```
String sourceName = imgObj.getSource( );
```

## getSourceLocation( )

### Format

public String getSourceLocation( )

### Description

Returns the value of the srcLocation attribute.

### Parameters

None.

### Return Value

This method returns the value of the srcLocation attribute, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the srcLocation attribute.

### Examples

```
String location = imgObj.getSourceLocation( );
```

## getSourceName( )

### Format

public String getSourceName( )

### Description

Returns the value of the srcName attribute.

### Parameters

None.

### Return Value

This method returns the value of the srcName attribute, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the srcName attribute.

### Examples

```
String name = imgObj.getSourceName( );
```

## getSourceType( )

### Format

public String getSourceType( )

### Description

Returns the value of the srcType attribute.

### Parameters

None.

### Return Value

This method returns the value of the srcType attribute, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the srcType attribute.

### Examples

```
String type = imgObj.getSourceType( );
```

## getUpdateTime( )

### Format

public java.sql.Timestamp getUpdateTime( )

### Description

Returns the value of the updateTime attribute.

### Parameters

None.

### Return Value

This method returns the value of the updateTime attribute as a java.sql.Timestamp object.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the updateTime attribute.

### Examples

```
Timestamp time = imgObj.getUpdateTime( );
```

## getWidth( )

### Format

public int getWidth( )

### Description

Returns the value of the width attribute.

### Parameters

None.

### Return Value

This method returns the value of the width attribute, as an integer.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the width attribute.

### Examples

```
int width = imgObj.getWidth( );
```

## importData( )

### Format

public void importData(byte[ ] [ ] ctx)

### Description

Imports data from an external source into the database BLOB specified by the localData attribute. The external data source is specified by the srcType, srcLocation, and srcName attributes. After importing the image data, by default, this method automatically calls the setProperties( ) method in the database to set the property attributes. If you are importing a foreign image whose format is not understood by *inter*Media, call the setFormat( ) method to set the fileFormat to a String beginning with "other" to disable the automatic call to the setProperties( ) method.

### Parameters

**ctx**
The source plug-in context information.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding import( ) method or the setProperties( ) method in the database.

### Examples

```
byte [ ] [ ] ctx = new byte[1][64];
imgObj.importData(ctx)
```

where:

- ctx: contains the source plug-in context information.

## importFrom( )

### Format

public void importFrom(byte[ ] [ ] ctx, String srcType, String srcLocation, String srcName)

### Description

Imports data from an external source into the database BLOB specified by the localData attribute. The external data source is specified by the srcType, srcLocation, and srcName parameters. The srcType, srcLocation, and srcName attributes are updated with values of the srcType, srcLocation, and srcName parameters passed to the importFrom( ) method. After importing the image data, by default, this method automatically calls the setProperties( ) method in the database to set the property attributes. If you are importing a foreign image whose format is not understood by *inter*Media, call the setFormat( ) method to set the fileFormat to "other" to disable the automatic call to the setProperties( ) method.

### Parameters

**ctx**
The source plug-in context information. See *Oracle interMedia User’s Guide and Reference* for more information.

**srcType**
The source type from which the data will be imported.

**srcLocation**
The source location from which the data will be imported.

**srcName**
The source name from which the data will be imported.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding importFrom( ) method or the setProperties( ) method in the database.

## Examples

```
byte [ ] [ ] ctx = new byte[1][64];
imgObj.importFrom(ctx,"file","IMAGEDIR","testimg.dat");
```

where:

- ctx: contains the source plug-in context information.
- file: is the source plug-in used to import the data.
- IMAGEDIR: is the location of the file from which the data will be imported.
- testimg.dat: is the file from which the data will be imported.

## isLocal( )

### Format

public boolean isLocal( )

### Description

Indicates if the image data is stored locally in the database in a BLOB specified by the localData attribute.

### Parameters

None.

### Return Value

This method returns true if the data is stored locally in the database in a BLOB; false otherwise.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the localData attribute.

### Examples

```
if(imgObj.isLocal( ))
     System.out.println("local attribute is true");
else
     System.out.println("local attribute is false");
```

## loadDataFromByteArray( )

### Format

public boolean loadDataFromByteArray(byte[ ] byteArr)

### Description

Loads data from a byte array into the database BLOB specified by the localData attribute. Before loading the data, this method calls the deleteContent( ) method to delete any existing data in the BLOB. It also calls the setLocal( ) method to set the local flag. In addition, this method calls the setUpdateTime( ) method to set the updateTime attribute to the database server's current SYSDATE time.

### Parameters

**byteArr**
The name of the byte array from which the data will be loaded.

### Return Value

This method returns true if the data is loaded successfully; otherwise, an exception is raised if an error occurs. This method never returns false.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing an object attribute or executing a method in the database.

java.io.IOException

This exception is thrown if an error occurs reading the byte array.

### Examples

See Section 2.3.2.8 for an example of this method.

## loadDataFromFile( )

### Format

public boolean loadDataFromFile(String filename)

### Description

Loads data from a file into the database BLOB specified by the localData attribute. Before loading the data, this method calls the deleteContent( ) method to delete any existing data in the BLOB. It also calls the setLocal( ) method to set the local flag. In addition, this method calls the setUpdateTime( ) method to set the updateTime attribute to the database server's current SYSDATE time.

### Parameters

**filename**
The name of the file from which the data will be loaded.

### Return Value

This method returns true if the data is loaded successfully; otherwise, an exception is raised if an error occurs. This method never returns false.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing an object attribute or executing a method in the database.

java.io.IOException

This exception is thrown if an error occurs reading the data file.

### Examples

See Section 2.3.2.6 for an example of this method.

## loadDataFromInputStream( )

### Format

public boolean loadDataFromInputStream(InputStream inpStream)

### Description

Loads data from an InputStream object into the database BLOB specified by the localData attribute. Before loading the data, this method calls the deleteContent( ) method to delete any existing data in the BLOB. It also calls the setLocal( ) method to set the local flag. In addition, this method calls the setUpdateTime( ) method to set the updateTime attribute to the database server's current SYSDATE time.

### Parameters

**inpStream**
The name the InputStream object from which the data will be loaded.

### Return Value

This method returns true if the data is loaded successfully; otherwise, an exception is raised if an error occurs. This method never returns false.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing an object attribute or executing a method in the database.

java.io.IOException

This exception is thrown if an error occurs reading the InputStream object.

### Examples

See Section 2.3.2.7 for an example of this method.

## process( )

### Format

public void process(String cmd)

### Description

Performs one or more image processing operations on the image data in the database BLOB specified by the localData attribute. This method calls the corresponding process( ) method in the database to perform the image processing operations specified by the cmd parameter.

See *Oracle interMedia User's Guide and Reference* for more information on the various image processing operations that can be performed on an image.

### Parameters

**cmd**
A String that specifies a list of image processing operations to perform on the image.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding process( ) method in the database.

### Examples

```
imgObj.process("fileFormat=JFIF");
```

where:

- fileFormat=JFIF: is the command to be executed.

## processCopy( )

### Format

public void processCopy(String cmd, OrdImage dest)

### Description

Copies the image data to the destination object and performs one or more image processing operations on the image data. If the source image data is stored externally to the database, then it is imported into the database BLOB specified by the localData attribute in the destination OrdImage object. Otherwise, the image data is copied from the BLOB specified by localData attribute in the current OrdImage object to the BLOB specified by the localData attribute in the destination object.

### Parameters

**cmd**
A String that specifies a list of image processing operations to perform on the image.

**dest**
The destination OrdImage object.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs calling the corresponding processCopy( ) method in the database.

### Examples

```
//create and populate an OrdImage object named imgObj2
imgObj.processCopy("maxScale=32 32, fileFormat= GIFF", imgObj2);
```

where:

- maxScale=32 32, fileFormat= GIFF: is the command to be executed.
- imgObj2: is the object that will receive the results of executing the command.

## setCompressionFormat( )

### Format

```
public void setCompressionFormat(String CompressionFormat)
```

### Description

Sets the value of the compressionFormat attribute.

The setProperties( ) method sets this attribute automatically for certain media formats; use this method only if you are not using the setProperties( ) method. This method sets only the attribute value; it does not modify the media data itself.

### Parameters

**CompressionFormat**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the compressionFormat attribute.

### Examples

None.

## setContentFormat( )

### Format

public void setContentFormat(String ContentFormat)

### Description

Sets the value of the contentFormat attribute.

The setProperties( ) method sets this attribute automatically for certain media formats; use this method only if you are not using the setProperties( ) method. This method sets only the attribute value; it does not modify the media data itself.

### Parameters

**ContentFormat**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the contentFormat attribute.

### Examples

None.

## setContentLength( )

### Format

public void setContentLength(int ContentLength)

### Description

Sets the value of the contentLength attribute.

The setProperties( ) method sets this attribute automatically for certain media formats; use this method only if you are not using the setProperties( ) method. This method sets only the attribute value; it does not modify the media data itself.

### Parameters

**ContentLength**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the contentLength attribute.

### Examples

None.

## setFormat( )

### Format

public void setFormat(String Format)

### Description

Sets the value of the fileFormat attribute.

The setProperties( ) method sets this attribute automatically for certain media formats; use this method only if you are not using the setProperties( ) method. This method sets only the attribute value; it does not modify the media data itself. Set the fileFormat attribute to a String beginning with "other" to disable the automatic call to the setProperties( ) method by the importData( ) and importFrom( ) methods.

### Parameters

**Format**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the fileFormat attribute.

### Examples

None.

## setHeight( )

### Format

public void setHeight(int Height)

### Description

Sets the value of the height attribute.

The setProperties( ) method sets this attribute automatically for certain image formats; use this method only if you are not using the setProperties( ) method. This method sets only the attribute value; it does not modify the image data itself.

### Parameters

**Height**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the height attribute.

### Examples

```
imgObj.setHeight(24);
```

where:

- 24: is the value to be set, in pixels, in the height attribute.

## setLocal( )

### Format

public void setLocal( )

### Description

Sets the value of the local attribute to indicate that the image data is stored locally in the database in a BLOB specified by the localData attribute.

### Parameters

None.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the localData attribute.

### Examples

```
imgObj.setLocal( );
```

## setMimeType( )

### Format

public void setMimeType(String mimeType)

### Description

Sets the value of the mimeType attribute.

The setProperties( ) method sets this attribute automatically for certain media formats; use this method only if you are not using the setProperties( ) method. This method sets only the attribute value; it does not modify the media data itself.

### Parameters

**mimeType**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the mimeType attribute.

### Examples

```
imgObj.setMimeType("image/bmp");
```

where:

- image/bmp: is the MIME type to be set.

## setProperties( )

### Format

public void setProperties( )

### Description

Parses the image data properties and sets the values of the attributes in the OrdImage Java object. This method sets the values of the height, width, contentLength, fileFormat, contentFormat, compressionFormat, and mimeType attributes. An attribute is set to null if the corresponding property cannot be extracted for a specific image format. This method throws a SQLException error if the image format is not recognized.

### Parameters

None.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding setProperties( ) method in the database.

### Examples

```
imgObj.setProperties( );
```

## setProperties(String)

### Format

public void setProperties(String description)

### Description

Writes the characteristics of a foreign image into the appropriate attribute fields. This method sets the values of various attributes of the OrdImage object based on a set of characteristics that describe the image properties. With this information, *inter*Media is able to process certain foreign image formats. For more information on setting image characteristics for foreign images, see *Oracle interMedia User's Guide and Reference*.

### Parameters

**description**
A String that specifies the image characteristics to set for the foreign image.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding setProperties( ) method in the database.

### Examples

```
String properties = "width=123 height=321 compressionformat=NONE
     userString=DJM dataOffset=128 scanlineOrder=INVERSE
     pixelOrder=REVERSE interleaving=BIL numberOfBands=1
     defaultChannelSelection=1";
imgObj.setProperties(properties);
```

where:

- properties: is a String that contains the properties to be set.

## setSource( )

### Format

public void setSource(String srcType, String srcLocation, String srcName)

### Description

Sets the values of the srcType, srcLocation, and srcName attributes.

### Parameters

**srcType**
The type of the source.

**srcLocation**
The location of the source.

**srcName**
The name of the source.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the srcType, srcLocation, or srcName attributes.

### Examples

```
imgObj.setSource("file","IMAGEDIR","jdoe.gif");
```

where:

- file: is the source type.
- IMAGEDIR: is the source location.
- jdoe.gif: is the source name.

## setUpdateTime( )

### Format

public void setUpdateTime(java.sql.Timestamp currentTime)

### Description

Sets the value of the updateTime attribute. This method sets the value of the updateTime attribute to the specified time, or to the database server's current SYSDATE time if the currentTime attribute is specified as null.

### Parameters

**currentTime**
The update time, or the null value, used to set the value of the updateTime attribute to the database server's current SYSDATE time.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding setUpdateTime( ) method in the database.

### Examples

```
imgObj.setUpdateTime(null);
```

## setWidth( )

### Format

public void setWidth(int Width)

### Description

Sets the value of the width attribute.

The setProperties( ) method sets this attribute automatically for certain image formats; use this method only if you are not using the setProperties( ) method. This method sets only the attribute value; it does not modify the image data itself.

### Parameters

**Width**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the width attribute.

### Examples

```
imgObj.setWidth(24);
```

where:

- 24: is the value to be set, in pixels, in the width attribute.

# 6 OrdImageSignature Reference Information

The OrdImageSignature class is used to represent an instance of the ORDSYS.ORDImageSignature database type in a Java application. The OrdImageSignature class includes a method to generate an image signature, as well as static methods to compare two image signatures.

If your application modifies the OrdImageSignature object, you must perform a SQL update operation to make those changes permanent.

---

**Note:** To take advantage of the increased performance that is possible using image matching with image signature indexes on the underlying tables, use the IMGSimilar and IMGScore SQL operators.

See *Oracle interMedia User's Guide and Reference* for more information about image signature indexes.

---

## 6.1 Prerequisites

You will need to include the following import statements in your Java file to run *inter*Media methods:

```
import java.sql.*;
import java.io.*;
import oracle.jdbc.driver.*;
import oracle.sql.*;
import oracle.ord.im.*;
```

The examples in this reference chapter are based on the assumption that the following operations have already been performed:

- A connection has been made to a table that contains a column of type OrdImageSignature.
- A local OrdImageSignature object named matchObj has been created and populated with data.

## 6.2 Reference Information

This section presents reference information on the methods that operate on OrdImageSignature objects.

## evaluateScore( )

### Format

public static float evaluateScore(OrdImageSignature signature1, OrdImageSignature signature2,
String attrWeights)

### Description

Compares two image signatures, returning a score that indicates the degree of difference between the image signatures. This method compares the image signatures in signature1 and signature2 using weights specified for one or more visual attributes. Returns a score between 0.0 and 100.0, where a lower value indicates a closer match.

Specify a weight in the range 0.0 to 1.0 for one or more of the following visual attributes: color, shape, texture, location.

You must specify a value greater than 0.0 for at least one of the following attributes: color, shape, or texture. The location attribute indicates the importance of the distribution of the color, shape, or texture features in the images. During processing, the values are normalized such that they total 1.0. For example:

```
color=0.7,shape-0.3
```

---

**Note:** The ORDImageSignature evaluateScore( ) method operates on two image signatures, not on indexes on database tables. Therefore, this method cannot take advantage of the increased performance that is possible using image matching with image signature indexes on the underlying tables. To use image signature indexes, use the IMGSimilar and IMGScore SQL operators.

See *Oracle interMedia User's Guide and Reference* for more information about image signature indexes.

---

### Parameters

**signature1**
The first OrdImageSignature.

**signature2**
An OrdImageSignature to be compared to signature1.

**attrWeights**
A String that specifies a list of one or more visual attributes and the weight to be applied to each attribute.

## Return Value

This method returns the score, as a floating-point value.

## Exceptions

java.sql.SQLException

This exception is thrown if an error occurs calling the evaluateScore( ) method in the database.

## Examples

```
float score = matchObj.evaluateScore(signature1, signature2, "color=1.0");
```

where:

- signature1: is the first signature.
- signature2: is the second signature, which will be compared to signature1.
- color=1.0: is the weight to apply to the color visual attribute.

## generateSignature( )

### Format

public void generateSignature(OrdImage img)

### Description

Generates an image signature for the specified image.

### Parameters

**img**
An OrdImage object from which to generate the signature.

### Return Value

None.

### Exceptions

SQLException

This exception is thrown if an error occurs calling the generateSignature( ) method in the database.

### Examples

```
matchObj.generateSignature(imgObj);
```

where:

- imgObj: is the image object whose signature is to be generated.

## getFactory( )

### Format

public static oracle.sql.CustomDatumFactory getFactory( )

### Description

Returns the OrdImageSignature CustomDatumFactory interface for use by the getCustomDatum( ) method. Specify the getFactory( ) method as the factory parameter of the getCustomDatum( ) method when retrieving an OrdImageSignature object from an OracleResultSet or OracleCallableStatement object.

### Parameters

None.

### Return Value

This method returns the OrdImageSignature implementation of the CustomDatumFactory interface.

### Exceptions

None.

### Examples

```
OrdImageSignature sig = (OrdImageSignature)rset.getCustomDatum( 1,
OrdImageSignature.getFactory() );
```

## isSimilar( )

### Format

```
public static int isSimilar(OrdImageSignature signature1, OrdImageSignature signature2,
    String attrWeights, float threshold)
```

### Description

Compares two image signatures, returning a status that indicates if the degree of difference between the image signatures is within a specified threshold. This method compares the image signatures in signature1 and signature2 using weights specified for one or more visual attributes. The result of the comparison is a score between 0.0 and 100.0, where a lower value indicates a closer match. If the score is less than or equal to the specified threshold, the images are considered a match and the method returns 1; otherwise, the method returns 0.

Specify a weight in the range 0.0 to 1.0 for one or more of the following visual attributes: color, shape, texture, location.

You must specify a value greater than 0.0 for at least one of the following attributes: color, shape, or texture. The location attribute indicates the importance of the distribution of the color, shape, or texture features in the images. During processing, the values are normalized such that they total 1.0. For example:

```
color=0.7,shape-0.3
```

**Note:** The ORDImageSignature isSimilar( ) method operates on two image signatures, not on indexes on database tables. Therefore, this method cannot take advantage of the increased performance that is possible using image matching with image signature indexes on the underlying tables. To use image signature indexes, use the IMGSimilar and IMGScore SQL operators.

See *Oracle interMedia User's Guide and Reference* for more information about image signature indexes.

### Parameters

**signature1**
The first OrdImageSignature.

**signature2**
An OrdImageSignature to be compared to signature1.

**attrWeights**
A String that specifies a list of one or more visual attributes and the weight to be applied to each attribute.

**threshold**
A floating-point value that specifies the degree of similarity required for the two images to be considered a match.

## Return Value

This method returns an integer value of 1 if the images match; otherwise, it returns 0.

## Exceptions

java.sql.SQLException

This exception is thrown if an error occurs calling the isSimilar( ) method in the database.

## Examples

```
int i = matchObj.isSimilar(signature1, signature2, "color=0.5,shape=0.5", 10);
```

where:

- signature1: is the first signature.
- signature2: is the second signature, which will be compared to signature1.
- color=0.5,shape=0.5: is the weight to apply to these visual attributes.
- 10: is the value that the score must be less than or equal to in order to be considered a match.

# 7 JAI Input and Output Stream Reference Information

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):

- BfileInputStream, which allows you to read data from an Oracle BFILE associated with the stream.
- BlobInputStream, which allows you to read data from an Oracle BLOB associated with the stream.
- BlobOutputStream, which allows you to write data from a buffer to an Oracle BLOB associated with the stream

## 7.1 Prerequisites

You will need to include the following import statements in your Java file in order to use the JAI stream objects:

```
import oracle.sql.BLOB;
import oracle.sql.BFILE;
import oracle.ord.media.jai.io.*;
```

## BfileInputStream Object

This section presents reference information on the methods associated with the BfileInputStream object, which provides an interface for JAI to read data from BFILEs. It is a subclass of com.sun.media.jai.codec.SeekableStream and java.io.InputStream; it implements java.io.DataInput.

Some examples in this reference chapter are based on the assumption that the following operations have already been performed:

- The following import statements have been included:

  ```
  import javax.media.jai.JAI;
  import java.awt.image.RenderedImage;
  ```

- A local BfileInputStream object named inStream has been created.

## BfileInputStream(BFILE)

### Format

public BfileInputStream(oracle.sql.BFILE bfile)

### Description

Creates a BfileInputStream object that reads from the specified BFILE. The constructor uses the maximum chunk size defined for a BFILE. The BFILE will be opened after this constructor executes.

### Parameters

**bfile**
The BFILE from which data will be read.

### Return Value

None.

### Exceptions

java.io.IOException

java.sql.SQLException

### Example

```
BfileInputStream inStream = new BfileInputStream(bfile);
RenderedImage image = JAI.create("stream",inStream);
```

## BfileInputStream(BFILE, int)

### Format

public BfileInputStream(oracle.sql.BFILE bfile, int chunkSize)

### Description

Creates a BfileInputStream object that reads from the specified BFILE. The constructor uses the specified chunk size. The BFILE will be opened after this constructor executes.

### Parameters

**bfile**
The BFILE from which data will be read.

**chunkSize**
The maximum amount of data to read from the BFILE at one time.

### Return Value

None.

### Exceptions

java.io.IOException

java.sql.SQLException

### Example

```
BfileInputStream inStream = new BfileInputStream(bfile,4096);
RenderedImage image = JAI.create("stream",inStream);
```

where:

- 4096: is the maximum number of bytes to be read at one time.

## canSeekBackwards( )

### Format

public boolean canSeekBackwards( )

### Description

Checks whether or not the stream can read backwards. Because the BfileInputStream object can read backwards, this method will always return true.

### Parameters

None.

### Return Value

This method returns true.

### Exceptions

None.

### Example

None.

## close( )

### Format

public void close( )

### Description

Closes the BfileInputStream, releasing any resources being used. The BFILE automatically closes after the stream closes.

### Parameters

None.

### Return Value

None.

### Exceptions

java.io.IOException

### Example

```
inStream.close( )
```

## getBFILE( )

### Format

public oracle.sql.BFILE getBFILE( )

### Description

Returns the BFILE associated with the BfileInputStream.

### Parameters

None.

### Return Value

This method returns the BFILE associated with the BfileInputStream.

### Exceptions

None.

### Example

```
BFILE imageBFILE = inStream.getBFILE( );
```

## getFilePointer( )

### Format

public long getFilePointer( )

### Description

Returns the offset from the beginning of the BFILE at which the next read will occur.

### Parameters

None.

### Return Value

This method returns the offset from the beginning of the BFILE at which the next read will occur, in bytes.

### Exceptions

java.io.IOException

### Example

```
long offset = inStream.getFilePointer( );
```

## mark( )

### Format

public void mark(int readLimit)

### Description

Marks the current position in the BfileInputStream. A call to the reset( ) method will return you to the last marked position in the BfileInputStream.

### Parameters

**readLimit**
This argument is ignored by the class.

### Return Value

None.

### Exceptions

None.

### Example

```
inStream.mark(1);
```

## markSupported( )

### Format

```
public boolean markSupported( )
```

### Description

Checks whether or not the BfileInputStream supports marking. Because the BfileInputStream object supports marking, this method will always return true.

### Parameters

None.

### Return Value

This method returns true.

### Exceptions

None.

### Example

None.

## read( )

### Format

public int read( )

### Description

Reads a single byte from the BFILE associated with the BfileInputStream.

### Parameters

None.

### Return Value

This method returns the byte of data that is read, or -1 if the end of the BFILE has been reached.

### Exceptions

java.io.IOException

### Example

```
int i = inStream.read( );
```

## read(byte[ ])

### Format

public int read(byte[ ] buffer)

### Description

Reads data from the BFILE into the specified buffer.

### Parameters

**buffer**
The buffer into which the data is read.

### Return Value

This method returns the number of bytes read into the buffer, or -1 if the end of the BFILE was reached before any data was read. The value cannot exceed the length of the buffer.

### Exceptions

java.io.IOException

### Example

```
byte[ ] buffer = new byte[4000];
int i = inStream.read(buffer);
```

where:

- buffer: is the buffer into which the data is read.

## read(byte[ ], int, int)

### Format

public int read(byte[ ]buffer, int off, int len)

### Description

Reads up to the specified length of data from the BFILE into the specified buffer, starting from the specified offset.

### Parameters

**buffer**
The buffer into which the data is read.

**off**
The offset from the beginning of the buffer at which data will be written, in bytes.

**len**
The maximum number of bytes to be read into the buffer.

### Return Value

This method returns the number of bytes read, or -1 if the end of the BFILE was reached before any data was read. The value cannot exceed the length of the buffer.

### Exceptions

java.io.IOException

### Example

```
byte[ ] buffer = new byte[4000];
int i = inStream.read(buffer,75,50);
```

where:

- buffer: is the buffer into which the data is read.
- 75: is the offset from the beginning of the buffer at which reading will begin.
- 50: is the number of bytes to be read into the buffer.

## remaining( )

### Format

public long remaining( )

### Description

Returns the number of unread bytes remaining in the BFILE.

### Parameters

None.

### Return Value

This method returns the number of unread bytes in the BFILE.

### Exceptions

None.

### Example

```
long remain = inStream.remaining( );
```

## reset( )

### Format

public void reset( )

### Description

Repositions the stream to the position of the last valid mark.

### Parameters

None.

### Return Value

None.

### Exceptions

java.io.IOException

### Example

```
inStream.reset( );
```

## seek( )

### Format

public void seek(long pos)

### Description

Sets the offset from the beginning of the BFILE at which the next read should occur.

### Parameters

**pos**
The offset from the beginning of the BFILE at which the next read should occur.

### Return Value

None.

### Exceptions

java.io.IOException

### Example

```
inStream.seek(75);
```

where:

- 75: is the offset from the beginning of the BFILE at which the next read should occur.

## skip( )

### Format

public long skip(long n)

### Description

Attempts to skip over the specified number of bytes in the BFILE.

The number of bytes skipped may be smaller than the specified number; for example, the number would be smaller if the end of the file is reached.

### Parameters

**n**
The number of bytes to be skipped.

### Return Value

This method returns the number of bytes that are actually skipped.

### Exceptions

java.io.IOException

### Example

```
long skipped = inStream.skip(100);
```

where:

- 100: is the number of bytes to be skipped.

## BlobInputStream Object

This section presents reference information on the methods associated with the BlobInputStream object, which provides an interface for JAI to read data from BLOBs. It is a subclass of com.sun.media.jai.codec.SeekableStream and java.io.InputStream; it implements java.io.DataInput.

Some examples in this reference chapter are based on the assumption that the following operations have already been performed:

- The following import statements have been included:

  ```
  import javax.media.jai.JAI;
  import java.awt.image.RenderedImage;
  ```

- A local BlobInputStream object named inStream has been created.

## BlobInputStream(BLOB)

### Format

public BlobInputStream(oracle.sql.BLOB blob)

### Description

Creates a BlobInputStream object that reads from the specified BLOB. The constructor uses an optimal chunk size that is determined by the database.

### Parameters

**blob**
The BLOB from which data will be read.

### Return Value

None.

### Exceptions

java.io.IOException

java.sql.SQLException

### Example

```
BlobInputStream inStream = new BlobInputStream(blob);
RenderedImage image = JAI.create("stream",inStream);
```

## BlobInputStream(BLOB, int)

### Format

public BlobInputStream(oracle.sql.BLOB blob, int chunkSize)

### Description

Creates a BlobInputStream object that reads from the specified BLOB. The constructor uses the specified chunk size.

### Parameters

**blob**
The BLOB from which data will be read.

**chunkSize**
The maximum amount of data to read from the BLOB at one time.

### Return Value

None.

### Exceptions

java.io.IOException

java.sql.SQLException

### Example

```
BlobInputStream inStream = new BlobInputStream(blob,4096);
RenderedImage image = JAI.create("stream",inStream);
```

where:

- 4096: is the maximum number of bytes to be read at one time.

## canSeekBackwards( )

### Format

public boolean canSeekBackwards( )

### Description

Checks whether or not the stream can read backwards. Because the BlobInputStream object can read backwards, this method will always return true.

### Parameters

None.

### Return Value

This method returns true.

### Exceptions

None.

### Example

None.

## close( )

### Format

public void close( )

### Description

Closes the BlobInputStream, releasing any resources being used.

### Parameters

None.

### Return Value

None.

### Exceptions

java.io.IOException

### Example

```
inStream.close( )
```

## getBLOB( )

### Format

public oracle.sql.BLOB getBLOB( )

### Description

Returns the BLOB associated with the BlobInputStream.

### Parameters

None.

### Return Value

This method returns the BLOB associated with the BlobInputStream.

### Exceptions

None.

### Example

```
BLOB imageBLOB = inStream.getBLOB( );
```

## getFilePointer( )

### Format

public long getFilePointer( )

### Description

Returns the offset from the beginning of the BLOB at which the next read will occur.

### Parameters

None.

### Return Value

This method returns the offset from the beginning of the BLOB at which the next read will occur, in bytes.

### Exceptions

java.io.IOException

### Example

```
long offset = inStream.getFilePointer( );
```

## mark( )

### Format

public void mark(int readLimit)

### Description

Marks the current position in the BlobInputStream. A call to the reset( ) method will return you to the last marked position in the BlobInputStream.

### Parameters

**readLimit**
This argument is ignored by the class.

### Return Value

None.

### Exceptions

None.

### Example

```
inStream.mark(1);
```

## markSupported( )

### Format

public boolean markSupported( )

### Description

Checks whether or not the BlobInputStream supports marking. Because the BlobInputStream object supports marking, this method will always return true.

### Parameters

None.

### Return Value

This method returns true.

### Exceptions

None.

### Example

None.

## read( )

### Format

public int read( )

### Description

Reads a single byte from the BLOB associated with the BlobInputStream.

### Parameters

None.

### Return Value

This method returns the byte of data that is read, or -1 if the end of the BLOB has been reached.

### Exceptions

java.io.IOException

### Example

```
int i = inStream.read( );
```

## read(byte[ ])

### Format

public int read(byte[ ] buffer)

### Description

Reads data from the BLOB into the specified buffer.

### Parameters

**buffer**
The buffer into which the data is read.

### Return Value

This method returns the number of bytes read into the buffer, or -1 if the end of the BLOB was reached before any data was read. The value cannot exceed the length of the buffer.

### Exceptions

java.io.IOException

### Example

```
byte[ ] buffer = new byte[4000];
int i = inStream.read(buffer);
```

where:

- buffer: is the buffer into which the data is read.

## read(byte[ ], int, int)

### Format

public int read(byte[ ]buffer, int off, int len)

### Description

Reads up to the specified length of data from the BLOB into the specified buffer, starting from the specified offset.

### Parameters

**buffer**
The buffer into which the data is read.

**off**
The offset from the beginning of the buffer at which data will be written, in bytes.

**len**
The maximum number of bytes to be written to the buffer.

### Return Value

This method returns the number of bytes read into the buffer, or -1 if the end of the BLOB has been reached. The value cannot exceed the length of the buffer.

### Exceptions

java.io.IOException

### Example

```
byte[ ] buffer = new byte[4000];
int i = inStream.read(buffer,75,50);
```

where:

- buffer: is the buffer into which the data is read.
- 75: is the offset from the beginning of the buffer at which data will be written.
- 50: is the number of bytes to be read into the buffer.

## remaining( )

### Format

public long remaining( )

### Description

Returns the number of unread bytes remaining in the BLOB.

### Parameters

None.

### Return Value

This method returns the number of unread bytes in the BLOB.

### Exceptions

None.

### Example

```
long remain = inStream.remaining( );
```

## reset( )

### Format

public void reset( )

### Description

Repositions the stream to the position of the last valid mark.

### Parameters

None.

### Return Value

None.

### Exceptions

java.io.IOException

### Example

```
inStream.reset( );
```

## seek( )

### Format

public void seek(long pos)

### Description

Sets the offset from the beginning of the BLOB at which the next read should occur.

### Parameters

**pos**
The offset from the beginning of the BLOB at which the next read should occur.

### Return Value

None.

### Exceptions

java.io.IOException

### Example

```
inStream.seek(75);
```

where:

- 75: is the offset from the beginning of the BLOB at which the next read should occur.

## skip( )

### Format

public long skip(long n)

### Description

Attempts to skip over the specified number of bytes in the BLOB.

The number of bytes skipped may be smaller than the specified number; for example, the number would be smaller if the end of the file is reached.

### Parameters

**n**
The number of bytes to be skipped.

### Return Value

This method returns the number of bytes that are actually skipped.

### Exceptions

java.io.IOException

### Example

```
long skipped = inStream.skip(100);
```

where:

- 100: is the number of bytes to be skipped.

## BlobOutputStream Object

This section presents reference information on the methods associated with the BlobOutputStream object, which provides an interface for JAI to write data to BLOBs. It is a subclass of java.io.OutputStream.

Some examples in this reference chapter are based on the assumption that the following operations have already been performed:

- The following import statements have been included:

  ```
  import javax.media.jai.JAI;
  import java.awt.image.RenderedImage;
  ```

- A local BlobOutputStream object named outStream has been created.

## BlobOutputStream(BLOB)

### Format

public BlobOutputStream (oracle.sql.BLOB blob)

### Description

Creates a BlobOutputStream object that writes to the specified BLOB, using an optimal chunk size that is determined by the database. Creating an object of this type implicitly trims the data in the BLOB to a length of 0.

### Parameters

**blob**
The BLOB to which data will be written.

### Return Value

None.

### Exceptions

java.io.IOException

java.sql.SQLException

### Example

```
RenderedImage = JAI.create("fileload","sample.jpg");
BlobOutputStream outStream = new BlobOutputStream(blob);
JAI.create("encode",image,"bmp")
```

## BlobOutputStream(BLOB, int)

### Format

public BlobOutputStream(oracle.sql.BLOB blob, int chunkSize)

### Description

Creates a BlobOutputStream object that writes to the specified BLOB, using the given integer as the maximum chunk size. Creating an object of this type implicitly trims the data in the BLOB to a length of zero.

### Parameters

**blob**
The BLOB to which data will be written

**chunkSize**
The maximum amount of data to write to the BLOB at one time.

### Return Value

None.

### Exceptions

java.io.IOException

java.sql.SQLException

### Example

```
RenderedImage = JAI.create("fileload","sample.jpg");
BlobOutputStream outStream = new BlobOutputStream(blob,4096);
JAI.create("encode",image,"bmp")
```

where:

- 4096: is the maximum amount of data that will be written at one time.

## close( )

### Format

public void close( )

### Description

Closes the output stream and releases any system resources associated with this stream. Before closing the stream, this method automatically calls flush( ) to write any buffered bytes to the BLOB.

### Parameters

None.

### Return Value

None.

### Exceptions

java.io.IOException

### Example

```
outStream.close( );
```

## flush( )

### Format

public void flush( )

### Description

Flushes the output stream and forces any buffered output bytes to be written to the BLOB.

### Parameters

None.

### Return Value

None.

### Exceptions

java.io.IOException

### Example

```
outStream.flush( );
```

## getFilePointer( )

### Format

public long getFilePointer( )

### Description

Returns the offset from the beginning of the BLOB at which the next write will occur.

### Parameters

None.

### Return Value

This method returns the offset from the beginning of the BLOB at which the next write will occur, in bytes.

### Exceptions

java.io.IOException

### Example

```
long offset = outStream.getFilePointer( );
```

## length( )

### Format

public long length( )

### Description

Returns the current length of the output stream.

### Parameters

None.

### Return Value

This method returns the current length of the output stream.

### Exceptions

java.io.IOException

### Example

```
long length = outStream.length( );
```

## seek( )

### Format

public void seek(long pos)

### Description

Sets the file-pointer offset, measured from the beginning of this stream, at which the next write occurs.

The offset may be set beyond the end of the stream. Setting the offset beyond the end of the stream does not change the stream length; the stream length will change only by writing after the offset has been set beyond the end of the stream.

### Parameters

**pos**
The offset position, measured in bytes from the beginning of the stream, at which to set the file pointer.

### Return Value

None.

### Exceptions

java.io.IOException

### Example

```
outStream.seek(4096);
```

## write(byte[ ])

### Format

public void write(byte[ ] buffer)

### Description

Writes all bytes in the specified byte array to the BLOB.

### Parameters

**buffer**
An array of bytes to be written to the BLOB.

### Return Value

None.

### Exceptions

java.io.IOException

### Example

```
//create a byte array named buffer and populate it with data
outStream.write(buffer);
```

where:

- buffer: is the array of bytes that will be written to the BLOB.

## write(byte[ ], int, int)

### Format

public void write(byte[ ] buffer, int off, int len)

### Description

Writes the specified number of bytes from the specified byte array to the BLOB.

### Parameters

**buffer**
The buffer containing the data to be written to the BLOB.

**off**
The start offset in the buffer.

**len**
The number of bytes to write to the BLOB.

### Return Value

None.

### Exceptions

java.io.IOException

### Example

```
//create a byte array named buffer and populate it with data
outStream.write(buffer,75,50);
```

where:

- buffer: is the array of bytes that will be written to the BLOB.
- 75: is the offset from the beginning of the buffer at which data will be read.
- 50: is the number of bytes to be written.

## write(int)

### Format

public void write(int b)

### Description

Writes the specified byte to the BLOB.

### Parameters

**b**
The byte to be written to the BLOB. Only the low-order byte is written; the upper 24 bits are ignored.

### Return Value

None.

### Exceptions

java.io.IOException

### Example

```
outStream.write(50);
```

where:

- 50: is an integer whose low-order byte will be written to the BLOB.

# 8 OrdVideo Reference Information

The OrdVideo class is used to represent an instance of the ORDSYS.ORDVideo database type in a Java application. The OrdVideo class includes a set of methods to get and set various object attributes, as well as a set of methods that perform various operations on an OrdVideo Java object.

Almost all methods operate on the attributes of the OrdVideo Java object in the application. The exceptions are those methods that access the video data for read or write purposes, which are described in the following list:

- Methods that operate on the database BLOB specified by the localData attribute, read and write data stored in the database BLOB.
- Methods that operate on the database BFILE specified by the srcLocation and srcName attributes when the srcType attribute is "file," read data from the specified file at the database server.
- Methods that operate on the URL specified by the srcType, srcLocation, and srcName attributes when the srcType attribute is "http," read data from the resource at the specified URL.

If your application modifies the OrdVideo Java object, or the video data in the database, you must update the ORDVideo SQL object in the database to make those changes permanent.

Some methods in the OrdVideo Java class are handed off to a database source plug-in or database format plug-in for processing; these methods have `byte [ ] [ ] ctx` as a context parameter. Applications should allocate a 64-byte array to hold any context information that may be required by a source plug-in or a format plug-in. For example, a plug-in may initialize the context information in one call and use that information in a subsequent call. The source plug-in context requires one array; the format plug-in context requires another array. For most plug-ins, 64 bytes should be sufficient. Some user-defined plug-ins may need additional space. The following example shows how to allocate a plug-in context information array:

```
byte [] [] ctx = new byte[1][64];
```

---

**Note:** In the current release, no Oracle-supplied source plug-ins or format plug-ins maintain context. Also, not all user-written source plug-ins or format plug-ins maintain context. However, if you include the context parameter as described, your application should work with any current or future source plug-ins or format plug-ins.

---

See *Oracle interMedia User's Guide and Reference* for more information about plug-ins.

## 8.1 Prerequisites

You will need to include the following import statements in your Java file in order to run *inter*Media methods:

```
import java.sql.*;
import java.io.*;
import oracle.jdbc.driver.*;
import oracle.sql.*;
import oracle.ord.im.*;
```

The examples in this reference chapter are based on the assumption that the following operations have already been performed:

- A connection has been made to a table that contains a column of type OrdVideo.
- A local OrdVideo object named vidObj has been created and populated with data.

For examples of making a connection and populating a local object, see Section 2.4.2.

## 8.2 Reference Information

This section presents reference information on the methods that operate on OrdVideo objects.

## checkProperties( )

### Format

public boolean checkProperties(byte[ ] [ ] ctx)

### Description

Checks if the properties of the video data are consistent with the attributes of the OrdVideo Java object.

### Parameters

**ctx**
The format plug-in context information.

### Return Value

This method returns true if the properties of the video data are consistent with the attributes of the OrdVideo Java object; false otherwise.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding checkProperties( ) method in the database.

### Examples

```
byte [ ] [ ] ctx = new byte[1][64];
if(vidObj.checkProperties(ctx))
     System.out.println("checkProperties successful");
```

where:

- ctx: contains the format plug-in context information.

## clearLocal( )

### Format

public void clearLocal( )

### Description

Clears the local attribute to indicate that the video data is stored externally.

### Parameters

None.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the local attribute.

### Examples

```
vidObj.clearLocal( );
```

## closeSource( )

### Format

public int closeSource(byte[ ] [ ] ctx)

### Description

Closes a data source.

### Parameters

**ctx**
The source plug-in context information.

### Return Value

This method returns the status as an integer, where zero indicates success and a non-zero value indicates a failure code specific to the source plug-in.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding closeSource( ) method in the database.

### Examples

```
byte [ ] [ ] ctx = new byte[1][64];
int i = vidObj.closeSource(ctx);
if(i == 0)
     System.out.println("closeSource successful");
```

where:

- ctx: contains the source plug-in context information.

## deleteContent( )

### Format

public void deleteContent( )

### Description

Deletes any data stored in the database BLOB specified by the localData attribute.

### Parameters

None.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding deleteContent( ) method in the database.

### Examples

```
vidObj.deleteContent( );
```

## export( )

### Format

public void export (byte[ ] [ ] ctx, String srcType, String srcLocation, String srcName)

### Description

Exports the data from the BLOB specified by the localData attribute. This method calls the corresponding export( ) method in the database to export the video data to a location specified by the srcType, srcLocation, and srcName parameters.

Not all source plug-ins support this method. Only the "file" source type is natively supported.

This method will work only if you are running Oracle database server release 8.1.7 or later.

The remainder of this description describes the use of the export( ) method and the Oracle-supplied "file" source plug-in. User-written plug-ins will behave differently.

The export( ) method implemented by the "file" source plug-in copies, but does not modify, the video data stored in the database BLOB specified by the localData attribute.

After exporting the video data, all the video property attributes remain unchanged. However, the srcType, srcLocation, and srcName attributes are updated with values of the srcType, srcLocation, and srcName parameters passed to the export( ) method. After calling the export( ) method, if you no longer intend to manage the video data within the database, call the clearLocal( ) method to indicate the video data is stored outside the database, and call the deleteContent( ) method to delete the video data stored in the database BLOB.

The export( ) method in the database writes only to a database directory object that the user has privileges to access. That is, you can access a directory that you have created using the SQL CREATE DIRECTORY statement, or one to which you have been granted READ access. To execute the CREATE DIRECTORY statement, you must have the CREATE ANY DIRECTORY privilege. In addition, you must use the DBMS_JAVA.GRANT_PERMISSION method to specify which files can be written.

For example, the following SQL*Plus command grants the user, MEDIAUSER, the permission to write to the file named movie1.mov:

```
CALL DBMS_JAVA.GRANT.PERMISSION(
     'MEDIAUSER',
     'java.io.FilePermission',
     '/videos/filmclips/movie1.mov',
     'write');
```

The previous example shows how to authorize access to write to a single file. In addition, there are various wildcard path specifications that authorize write access to multiple directories and file names. For example, a path specification that ends in a slash and asterisk (/*), where the slash is the file-separator character that is operating-system dependent, indicates all the files contained in the specified directory. A path specification that ends with a slash and hyphen (/-) indicates all files contained in the specified directory and all its subdirectories. A path name consisting of the special token <<ALL FILES>> authorizes access to any file.

See *Oracle9i Java Developer's Guide* and the java.io.FilePermission class in the Java API for more information about security and performance. See *Oracle interMedia User's Guide and Reference* for more information about the required privileges.

## Parameters

**ctx**
The source plug-in context information.

**srcType**
The source type to which the content will be exported.

**srcLocation**
The source location to which the content will be exported.

**srcName**
The source name to which the content will be exported.

## Return Value

None.

## Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding export( ) method in the database.

## Examples

```
byte [ ] [ ] ctx = new byte[1][64];
vidObj.export(ctx,"file","VIDEODIR","complete.dat");
```

where:

- ctx: contains the source plug-in context information.
- file: is the source plug-in used to export the content.
- VIDEODIR: is the location to which the content will be exported.
- complete.dat: is the file to which the content will be exported.

## getAllAttributes( )

### Format

public oracle.sql.CLOB getAllAttributes(byte[ ] [ ] ctx)

### Description

Returns the values of the video properties in a temporary CLOB in a form defined by the format plug-in. For natively supported formats, the information is presented as a comma-separated list of attributes in the form attributeName=attributeValue, where the list contains the following attributes: format, mimeType, width, height, frameResolution, frameRate, videoDuration, numberOfFrames, compressionType, numberOfColors, and bitRate. For user-defined formats, the information is presented in a form defined by the format plug-in.

**Note:** The application must free the temporary CLOB after reading the information it contains.

### Parameters

**ctx[ ]**
The format plug-in context information.

### Return Value

This method returns the values of the attributes as a temporary oracle.sql.CLOB.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding getAllAttributes( ) method in the database.

### Examples

```
byte [ ] [ ] ctx = new byte[1][64];
CLOB attributes = vidObj.getAllAttributes(ctx);
```

where:

- ctx: contains the format plug-in context information.

## getAttribute( )

### Format

public String getAttribute(byte[ ] [ ] ctx, String name)

### Description

Returns the value of the requested video property. This method is used by user-defined format plug-ins to return the value of a video property that is not available as an attribute of the OrdVideo Java object. This method is not implemented by any Oracle-supplied format plug-ins.

### Parameters

**ctx**
The format plug-in context information.

**name**
The property or attribute name.

### Return Value

This method returns the value of the attribute, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding getAttribute( ) method in the database.

### Examples

```
byte [ ] [ ] ctx = new byte[1][64];
String attribute = vidObj.getAttribute(ctx, video_duration);
```

where:

- ctx: contains the format plug-in context information.
- video_duration: is the value of the attribute to get.

## getBFILE( )

### Format

public oracle.sql.BFILE getBFILE( )

### Description

Returns a BFILE locator from the database when the srcType attribute is "file". This method calls the corresponding getBFILE( ) method in the database, which creates the BFILE using the srcLocation and srcName attributes.

### Parameters

None.

### Return Value

This method returns an oracle.sql.BFILE locator.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding getBFILE( ) method in the database.

### Examples

```
BFILE videoBFILE = vidObj.getBFILE( );
```

## getBitRate( )

### Format

public int getBitRate( )

### Description

Returns the value of the bitRate attribute.

### Parameters

None.

### Return Value

This method returns the value of the bitRate attribute, as an integer.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the bitRate attribute.

### Examples

```
int bitRate = vidObj.getBitRate( );
```

## getComments( )

### Format

public oracle.sql.CLOB getComments( )

### Description

Returns the CLOB locator from the comments attribute.

### Parameters

None.

### Return Value

This method returns the value of the comments attribute as an oracle.sql.CLOB locator.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the comments attribute.

### Examples

```
CLOB comments = vidObj.getComments( );
```

## getCompressionType( )

### Format

public String getCompressionType( )

### Description

Returns the value of the compressionType attribute.

### Parameters

None.

### Return Value

This method returns the value of the compressionType attribute, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the compressionType attribute.

### Examples

```
String compressionType = vidObj.getCompressionType( );
```

## getContent( )

### Format

public oracle.sql.BLOB getContent( )

### Description

Returns the BLOB locator from the localData attribute.

### Parameters

None.

### Return Value

This method returns an oracle.sql.BLOB locator.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the localData attribute.

### Examples

```
BLOB localContent = vidObj.getContent( );
```

## getContentInLob( )

### Format

public oracle.sql.BLOB getContentInLob(byte[ ] [ ] ctx, String mimetype[ ], String format[ ])

### Description

Returns the data from the BLOB specified by the localData attribute in a temporary BLOB in the database. This method creates a temporary BLOB in the database, reads the data from the BLOB specified by the localData attribute, writes the data to the temporary BLOB, then returns the temporary BLOB locator to the caller.

**Note:** The application must free the temporary BLOB after accessing the data it contains.

### Parameters

**ctx**
The source plug-in context information.

**mimetype**
A String array, 1 element in length, into which the mimeType attribute is written as element 0.

**format**
A String array, 1 element in length, into which the format attribute is written as element 0.

### Return Value

This method returns the video data in a temporary oracle.sql.BLOB locator.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs creating the temporary BLOB or executing the corresponding getContentInLob( ) method in the database.

## Examples

```
byte [ ] [ ] ctx = new byte[1][64];
String mimetype[ ] = new String[1];
String format[ ] = new String[1];
BLOB localContent = vidObj.getContentInLob(ctx,mimetype,format);
```

where:

- ctx: contains the source plug-in context information.
- mimetype: is an array of Strings whose first value contains the MIME type. This value is generated by the server.
- format: is an array of Strings whose first value contains the format. This value is generated by the server.

## getContentLength( )

### Format

public int getContentLength( )

### Description

Returns the length of the video data. This method calls the corresponding getContentLength( ) method in the database.

This method is not supported for all source types. For example, the "http" source type does not support this method.

### Parameters

None.

### Return Value

This method returns the value of the contentLength attribute, as an integer.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding getContentLength( ) method in the database.

### Examples

```
int contentLength = vidObj.getContentLength( );
```

## getContentLength(byte[ ][ ])

### Format

public int getContentLength(byte[ ] [ ] ctx)

### Description

Returns the length of the video data using source plug-in context information. This method calls the corresponding getContentLength( ) method in the database.

### Parameters

**ctx**
The source plug-in context information.

### Return Value

This method returns the value of the contentLength attribute, as an integer.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding getContentLength( ) method in the database.

### Examples

```
byte [ ] [ ] ctx = new byte[1][64];
int contentLength = vidObj.getContentLength(ctx);
```

where:

- ctx: contains the source plug-in context information.

## getDataInByteArray( )

### Format

public byte[ ] getDataInByteArray( )

### Description

Returns a byte array containing the data from the database BLOB specified by the localData attribute.

### Parameters

None.

### Return Value

This method returns a byte array containing the data.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing an object attribute.

java.io.IOException

This exception is thrown if an error occurs reading the data from the BLOB.

java.lang.OutOfMemoryError

This exception is thrown if sufficient memory cannot be allocated to hold the data.

### Examples

```
byte[ ] byteArr = vidObj.getDataInByteArray( );
```

## getDataInFile( )

### Format

public boolean getDataInFile(String filename)

### Description

Writes the data from the database BLOB specified by the localData attribute to a local file.

### Parameters

**filename**
The name of the file to which the data will be written.

### Return Value

This method returns true if the data is written to the file successfully; otherwise, an exception is raised if an error occurs. This method never returns false.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing an object attribute.

java.io.IOException

This exception is thrown if an error occurs reading the data from the BLOB or writing the data to the output file.

### Examples

```
boolean load = vidObj.getDataInFile("output1.dat");
if(load)
     System.out.println("getDataInFile completed successfully");
else
     System.out.println("Error in getDataInFile");
```

where:

- output1.dat: is the file into which the data will be loaded.

## getDataInStream( )

### Format

public InputStream getDataInStream( )

### Description

Returns an InputStream object from which the data in the database BLOB specified by the localData attribute can be read.

### Parameters

None.

### Return Value

This method returns an InputStream object from which the data will be read.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing an object attribute.

### Examples

```
InputStream inpStream = vidObj.getDataInStream( );
```

## getDescription( )

### Format

public String getDescription( )

### Description

Returns the value of the description attribute.

### Parameters

None.

### Return Value

This method returns the value of the description attribute, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the description attribute.

### Examples

```
String description = vidObj.getDescription( );
```

## getFactory( )

### Format

public static oracle.sql.CustomDatumFactory getFactory( )

### Description

Returns the OrdVideo CustomDatumFactory interface for use by the getCustomDatum( ) method. Specify the getFactory( ) method as the factory parameter of the getCustomDatum( ) method when retrieving an OrdVideo object from an OracleResultSet or OracleCallableStatement object.

### Parameters

None.

### Return Value

This method returns the OrdVideo implementation of the CustomDatumFactory interface.

### Exceptions

None.

### Examples

```
OrdVideo vid = (OrdVideo)rset.getCustomDatum( 1, OrdVideo.getFactory() );
```

## getFormat( )

### Format

public String getFormat( )

### Description

Returns the value of the format attribute.

### Parameters

None.

### Return Value

This method returns the value of the format attribute, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the format attribute.

### Examples

```
String format = vidObj.getFormat( );
```

## getFrameRate( )

### Format

public int getFrameRate( )

### Description

Returns the value of the frameRate attribute.

### Parameters

None.

### Return Value

This method returns the value of the frameRate attribute, as an integer.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the frameRate attribute.

### Examples

```
int frameRate = vidObj.getFrameRate( );
```

## getFrameResolution( )

### Format

public int getFrameResolution( )

### Description

Returns the value of the frameResolution attribute.

### Parameters

None.

### Return Value

This method returns the value of the frameResolution attribute, as an integer.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the frameResolution attribute.

### Examples

```
int frameResolution = vidObj.getFrameResolution( );
```

## getHeight( )

### Format

public int getHeight( )

### Description

Returns the value of the height attribute.

### Parameters

None.

### Return Value

This method returns the value of the height attribute, as an integer.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the height attribute.

### Examples

```
int height = vidObj.getHeight( );
```

## getMimeType( )

### Format

public String getMimeType( )

### Description

Returns the value of the mimeType attribute.

### Parameters

None.

### Return Value

This method returns the value of the mimeType attribute, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the mimeType attribute.

### Examples

```
String mimeType = vidObj.gtMimeType( );
```

## getNumberOfColors( )

### Format

public int getNumberOfColors( )

### Description

Returns the value of the numberOfColors attribute.

### Parameters

None.

### Return Value

This method returns the value of the numberOfColors attribute, as an integer.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the numberOfColors attribute.

### Examples

```
int numberOfColors = vidObj.getNumberOfColors( );
```

## getNumberOfFrames( )

### Format

public int getNumberOfFrames( )

### Description

Returns the value of the numberOfFrames attribute.

### Parameters

None.

### Return Value

This method returns the value of the numberOfFrames attribute, as an integer.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the numberOfFrames attribute.

### Examples

```
int numberOfFrames = vidObj.getNumberOfFrames( );
```

## getSource( )

### Format

public String getSource( )

### Description

Returns the source information in the form srcType://srcLocation/srcName.

### Parameters

None.

### Return Value

This method returns the source information, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding getSource( ) method in the database.

### Examples

```
String source = viObj.getSource( );
```

## getSourceLocation( )

### Format

public String getSourceLocation( )

### Description

Returns the value of the srcLocation attribute.

### Parameters

None.

### Return Value

This method returns the value of the srcLocation attribute, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the srcLocation attribute.

### Examples

```
String location = vidObj.getSourceLocation( );
```

## getSourceName( )

### Format

public String getSourceName( )

### Description

Returns the value of the srcName attribute.

### Parameters

None.

### Return Value

This method returns the value of the srcName attribute, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the srcName attribute.

### Examples

```
String name = vidObj.getSourceName( );
```

## getSourceType( )

### Format

public String getSourceType( )

### Description

Returns the value of the srcType attribute.

### Parameters

None.

### Return Value

This method returns the value of the srcType attribute, as a String.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the srcType attribute.

### Examples

```
String type = vidObj.getSourceType( );
```

## getUpdateTime( )

### Format

public java.sql.Timestamp getUpdateTime( )

### Description

Returns the value of the updateTime attribute.

### Parameters

None.

### Return Value

This method returns the value of the updateTime attribute as a java.sql.Timestamp object.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the updateTime attribute.

### Examples

```
Timestamp time = vidObj.getUpdateTime( );
```

## getVideoDuration( )

### Format

public int getVideoDuration( )

### Description

Returns the value of the videoDuration attribute.

### Parameters

None.

### Return Value

This method returns the value of the videoDuration attribute, as an integer.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the videoDuration attribute.

### Examples

```
int videoDuration = vidObj.getVideoDuration( );
```

## getWidth( )

### Format

public int getWidth( )

### Description

Returns the value of the width attribute.

### Parameters

None.

### Return Value

This method returns the value of the width attribute, as an integer.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the width attribute.

### Examples

```
int width = vidObj.getWidth( );
```

## importData( )

### Format

public void importData(byte[ ] [ ] ctx)

### Description

Imports data from an external source into the database BLOB specified by the localData attribute. The external data source is specified by the srcType, srcLocation, and srcName attributes.

### Parameters

**ctx**
The source plug-in context information.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding import( ) method in the database.

### Examples

```
byte [ ] [ ] ctx = new byte[1][64];
vidObj.importData(ctx);
```

where:

- ctx: contains the source plug-in information.

## importFrom( )

### Format

public void importFrom(byte[ ] [ ] ctx, String srcType, String srcLocation, String srcName)

### Description

Imports data from an external source into the database BLOB specified by the localData attribute. The external data source is specified by the srcType, srcLocation, and srcName parameters. The srcType, srcLocation, and srcName attributes are updated with values of the srcType, srcLocation, and srcName parameters passed to the importFrom( ) method.

### Parameters

**ctx**
The source plug-in context information. See *Oracle interMedia User's Guide and Reference* for more information.

**srcType**
The source type from which the data will be imported.

**srcLocation**
The source location from which the data will be imported.

**srcName**
The source name from which the data will be imported.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding importFrom( ) method in the database.

### Examples

```
byte [ ] [ ] ctx = new byte[1][64];
```

```
vidObj.importFrom(ctx,"file","VIDEODIR","testvid.dat");
```

where:

- ctx: contains the source plug-in context information.
- file: is the source plug-in used to import the data.
- VIDEODIR: is the location of the file from which the data will be imported.
- testvid.dat: is the file from which the data will be imported.

## isLocal( )

### Format

public boolean isLocal( )

### Description

Indicates if the video data is stored locally in the database in a BLOB specified by the localData attribute.

### Parameters

None.

### Return Value

This method returns true if the data is stored locally in the database in a BLOB; false otherwise.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the localData attribute.

### Examples

```
if(vidObj.isLocal( ))
     System.out.println("local attribute is set to true");
else
     System.out.println("local attribute is set to false");
```

## loadDataFromByteArray( )

### Format

public boolean loadDataFromByteArray(byte[ ] byteArr)

### Description

Loads data from a byte array into the database BLOB specified by the localData attribute. Before loading the data, this method calls the deleteContent( ) method to delete any existing data in the BLOB. It also calls the setLocal( ) method to set the local flag. In addition, this method calls the setUpdateTime( ) method to set the updateTime attribute to the database server's current SYSDATE time.

### Parameters

**byteArr**
A byte array from which the data will be loaded.

### Return Value

This method returns true if the data is loaded successfully; otherwise, an exception is raised if an error occurs. This method never returns false.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing an object attribute or executing a method in the database.

java.io.IOException

This exception is thrown if an error occurs reading the byte array.

### Examples

```
byte[ ] data = new byte[32000];
FileInputStream fStream = new FileInputStream("testvid.dat");
fStream.read(data,0,32000);
boolean success = vidObj.loadDataFromByteArray(data);
if(success)
     System.out.println("loadDataFromByteArray was successful");
```

```
else
     System.out.println("loadDataFromByteArray was unsuccessful");
```

where:

- data: is the local byte array from which the data will be loaded.
- testvid.dat: is a local file that contains 32,000 bytes of data.

## loadDataFromFile( )

### Format

public boolean loadDataFromFile(String filename)

### Description

Loads data from a file into the database BLOB specified by the localData attribute. Before loading the data, this method calls the deleteContent( ) method to delete any existing data in the BLOB. It also calls the setLocal( ) method to set the local flag. In addition, this method calls the setUpdateTime( ) method to set the updateTime attribute to the database server's current SYSDATE time.

### Parameters

**filename**
The name of the file from which the data will be loaded.

### Return Value

This method returns true if the data is loaded successfully; otherwise, an exception is raised if an error occurs. This method never returns false.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing an object attribute or executing a method in the database.

java.io.IOException

This exception is thrown if an error occurs reading the data file.

### Examples

```
vidObj.loadDataFromFile("testvid.dat");
```

where:

- testvid.dat: is a local file that contains video data.

## loadDataFromInputStream( )

### Format

public boolean loadDataFromInputStream(InputStream inpStream)

### Description

Loads data from an InputStream object into the database BLOB specified by the localData attribute. Before loading the data, this method calls the deleteContent( ) method to delete any existing data in the BLOB. It also calls the setLocal( ) method to set the local flag. In addition, this method calls the setUpdateTime( ) method to set the updateTime attribute to the database server's current SYSDATE time.

### Parameters

**inpStream**
The InputStream object from which the data will be loaded.

### Return Value

This method returns true if the data is loaded successfully; otherwise, an exception is raised if an error occurs. This method never returns false.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing an object attribute or executing a method in the database.

java.io.IOException

This exception is thrown if an error occurs reading the InputStream object.

### Examples

```
FileInputStream fStream = new FileInputStream("testvid.dat");
vidObj.loadDataFromInputStream(fStream);
```

where:

- testvid.dat: is a local file that contains video data.

- fStream: is the local input stream that will load video data into the OrdVideo object.

## openSource( )

### Format

public int openSource(byte[ ] userarg, byte[ ] [ ] ctx)

### Description

Opens a data source.

### Parameters

**userarg**
Additional source plug-in information that may be required by user-defined source plug-ins.

**ctx**
The source plug-in context information. See *Oracle interMedia User's Guide and Reference* for more information.

### Return Value

This method returns the status as an integer, where zero indicates success and a non-zero value indicates a failure code specific to the source plug-in.

### Exceptions

java.lang.Exception

This exception is thrown if an error occurs executing the corresponding openSource( ) method in the database.

### Examples

```
byte[ ] userarg = new byte[4000];
byte [ ] [ ] ctx = new byte[1][64];
int i = vidObj.openSource(userarg,ctx);
if(i == 0)
     System.out.println("openSource successful");
else
     System.out.println("openSource unsuccessful");
```

where:

- userarg: contains permission-related parameters.
- ctx: contains the source plug-in context information.

## processSourceCommand( )

### Format

public byte[ ] processSourceCommand(byte[ ] [ ] ctx, String cmd, String args, byte[ ] [ ] result)

### Description

Calls the source plug-in in the database to execute a command. This method is used with user-written plug-ins only; this method raises an exception if used with the source plug-ins supplied by Oracle.

### Parameters

**ctx**
The source plug-in context information. See *Oracle interMedia User's Guide and Reference* for more information.

**cmd**
The command to be executed by the source plug-in.

**args**
The command arguments.

**result**
A byte array of the form [1][*n*] into which the result of the command execution is written.

### Return Value

This method returns the results of executing the command.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding processSourceCommand( ) method in the database.

### Examples

```
byte [ ] [ ] ctx = new byte[1][64];
String cmd;
String args;
byte [ ] [ ] result;
//assign a command value to cmd
//assign any arguments to args
byte[ ] commandResults = vidObj.processSourceCommand(ctx,cmd,
     args,result);
```

where:

- ctx: contains the source plug-in information.
- cmd: is the command to be run.
- args: contains any arguments required by the command.
- result: is the results of executing the command.

## processVideoCommand( )

### Format

public byte[ ] processVideoCommand(byte[ ] [ ] ctx, String cmd, String args, byte[ ] [ ] result)

### Description

Calls the format plug-in in the database to execute a command. This method is used with user-written format plug-ins only; this method raises an exception if used with the format plug-ins supplied by Oracle.

### Parameters

**ctx**
The format plug-in context information.

**cmd**
The command to be executed by the format plug-in.

**args**
The command arguments.

**result**
A byte array of the form [1][*n*] into which the result of the command execution is written.

### Return Value

This method returns the results of executing the command.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding processVideoCommand( ) method in the database.

### Examples

```
byte [ ] [ ] ctx = new byte[1][64]
String cmd;
String args;
```

```
byte [ ] [ ] result;
//assign a command value to cmd
//assign any arguments to args
byte[ ] commandResults = vidObj.processVideoCommand(ctx,cmd,
     args,result);
```

where:

- ctx: contains the format plug-in information.
- cmd: is the command to be run.
- args: contains any arguments required by the command.
- result: is the results of executing the command.

## readFromSource( )

### Format

public int readFromSource(byte[ ] [ ] ctx, int startpos, int numbytes, byte[ ] [ ] buffer)

### Description

Reads data from the data source. This method reads the specified number of bytes into the application buffer from the data source starting at the specified position in the data source.

Not all source plug-ins require that the data source be opened before it can be read. However, to ensure that an application will work with any current or future source plug-ins, call the openSource( ) method before calling this method.

### Parameters

**ctx**
The source plug-in context information. See *Oracle interMedia User's Guide and Reference* for more information.

**startpos**
The start position in the data source.

**numbytes**
The number of bytes to be read from the data source.

**buffer**
A byte array of the form [1][*n*], where *n* is greater than or equal to numbytes.

### Return Value

This method returns the number of bytes read, as an integer.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding readFromSource( ) method in the database.

## Examples

```
byte [ ] [ ] ctx = new byte[1][64];
byte [ ] [ ] buffer = new byte[12];
int i = vidObj.readFromSource(ctx,0,12,buffer);
```

where:

- ctx: contains the source plug-in context information.
- 0: is the position to begin reading from the localData field.
- 12: is the number of bytes to be read.
- buffer: is the location to which the data will be read.

## setBitRate( )

### Format

public void setBitRate(int bitRate)

### Description

Sets the value of the bitRate attribute.

The setProperties( ) method sets this attribute automatically for certain video formats; use this method only if you are not using the setProperties( ) method. This method sets only the attribute value; it does not modify the video data itself.

### Parameters

**bitRate**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the bitRate attribute.

### Examples

```
vidObj.setBitRate(1500);
```

where:

- 1500: is the value to be set, in bits per second, in the bitRate attribute.

## setComments( )

### Format

public void setComments(oracle.sql.CLOB comments)

### Description

Sets the value of the comments attribute.

The comments attribute is reserved for use by *inter*Media. You can set your own value, but it could be overwritten by Oracle *inter*Media Annotator or by the setProperties( ) method.

### Parameters

**comments**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the comments attribute.

### Examples

```
vidObj.setComments(commentsData);
```

where:

- commentsData: is a CLOB that contains data to be set in the comments attribute.

## setCompressionType( )

### Format

public void setCompressionType(String compressionType)

### Description

Sets the value of the compressionType attribute.

The setProperties( ) method sets this attribute automatically for certain video formats; use this method only if you are not using the setProperties( ) method. This method sets only the attribute value; it does not modify the video data itself.

### Parameters

**compressionType**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the compressionType attribute.

### Examples

```
vidObj.setCompressionType("Cinepak");
```

where:

- Cinepak: is the value to be set in the compressionType attribute.

## setDescription( )

### Format

public void setDescription(String description)

### Description

Sets the value of the description attribute.

### Parameters

**description**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the description attribute.

### Examples

```
vidObj.setDescription("My video file");
```

where:

- My video file: is the value to be set in the description attribute.

## setFormat( )

### Format

public void setFormat(String format)

### Description

Sets the value of the format attribute.

The format attribute determines which format plug-in is used to handle calls to methods that operate on the video data. In particular, the setProperties( ) method uses the format attribute to determine which format plug-in to call to parse the video data properties. See the setProperties( ) method for more information on how to initialize the format attribute before calling the setProperties( ) method, and for information on how the setProperties( ) method in the default, Oracle-supplied plug-in, sets the value of the format attribute. Calling the setFormat( ) method sets only the attribute value; it does not modify the video data itself.

### Parameters

**format**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the format attribute.

### Examples

```
vidObj.setFormat("MOOV");
```

where:

- MOOV: is the value to be set in the format attribute.

## setFrameRate( )

### Format

public void setFrameRate(int frameRate)

### Description

Sets the value of the frameRate attribute.

The setProperties( ) method sets this attribute automatically for certain video formats; use this method only if you are not using the setProperties( ) method. This method sets only the attribute value; it does not modify the video data itself.

### Parameters

**frameRate**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the frameRate attribute.

### Examples

```
vidObj.setFrameRate(5);
```

where:

- 5: is the value to be set, in frames per second, in the frameRate attribute.

## setFrameResolution( )

### Format

public void setFrameResolution(int frameResolution)

### Description

Sets the value of the frameResolution attribute.

The setProperties( ) method sets this attribute automatically for certain video formats; use this method only if you are not using the setProperties( ) method. This method sets only the attribute value; it does not modify the video data itself.

### Parameters

**frameResolution**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the frameResolution attribute.

### Examples

```
vidObj.setFrameResolution(4);
```

where:

- 4: is the value to be set in the frameResolution attribute.

## setHeight( )

### Format

public void setHeight(int height)

### Description

Sets the value of the height attribute.

The setProperties( ) method sets this attribute automatically for certain video formats; use this method only if you are not using the setProperties( ) method. This method sets only the attribute value; it does not modify the video data itself.

### Parameters

**height**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the height attribute.

### Examples

```
vidObj.setHeight(24);
```

where:

- 24: is the value to be set, in pixels, in the height attribute.

## setKnownAttributes( )

### Format

```
public void setKnownAttributes(String format, int width, int height, int frameResolution,
                               int frameRate, int videoDuration, int numberOfFrames,
                               String compressionType, int numberOfColors,
                                   int bitRate)
```

### Description

Sets the values of the known attributes of the OrdVideo Java object.

The setProperties( ) method sets the values of the following attributes automatically for certain video formats: format, width, height, frameResolution, frameRate, videoDuration, numberOfFrames, compressionType, numberOfColors, and bitRate. Use this method only if you are not using the setProperties( ) method. This method sets only the specified attribute values; it does not modify the video data itself.

### Parameters

**format**
The new attribute value, as a String.

**width**
The new attribute value, as an integer.

**height**
The new attribute value, as an integer.

**frameResolution**
The new attribute value, as an integer.

**frameRate**
The new attribute value, as an integer.

**videoDuration**
The new attribute value, as an integer.

**numberOfFrames**
The new attribute value, as an integer.

**compressionType**
The new attribute value, as a String.

**numberOfColors**
The new attribute value, as an integer.

**bitRate**
The new attribute value, as an integer.

## Return Value

None.

## Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding setKnownAttributes( ) method in the database.

## Examples

```
vidObj.setKnownAttributes("MOOV",1,2,4,5,20,8,"Cinepak",256,1500);
```

where:

- MOOV: is the value to be set in the format attribute.
- 1: is the value to be set, in pixels, in the width attribute.
- 2: is the value to be set, in pixels, in the height attribute.
- 4: is the value to be set in the frameResolution attribute.
- 5: is the value to be set, in frames per second, in the frameRate attribute.
- 20: is the value to be set in the videoDuration attribute.
- 8: is the value to be set in the numberOf Frames attribute.
- Cinepak: is the value to be set in the compressionType attribute.
- 256: is the value to be set in the numberOfColors attribute.
- 1500: is the value to be set, in bits per second, in the bitRate attribute.

## setLocal( )

### Format

public void setLocal( )

### Description

Sets the value of the local attribute to indicate that the video data is stored locally in the database in a BLOB specified by the localData attribute.

### Parameters

None.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the localData attribute.

### Examples

```
vidObj.setLocal( );
```

## setMimeType( )

### Format

public void setMimeType(String mimeType)

### Description

Sets the value of the mimeType attribute.

The setProperties( ) method sets this attribute automatically for certain video formats; use this method only if you are not using the setProperties( ) method. This method sets only the attribute value; it does not modify the video data itself.

### Parameters

**mimeType**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the mimeType attribute.

### Examples

```
vidObj.setMimeType("video/avi");
```

where:

- video/avi: is the MIME type to be set.

## setNumberOfColors( )

### Format

public void setNumberOfColors(int numberOfColors)

### Description

Sets the value of the numberOfColors attribute.

The setProperties( ) method sets this attribute automatically for certain video formats; use this method only if you are not using the setProperties( ) method. This method sets only the attribute value; it does not modify the video data itself.

### Parameters

**numberOfColors**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the numberOfColors attribute.

### Examples

```
vidObj.setNumberOfColors(256);
```

where:

- 256: is the value to be set in the numberOfColors attribute.

## setNumberOfFrames( )

### Format

public void setNumberOfFrames(int numberOfFrames)

### Description

Sets the value of the numberOfFrames attribute.

The setProperties( ) method sets this attribute automatically for certain video formats; use this method only if you are not using the setProperties( ) method. This method sets only the attribute value; it does not modify the video data itself.

### Parameters

**numberOfFrames**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the numberOfFrames attribute.

### Examples

```
vidObj.setNumberOfFrames(8);
```

where:

- 8: is the the value to be set in the numberOfFrames attribute.

## setProperties(byte[ ][ ])

### Format

public void setProperties(byte[ ] [ ] ctx)

### Description

Parses the video data properties and sets the values of the attributes in the OrdVideo Java object. This method sets the values of the format, mimeType, width, height, frameResolution, frameRate, videoDuration, numberOfFrames, compressionType, numberOfColors, and bitRate attributes. An attribute is set to null if the corresponding property cannot be extracted for a specific video format. This method throws a SQLException error if the video format is not recognized.

The format attribute determines which format plug-in is used to parse the video data properties. If the format attribute is null when the setProperties( ) method is called, then the default, Oracle-supplied, format plug-in is used to parse the video data properties and fill in various attributes, including the actual video data format, for supported video formats. See *Oracle interMedia User's Guide and Reference* for information on the video formats supported by the Oracle-supplied format plug-ins. Note that the ORDVideo.init methods in the database always set the value of the format attribute to null. If the format attribute is not null, then the format plug-in specified by the format attribute will be called when the setProperties( ) method is called.

### Parameters

**ctx**
The format plug-in context information.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding setProperties( ) method in the database.

## Examples

```
byte [ ] [ ] ctx = new byte[1][64];
vidObj.setProperties(ctx);
```

where:

- ctx: contains the format plug-in context information.

## setProperties(byte[ ][ ], boolean)

### Format

public void setProperties(byte[ ] [ ] ctx, boolean setComments)

### Description

Parses the video data properties, sets the values of the attributes in the OrdVideo Java object, and optionally populates the CLOB specified by the comments attribute. This method sets the values of the format, mimeType, width, height, frameResolution, frameRate, videoDuration, numberOfFrames, compressionType, numberOfColors, and bitRate attributes. An attribute is set to null if the corresponding property cannot be extracted for a specific video format. If the setComments parameter is true, this method also populates the CLOB specified by the comments attribute with all extracted properties in XML form. If the setComments parameter is false, the comments attribute is not modified. This method throws a SQLException error if the video format is not recognized.

The format attribute determines which format plug-in is used to parse the video data properties. If the format attribute is null when the setProperties( ) method is called, then the default, Oracle-supplied, format plug-in is used to parse the video data properties and fill in various attributes, including the actual video data format, for supported video formats. See *Oracle interMedia User's Guide and Reference* for information on the video formats supported by the Oracle-supplied format plug-ins. Note that the ORDVideo.init methods in the database always set the value of the format attribute to null. If the format attribute is not null, then the format plug-in specified by the format attribute will be called when the setProperties( ) method is called.

### Parameters

**ctx**
The format plug-in context information.

**setComments**
A Boolean value that specifies whether or not to populate the CLOB specified by the comments attribute.

## Return Value

None.

## Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding setProperties( ) method in the database.

## Examples

```
byte [ ] [ ] ctx = new byte[1][64];
vidObj.setProperties(ctx,true);
```

where:

- ctx: contains the format plug-in context information.
- true: indicates that the comments field will be populated.

## setSource( )

### Format

public void setSource(String srcType, String srcLocation, String srcName)

### Description

Sets the values of the srcType, srcLocation, and srcName attributes.

### Parameters

**srcType**
The type of the source.

**srcLocation**
The location of the source.

**srcName**
The name of the source.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the srcType, srcLocation, or srcName attributes.

### Examples

```
vidObj.setSource("LOCAL","VIDEODIR","video.dat");
```

where:

- LOCAL: is the source type.
- VIDEODIR: is the source location.
- video.dat: is the source name.

## setUpdateTime( )

### Format

public void setUpdateTime(java.sql.Timestamp currentTime)

### Description

Sets the value of the updateTime attribute. This method sets the value of the updateTime attribute to the specified time, or to the database server's current SYSDATE time if currentTime is specified as null.

### Parameters

**currentTime**
The update time, or the null value, used to set the value of the updateTime attribute to the database server's current SYSDATE time.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding setUpdateTime( ) method in the database.

### Examples

```
vidObj.setUpdateTime(null);
```

## setVideoDuration( )

### Format

public void setVideoDuration(int videoDuration)

### Description

Sets the value of the videoDuration attribute.

The setProperties( ) method sets this attribute automatically for certain video formats; use this method only if you are not using the setProperties( ) method. This method sets only the attribute value; it does not modify the video data itself.

### Parameters

**videoDuration**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the videoDuration attribute.

### Examples

```
vidObj.setVideoDuration(20);
```

where:

- 20: is the value to be set in the videoDuration attribute.

## setWidth( )

### Format

public void setWidth(int width)

### Description

Sets the value of the width attribute.

The setProperties( ) method sets this attribute automatically for certain video formats; use this method only if you are not using the setProperties( ) method. This method sets only the attribute value; it does not modify the video data itself.

### Parameters

**width**
The new attribute value.

### Return Value

None.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs accessing the width attribute.

### Examples

```
vidObj.setWidth(24);
```

where:

- 24: is the value to be set,  in pixels, in the width attribute.

## trimSource( )

### Format

```
public int trimSource(byte[ ] [ ] ctx, int newLen)
```

### Description

Trims the data to the specified length.

Not all source plug-ins support trim operations. For example, applications can trim the data stored in a BLOB specified by the localData attribute; however, the "file" and "http" data source types do not support write access, and so do not support this method. Furthermore, those source plug-ins that do support write access may not support the trim operation.

Not all source plug-ins require that the data source be opened before it can be modified. However, to ensure that an application will work with any current or future source plug-ins, call the openSource( ) method before calling this method.

### Parameters

**ctx**
The source plug-in context information. See *Oracle interMedia User's Guide and Reference* for more information.

**newLen**
The length to which the data is to be trimmed.

### Return Value

This method returns the status as an integer, where zero indicates success and a non-zero value indicates a failure code specific to the source plug-in.

### Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding trimSource( ) method in the database.

## Examples

```
byte [ ] [ ] ctx = new byte[1][64];
int i = vidObj.trimSource(ctx,10);
if (i == 0)
     System.out.println("trimSource successful");
else
     System.out.println("trimSource unsuccessful");
```

where:

- ctx: contains the source plug-in context information.
- 10: is the new length of the source.

## writeToSource( )

### Format

```
public int writeToSource(byte[ ] [ ] ctx, int startpos, int numbytes, byte[ ] buffer)
```

### Description

Writes data to the data source. This method writes the specified number of bytes from the application buffer to the data source starting at the specified position in the data source.

Not all source plug-ins support write operations. For example, applications can write to a BLOB specified by the localData attribute; however, the "file" and "http" data source types do not support write access, and so do not support this method. Furthermore, those source plug-ins that do support write access may support only sequential write access, and may not support write access to arbitrary starting positions within the data source.

Not all source plug-ins require that the data source be opened before it can be written. However, to ensure that an application will work with any current or future source plug-ins, call the openSource( ) method before calling this method.

### Parameters

**ctx**
The source plug-in context information. See *Oracle interMedia User's Guide and Reference* for more information.

**startpos**
The start position in the data source.

**numbytes**
The number of bytes to be written to the data source.

**buffer**
A byte array containing the data to be written.

### Return Value

This method returns the number of bytes written, as an integer.

## Exceptions

java.sql.SQLException

This exception is thrown if an error occurs executing the corresponding writeToSource( ) method in the database.

## Examples

```
byte [ ] [ ] ctx= new byte[1][64];
byte[ ] data = new byte[20];
//populate data with 20 bytes of content
int i = vidObj.writeToSource(ctx,1,20,data);
```

where:

- ctx: contains the source plug-in context information.
- 1: is the position in the comments field where writing will begin.
- 20: is the number of bytes to be written.
- data: contains the content to be written.

# 9 Java Classes for Servlets and JSP Reference Information

Oracle *inter*Media Java Classes for servlets and JavaServer Pages (JSP) facilitates retrieving and uploading media data from and to an Oracle database.

The OrdHttpResponseHandler class facilitates the retrieval of media data from an Oracle database, and its delivery to a browser or other HTTP client from a Java servlet. The OrdHttpJspResponseHandler class provides the same features for JSP.

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 Java servlets or JSP. The handling of uploaded files is facilitated by the OrdHttpUploadFile class, which provides an easy-to-use API that applications call to load image, audio, and video data into a database.

## 9.1 Prerequisites

You will need to include the following import statements in your Java file in order to execute *inter*Media methods:

```
import java.sql.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import oracle.jdbc.driver.*;
import oracle.sql.*;
import oracle.ord.im.*;
```

## OrdHttpResponseHandler Reference Information

This section presents reference information on the methods of the OrdHttpResponseHandler class.

The OrdHttpResponseHandler class facilitates the retrieval of media data from an Oracle database, and its delivery to a browser or other HTTP client from a Java servlet.

An *inter*Media Java object, such as an OrdImage object, is not dependent on the JDBC statement or result set from which it was obtained. However, an *inter*Media Java object is dependent on the JDBC connection it is using and on which the SQL statement was executed, or from which the result set was obtained. Therefore, having obtained an *inter*Media Java object from the database, an application must not release the JDBC connection before delivering the media data to the browser.

This class contains the following field:

- public static final int DEFAULT_BUFFER_SIZE

  The OrdHttpResponseHandler class uses a default buffer size of 32768 to retrieve LOB data from the database and deliver it to the client. You can override this value with the setBufferSize( ) method.

The following example shows how to use the OrdHttpResponseHandler class to retrieve an image from a database and deliver it to a browser:

```
PreparedStatement stmt = conn.prepareStatement("select photo from photo_album
     where id = ?");
stmt.setString(1, request.getParameter("photo_id"));
OracleResultSet rset = (OracleResultSet)stmt.executeQuery( );
if (rset.next( )){
     OrdImage media = (OrdImage)rset.getCustomDatum(1, OrdImage.getFactory( ));
     OrdHttpResponseHandler handler = new OrdHttpResponseHandler(request,
          response);
     handler.sendImage(media);
}
else{
     response.setStatus(response.SC_NOT_FOUND);
}
rset.close( );
stmt.close( );
```

### A Note on the Use of Charsets Other Than ISO-8859-1 (Latin-1)

If you wish to retrieve from an OrdDoc object a text-based document with a character set (charset) other than ISO-8859-1 (also called Latin-1), and deliver that document to a browser, your application must specify the charset name in the HTTP Content-Type header.

If the charset specification is included in the MIME type attribute in the OrdDoc object, then your application needs to call only the sendDoc( ) method to retrieve the document and deliver it to the browser. For example, an HTML page that is written in Japanese might be stored in the OrdDoc object with a MIME type of `text/html; charset=Shift_JIS`. In this case, calling the sendDoc( ) method will send the appropriate Content-Type header, allowing the browser to display the page correctly.

However, if the MIME type in the OrdDoc object does not include the charset specification, then you must call one of the sendResponse( ) methods and specify the MIME type explicitly. For example, if the MIME type of an HTML page written in Japanese is stored in the OrdDoc object as `text/html`, and the charset name is specified in a separate column, then the application must append the charset specification to the MIME type before calling a sendResponse( ) method. For example:

```
OraclePreparedStatement stmt = (OraclePreparedStatement)conn.prepareStatement(
     "select doc, charset from documents where id = ?");
stmt.setString(1, request.getParameter("id");
OracleResultSet rset = (OracleResultSet)stmt.executeQuery( );
if (rset.next( )){
     OrdDoc doc = (OrdDoc)rset.getCustomDatum(1, OrdDoc.getFactory( ));
     String charset = rset.getString(2);
     String mimeType = doc.getMimeType( ) + "; charset=" + charset;
     OrdHttpResponseHandler handler = new OrdHttpResponseHandler(request,
          response);
     handler.sendResponse(mimeType, doc.getContentLength( ), doc.getContent( ),
          doc.getUpdateTime( ));
}
else{
     response.setStatus(response.SC_NOT_FOUND);
}
rset.close( );
stmt.close( );
```

## OrdHttpResponseHandler( )

### Format

public OrdHttpResponseHandler( )

### Description

Creates an OrdHttpResponseHandler object to handle the response to a multimedia retrieval request. The application must subsequently specify the HttpServletResponse object by calling the setServletResponse( ) method, and can optionally specify the HttpServletRequest object by calling the setServletRequest( ) method.

### Parameters

None.

### Return Value

None.

### Exceptions

None.

### Examples

See setServletRequest( ) for an example of this method.

## OrdHttpResponseHandler(HttpServletRequest,HttpServletResponse)

### Format

```
public OrdHttpResponseHandler(javax.servlet.http.HttpServletRequest request,
                              javax.servlet.http.HttpServletResponse response)
```

### Description

Creates an OrdHttpResponseHandler object to handle the response to a multimedia retrieval request and specifies the HttpServletRequest and HttpServletResponse objects for the response handler.

### Parameters

**request**
An object of type HttpServletRequest.

**response**
An object of type HttpServletResponse.

### Return Value

None.

### Exceptions

None.

### Examples

See sendAudio( ) for an example of this method.

## sendAudio( )

### Format

public void sendAudio(oracle.ord.im.OrdAudio media)

### Description

Retrieves an audio clip from an OrdAudio object and delivers it to the browser.

This method supports browser content caching by supporting the If-Modified-Since and Last-Modified headers.

### Parameters

**media**
An object of type oracle.ord.im.OrdAudio.

### Return Value

None.

### Exceptions

java.lang.IllegalStateException

This exception is thrown if HttpServletRequest or HttpServletResponse has not been specified.

OrdHttpResponseException

This exception is thrown if the source type is not recognized.

javax.servlet.ServletException

This exception is thrown if an error occurs accessing the binary output stream.

java.sql.SQLException

This exception is thrown if an error occurs obtaining an InputStream object to read the media data.

java.io.IOException

This exception is thrown if an error occurs reading the media data.

## Examples

```
OraclePreparedStatement stmt = (OraclePreparedStatement)
     conn.prepareStatement("select media from sounds where id = ?");
stmt.setString(1, request.getParameter("id"));
OracleResultSet rset = (OracleResultSet)stmt.executeQuery( );
if (rset.next( )){
     OrdAudio media = (OrdAudio)rset.getCustomDatum(1, OrdAudio.getFactory( ));
     OrdHttpResponseHandler handler = new OrdHttpResponseHandler
          (request, response);
     handler.sendAudio(media);
     return;
}
else{
     response.setStatus(response.SC_NOT_FOUND);
}
```

where:

- media: represents an object of type oracle.ord.im.OrdAudio.

## sendDoc( )

### Format

public void sendDoc(oracle.ord.im.OrdDoc media)

### Description

Retrieves media data from an OrdDoc object and delivers it to the browser.

This method supports browser content caching by supporting the If-Modified-Since and Last-Modified headers.

### Parameters

**media**
An object of type oracle.ord.im.OrdDoc.

### Return Value

None.

### Exceptions

java.lang.IllegalStateException

This exception is thrown if HttpServletRequest or HttpServletResponse has not been specified.

OrdHttpResponseException

This exception is thrown if the source type is not recognized.

javax.servlet.ServletException

This exception is thrown if an error occurs accessing the binary output stream.

java.sql.SQLException

This exception is thrown if an error occurs obtaining an InputStream object to read the media data.

java.io.IOException

This exception is thrown if an error occurs reading the media data.

## Examples

```
OraclePreparedStatement stmt = (OraclePreparedStatement)
     conn.prepareStatement("select media from documents where id = ?");
stmt.setString(1, request.getParameter("id"));
OracleResultSet rset = (OracleResultSet)stmt.executeQuery( );
if (rset.next( )){
     OrdDoc media = (OrdDoc)rset.getCustomDatum(1, OrdDoc.getFactory( ));
     OrdHttpResponseHandler handler = new OrdHttpResponseHandler
          (request, response);
     handler.sendDoc(media);
     return;
}
else{
     response.setStatus(response.SC_NOT_FOUND);
}
```

where:

- media: represents an object of type oracle.ord.im.OrdDoc.

## sendImage( )

### Format

public void sendImage(oracle.ord.im.OrdImage media)

### Description

Retrieves an image from an OrdImage object and delivers it to the browser.

This method supports browser content caching by supporting the If-Modified-Since and Last-Modified headers.

### Parameters

**media**
An object of type oracle.ord.im.OrdImage.

### Return Value

None.

### Exceptions

java.lang.IllegalStateException

This exception is thrown if HttpServletRequest or HttpServletResponse has not been specified.

OrdHttpResponseException

This exception is thrown if the source type is not recognized.

javax.servlet.ServletException

This exception is thrown if an error occurs accessing the binary output stream.

java.sql.SQLException

This exception is thrown if an error occurs obtaining an InputStream object to read the media data.

java.io.IOException

This exception is thrown if an error occurs reading the media data.

## Examples

```
OraclePreparedStatement stmt = (OraclePreparedStatement)
     conn.prepareStatement("select media from photos where id = ?");
stmt.setString(1, request.getParameter("id"));
OracleResultSet rset = (OracleResultSet)stmt.executeQuery( );
if (rset.next( )){
     OrdImage media = (OrdImage)rset.getCustomDatum(1, OrdImage.getFactory( ));
     OrdHttpResponseHandler handler = new OrdHttpResponseHandler
          (request, response);
     handler.sendImage(media);
     return;
}
else{
     response.setStatus(response.SC_NOT_FOUND);
}
```

where:

- media: represents an object of type oracle.ord.im.OrdImage.

## sendResponse(String,int,BFILE,Timestamp)

### Format

```
public void sendResponse(String contentType, int length, oracle.sql.BFILE bfile,
                         java.sql.Timestamp lastModified)
```

### Description

Builds the HTTP response header, then retrieves the contents of the BFILE from the database and delivers it to the browser.

This method supports browser content caching by supporting the If-Modified-Since and Last-Modified headers.

### Parameters

**contentType**
A string that specifies the MIME type of the content.

**length**
An integer that specifies the length of the data.

**bfile**
An oracle.sql.BFILE from which the media data is retrieved.

**lastModified**
A java.sql.Timestamp object that specifies the date and time when the data was last modified, or null if no last modified date and time are available.

### Return Value

None.

### Exceptions

java.lang.IllegalStateException

This exception is thrown if HttpServletRequest or HttpServletResponse has not been specified.

java.lang.IllegalArgumentException

This exception is thrown if the length is negative.

javax.servlet.ServletException

This exception is thrown if an error occurs accessing the binary output stream.

java.sql.SQLException

This exception is thrown if an error occurs obtaining an InputStream object to read the media data.

java.io.IOException

This exception is thrown if an error occurs reading the media data.

## Examples

```
OraclePreparedStatement stmt = (OraclePreparedStatement)conn.prepareStatement
     ("select mimetype, len, doc, updatetime from docfiles where id = ?");
stmt.setString( 1, request.getParameter("id"));
OracleResultSet rset = (OracleResultSet)stmt.executeQuery( );
if (rset.next( )){
     String mimeType = rset.getString(1);
     int len = rset.getInt(2);
     BFILE bfile = rset.getBFILE(3);
     Timestamp updateTime = rset.getTimestamp(4);
     OrdHttpResponseHandler handler = new OrdHttpResponseHandler
          (request, response);
     handler.sendResponse(mimeType, len, bfile, updateTime);
     return;
}
else{
     response.setStatus(response.SC_NOT_FOUND);
}
```

## sendResponse(String,int,BLOB,Timestamp)

### Format

```
public void sendResponse(String contentType, int length, oracle.sql.BLOB blob,
                         java.sql.Timestamp lastModified)
```

### Description

Builds the HTTP response header, then retrieves the contents of the BLOB from the database and delivers it to the browser.

This method supports browser content caching by supporting the If-Modified-Since and Last-Modified headers.

### Parameters

**contentType**
A string that specifies the MIME type of the content.

**length**
An integer that specifies the length of the data.

**blob**
An oracle.sql.BLOB from which the media data is retrieved.

**lastModified**
A java.sql.Timestamp object that specifies the date and time when the data was last modified, or null if no last modified date and time are available.

### Return Value

None.

### Exceptions

java.lang.IllegalStateException

This exception is thrown if HttpServletRequest or HttpServletResponse has not been specified.

java.lang.IllegalArgumentException

This exception is thrown if the length is negative.

javax.servlet.ServletException

This exception is thrown if an error occurs accessing the binary output stream.

java.sql.SQLException

This exception is thrown if an error occurs obtaining an InputStream object to read the media data.

java.io.IOException

This exception is thrown if an error occurs reading the media data.

## Examples

```
OraclePreparedStatement stmt = (OraclePreparedStatement)conn.prepareStatement
     ("select mimetype, len, doc, updatetime from docblobs where id = ?");
stmt.setString(1, request.getParameter("id"));
OracleResultSet rset = (OracleResultSet)stmt.executeQuery( );
if (rset.next( )){
     String mimeType = rset.getString(1);
     int len = rset.getInt(2);
     BLOB blob = rset.getBLOB(3);
     Timestamp updateTime = rset.getTimestamp(4);
     OrdHttpResponseHandler handler = new OrdHttpResponseHandler
          (request, response);
     handler.sendResponse(mimeType, len, blob, updateTime);
     return;
}
else{
     response.setStatus(response.SC_NOT_FOUND);
}
```

## sendResponse(String,int,InputStream,Timestamp)

### Format

```
public void sendResponse(String contentType, int length, java.io.InputStream in,
                         java.sql.Timestamp lastModified)
```

### Description

Builds the HTTP response header, then retrieves the contents of the InputStream object and delivers it to the browser.

This method supports browser content caching by supporting the If-Modified-Since and Last-Modified headers.

### Parameters

**contentType**
A String that specifies the MIME type of the content.

**length**
An integer that specifies the length of the data.

**in**
An InputStream object from which the media data is retrieved.

**lastModified**
A java.sql.Timestamp object that specifies the date and time when the data was last modified, or null if no last modified date and time are available.

### Return Value

None.

### Exceptions

java.lang.IllegalStateException

This exception is thrown if HttpServletRequest or HttpServletResponse has not been specified.

java.lang.IllegalArgumentException

This exception is thrown if the length is negative.

javax.servlet.ServletException

This exception is thrown if an error occurs accessing the binary output stream.

java.io.IOException

This exception is thrown if an error occurs reading the media data.

## Examples

```
OraclePreparedStatement stmt = (OraclePreparedStatement)conn.prepareStatement
     ("select mimetype, len, doc, updatetime from docblobs where id = ?");
stmt.setString(1, request.getParameter("id"));
OracleResultSet rset = (OracleResultSet)stmt.executeQuery( );
if ( rset.next( ) ){
     String mimeType = rset.getString(1);
     int len = rset.getInt(2);
     BLOB blob = rset.getBLOB(3);
     Timestamp updateTime = rset.getTimestamp(4);
     InputStream blobInputStream = blob.getBinaryStream( );
     OrdHttpResponseHandler handler = new OrdHttpResponseHandler
          (request, response);
     handler.sendResponse(mimeType, len, blobInputStream, updateTime);
     return;
}
else{
     response.setStatus(response.SC_NOT_FOUND);
}
```

## sendResponseBody(int,BFILE)

### Format

public void sendResponseBody(int length, oracle.sql.BFILE bfile)

### Description

Retrieves the contents of a BFILE from the database and delivers it as the response body to the browser. The caller is responsible for building the HTTP header.

### Parameters

**length**
An integer that specifies the length of the data.

**bfile**
An oracle.sql.BFILE from which the content is retrieved.

### Return Value

None.

### Exceptions

java.lang.IllegalStateException

This exception is thrown if HttpServletRequest has not been specified.

java.lang.IllegalArgumentException

This exception is thrown if the length is negative.

javax.servlet.ServletException

This exception is thrown if an error occurs accessing the binary output stream.

java.sql.SQLException

This exception is thrown if an error occurs obtaining an InputStream object to read the media data.

java.io.IOException

This exception is thrown if an error occurs reading the media data.

## Examples

```
OraclePreparedStatement stmt = (OraclePreparedStatement)conn.prepareStatement
     ("select mimetype, len, doc from docfiles where id = ?");
stmt.setString(1, request.getParameter("id"));
OracleResultSet rset = (OracleResultSet)stmt.executeQuery( );
if (rset.next( )){
     String mimeType = rset.getString(1);
     int len = rset.getInt(2);
     BFILE bfile = rset.getBFILE(3);
     response.setContentLength(len);
     response.setContentType(mimeType);
     OrdHttpResponseHandler handler = new OrdHttpResponseHandler( );
     handler.setServletResponse(response);
     handler.sendResponseBody(len, bfile);
     return;
}
else{
     response.setStatus(response.SC_NOT_FOUND);
}
```

## sendResponseBody(int,BLOB)

### Format

```
public void sendResponseBody(int length, oracle.sql.BLOB blob)
```

### Description

Retrieves the contents of a BLOB from the database and delivers it as the response body to the browser. The caller is responsible for building the HTTP header.

### Parameters

**length**
An integer that specifies the length of the data.

**blob**
An oracle.sql.BLOB from which the content is retrieved.

### Return Value

None.

### Exceptions

java.lang.IllegalStateException

This exception is thrown if HttpServletRequest has not been specified.

java.lang.IllegalArgumentException

This exception is thrown if the length is negative.

javax.servlet.ServletException

This exception is thrown if an error occurs accessing the binary output stream.

java.sql.SQLException

This exception is thrown if an error occurs obtaining an InputStream object to read the media data.

java.io.IOException

This exception is thrown if an error occurs reading the media data.

## Examples

```
OraclePreparedStatement stmt = (OraclePreparedStatement)conn.prepareStatement
     ("select mimetype, len, doc from docblobs where id = ?");
stmt.setString( 1, request.getParameter("id"));
OracleResultSet rset = (OracleResultSet)stmt.executeQuery( );
if (rset.next( )){
     String mimeType = rset.getString(1);
     int len = rset.getInt(2);
     BLOB blob = rset.getBLOB(3);
     response.setContentLength(len);
     response.setContentType(mimeType);
     OrdHttpResponseHandler handler = new OrdHttpResponseHandler( );
     handler.setServletResponse(response);
     handler.sendResponseBody(len, blob);
     return;
}
else{
     response.setStatus(response.SC_NOT_FOUND);
}
```

## sendResponseBody(int,InputStream)

### Format

public void sendResponseBody(int length, java.io.InputStream in)

### Description

Retrieves the contents of the InputStream object and delivers it to the client. The caller is responsible for building the HTTP header.

### Parameters

**length**
An integer that specifies the length of the data.

**in**
An InputStream object from which the media data is retrieved.

### Return Value

None.

### Exceptions

java.lang.IllegalStateException

This exception is thrown if HttpServletRequest has not been specified.

java.lang.IllegalArgumentException

This exception is thrown if the length is negative.

javax.servlet.ServletException

This exception is thrown if an error occurs accessing the binary output stream.

java.io.IOException

This exception is thrown if an error occurs reading the media data.

### Examples

```
OraclePreparedStatement stmt = (OraclePreparedStatement)conn.prepareStatement
     ("select mimetype, len, doc from docblobs where id = ?");
stmt.setString(1, request.getParameter("id"));
```

```
OracleResultSet rset = (OracleResultSet)stmt.executeQuery( );
if (rset.next( )){
     String mimeType = rset.getString(1);
     int len = rset.getInt(2);
     BLOB blob = rset.getBLOB(3);
     response.setContentLength(len);
     response.setContentType(mimeType);
     InputStream blobInputStream = blob.getBinaryStream( );
     OrdHttpResponseHandler handler = new OrdHttpResponseHandler( );
     handler.setServletResponse(response);
     handler.sendResponseBody(len, blobInputStream);
     return;
}
else{
     response.setStatus(response.SC_NOT_FOUND);
}
```

## sendVideo( )

### Format

public void sendVideo(oracle.ord.im.OrdVideo media)

### Description

Retrieves a video clip from an OrdVideo object and delivers it to the browser.

This method supports browser content caching by supporting the If-Modified-Since and Last-Modified headers.

### Parameters

**media**
An object of type oracle.ord.im.OrdVideo.

### Return Value

None.

### Exceptions

java.lang.IllegalStateException

This exception is thrown if HttpServletRequest or HttpServletResponse has not been specified.

OrdHttpResponseException

This exception is thrown if the source type is not recognized.

javax.servlet.ServletException

This exception is thrown if an error occurs accessing the binary output stream.

java.sql.SQLException

This exception is thrown if an error occurs obtaining an InputStream object to read the media data.

java.io.IOException

This exception is thrown if an error occurs reading the media data.

## Examples

```
OraclePreparedStatement stmt = (OraclePreparedStatement)
     conn.prepareStatement("select video from movies where id = ?");
stmt.setString(1, request.getParameter("id"));
OracleResultSet rset = (OracleResultSet)stmt.executeQuery( );
if (rset.next( )){
     OrdVideo media = (OrdVideo)rset.getCustomDatum(1, OrdVideo.getFactory( ));
     OrdHttpResponseHandler handler = new OrdHttpResponseHandler
          (request, response);
     handler.sendVideo(media);
     return;
}
else{
     response.setStatus( response.SC_NOT_FOUND );
}
```

where:

- media: represents an object of type oracle.ord.im.OrdVideo.

## setBufferSize( )

### Format

public void setBufferSize(int bufferSize)

### Description

Sets the buffer size for LOB read and response write operations.

### Parameters

**bufferSize**
An integer that specifies the buffer size.

### Return Value

None.

### Exceptions

java.lang.IllegalArgumentException

This exception is thrown if the buffer size is negative or zero.

### Examples

```
OrdHttpResponseHandler handler = new OrdHttpResponseHandler(request, response);
handler.setBufferSize(16000);
```

## setServletRequest( )

### Format

public void setServletRequest(javax.servlet.http.HttpServletRequest request)

### Description

Specifies the HttpServletRequest object for this response handler. You must call this method if you did not specify the HttpServletRequest object in the constructor and you want to call any of the send methods other than the sendResponseBody methods. You do not need to call this method if you call only the sendResponseBody methods.

### Parameters

**request**
An object of type HttpServletRequest.

### Return Value

None.

### Exceptions

None.

### Examples

```
OrdHttpResponseHandler handler = new OrdHttpResponseHandler( );
handler.setServletRequest(request);
handler.setServletResponse(response);
```

## setServletResponse( )

### Format

```
public void setServletResponse(javax.servlet.http.HttpServletResponse response)
```

### Description

Specifies the HttpServletResponse object for this response handler. You must call this method before calling any of the send methods if you did not specify the HttpServletResponse object in the constructor.

### Parameters

**response**
An object of type HttpServletResponse.

### Return Value

None.

### Exceptions

None.

### Examples

See setServletRequest( ) for an example of this method.

## OrdHttpJspResponseHandler Reference Information

This section presents reference information on the methods of the OrdHttpJspResponseHandler class.

The OrdHttpJspResponseHandler class facilitates the retrieval of media data from an Oracle database, and its delivery to a browser or another HTTP client from a JSP.

This class inherits the DEFAULT_BUFFER_SIZE field from the OrdHttpResponseHandler class.

**An Important Note on JSP Engines**

JSP engines are not required to support access to the servlet binary output stream. Therefore, not all JSP engines support the delivery of media data using the OrdHttpJspResponseHandler class.

All media data stored in the database using the *inter*Media types, including text documents stored using the OrdDoc type, is stored using a binary LOB data type. Media data stored internally in the database is stored using a BLOB. Media data stored in an operating system file outside the database is stored using a BFILE. Therefore, all media data is delivered to the browser through the servlet binary output stream, using the ServletOutputStream class.

All the send methods in the OrdHttpJspResponseHandler class mirror the initial processing of the jsp:forward tag. Specifically, these send methods call the JspWriter.clear method to clear the output buffer of the page prior to obtaining the binary output stream. However, JSP engines are not required to support a call to the ServletResponse.getOutputStream method from within a JSP. A JSP engine that does not support this typically throws an IllegalStateException error from the getOutputStream method. However, the exact behavior is implementation-specific.

If your JSP engine does not support access to the binary output stream from within a JSP, then you must use a servlet to deliver media data. For example, perform one of the following operations:

- Use the jsp:forward tag to forward a multimedia retrieval request to a servlet.
- Construct multimedia retrieval URLs to retrieve the data directly from a servlet.

**An Important Note on Return Statements**

When delivering media data from a JSP, a return statement is always required following a call to any of the send methods of the OrdHttpJspResponseHandler class. The return statement is necessary to ensure that no other data is written to the output stream of the JSP following the media data.

An `if ( true ) { ... return; }` construct may be used to avoid the "statement not reachable" error that may result from the presence of additional code, generated by the JSP engine, at the end of a compiled page. This construct, which mirrors exactly the code produced by some JSP engines to handle the `<jsp:forward ... >` directive, is shown in the example provided later in this section.

---

**Note:** An *inter*Media Java object, such as an OrdImage object, is not dependent on the JDBC statement or result set from which it was obtained. However, an *inter*Media Java object is dependent on the JDBC connection it is using and on which the SQL statement was executed or from which the result set was obtained. Therefore, having obtained an *inter*Media Java object from the database, an application must not release the JDBC connection before delivering the media data to the browser.

---

All the send methods in this class call the JspWriter.clear method to clear the page's output buffer prior to delivering the image. Therefore, the page must use the buffered output model, which is the default.

The following example demonstrates how to use the OrdHttpJspResponseHandler class to retrieve an image from a database and deliver it to a browser. The return statement ensures that the trailing newline characters following the final end tag (represented by a percent mark and right-angle bracket, or %> ) are not transmitted to the browser as part of the image.

The `if ( true ) { ... return; }` construct is used to avoid the "statement not reachable" error that would otherwise be produced by this example due to the additional statements, generated by the JSP engine, at the end of the compiled page.

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

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

<%
```

```
        // Select the entry from the table using the ID request parameter,
        // then fetch the row.

        photos.selectRowById(request.getParameter("id"));
        if (!photos.fetch( )){
             response.setStatus(response.SC_NOT_FOUND);
             return;
        }

        // Set the page context for the retrieve request, then retrieve
        // the image from the database and deliver it to the browser. The
        // getImage( ) method returns an object of type oracle.ord.im.OrdImage.

        if (true){
             handler.setPageContext(pageContext);
             handler.sendImage(photos.getImage( ));
             return;
        }
   %>
```

## OrdHttpJspResponseHandler( )

### Format

```
public OrdHttpJspResponseHandler( )
```

### Description

Creates an OrdHttpJspResponseHandler object to handle the response to a multimedia retrieval request. The application must subsequently specify the PageContext object by calling the setPageContext( ) method.

### Parameters

None.

### Return Value

None.

### Exceptions

None.

### Examples

The default constructor is typically invoked implicitly when the OrdHttpJspResponseHandler class is used as a JavaBean. See setPageContext( ) for an example of the implicit use of the constructor.

## OrdHttpJspResponseHandler(PageContext)

### Format

public OrdHttpJspResponseHandler(javax.servlet.jsp.PageContext pageContext)

### Description

Creates an OrdHttpJspResponseHandler object to handle the response to a multimedia retrieval request, and specifies the PageContext object for the response handler.

### Parameters

**pageContext**
An object of type PageContext.

### Return Value

None.

### Exceptions

None.

### Examples

```
OrdHttpJspResponseHandler handler = new OrdHttpJspResponseHandler(pageContext);
```

## sendAudio( )

### Format

public void sendAudio(oracle.ord.im.OrdAudio media)

### Description

Retrieves an audio clip from an OrdAudio object and delivers it to the browser.

This method supports browser content caching by supporting the If-Modified-Since and Last-Modified headers. This method calls the JspWriter.clear method to clear the output buffer of the page prior to delivering the audio clip. Therefore, the page must use the buffered output model, which is the default.

### Parameters

**media**
An object of type oracle.ord.im.OrdAudio.

### Return Value

This method overrides the sendAudio( ) method in class OrdHttpResponseHandler.

### Exceptions

java.lang.IllegalStateException

This exception is thrown if PageContext has not been specified.

OrdHttpResponseException

This exception is thrown if the source type is not recognized.

javax.servlet.ServletException

This exception is thrown if an error occurs accessing the binary output stream.

java.sql.SQLException

This exception is thrown if an error occurs obtaining an InputStream object to read the media data.

java.io.IOException

This exception is thrown if an error occurs reading the media data.

## Examples

The OrdHttpJspResponseHandler.sendAudio( ) method extends the OrdHttpResponseHandler.sendAudio( ) method. See sendAudio( ) in OrdHttpResponseHandler Reference Information for an example of this method in the base class.

## sendDoc( )

### Format

public void sendDoc(oracle.ord.im.OrdDoc media)

### Description

Retrieves media data from an OrdDoc object and delivers it to the browser.

This method supports browser content caching by supporting the If-Modified-Since and Last-Modified headers. This method calls the JspWriter.clear method to clear the output buffer of the page prior to delivering the media. Therefore, the page must use the buffered output model, which is the default.

### Parameters

**media**
An object of type oracle.ord.im.OrdDoc.

### Return Value

This method overrides the sendDoc( ) method in class OrdHttpResponseHandler.

### Exceptions

java.lang.IllegalStateException

This exception is thrown if PageContext has not been specified.

OrdHttpResponseException

This exception is thrown if the source type is not recognized.

javax.servlet.ServletException

This exception is thrown if an error occurs accessing the binary output stream.

java.sql.SQLException

This exception is thrown if an error occurs obtaining an InputStream object to read the media data.

java.io.IOException

This exception is thrown if an error occurs reading the media data.

## Examples

The OrdHttpJspResponseHandler.sendDoc( ) method extends the OrdHttpResponseHandler.sendDoc( ) method. See sendDoc( ) in OrdHttpResponseHandler Reference Information for an example of this method in the base class.

## sendImage( )

### Format

public void sendImage(oracle.ord.im.OrdImage media)

### Description

Retrieves an image from an OrdImage object and delivers it to the browser.

This method supports browser content caching by supporting the If-Modified-Since and Last-Modified headers. This method calls the JspWriter.clear method to clear the output buffer of the page prior to delivering the image. Therefore, the page must use the buffered output model, which is the default.

### Parameters

**media**
An object of type oracle.ord.im.OrdImage.

### Return Value

This method overrides the sendImage( ) method in class OrdHttpResponseHandler.

### Exceptions

java.lang.IllegalStateException

This exception is thrown if PageContext has not been specified.

OrdHttpResponseException

This exception is thrown if the source type is not recognized.

javax.servlet.ServletException

This exception is thrown if an error occurs accessing the binary output stream.

java.sql.SQLException

This exception is thrown if an error occurs obtaining an InputStream object to read the media data.

java.io.IOException

This exception is thrown if an error occurs reading the media data.

## Examples

The OrdHttpJspResponseHandler.sendImage( ) method extends the OrdHttpResponseHandler.sendImage( ) method. See sendImage( ) in OrdHttpResponseHandler Reference Information for an example of this method in the base class.

## sendResponse(String,int,BFILE,Timestamp)

### Format

```
public void sendResponse(String contentType, int length, oracle.sql.BFILE bfile,
                         java.sql.Timestamp lastModified)
```

### Description

Builds the HTTP response header, then retrieves the contents of the BFILE from the database and delivers it to the browser.

This method supports browser content caching by supporting the If-Modified-Since and Last-Modified headers. This method calls the JspWriter.clear method to clear the output buffer of the page prior to delivering the image. Therefore, the page must use the buffered output model, which is the default.

### Parameters

**contentType**
A String that specifies the MIME type of the content.

**length**
An integer that specifies the length of the data.

**bfile**
An object of type oracle.sql.BFILE.

**lastModified**
A java.sql.Timestamp object that specifies the date and time when the data was last modified, or null if no last modified date and time are available.

### Return Value

This method overrides the sendResponse( ) method in class OrdHttpResponseHandler.

### Exceptions

java.lang.IllegalStateException

This exception is thrown if PageContext has not been specified.

javax.servlet.ServletException

This exception is thrown if an error occurs accessing the binary output stream.

java.sql.SQLException

This exception is thrown if an error occurs obtaining an InputStream object to read the media data.

java.io.IOException

This exception is thrown if an error occurs reading the media data.

java.lang.IllegalArgumentException

This exception is thrown if the length is negative.

## Examples

The OrdHttpJspResponseHandler.sendResponse(String, int, BFILE, Timestamp) method extends the OrdHttpResponseHandler.sendResponse(String, int, BFILE, Timestamp) method. See sendResponse(String,int,BFILE,Timestamp) in OrdHttpResponseHandler Reference Information for an example of this method in the base class.

## sendResponse(String,int,BLOB,Timestamp)

### Format

```
public void sendResponse(String contentType, int length, oracle.sql.BLOB blob,
                         java.sql.Timestamp lastModified)
```

### Description

Builds the HTTP response header, then retrieves the contents of the BLOB from the database and delivers it to the browser.

This method supports browser content caching by supporting the If-Modified-Since and Last-Modified headers. This method calls the JspWriter.clear method to clear the output buffer of the page prior to delivering the image. Therefore, the page must use the buffered output model, which is the default.

### Parameters

**contentType**
A String that specifies the MIME type of the content.

**length**
An integer that specifies the length of the data.

**blob**
An object of type oracle.sql.BLOB.

**lastModified**
A java.sql.Timestamp object that specifies the date and time when the data was last modified, or null if no last modified date and time are available.

### Return Value

This method overrides the sendResponse( ) method in class OrdHttpResponseHandler.

### Exceptions

java.lang.IllegalStateException

This exception is thrown if PageContext has not been specified.

javax.servlet.ServletException

This exception is thrown if an error occurs accessing the binary output stream.

java.sql.SQLException

This exception is thrown if an error occurs obtaining an InputStream object to read the media data.

java.io.IOException

This exception is thrown if an error occurs reading the media data.

java.lang.IllegalArgumentException

This exception is thrown if the length is negative.

## Examples

The OrdHttpJspResponseHandler.sendResponse(String, int, BLOB, Timestamp) method extends the OrdHttpResponseHandler.sendResponse(String, int, BLOB, Timestamp) method. See sendResponse(String,int,BLOB,Timestamp) in OrdHttpResponseHandler Reference Information for an example of this method in the base class.

## sendResponse(String,int,InputStream,Timestamp)

### Format

```
public void sendResponse(String contentType, int length, java.io.InputStream in,
                         java.sql.Timestamp lastModified)
```

### Description

Builds the HTTP response header, then retrieves the contents of the InputStream object and delivers it to the browser.

This method supports browser content caching by supporting the If-Modified-Since and Last-Modified headers. This method calls the JspWriter.clear method to clear the output buffer of the page prior to delivering the image. Therefore, the page must use the buffered output model, which is the default.

### Parameters

**contentType**
A String that specifies the MIME type of the content.

**length**
An integer that specifies the length of the data.

**in**
An InputStream object from which the media data is retrieved.

**lastModified**
A java.sql.Timestamp object that specifies the date and time when the data was last modified, or null if no last modified date and time are available.

### Return Value

This method overrides the sendResponse( ) method in class OrdHttpResponseHandler.

### Exceptions

java.lang.IllegalStateException

This exception is thrown if PageContext has not been specified.

javax.servlet.ServletException

This exception is thrown if an error occurs accessing the binary output stream.

java.io.IOException

This exception is thrown if an error occurs reading the media data.

java.lang.IllegalArgumentException

This exception is thrown if the length is negative.

## Examples

The OrdHttpJspResponseHandler.sendResponse(String, int, InputStream, Timestamp) method extends the OrdHttpResponseHandler.sendResponse(String, int, InputStream, Timestamp) method. See sendResponse(String,int,InputStream,Timestamp) in OrdHttpResponseHandler Reference Information for an example of this method in the base class.

## sendVideo( )

### Format

public void sendVideo(oracle.ord.im.OrdVideo media)

### Description

Retrieves a video clip from an OrdVideo object and delivers it to the browser.

This method supports browser content caching by supporting the If-Modified-Since and Last-Modified headers. This method calls the JspWriter.clear method to clear the output buffer of the page prior to delivering the video clip. Therefore, the page must use the buffered output model, which is the default.

### Parameters

**media**
An object of type oracle.ord.im.OrdVideo.

### Return Value

This method overrides the sendVideo( ) method in class OrdHttpResponseHandler.

### Exceptions

java.lang.IllegalStateException

This exception is thrown if PageContext has not been specified.

OrdHttpResponseException

This exception is thrown if the source type is not recognized.

javax.servlet.ServletException

This exception is thrown if an error occurs accessing the binary output stream.

java.sql.SQLException

This exception is thrown if an error occurs obtaining an InputStream object to read the media data.

java.io.IOException

This exception is thrown if an error occurs reading the media data.

## Examples

The OrdHttpJspResponseHandler.sendVideo( ) method extends the OrdHttpResponseHandler.sendVideo( ) method. See sendVideo( ) in OrdHttpResponseHandler Reference Information for an example of this method in the base class.

## setPageContext( )

### Format

public void setPageContext(javax.servlet.jsp.PageContext pageContext)

### Description

Specifies the PageContext object for this response handler. You must call this method before calling any of the send methods if you did not specify the PageContext object in the constructor.

### Parameters

**pageContext**
An object of type PageContext.

### Return Value

None.

### Exceptions

None.

### Examples

```
<jsp:useBean id="handler" scope="page"
             class="oracle.ord.im.OrdHttpJspResponseHandler"/>
<%
     OraclePreparedStatement stmt = (OraclePreparedStatement)
          conn.prepareStatement("select image from photos where id = ?");
     stmt.setString(1, request.getParameter("id"));
     OracleResultSet rset = (OracleResultSet)stmt.executeQuery( );
     if (rset.next( )){
          OrdImage media = (OrdImage)rset.getCustomDatum(1,
               OrdImage.getFactory( ));
          handler.setPageContext(pageContext);
          handler.sendImage(image);
          return;
     }else{
          response.setStatus(response.SC_NOT_FOUND);
     }
%>
```

## OrdHttpUploadFile Reference Information

This section presents reference information on the methods of the OrdHttpUploadFile class.

Form-based file uploading using HTML forms encodes form data and uploaded files in POST requests using the multipart/form-data format. The OrdHttpUploadFile class is used to represent an uploaded file that has been parsed by the OrdHttpUploadFormData class (see OrdHttpUploadFormData Reference Information for more information on the OrdHttpUploadFormData class). The OrdHttpUploadFile class provides methods to obtain information about the uploaded file, to access the contents of the file directly, and to facilitate loading the contents into an *inter*Media object in a database.

Every input field of type FILE in an HTML form will produce a parameter of type OrdHttpUploadFile, whether or not a user enters a valid file name into such a field. Depending on the requirements, applications can test the length of the file name, the length of the content, or both to determine if a valid file name was entered by a user and if the file was successfully uploaded by the browser. For example, if a user does not enter a file name, the length of the String returned by the getOriginalFileName( ) method will be zero. However, if a user enters an invalid file name or the name of an empty (zero-length) file, then the content length returned by the getContentLength( ) method will be zero, even though the length of the file name will not be zero.

## getContentLength( )

### Format

public int getContentLength( )

### Description

Returns the length of the uploaded media file. If you enter an invalid file name, the name of a nonexistent file, or the name of an empty file, the length returned is zero.

### Parameters

None.

### Return Value

This method returns the length of the uploaded file.

### Exceptions

None.

### Examples

```
OrdHttpUploadFormData formData = new OrdHttpUploadFormData(request);
formData.parseFormData( );
...
OrdHttpUploadFile photo = formData.getFileParameter("photo");
String mimeType = photo.getMimeType( );
int i = photo.getContentLength( );
if (i == 0){
     displayUserError("The file is empty, invalid, or nonexistent");
}
...
photo.release( );
```

## getInputStream( )

### Format

public java.io.InputStream getInputStream( )

### Description

Returns an InputStream object that can be used to read uploaded data directly. Applications should close the stream with the close( ) method when finished.

### Parameters

None.

### Return Value

This method returns an InputStream object, from which to read the contents.

### Exceptions

java.lang.IllegalStateException

This exception is thrown if the uploaded file is no longer available because it has been released.

java.io.IOException

This exception is thrown if an error occurs opening the temporary file.

### Examples

```
OrdImage dbImage = (OrdImage)rset.getCustomDatum(1, OrdImage.getFactory( ));
OrdHttpUploadFile uploadImage = formData.getFileParameter("photo");
InputStream photoInputStream = uploadImage.getInputStream( );
try{
     dbImage.loadDataFromInputStream(photoInputStream);
}
finally{
     photoInputStream.close( );
}
```

## getMimeType( )

### Format

```
public String getMimeType( )
```

### Description

Returns the MIME type of the media file, as determined by the browser when the file is uploaded.

Some browsers return a default MIME type even if you do not supply a file name; therefore, the application should check the file name or content length to ensure the file was uploaded successfully.

### Parameters

None.

### Return Value

This method returns the MIME type of the file, as a String.

### Exceptions

None.

### Examples

See getContentLength( ) for an example of this method.

## getOriginalFileName( )

### Format

public String getOriginalFileName( )

### Description

Returns the original file name, as provided by the browser. If you do not supply a file name, an empty String is returned.

### Parameters

None.

### Return Value

This method returns the file name, as a String.

### Exceptions

None.

### Examples

```
OrdHttpUploadFormData formData = new OrdHttpUploadFormData(request);
formData.parseFormData( );
...
OrdHttpUploadFile photo = formData.getFileParameter("photo");
String originalName = photo.getOriginalFileName( );
...
formData.release( );
```

## getSimpleFileName( )

### Format

public String getSimpleFileName( )

### Description

Returns the simple file name (that is, the name of the file and the extension). If you do not supply a file name, an empty String is returned.

### Parameters

None.

### Return Value

This method returns the simple file name, as a String.

### Exceptions

None.

### Examples

```
OrdHttpUploadFormData formData = new OrdHttpUploadFormData(request);
formData.parseFormData( );
...
OrdHttpUploadFile photo = formData.getFileParameter("photo");
String name = photo.getSimpleFileName( );
...
formData.release( );
```

## loadAudio(OrdAudio)

### Format

public void loadAudio(oracle.ord.im.OrdAudio media)

### Description

Loads the uploaded file into an OrdAudio Java object and sets the properties based on the audio data. This method loads the audio data into the database and calls the OrdAudio.setProperties( ) method to set the properties, such as the MIME type. This method does not use any existing format plug-in context information and does not set any comments when setting the properties. To use this method, the application fetches an initialized OrdAudio object from the database, calls this method to load the audio data into the database, and then updates the OrdAudio object in the database.

If the call to the setProperties( ) method fails because the audio format is not recognized, this method sets the following properties:

- MIME type (to the value specified by the browser)
- update time (to the current date and time)

### Parameters

**media**
An oracle.ord.im.OrdAudio object into which the audio data will be loaded.

### Return Value

None.

### Exceptions

java.io.IOException

This exception is thrown if an error occurs while reading or writing the media data.

java.sql.SQLException

This exception is thrown if an unrecognized error occurs while storing the media data.

java.lang.IllegalStateException

This exception is thrown if the uploaded file is no longer available because it has been released.

## Example

```
OrdHttpUploadFormData formData = new OrdHttpUploadFormData(request);
formData.parseFormData( );
String id = formData.getParameter("id");
OrdHttpUploadFile uploadFile = formData.getFileParameter("soundfile");

OraclePreparedStatement stmt = (OraclePreparedStatement)
     conn.prepareStatement("insert into songs (id,sound) values(?,
     ORDSYS.ORDAUDIO.INIT( ))");
stmt.setString(1, id);
stmt.executeUpdate( );
stmt.close( );

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

OrdAudio sound = (OrdAudio)rset.getCustomDatum(1, OrdAudio.getFactory( ));
uploadFile.loadAudio(sound);
formData.release( );

rset.close( );
stmt.close( );

stmt = (OraclePreparedStatement)conn.prepareStatement("update songs set
     sound = ? where id = ?");
stmt.setCustomDatum(1, sound);
stmt.setString(2, id);
stmt.execute( );
stmt.close( );
conn.commit( );
```

## loadAudio(OrdAudio,byte[ ][ ], boolean)

### Format

public void loadAudio(oracle.ord.im.OrdAudio media, byte[ ][ ] ctx, boolean setComments)

### Description

Loads the uploaded file into an OrdAudio Java object and sets the properties using an application-supplied, format plug-in context. This method loads the audio data into the database and calls the OrdAudio.setProperties( ) method to set the properties, such as the MIME type. The application provides the format plug-in context information and chooses whether or not to set the comments in the OrdAudio object. To use this method, the application fetches an initialized OrdAudio object from the database, calls this method to load the audio data into the database, and then updates the OrdAudio object in the database.

If the call to the setProperties( ) method fails because the audio format is not recognized, this method sets the following properties:

- MIME type (to the value specified by the browser)
- update time (to the current date and time)

### Parameters

**media**
An oracle.ord.im.OrdAudio object into which the audio data will be loaded.

**ctx**
The format plug-in context information.

**setComments**
A Boolean value indicating whether or not to set the comments in the OrdAudio object.

### Return Value

None.

### Exceptions

java.io.IOException

This exception is thrown if an error occurs while reading or writing the media data.

java.sql.SQLException

This exception is thrown if an unrecognized error occurs while storing the media data.

java.lang.IllegalStateException

This exception is thrown if the uploaded file is no longer available because it has been released.

## Examples

```
OrdHttpUploadFormData formData = new OrdHttpUploadFormData(request);
formData.parseFormData( );
String id = formData.getParameter("id");
OrdHttpUploadFile uploadFile = formData.getFileParameter("soundfile");

OraclePreparedStatement stmt = (OraclePreparedStatement)
     conn.prepareStatement("insert into songs (id,sound) values(?,
     ORDSYS.ORDAUDIO.INIT( ))");
stmt.setString(1, id);
stmt.executeUpdate( );
stmt.close( );

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

byte[ ] [ ] ctx = new byte[1][64];
OrdAudio sound = (OrdAudio)rset.getCustomDatum(1, OrdAudio.getFactory( ));
uploadFile.loadAudio(sound, ctx, true);
formData.release( );

rset.close( );
stmt.close( );

stmt = (OraclePreparedStatement)conn.prepareStatement("update songs set
     sound = ? where id = ?");
stmt.setCustomDatum(1, sound);
stmt.setString(2, id);
stmt.execute( );
stmt.close( );
conn.commit( );
```

## loadBlob( )

### Format

public void loadBlob(oracle.sql.BLOB blob)

### Description

Loads the uploaded media file into a BLOB.

### Parameters

**blob**
An oracle.sql.BLOB into which the data will be loaded.

### Return Value

None.

### Exceptions

java.io.IOException

This exception is thrown if an error occurs while reading or writing the media data.

java.sql.SQLException

This exception is thrown if an unrecognized error occurs while storing the media data.

java.lang.IllegalStateException

This exception is thrown if the uploaded file is no longer available because it has been released.

### Examples

```
OrdHttpUploadFormData formData = new OrdHttpUploadFormData(request);
formData.parseFormData( );
String id = formData.getParameter("id");
OrdHttpUploadFile uploadFile = formData.getFileParameter("docfile");

OraclePreparedStatement stmt = (OraclePreparedStatement)
     conn.prepareStatement("insert into docs (id,doc_blob) values
     (?,EMPTY_BLOB( ))");
stmt.setString(1, id);
stmt.executeUpdate( );
stmt.close( );

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

BLOB docBlob = rset.getBLOB(1);
uploadFile.loadBlob(docBlob);
formData.release( );

rset.close( );
stmt.close( );
conn.commit( );
```

## loadDoc(OrdDoc)

### Format

public void loadDoc(oracle.ord.im.OrdDoc media)

### Description

Loads the uploaded file into an OrdDoc Java object and sets the properties based on the contents of the document. This method loads the document into the database and calls the OrdDoc.setProperties( ) method to set the properties, such as the MIME type. This method does not use any existing format plug-in context information and does not set any comments when setting the properties. To use this method, the application fetches an initialized OrdDoc object from the database, calls this method to load the document into the database, and then updates the OrdDoc object in the database.

If the call to the setProperties( ) method fails because the document format is not recognized, this method sets the following properties:

- MIME type (to the value specified by the browser)
- content length (to the length of the uploaded file)
- update time (to the current date and time)

### Parameters

**media**
An oracle.ord.im.OrdDoc object into which the document will be loaded.

### Return Value

None.

### Exceptions

java.io.IOException

This exception is thrown if an error occurs while reading or writing the media data.

java.sql.SQLException

This exception is thrown if an unrecognized error occurs while storing the media data.

java.lang.IllegalStateException

This exception is thrown if the uploaded file is no longer available because it has been released.

## Examples

```
OrdHttpUploadFormData formData = new OrdHttpUploadFormData(request);
formData.parseFormData( );
String id = formData.getParameter("id");
OrdHttpUploadFile uploadFile = formData.getFileParameter("docfile");

OraclePreparedStatement stmt = (OraclePreparedStatement)
     conn.prepareStatement("insert into documents (id,doc) values(?,
     ORDSYS.ORDDOC.INIT( ))");
stmt.setString(1, id);
stmt.executeUpdate( );
stmt.close( );

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

OrdDoc media = (OrdDoc)rset.getCustomDatum(1, OrdDoc.getFactory( ));
uploadFile.loadDoc(media);
formData.release( );

rset.close( );
stmt.close( );

stmt = (OraclePreparedStatement)conn.prepareStatement("update documents set
     doc = ? where id = ?");
stmt.setCustomDatum(1, doc);
stmt.setString(2, id);
stmt.execute( );
stmt.close( );
conn.commit( );
```

## loadDoc(OrdDoc,byte[ ][ ],boolean)

### Format

public void loadDoc(oracle.ord.im.OrdDoc media, byte[ ][ ] ctx, boolean setComments)

### Description

Loads the uploaded file into an OrdDoc Java object and sets the properties using an application-supplied, format plug-in context. This method loads the document into the database and calls the OrdDoc.setProperties( ) method to set the properties, such as the MIME type. The application provides the format plug-in context information and chooses whether or not to set the comments in the OrdDoc object. To use this method, the application fetches an initialized OrdDoc object from the database, calls this method to load the document into the database, and then updates the OrdDoc object in the database.

If the call to the setProperties( ) method fails because the document format is not recognized, this method sets the following properties:

- MIME type (to the value specified by the browser)
- content length (to the length of the uploaded file)
- update time (to the current date and time)

### Parameters

**media**
An oracle.ord.im.OrdDoc object into which the document will be loaded.

**ctx**
The format plug-in context information.

**setComments**
A Boolean value indicating whether or not to set the comments in the object.

### Return Value

None.

### Exceptions

java.io.IOException

This exception is thrown if an error occurs while reading or writing the media data.

java.sql.SQLException

This exception is thrown if an unrecognized error occurs while storing the media data.

java.lang.IllegalStateException

This exception is thrown if the uploaded file is no longer available because it has been released.

## Examples

```
OrdHttpUploadFormData formData = new OrdHttpUploadFormData(request);
formData.parseFormData( );
String id = formData.getParameter("id");
OrdHttpUploadFile uploadFile = formData.getFileParameter("docfile");

OraclePreparedStatement stmt = (OraclePreparedStatement)
     conn.prepareStatement("insert into documents (id,doc) values(?,
     ORDSYS.ORDDOC.INIT( ))");
stmt.setString(1, id);
stmt.executeUpdate( );
stmt.close( );

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

byte[ ] [ ] ctx = new byte[1][64];
OrdDoc media = (OrdDoc)rset.getCustomDatum(1, OrdDoc.getFactory( ));
uploadFile.loadDoc(media, ctx, true);
formData.release( );

rset.close( );
stmt.close( );

stmt = (OraclePreparedStatement)conn.prepareStatement("update documents set
     doc = ? where id = ?");
stmt.setCustomDatum(1, doc);
stmt.setString(2, id);
stmt.execute( );
stmt.close( );
conn.commit( );
```

## loadImage(OrdImage)

### Format

public void loadImage(oracle.ord.im.OrdImage media)

### Description

Loads the uploaded file into an OrdImage Java object and sets the properties based on the image contents. This method loads the image content into the database and calls the OrdImage.setProperties( ) method to set the image's properties, such as the MIME type, length, height, and width. To use this method, the application fetches an initialized OrdImage object from the database, calls this method to load the image content into the database, and then updates the OrdImage object in the database.

If the call to the setProperties( ) method fails because the image format is not recognized, this method sets the following properties:

- MIME type (to the value specified by the browser)
- content length (to the length of the uploaded file)
- update time (to the current date and time)

### Parameters

**media**
An oracle.ord.im.OrdImage object into which the image data will be loaded.

### Return Value

None.

### Exceptions

java.io.IOException

This exception is thrown if an error occurs while reading or writing the media data.

java.sql.SQLException

This exception is thrown if an unrecognized error occurs while storing the media data.

java.lang.IllegalStateException

This exception is thrown if the uploaded file is no longer available because it has been released.

## Examples

```
OrdHttpUploadFormData formData = new OrdHttpUploadFormData(request);
formData.parseFormData( );
String id = formData.getParameter("id");
OrdHttpUploadFile uploadFile = formData.getFileParameter("photofile");

OraclePreparedStatement stmt = (OraclePreparedStatement)
     conn.prepareStatement("insert into photos (id,photo) values(?,
     ORDSYS.ORDIMAGE.INIT( ))");
stmt.setString(1, id);
stmt.executeUpdate( );
stmt.close( );

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

OrdImage photo = (OrdImage)rset.getCustomDatum(1, OrdImage.getFactory( ));
uploadFile.loadImage(photo);
formData.release( );

rset.close( );
stmt.close( );

stmt = (OraclePreparedStatement)conn.prepareStatement("update photos set
     photo = ? where id = ?");
stmt.setCustomDatum(1, photo);
stmt.setString(2, id);
stmt.execute( );
stmt.close( );
conn.commit( );
```

## loadImage(OrdImage,String)

### Format

public void loadImage(oracle.ord.im.OrdImage media, String cmd)

### Description

Loads the uploaded file into an OrdImage Java object and sets the properties according to an application-supplied command string. To use this method, the application fetches an initialized OrdImage object from the database, calls this method to load the image content into the database, and then updates the OrdImage object in the database.

### Parameters

**media**
An oracle-ord.im.OrdImage object into which the uploaded image data will be loaded.

**cmd**
A String that specifies the properties to be set.

### Return Value

None.

### Exceptions

java.io.IOException

This exception is thrown if an error occurs while reading or writing the media data.

java.sql.SQLException

This exception is thrown if an unrecognized error occurs while storing the media data.

java.lang.IllegalStateException

This exception is thrown if the uploaded file is no longer available because it has been released.

## Examples

```
OrdHttpUploadFormData formData = new OrdHttpUploadFormData(request);
formData.parseFormData( );
String id = formData.getParameter("id");
OrdHttpUploadFile uploadFile = formData.getFileParameter("photofile");

OraclePreparedStatement stmt = (OraclePreparedStatement)
     conn.prepareStatement("insert into photos (id,photo) values(?,
     ORDSYS.ORDIMAGE.INIT( ))");
stmt.setString(1, id);
stmt.executeUpdate( );
stmt.close( );

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

OrdImage photo = (OrdImage)rset.getCustomDatum(1, OrdImage.getFactory( ));
String cmd = getImagePropertiesCommand(photo);
uploadFile.loadImage(photo, cmd);
formData.release( );

rset.close( );
stmt.close( );

stmt = (OraclePreparedStatement)conn.prepareStatement("update photos set
     photo = ? where id = ?");
stmt.setCustomDatum(1, photo);
stmt.setString(2, id);
stmt.execute( );
stmt.close( );
conn.commit( );
```

## loadVideo(OrdVideo)

### Format

public void loadVideo(oracle.ord.im.OrdVideo media)

### Description

Loads the uploaded file into an OrdVideo Java object and sets the properties based on the video data. This method loads the video data into the database and calls the OrdVideo.setProperties( ) method to set the properties, such as the MIME type. This method does not use any existing format plug-in context information and does not set any comments when setting the properties. To use this method, the application fetches an initialized OrdVideo object from the database, calls this method to load the video data into the database, and then updates the OrdVideo object in the database.

If the call to the setProperties( ) method fails because the video format is not recognized, this method sets the following properties:

- MIME type (to the value specified by the browser)
- update time (to the current date and time)

### Parameters

**media**
An oracle.ord.im.OrdVideo object into which the uploaded video data will be loaded.

### Return Value

None.

### Exceptions

java.io.IOException

This exception is thrown if an error occurs while reading or writing the media data.

java.sql.SQLException

This exception is thrown if an unrecognized error occurs while storing the media data.

java.lang.IllegalStateException

This exception is thrown if the uploaded file is no longer available because it has been released.

## Examples

```
OrdHttpUploadFormData formData = new OrdHttpUploadFormData(request);
formData.parseFormData( );
String id = formData.getParameter("id");
OrdHttpUploadFile uploadFile = formData.getFileParameter("videofile");

OraclePreparedStatement stmt = (OraclePreparedStatement)
    conn.prepareStatement("insert into movies (id,video) values(?,
    ORDSYS.ORDVIDEO.INIT( ))");
stmt.setString(1, id);
stmt.executeUpdate( );
stmt.close( );

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

OrdVideo media = (OrdVideo)rset.getCustomDatum(1, OrdVideo.getFactory( ));
uploadFile.loadVideo(media);
formData.release( );

rset.close( );
stmt.close( );

stmt = (OraclePreparedStatement)conn.prepareStatement("update movies set
    video = ? where id = ?");
stmt.setCustomDatum(1, video);
stmt.setString(2, id);
stmt.execute( );
stmt.close( );
conn.commit( );
```

## loadVideo(OrdVideo,byte[ ][ ],boolean)

### Format

public void loadVideo(oracle.ord.im.OrdVideo media, byte[ ][ ] ctx, boolean setComments)

### Description

Loads the uploaded file into an OrdVideo Java object and sets the properties using an application-supplied, format plug-in context. This method loads the video data into the database and calls the OrdVideo.setProperties( ) method to set the properties, such as the MIME type. The application provides the format plug-in context information and chooses whether or not to set the comments in the OrdVideo object. To use this method, the application fetches an initialized OrdVideo object from the database, calls this method to load the video data into the database, and then updates the OrdVideo object in the database.

If the call to the setProperties( ) method fails because the video format is not recognized, this method sets the following properties:

- MIME type (to the value specified by the browser)
- update time (to the current date and time)

### Parameters

**media**
An oracle.ord.im.OrdVideo object into which the uploaded video data will be loaded.

**ctx**
The format plug-in context information.

**setComments**
A Boolean value indicating whether or not to set the comments in the object.

### Return Value

None.

### Exceptions

java.io.IOException

This exception is thrown if an error occurs while reading or writing the media data.

java.sql.SQLException

This exception is thrown if an unrecognized error occurs while storing the media data.

java.lang.IllegalStateException

This exception is thrown if the uploaded file is no longer available because it has been released.

## Examples

```
OrdHttpUploadFormData formData = new OrdHttpUploadFormData(request);
formData.parseFormData( );
String id = formData.getParameter("id");
OrdHttpUploadFile uploadFile = formData.getFileParameter("videofile");

OraclePreparedStatement stmt = (OraclePreparedStatement)
     conn.prepareStatement("insert into movies (id,video) values(?,
     ORDSYS.ORDVIDEO.INIT( ))");
stmt.setString(1, id);
stmt.executeUpdate( );
stmt.close( );

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

byte[ ] [ ] ctx = new byte[1][64];
OrdVideo media = (OrdVideo)rset.getCustomDatum(1, OrdVideo.getFactory( ));
uploadFile.loadVideo(media, ctx, true);
formData.release( );

rset.close( );
stmt.close( );

stmt = (OraclePreparedStatement)conn.prepareStatement("update movies set
     video = ? where id = ?");
stmt.setCustomDatum(1, video);
stmt.setString(2, id);
stmt.execute( );
stmt.close( );
conn.commit( );
```

## release( )

### Format

public void release( )

### Description

Releases all resources held by an OrdHttpUploadFile object. Specifically, this method releases the memory used to hold the contents of an uploaded file or deletes the temporary file used to hold the contents of the uploaded file. An application can optimize memory usage by calling this method to release any allocated memory, making it a candidate for garbage collection, after the application has finished processing an uploaded file.

### Parameters

None.

### Return Value

None.

### Exceptions

None.

### Examples

See getContentLength( ) for an example of this method.

## OrdHttpUploadFormData Reference Information

This section presents reference information on the methods of the OrdHttpUploadFormData class.

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 uploaded files readily accessible to a Java servlet or JavaServer Page (JSP). The OrdHttpUploadFormData class provides methods to access text-based form field parameters that are identical to the getParameter( ), getParameterValues( ), and getParameterNames( ) methods provided by the ServletRequest class. Access to uploaded files is provided by a similar set of methods, namely getFileParameter( ), getFileParameterValues( ), and getFileParameterNames( ). The OrdHttpUploadFile objects returned by the getFileParameter( ) and getFileParameterValues( ) methods provide simple access to the MIME type, length, and contents of each uploaded file.

For more information on the OrdHttpUploadFile class, see OrdHttpUploadFile Reference Information.

The following is an example of how to use the OrdHttpUploadFormData class:

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

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

// Prepare and execute a SQL statement to insert a new row into
// the table and return the sequence number for the new row.
// Disable auto-commit to allow the LOB to be written correctly.
//
conn.setAutoCommit(false);
PreparedStatement stmt = conn.prepareStatement("insert into photo_album (id,
     description, location, photo) " + " values (?, ?, ?, ORDSYS.ORDIMAGE.INIT(
))");
stmt.setString(1, id);
stmt.setString(2, description);
stmt.setString(3, location);
stmt.executeUpdate( );

// Prepare and execute a SQL statement to fetch the new OrdImage
// object from the database.
//
stmt = conn.prepareStatement("select photo from photo_album where id = ? for
     update");
stmt.setString(1, id);
OracleResultSet rset = (OracleResultSet)stmt.executeQuery( );
if (!rset.next( )){
     throw new ServletException("new row not found in table");
}
OrdImage media = (OrdImage)rset.getCustomDatum(1, OrdImage.getFactory( ));

// Load the photograph into the database and set the properties.
//
photo.loadImage(media);

// Prepare and execute a SQL statement to update the image object.
//
stmt = (OraclePreparedStatement)conn.prepareStatement("update photo_album set
     photo = ? where id = ?");
stmt.setCustomDatum(1, media);
stmt.setString(2 id);
stmt.execute( );
stmt.close( );

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

### A Note on the Handling of Query String Parameters and Text-Based HTML Form Field Parameters

Every parameter in the optional query string of a request produces a corresponding parameter of type String, whether or not any data is associated with the parameter name. Likewise, every text-based input field in an HTML form also produces a corresponding parameter of type String, whether or not any data is entered into a field. When processing query string parameters and text-based input fields, applications can test the length of the corresponding String object to determine if any data is present.

The parseFormData( ) method merges all query string and form field parameters into a single set of ordered parameters, where the query string parameters are processed first, followed by the form field parameters. Thus, query string parameters take precedence over form field parameters. For example, if a request is made with a query string of arg=hello&arg=world and the values ’greetings’ and ’everyone’ are entered into two HTML form fields named ’arg’, then the resulting parameter set would include the following entry: arg=(hello, world, greetings, everyone).

### A Note on the Handling of FILE-Type HTML Form Field Parameters

Every input field of type FILE in an HTML form produces a corresponding parameter of type OrdUploadFile, whether or not a valid file name is entered into the field. When processing a field of type FILE, applications can test either the length of the file name, the length of content, or a combination of the two to determine if a valid file name was entered by a user, and if the file was successfully uploaded by the browser. See the OrdHttpUploadFile class in OrdHttpUploadFile Reference Information for more information.

### A Note on the Use of Non-Western European Languages

Microsoft’s Internet Explorer browser allows data to be entered into an HTML form using a character set encoding that is different from that being used to view the form. For example, it is possible to copy Japanese Shift_JIS character set data from one browser window and paste it into a form being viewed using the Western European (ISO) character set in a different browser window. In this situation, Internet Explorer can sometimes transmit the form data twice in such a way that the multipart/form-data parser cannot detect the duplicated data. Furthermore, the non-Western European language form data is sometimes sent as a Unicode escape sequence, sometimes in its raw binary form, and sometimes duplicated using both formats in different portions of the POST data.

Although this same problem does not exist with the Netscape browser, care must still be taken to ensure that the correct character set is being used. For example, although it is possible to copy Japanese Shift_JIS character set data from one browser window and paste it into a form being viewed using the Western European (ISO) character set in a different browser window, when the data is pasted into the form field, the two bytes that comprise each Japanese Shift_JIS character are stored as two individual Western European (ISO) characters.

Therefore, care must be taken to view an HTML form using the correct character set, no matter which Web browser is used. For example, the HTML META tag can be used to specify the character set as follows:

```
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=Shift_JIS">
```

## enableParameterTranslation( )

### Format

public void enableParameterTranslation(java.lang.Sting encoding)

### Description

Enables the translation of all HTML form parameter names and all text-based HTML form parameter values using the specified character encoding when parsing the body of a multipart/form-data POST request.

Character encoding of request parameters is not well defined in the HTTP specification. Most servlet containers interpret them using the servlet default encoding, ISO-8859-1. Therefore, to provide an API that is compatible with such servlet containers, by default, the OrdHttpUploadFormData class also interprets multipart/form-data request parameters using the default encoding, ISO-8859-1.

Applications that process requests and responses using other character encodings can, prior to calling the parseFormData( ) method, call the enableParameterTranslation( ) method to specify the character encoding to be used to translate the names of all HTML form parameters, and the values of all text-based HTML form parameters when parsing the body of a multipart/form-data POST request.

**Notes:**

- Query string parameters that accompany multipart/form-data POST requests are *not* translated prior to being merged into the list of multipart/form-data parameters. This is because there is no way to determine if the underlying servlet container or JSP engine has decoded the query string or translated the parameter names and values already. Therefore, the application is responsible for translating any multibyte query string parameter names or values in the case where the underlying servlet container or JSP engine does not perform the translation.
- The contents of uploaded files are never translated; nor is the associated content type attribute, which is always represented using the ISO-8859-1 character encoding. However, the file name attribute of an uploaded file is translated.
- Query string parameters in GET requests and query string and POST data parameters in application/x-www-form-urlencoded POST requests are never translated.
- To correctly handle the translation of HTML form parameter names and values, applications must call the enableParameterTranslation( ) method for multipart/form-data POST requests, even if the servlet container or JSP engine translates parameter names and values for GET requests and application/x-www-form-urlencoded POST requests.
- Do not call the enableParameterTranslation( ) method if the application contains code that handles the translation of parameter names and values.
- The enableParameterTranslation( ) method *must* be called *before* calling the parseFormData( ) method.

Calling the enableParameterTranslation( ) method with a character encoding other than ISO-8859-1 affects the following methods when called for multipart/form-data POST requests:

- getParameter( ): parameter name argument and returned parameter value
- getParameterValues( ): parameter name argument and returned parameter values

- getParameterNames( ): returned parameter names
- getFileParameter( ): parameter name argument only
- getFileParameterValues( ): parameter name argument only
- getFileParameterNames( ): returned parameter names

For GET requests and application/x-www-form-urlencoded POST requests, calls to the getParameter( ), getParameterValues( ), and getParameterNames( ) methods are passed directly to the underlying servlet container or JSP engine. Please consult the servlet container or JSP engine documentation for information regarding any parameter translation functions that might be supported by the servlet container or JSP engine.

## Parameters

**encoding**
A String that specifies the character encoding.

## Return Value

None.

## Exceptions

None.

## Examples

```
OrdHttpUploadFormData formData = new OrdHttpUploadFormData(request);
formData.enableParameterTranslation("GB2312")
formData.parseFormData( );
```

## getFileParameter( )

### Format

public OrdHttpUploadFile getFileParameter(String parameterName)

### Description

Returns information about an uploaded file identified by parameter name, as an OrdHttpUploadFile object.

Every input field of type FILE in an HTML form will produce a parameter of type OrdHttpUploadFile, whether or not you enter a valid file name into the field.

### Parameters

**parameterName**
The name of the uploaded file parameter, as a String.

### Return Value

This method returns the uploaded file parameter, as an OrdHttpUploadFile object, or null if the parameter does not exist.

### Exceptions

java.lang.IllegalStateException

This exception is thrown if the ServletRequest object has not been specified, if the multipart form data has not been parsed, or if the upload request has been released.

### Examples

```
OrdHttpUploadFormData formData = new OrdHttpUploadFormData(request);
formData.parseFormData( );
...
OrdHttpUploadFile photo = formData.getFileParameter("photo");
photo.loadImage(media);
...
formData.release( );
```

## getFileParameterNames( )

### Format

public java.util.Enumeration getFileParameterNames( )

### Description

Returns an Enumeration of the names of all input fields of type FILE in an HTML form, or an empty Enumeration if there are no input fields of type FILE.

Every input field of type FILE in an HTML form will produce a parameter of type OrdHttpUploadFile, whether or not you enter a valid file name into the field.

### Parameters

None.

### Return Value

This method returns a list of uploaded file parameter names, as an Enumeration of Strings.

### Exceptions

java.lang.IllegalStateException

This exception is thrown if the ServletRequest object has not been specified, if the multipart form data has not been parsed, or if the upload request has been released.

### Examples

```
OrdHttpUploadFormData formData = new OrdHttpUploadFormData( request );
formData.parseFormData( );
...
Enumeration names = formData.getFileParameterNames( );
```

## getFileParameterValues( )

### Format

public OrdHttpUploadFile[ ] getFileParameterValues(String parameterName)

### Description

Returns an array of OrdHttpUploadFile objects that represent all files uploaded using the specified parameter name. Every input field of type FILE in an HTML form will produce a parameter of type OrdHttpUploadFile, whether or not you enter a valid file name in the field.

### Parameters

**parameterName**
The name of the uploaded file parameter, as a String.

### Return Value

This method returns the uploaded file parameters as an array of OrdHttpUploadFile objects, or null if the parameter does not exist.

### Exceptions

java.lang.IllegalStateException

This exception is thrown if the ServletRequest object has not been specified, if the multipart form data has not been parsed, or if the upload request has been released.

### Examples

```
OrdHttpUploadFormData formData = new OrdHttpUploadFormData( request );
formData.parseFormData( );
...
OrdHttpUploadFile[ ] photo = formData.getFileParameterValues("photo")
```

## getParameter( )

### Format

public String getParameter(String parameterName)

### Description

Returns the value of the first query string parameter or text-based form field parameter with the specified name, or null if the parameter does not exist. The query string parameters of the request are merged with the text-based form field parameters by the parseFormData( ) method.

This method calls the getParameterName( ) method in the ServletRequest class if the request is not a multipart/form-data upload request.

### Parameters

**parameterName**
A String that specifies the parameter name.

### Return Value

This method returns the value of the specified parameter, as a String, or null if the parameter does not exist.

### Exceptions

java.lang.IllegalStateException

This exception is thrown if the ServletRequest object has not been specified, if the multipart form data has not been parsed, or if the upload request has been released.

### Examples

```
OrdHttpUploadFormData formData = new OrdHttpUploadFormData( request );
formData.parseFormData( );
...
String id = formData.getParameter("id");
```

## getParameterNames( )

### Format

public java.util.Enumeration getParameterNames( )

### Description

Returns an Enumeration of all the query string parameter names and all the text-based form field data parameter names in the request, or an empty Enumeration if there are no query string parameters and no text-based form field parameters. The query string parameters of the request are merged with the text-based form field parameters by the parseFormData( ) method.

This method calls the getParameterNames( ) method in the ServletRequest class if the request is not a multipart/form-data upload request.

### Parameters

None.

### Return Value

This method returns a list of parameter names, as an Enumeration of Strings.

### Exceptions

java.lang.IllegalStateException

This exception is thrown if the ServletRequest object has not been specified, if the multipart form data has not been parsed, or if the upload request has been released.

### Examples

```
OrdHttpUploadFormData formData = new OrdHttpUploadFormData( request );
formData.parseFormData( );
...
Enumeration names = formData.getParameterNames( );
```

## getParameterValues( )

### Format

public String[ ] getParameterValues(String parameterName)

### Description

Returns an array of String objects containing the values of all the query string parameters and text-based form field data parameters with the specified parameter name, or null if the parameter does not exist. The query string parameters of the request are merged with the text-based form field parameters by the parseFormData( ) method.

This method calls the getParameterValues( ) method in the ServletRequest class if the request is not a multipart/form-data upload request.

### Parameters

**parameterName**
A String that specifies the parameter name.

### Return Value

This method returns the parameter value, as a String, or null if the parameter does not exist.

### Exceptions

java.lang.IllegalStateException

This exception is thrown if the ServletRequest object has not been specified, if the multipart form data has not been parsed, or if the upload request has been released.

### Examples

```
OrdHttpUploadFormData formData = new OrdHttpUploadFormData(request);
formData.parseFormData( );
...
String[ ] ids = formData.getParameterValues("id");
```

## isUploadRequest( )

### Format

public boolean isUploadRequest( )

### Description

Tests if the request was encoded using the multipart/form-data encoding format.

### Parameters

None.

### Return Value

This method returns true if the request body was encoded using the multipart/form-data encoding format; false otherwise.

### Exceptions

java.lang.IllegalStateException

This exception is thrown if HttpServletRequest has not been specified.

### Examples

```
OrdHttpUploadFormData formData = new OrdHttpUploadFormData(request);
if(formData.isUploadRequest( )){
     formData.parseFormData( );
     OrdHttpUploadFile uploadFile = formData.getFileParameter(...);
     ...
}
else{
     String param = request.getParameter(...);
     ...
}
```

## OrdHttpUploadFormData( )

### Format

public OrdHttpUploadFormData( )

### Description

Creates an OrdHttpFormData object to parse a multipart/form-data request. Subsequently, the application must specify the ServletRequest object.

### Parameters

None.

### Return Value

None.

### Exceptions

None.

### Examples

See setServletRequest( ) for an example of this method.

## OrdHttpUploadFormData(ServletRequest)

### Format

```
public OrdHttpUploadFormData(javax.servlet.ServletRequest request)
```

### Description

Creates an OrdHttpUploadFormData object to parse a multipart/form-data request.

### Parameters

**request**
An object of type ServletRequest.

### Return Value

None.

### Exceptions

None.

### Examples

See getFileParameter( ) for an example of this method.

## parseFormData( )

### Format

public void parseFormData( )

### Description

Parses the body of a POST request that is encoded using the multipart/form-data encoding. If the request is not an upload request, this method does nothing.

### Parameters

None.

### Return Value

None.

### Exceptions

java.lang.IllegalStateException

This exception is thrown if HttpServletRequest has not been specified.

java.io.IOException

This exception is thrown if an error occurs reading the request body or writing a temporary file.

OrdHttpUploadException

This exception is thrown if an error occurs parsing the multipart/form-data message.

### Examples

See getFileParameter( ) for an example of this method.

## release( )

### Format

public void release( )

### Description

Releases all resources held by an OrdHttpUploadFormData object, including temporary files used to hold the contents of uploaded files. An application that enables the use of temporary files must call this method.

### Parameters

None.

### Return Value

None.

### Exceptions

None.

### Examples

See getFileParameter( ) for an example of this method.

## setMaxMemory( )

### Format

public void setMaxMemory(int maxMemory, String tempFileDir)

### Description

Specifies the maximum amount of memory that the contents of uploaded files can consume before the contents are stored in a temporary directory.

By default, the contents of uploaded files are held in memory until stored in a database by the application. If users upload large files, such as large video clips, then it may be desirable to limit the amount of memory consumed, and to store temporarily the contents of such files on disk, before the contents are written to a database.

---

**Note:** Applications that use this mechanism must ensure that any temporary files are deleted when no longer required by using the release( ) method. See the release( ) method for more information.

---

### Parameters

**maxMemory**
An integer that specifies the maximum amount of memory to be consumed by all uploaded files in a request before the contents of the uploaded files are stored in temporary files.

**tempFileDir**
A String that specifies the temporary file directory where you will store temporary files. This parameter is optional if the java.io.tmpdir system property is present.

### Return Value

None.

### Exceptions

java.lang.IllegalArgumentException

This exception is thrown if the maxMemory parameter is negative, or if the tempFileDir parameter was specified as null and the java.io.tmpdir system property is not present.

## Examples

```
OrdHttpUploadFormData formData = new OrdHttpUploadFormData(request);
try{
     formData.setMaxMemory(65536,null);
     formData.parseFormData( );
     ...
     OrdHttpUploadFile photo = formData.getFileParameter("photo");
     photo.loadImage(media);
     ...
}
finally{
     formData.release( );
}
```

## setServletRequest( )

### Format

public void setServletRequest(javax.servlet.ServletRequest request)

### Description

Specifies the ServletRequest object for the request. The ServletRequest object must be specified either in the constructor, or by calling this method before parsing the request.

### Parameters

**request**
An object of type ServletRequest.

### Return Value

None.

### Exceptions

None.

### Examples

```
OrdHttpUploadFormData formData = new OrdHttpUploadFormData( );
...
formData.setServletRequest(request);
```

# A Running Java Classes Examples

Four sample files (programs written in Java) are provided in the installation of *inter*Media Java Classes. These files provide examples of how to build Java applications with *inter*Media. They demonstrate loading data from various sources into database objects, downloading data from database objects to the file system, and extracting and displaying metadata from the media content.

The names of the Java sample files are as follows:

- For audio: AudioExample.java
- For document: DocumentExample.java
- For image: ImageExample.java
- For video: VideoExample.java

Two additional Java sample files are provided in the installation of *inter*Media Java Classes. These sample files demonstrate using *inter*Media Java Classes for servlets and JSPs to upload and retrieve media data.

The names of the additional Java sample files are as follows:

- Oracle *inter*Media JavaServer Pages Photo Album Demo
- Oracle *inter*Media Java Servlet Photo Album Demo

See Section A.1, "Finding the Sample Files" for information about where to find the readme files that contain these sample files.

## A.1 Finding the Sample Files

After installing Oracle *inter*Media, you can locate the Java sample files and instructions, included with *inter*Media Java Classes, from the appropriate readme file in the Oracle home directory as follows:

- For audio:

  ```
  <ORACLE_HOME>/ord/aud/demo/java/README.txt (on UNIX)
  <ORACLE_HOME>\ord\aud\demo\java\README.txt (on Windows NT)
  ```

- For document:

  ```
  <ORACLE_HOME>/ord/doc/demo/java/README.txt (on UNIX)
  <ORACLE_HOME>\ord\doc\demo\java\README.txt (on Windows NT)
  ```

- For image:

  ```
  <ORACLE_HOME>/ord/img/demo/java/README.txt (on UNIX)
  <ORACLE_HOME>\ord\img\demo\java\README.txt (on Windows NT)
  ```

- For video:

  ```
  <ORACLE_HOME>/ord/vid/demo/java/README.txt (on UNIX)
  <ORACLE_HOME>\ord\vid\demo\java\README.txt (on Windows NT)
  ```

- For the *inter*Media JavaServer Pages Photo Album Demo:

  ```
  <ORACLE_HOME>/ord/http/demo/jsp/README.txt (on UNIX)
  <ORACLE_HOME>\ord\http\demo\jsp\README.txt (on Windows NT)
  ```

- For the *inter*Media Java Servlet Photo Album Demo:

  ```
  <ORACLE_HOME>/ord/http/demo/servlet/README.txt (on UNIX)
  <ORACLE_HOME>\ord\http\demo\servlet\README.txt (on Windows NT)
  ```

## A.2 Before Running the Sample Files

Before you can run the Java sample files included with *inter*Media Java Classes, you must perform the following operations:

1. Install Oracle9*i* with Oracle *inter*Media.

   You must have the Oracle9*i* database server that includes Oracle *inter*Media installed on a server machine.

2. Ensure that the Java environment is correct and set up to compile and run Java programs.

3. See the appropriate readme file described in Section A.1, "Finding the Sample Files" for further instructions about running the sample files.

# B Exceptions and Errors

This appendix contains information on the exceptions and errors that can be raised by i*inter*Media Java Classes.

## B.1 Exception Class

The Exception class (and its subclasses, including SQLException) indicates conditions of interest to the user.

```
public class java.lang.Exception extends java.lang.Throwable {
     //Constructs an Exception with no detailed message
     public Exception();

     //Constructs an Exception with a detailed message
     public Exception(String s);
}
```

## B.2 IllegalArgumentException Class

The IllegalArgumentException class signals that a method has passed an invalid or inappropriate argument.

```
public class IllegalArgumentException extends RuntimeException{
     //Constructs an IllegalArgumentException with no detailed message
     public IllegalAgumentException( )
     //Constructs an IllegalArgumentExceptuon with the specified detailed
     //message
     public IllegalArgumentException(String s)
}
```

## B.3 IllegalStateException Class

The IllegalStateException class signals that a method has been invoked at an invalid or inappropriate time; the Java environment or application is not in an appropriate state for the requested operation.

```
public class IllegalStateException extends java.lang.RuntimeException {
     //Constructs an IllegalStateException with no detailed message
     public IllegalStateException();

     //Constructs an IllegalStateException with the specified detailed message
     public IllegalStateException(String s);
}
```

## B.4 IOException Class

The IOException class signals that an I/O exception of some sort has occurred.

```
public class java.io.IOException extends java.lang.Exception {
     //Constructs an IOException with no detailed message
     public IOException();

     //Constructs an IOException with the specified detailed message
     public IOException(String s);
}
```

## B.5 OutOfMemoryError Class

The OutOfMemoryError class signals that the Java Virtual Machine cannot allocate an object because it is both out of memory and unable to make more memory available through garbage collecting (that is, through deleting objects that are no longer being used).

```
public class java.lang.OutOfMemoryError extends java.lang.VirtualMachineError {
     //Constructs an OutOfMemoryError with no detailed message
     public OutOfMemoryError();

     //Constructs an OutOfMemoryError with a detailed message
     public OutOfMemoryError(string s);
}
```

## B.6 OrdHttpResponseException Class

The oracle.ord.im.OrdHttpResponse class extends ServletException to report errors encountered during the retrieval and delivery of media data from a database to an HTTP client.

## B.7 OrdHttpUploadException Class

The oracle.ord.im.OrdHttpUploadException class extends IOException to report errors encountered during the uploading of media data from an HTTP client to a database. Its primary purpose is to allow the localization of error message text.

## B.8 ServletException Class

The ServletException class defines a general exception that a servlet can throw when it encounters difficulty.

```
public class ServletException extends java.lang.Exception{
     //Constructs a new ServletException
     public ServletException( )

     //Constructs a new ServletException with the specified message
     public ServletException(String message)

     //Constructs a new ServletException with the specified message and
     //the "root cause" exception that interfered with the normal operation
     //of the servlet
     public ServletException(String message, java.lang.Throwable rootCause)

     //Constructs a new ServletException with the "root cause" exception that
     //interfered with the normal operation of the servlet
     public ServletException(java.lang.Throwable rootCause)

     //Gets the exception that caused the ServletException
     public java.lang.Throwable getRootCause( )
}
```

## B.9 SQLException Class

The SQLException class provides information on a database access error.

```
public class java.sql.SQLException extends java.lang.Exception {
     //The following four methods are public constructors:
     //Constructs a fully specified SQLException
     public SQLException(String reason, String SQLState, int vendorCode);

     //Constructs a SQLException with vendorCode value of 0
     public SQLException(String reason, String SQLState);

     //Constructs a SQLException with vendorCode value of 0 and a null SQLState
     public SQLException(String reason);

     //Constructs a SQLException with vendorCode of 0, a null SQLState. and a
     //null reason
     public SQLException();

     //The following four methods are public instance methods:

     //Gets the vendor-specific exception code
     public int getErrorCode();

     //Gets the exception connected to this one
     public SQLException getNextException();

     //Gets the SQL state
     public String getSQLState();

     //Adds a SQLException to the end
     public synchronized void setNextException(SQLException ex);
}
```

# C
# Deprecated Methods

The following list shows a list of the methods that have been deprecated since release 8.1.5 of Oracle *inter*Media Audio, Image, and Video Java Client.

- public void appendToComments(int amount, String buffer)
- public int compareComments(CLOB dest, int amount, int start_in_comment, int start_in_compare_comment )
- public CLOB copyCommentsOut(CLOB dest, int amount, int from_loc, int to_loc)
- public void deleteComments( )
- public int eraseFromComments(int amount, int offset)
- public void flush( ) throws SQLException
- public String getAllAttributesAsString(byte[ ] ctx)
- public int getAudioDuration(byte[ ] ctx)
- public int getBitRate(byte[ ] ctx)
- public int getCommentLength( )
- public String getCommentsAsString( )
- public String getCompressionType(byte[ ] ctx)
- public byte[ ] getData(String tableName, String columnName, String condition)
- public String getEncoding(byte[ ] ctx)
- public String getFormat(byte[ ] ctx)
- public int getFrameRate(byte[ ] ctx)
- public int getFrameResolution(byte[ ] ctx)

- public FrameDimension getFrameSize(byte[ ] ctx)
- public int getNumberOfChannels(byte[ ] ctx)
- public int getNumberOfColors(byte[ ] ctx)
- public int getNumberOfFrames(byte[ ] ctx)
- public int getSampleSize(byte[ ] ctx)
- public int getSamplingRate(byte[ ] ctx)
- public int getVideoDuration(byte[ ] ctx)
- public boolean loadComments(String filename)
- public void loadCommentsFromFile(String loc, String fileName, int amount, int from_loc, int to_loc)
- public boolean loadData(String fileName, String tableName, String columnName, String condition)
- public int locateInComment(String pattern, int offset, int occurrence)
- public OrdAudio(Connection the_connection)
- public OrdVideo(Connection the_connection)
- public String readFromComments(int offset, int amount)
- public void refresh(boolean forUpdate)
- public void setBindParams(String tableName, String columnName, String condition)
- public void trimComments(int newlen)
- public void writeToComments(int offset, int amount, String buffer)

# Index