Oracle8*i*™ *inter*Media Audio, Image, and Video Java Client

User's Guide and Reference

Release 8.1.5

February 1999

Part No. A67296-01

Oracle8*i* *inter*Media Audio, Image, and Video is a component of Oracle8*i* *inter*Media, a product designed to manage multimedia Web content within Oracle8*i*.

ORACLE®

Enabling the Information Age™

Oracle8*i* *inter*Media Audio, Image, and Video Java Client User's Guide and Reference

Part No. A67296-01

Release 8.1.5

Primary Author: Max Chittister

Contributors: Donna Brown, Raja Chatterjee, Ashok Joshi, Deb Laderoute, Dan Mullen, Susan Shepard, Alok Srivastava, Rod Ward

Contents

4 ORDSource Reference Information

6 ORDImage Reference Information

7 ORDVideo and FrameDimension Reference Information

8 DataSource Reference Information

A Sample Program Information

B Exceptions and Errors

C Reference Information

Glossary

Index

List of Examples

List of Figures

List of Tables

Send Us Your Comments

Oracle8*i interM*edia Audio, Image, and Video Java Client User's Guide and Reference

Release 8.1.5

Part No. A67296-01

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 chapter, section, and page number (if available). You can send comments to us in the following ways:

- e-mail: nedc_doc@us.oracle.com
- FAX - 603.897.3316. Attn: Oracle8*i interM*edia Audio, Image, and Video Java Client Documentation
- postal service:
  Oracle Corporation
  Oracle8*i interM*edia Audio, Image, and Video Java Client Documentation
  One Oracle Drive
  Nashua, NH 03062
  USA

If you would like a reply, please give your name, address, and telephone number below.

_____

_____

_____

If you have problems with the software, please contact your local Oracle World Wide Support Center.

Preface

This guide describes how to use Oracle8*i* *inter*Media Audio, Image, and Video Java Client.

Oracle8*i* *inter*Media Audio, Image, and Video Java Client requires Oracle8*i* Enterprise Edition. Oracle8*i* and Oracle8*i* Enterprise Edition have the same basic features. However, several advanced features are available only with the Enterprise Edition, and some of these features are optional.

For information about the differences between Oracle8*i* and Oracle8*i* Enterprise Edition and the features and options that are available to you, see *Getting to Know Oracle8i*.

Intended Audience

This guide is intended for developers or database administrators who are interested in storing, retrieving, and manipulating audio, image, and video data in an Oracle database, including developers of audio, image, or video specialization cartridges. Users of this guide should have experience with the Java programming language.

Structure

This guide contains the following chapters and appendixes:

Chapter 1	Contains a general introduction.
Chapter 2	Contains examples of the audio, image, and video application programming interfaces (APIs).

Chapter 3	Contains reference information on the ORDMultiMedia and BindToTableParams classes.
Chapter 4	Contains reference information on the ORDSource class.
Chapter 5	Contains reference information on the ORDAudio class.
Chapter 6	Contains reference information on the ORDImage class.
Chapter 7	Contains reference information on the ORDVideo and FrameDimension classes.
Chapter 8	Contains reference information on the DataSource class.
Appendix A	Contains information about the MediaAnnotator sample program.
Appendix B	Contains information on possible exceptions and errors.
Appendix C	Contains reference information on which methods operate on the client-side object or the server-side object.

Related Documents

Note: For information added after the release of this guide, refer to one of the online README.TXT files in your *ORACLE_HOME* directory. Depending on your operating system, the files may be named as follows:

- *$ORACLE_HOME*`/ord/im/admin/README.TXT` (general information)
- *$ORACLE_HOME*`/ord/aud/admin/README.TXT` (audio-specific information)
- *$ORACLE_HOME*`/ord/img/admin/README.TXT` (image-specific information)
- *$ORACLE_HOME*`/ord/vid/admin/README.TXT` (video-specific information)

Please see your operating-system specific installation guide for more information.

For more information on Oracle8*i* *inter*Media Audio, Image, and Video Options on the server side, see *Oracle8i interMedia Audio, Image, and Video User's Guide and Reference*.

For more information about using these data options in a development environment, see the following documents in the Oracle8*i* documentation set:

- *Oracle Call Interface Programmer's Guide*
- *Oracle8i Application Developer's Guide - Fundamentals*
- *Oracle8i Concepts*
- *PL/SQL User's Guide and Reference*

Conventions

In this guide, Oracle8*i* *inter*Media Audio, Image, and Video Java Client is sometimes referred to as *inter*Media Java Client. Oracle8*i* *inter*Media Audio, Image and Video is sometimes referred to as *inter*Media.

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

The following conventions are also used in this guide:

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

1
Introduction

Oracle8*i* *inter*Media is an extension to Oracle8*i* and Oracle8*i* Enterprise Edition. This extension uses object types to manage multimedia data. The capabilities of *inter*Media include limited image-conversion capabilities, storage, retrieval, management, and manipulation of audio, image, and video data managed by Oracle8*i*. *inter*Media supports multimedia storage, retrieval, and management of the following:

- Binary large objects (BLOBs) containing audio, image, or video data stored locally in Oracle8*i*
- File-based large objects, or BFILEs, containing audio, image, or video data stored locally in operating system-specific file systems
- URLs containing audio, image, and video data stored on any HTTP server such as the Oracle Application Server, Netscape Application Server, Microsoft Internet Information Server, Apache server, and Spyglass server
- Streaming audio and video data stored on specialized media servers such as the Oracle Video Server, RealAudio Server, and Real Video Server
- Any other specially formatted multimedia data stored in any user-defined sources on other servers

*inter*Media is designed to be extensible. It supports a base set of popular audio, image, and video data characteristics for multimedia processing that can be extended to support additional formats, **codecs**, or even specialized data processing algorithms for audio and video data.

*inter*Media is a building block for various multimedia applications rather than being an end-user application. It consists of object types and their related methods for managing and processing multimedia data. The following are some example applications for *inter*Media:

- Managing internet music stores that provide CD-quality music samplings
- Managing digital sound repositories
- Managing dictation and telephone conversation repositories
- Managing audio archives and collections
- Digital art galleries
- Real estate marketing
- Document imaging
- Photograph collections
- Managing internet video stores and digital video-clip previews
- Managing digital video sources for streaming video delivery systems
- Managing digital video libraries, archives, and repositories

These applications have certain distinct requirements and some degree of commonality. The audio, image, and video object types accommodate the commonality and support extensions that address application-specific requirements. With *inter*Media, multimedia data can be managed as easily as standard attribute data.

*inter*Media supports storage of many popular file formats, including popular desktop image, streaming audio, and video formats, in Oracle8*i* databases. *inter*Media provides the means to add audio, image, and video columns or objects to existing tables, insert and retrieve multimedia data, and provide limited processing and conversion between application formats. Database designers can extend existing application databases with multimedia data or build new end-user multimedia database applications.

1.1 If You Already Understand *inter*Media Multimedia Options

If you are already familiar with *inter*Media, you may not need to read much of the conceptual information in this chapter.

*inter*Media lets you store audio, image, or video data as objects in the database or as references to the following:

- External BFILE files
- HTTP server-based URLs
- Sources on a specialized media data server
- User-defined sources on other servers

*inter*Media lets you store and manage media data, and manipulate the following attributes:

- Audio, image, and video data local to the database
- Audio, image, and video data source information including the source type, location (that is, if the source is local), and source name
- Audio, image, and video data update time stamp
- Audio and video data description
- Audio, image, and video data format
- MIME media type of the audio, image, and video data
- Audio and video comments
- Audio characteristics: encoding type, number of channels, sampling rate, sample size, compression type, and play time (duration)
- Image characteristics: height, width, image content length, image content format, and image compression format
- Video characteristics: frame width, frame height, frame resolution, frame rate, play time (duration), number of frames, compression type, number of colors, and bit rate

1.2 Audio Concepts

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

1.2.1 Digitized Audio

Audio may be produced by an audio recorder, a microphone, digitized audio, 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 recorded on magnetic media, and convert it into digital values with specific audio characteristics. These characteristics include format, encoding type, number of channels, sampling rate, sample size, compression type, and audio duration.

*inter*Media audio provides the mechanism to integrate the storage and retrieval of audio data in Oracle databases using Oracle8*i*.

*inter*Media audio supports applications that either play or process audio data that is in a particular file format. This file format has a specific sampling rate, encoding

type, sample size, compression type, and number of channels depending upon hardware or processing power. *inter*Media audio is extensible, and therefore can support any variety of special audio characteristics.

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.

Given that an audio file may or may not include compression type and source information, the minimal attributes carried along with an audio clip may include the file format, MIME media type, encoding type, number of channels, sampling rate, sample size, and audio duration. These data attributes describe the audio data as it was recorded or produced by the digitized recording device.

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. Each audio data characteristic is crucial to audio data access and represents the audio data quality.

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 Image Concepts

This section contains information about digital images and using *inter*Media image to build image applications or specialized *inter*Media image.

1.3.1 Digitized Images

*inter*Media image can help integrate the storage and retrieval of digital images in Oracle databases using Oracle8*i*.

*inter*Media image supports two-dimensional, static, digital 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.

1.3.2 Image Components

A digital image consists of attributes that describe the characteristics of the image, and the image data itself (the digitized bits). Occasionally, image applications will associate application-specific information with an image, such as including the name of the person pictured in a photograph.

Given that an image file may or may not include compression type and source information, the minimal attributes of an image may include its size (height in scan lines and width in pixels), the resolution at which it was sampled, and the number of bits per pixel in each of the colors sampled.

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 can read and write image data using a variety of the data formats in use today. In addition, certain foreign images (formats not natively supported by *inter*Media image) have limited support.

The storage space required for digital 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.

1.3.3 Interchange Formats

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 digital imaging world and many applications that create or utilize digital images within these. *inter*Media image supports reading and writing many interchange formats.

1.4 Video Concepts

This section contains information about digitized video and using *inter*Media video to build video applications or specialized *inter*Media video.

1.4.1 Digitized Video

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

*inter*Media video can help integrate the storage, retrieval, and management of digitized video data in Oracle databases using Oracle8*i*.

*inter*Media video supports applications that either play or process video data that is in a particular file format. This file format has a specific frame rate, frame size, frame resolution, compression type, video length, bit rate, and number of colors depending upon hardware or processing power. *inter*Media video is extensible, and therefore can support any variety of special video characteristics.

1.4.2 Video Components

Digitized video components consist 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.

Given that a video file may or may not include compression type and source information, the minimal attributes carried along with a video clip may include the format, MIME media type, file format, encoding type, frame rate, frame size, frame resolution, total length of playing time, total number of frames, number of colors, and bit rate. These data attributes describe the video data as it was recorded or produced by the digitized recording device.

The video data can have different formats, compression types, frame rates, frame sizes, frame resolutions, playing times, compression types, number of colors, and bit rates depending upon how the video data was digitally recorded. Each video

data characteristic is crucial to video data access and represents the video data quality.

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.

1.5 Client-Side Support

Oracle8*i* *inter*Media Audio, Image, and Video Options let you store your multimedia information in a database table. However, in addition to storing this data, you might want to manipulate it or modify it. Oracle8*i* *inter*Media Audio, Image, and Video Java Client lets you to use local (**client-side**) applications to manipulate and/or modify multimedia data stored in a network-accessible (**server-side**) database.

Oracle8*i* *inter*Media Audio, Image, and Video Java Client lets you connect to a server-side multimedia object, copy that object from the server side to the client side, perform various operations on the client-side object, and transfer the new multimedia object back to the server side.

For situations where you do not have permission to modify the server-side object, Oracle8*i* *inter*Media Audio, Image, and Video Java Client can retrieve the multimedia data from the server side for display purposes only.

In addition, Oracle8*i* *inter*Media Audio, Image, and Video Java Client gives you the ability to integrate multimedia objects with various media frameworks, such as the Java Media Framework (JMF), RealPlayer, or Oracle Video Client (OVC).

1.6 Client/Server Similarities and Differences

Client-side objects have many similarities to the server-side objects of the same types.

For each client-side object, there must be a corresponding server-side object. The objects are identical, and the client-side object contains all the APIs that the server-side object does.

There are some architectural differences between the two, however. Oracle8*i* *inter*Media Audio, Image, and Video Java Client makes use of Java inheritance to capture the common attributes and behaviors of the ORDAudio, ORDImage, and ORDVideo objects in the ORDMultiMedia abstract superclass. The ORDAudio, ORDImage, and ORDVideo classes are subclasses of the superclass.

There are a number of attributes that are included on the client side that are not present on the server side. These include the following:

- A connection attribute that connects to the server side
- Binding attributes that are used to bind to a specific server-side object
- A lock attribute that locks the server-side object so changes on the client side will not affect the server-side object

The client side also includes some APIs not included on the server side. These APIs are used to do the following:

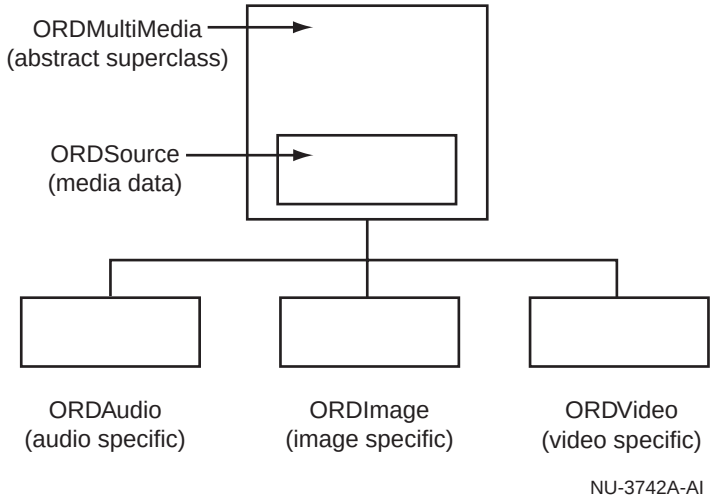
- Connect to the server-side object (setConnection( ))
- Bind to the server-side object (setBindParams( ))
- Update the client-side object from the server-side object (refresh( ))
- Update the server-side object from the client-side object (flush( ))
- Lock the server object (setLock( ))

1.7 Java Client Architecture

Oracle8*i* *inter*Media Audio, Image, and Video Java Client contains five basic classes: ORDMultiMedia, ORDSource, ORDAudio, ORDImage, and ORDVideo.

A diagram of the architecture containing these classes is shown in Figure 1–1.

Figure 1–1 Java Client Architecture



1.7.1 ORDMultiMedia Class

The ORDMultiMedia class is an abstract superclass. You will never create a ORDMultiMedia object. Instead, the ORDMultiMedia class contains methods and attributes common to ORDAudio, ORDImage, and ORDVideo.

The ORDMultiMedia class also includes two very important attributes: the binding parameters and the ORDSource object.

The **binding parameters** are the parameters used to select a server-side object to which the client-side object will be connected. This selection must be unique; that is, you cannot specify parameter values that select multiple objects and perform operations on all the objects at the same time. The binding must be in a one-to-one relationship. The binding parameters are tableName (the table in which the server-side object resides), columnName (the column in which the server-side object resides), and condition (a condition that must be met by the server-side object).

See Section 1.7.2 and Chapter 4 for more information on the ORDSource object.

1.7.2 ORDSource Class

The ORDSource object is created at the same time as the ORDMultiMedia object; you have no control over the creation of the ORDSource object.

The ORDSource object holds the digital media data.

1.7.3 ORDAudio Class

The ORDAudio class contains audio-specific attributes and methods.

As a subclass of the ORDMultiMedia class, the ORDAudio class inherits all the common attributes and methods from the ORDMultiMedia class.

1.7.4 ORDImage Class

The ORDImage class contains image-specific attributes and methods.

As a subclass of the ORDMultiMedia class, the ORDImage class inherits all the common attributes and methods from the ORDMultiMedia class.

1.7.5 ORDVideo Class

The ORDVideo class contains audio-specific attributes and methods.

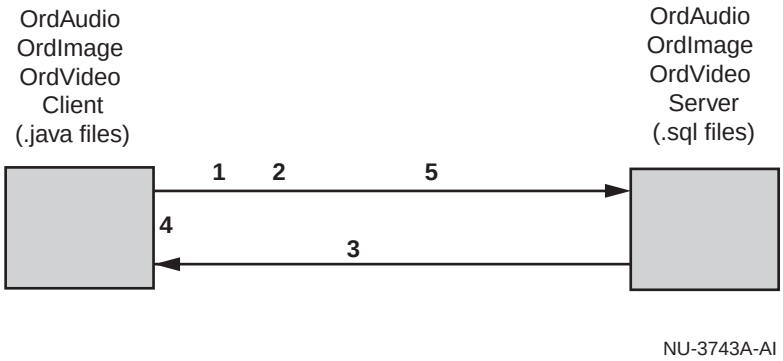
As a subclass of the ORDMultiMedia class, the ORDVideo class inherits all the common attributes and methods from the ORDMultiMedia class.

1.8 Client/Server Interaction

The client/server interaction described here assumes that you have constructed a media object on the client side.

Figure 1–2 is a diagram of the way that the client communicates with the server.

Figure 1–2 Client/Server Interaction



1. Make a connection from the client to the server with the setConnection( ) method.
2. Bind a client-side object to a matching server-side object based on the tableName, columnName, and condition variables.
3. **Refresh** the client-side object. The server sends information about the server-side object to the client-side object. The client-side object is then updated so it exactly matches the server-side object.

   It is possible to refresh in two separate modes: for read-only purposes and for update purposes. The difference between the two is the lock value. If the lock value is "false," then you do not have permission to make any changes and the object is locked. Therefore, the refresh is for read-only purposes. If the lock value is "true," then you have permission to make changes to the object.
4. Perform operations on the client-side object.

Note: If you execute a method that operates on the server-side object and changes the state of only the server-side object, you must explicitly refresh the client-side object in order to reflect that change on the client side.

---

5. **Flush** the client-side object. Once you finish using the client-side object, the client sends information about the client-side object to the server-side object. The server-side object is then updated so it exactly matches the client-side object.

   Note: If you execute a method that operates on the client-side object and changes the state of only the client-side object, you must explicitly flush the server-side object in order to reflect that change on the server-side.

   If you have set a lock, it is not automatically released upon flushing. You must release the lock manually by using the setLock( ) method. See Section 3.2.10 for more information on locking.

See Chapter 2 for examples of operations between the client and server.

1.9 Extending Oracle8*i interMedia*

1.9.1 Client-Side Extensibility

The ORDMultiMedia superclass can be extended in the same manner as any other Java class. The new class will inherit all methods and attributes from the ORDMultiMedia superclass. It will not, however, inherit any methods or attributes from the ORDAudio, ORDImage, and ORDVideo classes.

Your new class should have the following format:

```
package oracle.ord.media;
class OrdMedia extends OrdMultimedia {
     <common attributes...>
     .
     .
     .

     OrdMedia( ){
          super( );
     }

     <methods...>
     .
     .
     .
}
```

1.9.2 Server-Side Extensibility

See *Oracle8i interMedia Audio, Image, and Video User's Guide and Reference* for more information on extending *inter*Media on the server side.

Note: If you want to extend only a client-side object, you can extend it either within the media package or outside the media package. However, if you want to extend both the server-side and client-side objects, you must extend the client-side object within the media package.

2

Java Client Program Examples

This chapter provides examples that show common operations between the client and server. Examples are presented in two groups: audio/video and image.

All examples assume that you have created a database table on a server machine and the table is accessible from your client machine. For an example of creating the table, see *Oracle8i interMedia Audio, Image, and Video User's Guide and Reference*.

2.1 Example of Audio and Video APIs

Both the audio and video APIs work in the same way. Therefore, an example is provided for only the audio APIs. It can be assumed that the video APIs will behave in the same way and can be called in the same way as in the audio example.

For the audio example, the test class given here invokes the following APIs:

- Connecting to the database
- Binding to the database parameters
- Setting the properties
- Refreshing the client-side object
- Performing operations on the client-side object
  - Getting audio-related attributes
  - Making changes to the comments
  - Getting MIME type information
  - Getting source information
  - Setting and getting audio content-related information
- Flushing the client-side object

- Closing the connection

Reference information on the audio methods used in this example is presented in Chapter 5. Reference material on similar video methods is presented in Chapter 7.

2.1.1 Complete Audio Example

The following is a complete test class for invoking the audio related APIs:

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

public class demoProgram {
     public Connection connection;

     demoProgram( ) {}

     public void connect( ) throws Exception
     {
          String connectString;
          Class.forName ("oracle.jdbc.driver.OracleDriver");
          connectString = "jdbc:oracle:oci8:@";
          connection =
          DriverManager.getConnection(connectString,"ORDMEDIADEMO",
               "ORDMEDIADEMO");
          connection.setAutoCommit(false);
     }

     public static void main(String[ ] args)
     {
          byte[ ] ctx = new byte[4000];
          byte[ ] byteArray = new byte[80];
          String tableName;
          String columnName;
          String dataCondition;
          demoProgram client = new demoProgram( );

          try {
               client.connect( );
               System.out.println("after Connected");

               OrdAudio audioObj = new OrdAudio(client.connection);
```

```
System.out.println("Enter your search parameters");
System.out.println("Enter table Name");
System.in.read(byteArray);
tableName = new String(byteArray);
StringBuffer strBuffer = new StringBuffer(
     tableName.substring(0,tableName.indexOf("\n")));
tableName = new String(strBuffer);

System.out.println("Enter column Name");
System.in.read(byteArray);
columnName = new String(byteArray);
strBuffer = new StringBuffer(
     columnName.substring(0,columnName.indexOf("\n")));
columnName = new String(strBuffer);

System.out.println("Enter Data Condition");
System.in.read(byteArray);
dataCondition= new String(byteArray);
strBuffer = new StringBuffer(
     dataCondition.substring(0,dataCondition.indexOf("\n")));
dataCondition = new String(strBuffer);

audioObj.setBindParams(tableName, columnName, dataCondition);

audioObj.setProperties(ctx);

audioObj.refresh(true);

System.out.println("Encoding : " + audioObj.getEncoding(ctx));
System.out.println("Sample Size: " +
     audioObj.getSampleSize(ctx));
System.out.println("Sampling Rate: " +
     audioObj.getSamplingRate(ctx));
System.out.println("Format : " + audioObj.getFormat(ctx));

audioObj.appendToComments(5,"Drama");
System.out.println("Comments :" +
     audioObj.readFromComments(1,5));
audioObj.appendToComments(11," and Comedy");
System.out.println("Comments :" +
     audioObj.readFromComments(1,16));
System.out.println("Drama is in :" +
     audioObj.locateInComment("Drama",1,1));
System.out.println("Comment Length output  : " +
     audioObj.getCommentLength( ));
```

```
                    System.out.println("MimeType: " + audioObj.getMimeType( ));

                    System.out.println("SourceType : " + audioObj.getSourceType( ));
                    System.out.println("SourceLocation : " +
                         audioObj.getSourceLocation( ));
                    System.out.println("SourceName: " + audioObj.getSourceName( ));
                    System.out.println("SourceInformation: " +
                         audioObj.getSource( ));

                    System.out.println(" getContentLength output  : " +
                         audioObj.getContentLength(ctx));
                    audioObj.getDataInFile("output.dat");

                    audioObj.setDescription("Classic Collection");
                    System.out.println("Description : " +
                         audioObj.getDescription( ));

                    try {
                         audioObj.getCompressionType(ctx);
                    }
                    catch (Exception e) {
                         System.out.println("Exception raised in
                              getCompressionType");
                    }

                    audioObj.flush( );

                    client.connection.close( );

                    System.out.println("after close");
               }
               catch (Exception e) {
                    try {
                         System.out.println("Exception : " + e);
                         client.connection.close( );
                    } catch(Exception ex) {
                         System.out.println("Close Connection Exception : " + ex);
                    }
               }
          }
     }
```

2.1.2 Connecting to the Database

The following sections will provide more information on the preceding code. Throughout these examples, audioObj will refer to an audio object created on the client side.

Example 2–1 calls the connect( ) method, which makes a connection to the database.

Example 2–1 Connecting to the Database

```
client.connect( );
```

Example 2–2 shows the connect( ) method.

Example 2–2 The connect( ) Method

```
public void connect( ) throws Exception
{
     String connectString;
     Class.forName ("oracle.jdbc.driver.OracleDriver");
     connectString = "jdbc:oracle:oci8:@";
     connection = DriverManager.getConnection(connectString,"ORDMEDIADEMO",
          "ORDMEDIADEMO");
     connection.setAutoCommit(false);
}
```

2.1.3 Binding to the Database Parameters

Example 2–3 shows the code used to create an IPC connection. This includes obtaining the table name, column name, and data condition for the server-side object to which you will connect, as well as binding to these parameters.

Example 2–3 Binding to the Database Parameters

```
System.out.println("Enter your search parameters");
System.out.println("Enter table Name");
System.in.read(byteArray);
tableName = new String(byteArray);
StringBuffer strBuffer = new StringBuffer(
     tableName.substring(0,tableName.indexOf("\n")));
tableName = new String(strBuffer);

System.out.println("Enter column Name");
System.in.read(byteArray);
columnName = new String(byteArray);
strBuffer = new StringBuffer(
```

```
     columnName.substring(0,columnName.indexOf("\n")));
columnName = new String(strBuffer);

System.out.println("Enter Data Condition");
System.in.read(byteArray);
dataCondition= new String(byteArray);
strBuffer = new StringBuffer(
     dataCondition.substring(0,dataCondition.indexOf("\n")));
dataCondition = new String(strBuffer);

audioObj.setBindParams(tableName, columnName, dataCondition);
```

2.1.4 Setting the Properties

Example 2–4 sets the properties on the server side.

Example 2–4 Setting the Properties

```
audioObj.setProperties(ctx);
```

where:

- ctx: contains the context information of the caller. See Chapter 5 for more information on ctx.

2.1.5 Refreshing the Client-Side Object

Example 2–5 refreshes the client-side object with the attribute values from the client-side object.

Example 2–5 Refreshing the Client-Side Object

```
audioObj.refresh(true);
```

where:

- true: indicates that the object is refreshed for update.

2.1.6 Performing Basic Operations on the Client Object

The following examples demonstrate how the Java client program performs basic operations upon the ORDAudio object. Not all operations included in the complete example in Section 2.1.1 are shown here.

Example 2–6 gets the encoding, sample size, sampling rate, and format of the audio data.

Example 2–6 Getting Audio Attributes

```
System.out.println("Encoding : " + audioObj.getEncoding(ctx));
System.out.println("Sample Size: " + audioObj.getSampleSize(ctx));
System.out.println("Sampling Rate: " + audioObj.getSamplingRate(ctx));
System.out.println("Format : " + audioObj.getFormat(ctx));
```

where:

- ctx: contains the context information of the caller. See Chapter 5 for more information on ctx.

Example 2–7 performs operations upon the comments information.

Example 2–7 Comments Operations

```
audioObj.appendToComments(5,"Drama");
System.out.println("Comments :" + audioObj.readFromComments(1,5));
audioObj.appendToComments(11," and Comedy");
System.out.println("Comments :" + audioObj.readFromComments(1,16));
System.out.println("Drama is in :" + audioObj.locateInComment("Drama",1,1));
System.out.println("Comment Length output  : " + audioObj.getCommentLength( ));
```

See “appendToComments( ) Method”, “readFromComments( ) Method”, “locateInComment( ) Method”, and “getCommentLength( ) Method”, all of which are in Chapter 5, for more information.

Example 2–8 gets the MIME type information.

Example 2–8 Getting the MIME Type Information

```
System.out.println("MimeType: " +audioObj.getMimeType( ));
```

Example 2–9 gets source information.

Example 2–9 Getting Source Information

```
System.out.println("SourceType : " + audioObj.getSourceType( ));
System.out.println("SourceLocation : " + audioObj.getSourceLocation( ));
System.out.println("SourceName: " + audioObj.getSourceName( ));
System.out.println("SourceInformation: " +audioObj.getSource( ));
```

Example 2–10 sets and gets information pertaining to the audio content.

Example 2–10 Setting and Getting Information Pertaining to the Audio Content

```
System.out.println("getContentLength output : " +
     audioObj.getContentLength(ctx));
audioObj.getDataInFile("output.dat");
audioObj.setDescription("Classic Collection");
System.out.println("Description : " + audioObj.getDescription( ));
```

where:

- ctx: contains the context information of the caller. See Chapter 5 for more information on ctx.
- output.dat: is the file from which the data will be read.
- Classic Collection: is the description of the audio data to be set.

2.1.7 Flushing the Client Object

Example 2–11 flushes the client object to the server side.

Example 2–11 Flushing the Client Object

```
audioObj.flush( );
```

2.1.8 Closing the Connection

Example 2–12 closes the connection to the database.

Example 2–12 Closing the Connection

```
client.connection.close( );
```

2.2 Example of Image APIs

For the image example, the test class given here invokes the following APIs:

- Connecting to the database
- Binding to the database parameters
- Refreshing the client-side object

- Performing operations on the client object
  - Getting image-related attributes
  - Getting the content BFILE
  - Copying image data
  - Getting the content BLOB
  - Getting all image related attributes
  - Checking properties
  - Processing commands on the server side
  - Setting the MIME type
- Setting the server-side properties
- Flushing the client-side object
- Closing the connection

Reference information on the image methods used in this example is presented in Chapter 6.

2.2.1 Complete Image Example

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

public class doc {
     public Connection con;

     doc( ) {}

     // Get connection to server
     //
     public void connect( ) throws Exception
     {
          String connectString;
          Class.forName ("oracle.jdbc.driver.OracleDriver");
          connectString = "jdbc:oracle:oci8:@"; // for IPC
          DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver( ));
          con = DriverManager.getConnection(connectString,"SCOTT", "TIGER");
```

```
     con.setAutoCommit(false);
}

// main method: calls various OrdImage methods
//
public static void main(String[ ] args)
{
     doc cl = new doc( );
     byte[ ] ctx = new byte[4000];
     try {
          cl.connect( );

          // Create new client-side OrdImage object
          //
          OrdImage imgObj = new OrdImage(cl.con);

          // Specify table, column, and "where" clause to use when
          // fetching data into imgObj.
          //
          imgObj.setBindParams("ordimgtab", "imagebfile", " ID = 1 ");

          //Populate imgObj attributes with values from the database,
          //locking the row for update.
          //
          imgObj.refresh(true);

          // Display some attribute values.
          //
          System.out.println("ContentFormat : " +
               imgObj.getContentFormat( ));
          System.out.println("FileFormat : " + imgObj.getFormat( ));

          // Get the BFILE object on which imgObj is based, and display
          // the name of the file containing the image data.
          //
          BFILE bfile = imgObj.getBFILE( );
          String fileNameStr = bfile.getName( );
          if (fileNameStr.length( ) > 0)
          {
               System.out.println("bfile.getName = ");
               System.out.println(fileNameStr);
          }

          // Create and populate another client-side OrdImage object.
          //
```

```
OrdImage imgObj2 = new OrdImage(cl.con);
imgObj2.setBindParams("ordimgtab", "imageblob", " ID = 1 ");
imgObj2.refresh(true);

// Copy the contents of imgObj to imgObj2.
//
imgObj.copy(imgObj2);

//Import data from BFILE into BLOB in source of server-side
//object.
//
imgObj2.importData(ctx);
imgObj2.setLocal( );

// Get the BLOB containing image data and print its length.
//
BLOB blob = imgObj2.getContent( );
System.out.println("blob.length = " + blob.length( ));

// Set server object properties from the data in the BLOB.
//
imgObj2.setProperties( );

// Display new attribute values.
//
System.out.println("fileFormat : " + imgObj2.getFormat( ));
System.out.println("width : " + imgObj2.getWidth( ));
System.out.println("height : " + imgObj2.getHeight( ));
System.out.println("contentFormat : " +
     imgObj2.getContentFormat( ));
System.out.println("compressionFormat : " +
     imgObj2.getCompressionFormat( ));

// Get all attribute settings in a single string.
//
System.out.println("getAllAttributesAsString : " +
     imgObj2.getAllAttributesAsString( ));

// Check that attribute values match those in BLOB data.
//
imgObj2.setProperties( );
if (imgObj2.checkProperties( ) == true)
     System.out.println("Attribute and data properties match.");
else
     System.out.println("Attribute and data properties do not
```

```
                    match.");

          // Make modifications to image data based on command in
          //cmdStr.
          //
          String cmdStr = "scale=2 cut=100 100 100 100";
          imgObj2.process(cmdStr);
          imgObj2.setProperties( );
          System.out.println("After imgObj2.process(scale=2 cut=100 100
               100 100)");
          System.out.println("width : " + imgObj2.getWidth( ));
          System.out.println("height : " + imgObj2.getHeight( ));

          // Load image data from local client file into imgObj2 BLOB.
               if (imgObj2.loadData("/OraHome/ord/img/demo/imgdemo.dat"))
          {
               System.out.println("loadData succeeded");
               blob = imgObj2.getContent( );
               System.out.println("blob.length = " + blob.length( ));
               imgObj2.setProperties( );
          }
          else
               System.out.println("loadData failed ***");
     }
     catch (Exception e)
     {
          System.out.println("Exception : " + e);
     }

     //Set the MIME type of imgObj2.
     imgObj2.setMimeType("image/bmp");

     //Save the new MIME type setting in the server object.
     imgObj2.flush( );

     finally
     {
          // Always close the connection before exiting.
          //
          try
          {
               cl.con.close( );
          }
          catch(Exception ex)
          {
```

```
                    System.out.println("Close Connection Exception : " + ex);
                }
            }
        }
}
```

2.2.2 Connecting to the Database

The following sections will provide more information about the preceding code. Throughout these examples, imgObj and imgObj2 will refer to ORDImage objects created on the client side.

Example 2–13 calls the connect( ) method, which makes a connection to the database.

Example 2–13 Connecting to the Database

```
doc cl = new doc( );
.
.
.
cl.connect( );
```

Example 2–14 shows the connect( ) method.

Example 2–14 The connect( ) Method

```
public void connect( ) throws Exception
{
     String connectString;
     Class.forName ("oracle.jdbc.driver.OracleDriver");
     connectString = "jdbc:oracle:oci8:@"; // for IPC
     DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver( ));
     con = DriverManager.getConnection(connectString,"SCOTT", "TIGER");
     con.setAutoCommit(false);
}
```

where:

- SCOTT: is your username for connecting to the Oracle8*i* database.
- TIGER: is your password for connecting to the Oracle8*i* database.

2.2.3 Binding to the Database Parameters

Example 2–15 shows the code used to bind to the database parameters. This includes obtaining the table name, column name, and data condition for the server-side object to which you will connect, as well as binding to the parameters.

Example 2–15 Binding to the Database Parameters

```
imgObj.setBindParams("ordimgtab", "imagebfile", " ID = 1 ");
```

where:

- ordimgtab: is the name of the table to which the client object will be bound.
- imgbfile: is the name of the column to which the client will be bound.
- ID = 1: is the condition to determine the row to which the client will be bound.

2.2.4 Refreshing the Client-Side Object

Example 2–16 refreshes the client-side object.

Example 2–16 Refreshing the Client-Side Object

```
imgObj.refresh(true);
```

where:

- true: indicates that the object is refreshed for update.

2.2.5 Performing Basic Operations on the Client Object

The following examples demonstrate how the Java client program performs some basic operations upon the ORDImage object. Not all operations included in the complete example in Section 2.2.1 are shown here.

Example 2–17 gets the content format and file format of the image data.

Example 2–17 Getting Specific Image-Related Attributes

```
System.out.println("contentFormat : " + imgObj2.getContentFormat( ));
System.out.println("fileFormat : " + imgObj2.getFormat( ));
```

Example 2–18 gets a content BFILE that holds image data and prints the file name to the screen.

Example 2–18 Getting the Content BFILE

```
BFILE bfile = imgObj.getBFILE( );
String fileNameStr = bfile.getName( );
if (fileNameStr.length( ) > 0)
{
     System.out.println("bfile.getName = ");
     System.out.println(fileNameStr);
}
```

Example 2–19 copies the image data from the client-side ORDImage object source to the server-side ORDImage object source.

Example 2–19 Copying Image Data

```
imgObj.copy(imgObj2);
```

Example 2–20 gets a content BLOB that holds image data.

Example 2–20 Getting the Content BLOB

```
BLOB blob = imgObj2.getContent( );
```

Example 2–21 gets all the attributes of the image data as a String.

Example 2–21 Getting All Image-Related Attributes as a String

```
System.out.println("getAllAttributesAsString : " +
     imgObj2.getAllAttributesAsString( ));
```

Example 2–22 checks that the values of the database ORDImage object are consistent with the values stored in the content header.

Example 2–22 Checking Properties

```
imgObj2.setProperties( );
if (imgObj2.checkProperties( ) == true)
     System.out.println("Attribute and data properties match");
else
     System.out.println("Attribute and data properties do not match");
```

Example 2–23 executes the given commands on the server-side ORDImage object; that is, it makes the modifications included in cmdStr on the image data stored in imgObj2. See "process( ) Method" in Chapter 5 for more information.

Example 2–23 Processing Commands on the Server-Side Object

```
String cmdStr = "scale=2, cut=100 100 100 100";
imgObj2.process(cmdStr);
```

Example 2–24 sets the MIME type of the client-side ORDImage object.

Example 2–24 Setting the MIME Type

```
imgObj2.setMimeType("image/bmp");
```

where:

- image/bmp: is the MIME type to be set.

2.2.6 Setting the Properties

Example 2–25 updates the server-side ORDImage object with the setProperties( ) method, which updates the server-side ORDImage attributes to reflect the changes made on the client side.

Example 2–25 Setting the Properties

```
imgObj2.setProperties( );
```

2.2.7 Flushing the Client Object

Example 2–26 updates the server-side ORDImage object by flushing the client object to the server side.

Example 2–26 Flushing the Client Object

```
imgObj2.flush( );
```

2.2.8 Closing the Connection

Example 2–27 closes the connection to the database.

Example 2–27 Closing the Connection

```
cl.con.close( );
```

3

ORDMultiMedia and BindToTableParams Reference Information

Oracle8*i* *inter*Media Audio, Image, and Video Java Client contains information about the ORDMultiMedia:

- object type -- see "ORDMultiMedia Object Type" in Section 3.1.
- methods -- see Section 3.2.

Oracle8*i* *inter*Media Audio, Image, and Video Java Client contains information about the BindToTableParams:

- object type -- see "BindToTableParams Object Type" in Section 3.1.
- methods -- see Section 3.3.

Methods invoked at the ORDMultiMedia level that are handed off for processing to the server-side source plug-in have `byte[ ] ctx` as a context parameter. The space for the parameter is created by the client (in the reference examples, 4000 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.

> **Note:** In the current release, not all source plug-ins will use or generate the context parameter, but if you include the parameter as previously described, your application should work with any current or future source plug-in.

See *Oracle8i interMedia Audio, Image, and Video User's Guide and Reference* for more information.

3.1 Object Types

Oracle8*i* *inter*Media Audio, Image, and Video Java Client describes the ORDMultiMedia object type, which is the abstract superclass for all the other objects, namely ORDAudio, ORDImage, and ORDVideo.

It also describes the BindToTableParams object type, which contains information about binding a client-side object to a database object.

ORDMultiMedia Object Type

The ORDMultiMedia object type contains the common attributes and methods of the ORDAudio, ORDImage, and ORDVideo object types. It is an abstract superclass of ORDAudio, ORDImage, and ORDVideo.

> **Note:** In the following code, some methods will begin with *public*. This indicates that the method can be called by all classes in all packages. Other methods are designated *protected*, which means that the method can be called by all classes in the package, plus all subclasses anywhere. The methods that are not designated as public have the default protection, which means that they can be called only by other methods in the package.

The object type is defined as follows:

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

public abstract class OrdMultiMedia {
     BindToTableParams bindParams;
     String mediaType; // OrdAudio, OrdVideo, OrdImage
     Connection connection;

     OrdSource source;
     String mimeType = "";
     String format = "" ;

     protected OrdMultiMedia( )
     { }

     abstract String getMediaType( );
     { }

     abstract String getFormatStr( )
     { }

     abstract String getUpdStr( )
```

```
{ }

abstract String getSourceStr( )
{ }

abstract String getContentLengthAPI( )
{ }

public abstract int getContentLength( )
throws SQLException
{ }

protected void setConnection(Connection the_connection)
{ }

protected Connection getConnection( )
{ }

public void setBindParams(String tableName, String columnName,
     String condition)
{ }

String getSQLConstructor(boolean updateOption, String objName)
{ }

String defineSQLResults(String var)
{ }

int declareSQLResults(int start, OracleCallableStatement stmt)
throws SQLException
{ }

int getSQLResults(int start, OracleCallableStatement stmt)
throws SQLException
{ }

String setSQLParams(String var)
{ }

int bindInSQLParams(int start, OracleCallableStatement stmt)
throws SQLException
{ }

public abstract void refresh(boolean forUpdate)
throws SQLException
```

```
{ }

void setLock(boolean lockOption)
{ }

boolean isLocked( )
{ }

String getUpdateStr(String objName)
{ }

public abstract void flush( )
throws SQLException
{ }

String getPlSqlStmtBlockTemplate(boolean updateOption, String statement,
     String objName)
{ }

public String getMimeType( )
{ }

public void setMimeType(String the_mimeType)
{ }

public Timestamp getUpdateTime( )
{ }

public void setUpdateTime( )
throws SQLException
{ }

public void setFormat(String the_format)
{ }

public String getFormat( )
{ }

OrdSource getSourceObject( )
{ }

public String getSource( )
{ }

public void setSource(String type, String loc, String name)
```

```
{ }

public String getSourceType( )
{ }

public String getSourceLocation( )
{ }

public String getSourceName( )
{ }

public void importData(byte[ ] ctx)
throws SQLException
{ }

public void importFrom(byte[ ] ctx, String type, String loc,
     String name)
throws SQLException
{ }

public void export(byte[ ] ctx, String type, String loc, String name)
throws SQLException
{ }

public BLOB getContent( )
throws SQLException
{ }

public int getContentLength(byte[ ] ctx)
throws SQLException
{ }

public void deleteContent( )
throws SQLException
{ }

public byte[ ] getData(String tableName, String columnName,
     String condition)
throws SQLException, OutOfMemoryError
{ }

public byte[ ] getData( )
throws SQLException, OutOfMemoryError
{ }
```

```
    public void getDataInFile(String fileName)
    throws SQLException, IOException
    { }

    public InputStream getDataInStream( )
    throws SQLException
    { }

    public boolean loadData(String fileName, String tableName,
         String columnName, String condition)
    throws SQLException, FileNotFoundException, IOException
    { }

    public boolean loadData(String filename)
    throws SQLException, IOException, SecurityException
    { }

    boolean loadDataInChunks(String filename, int fileLength)
    throws SQLException, IOException
    { }

    public void setLocal( )
    { }

    public void clearLocal( )
    { }

    public boolean isLocal( )
    { }

    public int getContentLength( )
    throws SQLException
    { }

    public BFILE getBFILE( )
    throws SQLException
    { }

    public void setSourceInformation(String sourceType,
         String sourceLocation, String sourceName)
    { }
}
```

where the class attributes are defined as:

- bindParams: contains the parameters used to bind the client-side object to a server-side object. For more information on the BindToTableParams class, see "BindToTableParams Object Type" and Section 3.3.
- mediaType: defines the media type of the multimedia object (either ORDAudio, ORDImage, or ORDVideo).
- connection: is the connection to the database.
- source: is the source information of the multimedia object. It is of type ORDSource. For more information on the ORDSource object, see Chapter 4.
- mimeType: is the MIME type of the multimedia data.
- format: is the format of the multimedia data.

BindToTableParams Object Type

The BindToTableParams object type contains information about binding a client-side object to a database object.

> **Note:** In the following code, the methods begin with *public*. This indicates that the method can be called by all classes in all packages.

The object type is defined as follows:

```
package oracle.ord.media;

public class BindToTableParams {
     String tableName;
     String columnName;
     String dataCondition;

     public BindToTableParams( )
     { }

     public String getTableName( )
     { }

     public String getColumnName( )
     { }

     public String getDataCondition( )
     { }

     public void setTableName(String the_tableName)
     { }

     public void setColumnName(String the_columnName)
     { }

     public void setDataCondition(String the_dataCondition)
     { }
}
```

where the class attributes are defined as:

- tableName: is the name of the table to which you are binding.
- columnName: is the name of the column to which you are binding.
- dataCondition: is the data condition that selects a specific row in the selected column. This condition must be unique.

3.2 ORDMultiMedia Methods

This section presents reference information on the methods contained in the ORDMultiMedia superclass. These methods are described in the following groupings:

ORDMultiMedia Methods Associated with the bindParams Attribute

- public void setBindParams( ): set the data location parameters.

ORDMultiMedia Methods Associated with the connection Attribute

- protected void setConnection( ): set the connection with the database.
- protected Connection getConnection( ): get the connection with the database.

ORDMultiMedia Methods Associated with the source Attribute

- OrdSource getSourceObject( ): get the client-side OrdSource object.
- public void setSource( ): set the client-side ORDMultiMedia object source information, which is of the form srcType://srcLocation/srcName.
- public String getSource( ): get the client-side ORDMultiMedia object source information, which is of the form srcType://srcLocation/srcName.
- public void setSourceInformation( ): sets the source-related attribute values of the client-side ORDMultiMedia object .
- public String getSourceType( ): get the client-side ORDMultiMedia object source type information.
- public String getSourceLocation( ): get the client-side ORDMultiMedia object source location information.
- public String getSourceName( ): get the client-side ORDMultiMedia object source name information.
- public void importData( ): import data from the source information provided in the appropriate attributes of the server-side ORDMultiMedia object into the server-side ORDMultiMedia object.
- public void importFrom( ): import data from the source information provided in the appropriate parameters of the server-side ORDMultiMedia object into the server-side ORDMultiMedia object.
- public void export( ): export the data in the server-side ORDMultiMedia object localData attribute to the target specified in the parameters. Currently not supported at the server side.

- public BLOB getContent( ): copy the binary large object (BLOB) from the server side to the client side and return the LOB locator from the client cache.
- public int getContentLength(byte[ ]): return the content length of the media data.
- public int getContentLength( ): get the content length of the source localData attribute.
- public void deleteContent( ): delete the media data in the server-side ORDMultiMedia object.
- public void setLocal( ): set the client-side ORDMultiMedia object source local attribute to *true*.
- public void clearLocal( ): set the client-side ORDMultiMedia object source local setting to *false*.
- public boolean isLocal( ): get the client-side ORDMultiMedia object local attribute.
- public BFILE getBFILE( ): get the server-side ORDMultiMedia object source BFILE attribute.

ORDMultiMedia Methods Associated with the mimeType Attribute

- public void setMimeType( ): set the MIME type of the client-side ORDMultiMedia object.
- public String getMimeType( ): get the MIME type of the client-side ORDMultiMedia object.

ORDMultiMedia Methods Associated with the format Attribute

- public void setFormat( ): set the format of the client-side ORDMultiMedia object.
- public String getFormat( ): get the format of the client-side ORDMultiMedia object.

ORDMultiMedia Methods Associated with the updateTime attribute

- public void setUpdateTime( ): sets the source updateTime attribute to the current time in both the server-side and client-side ORDMultiMedia objects.
- public Timestamp getUpdateTime( ): return the value of the client-side ORDMultiMedia object source updateTime attribute.

ORDMultiMedia Methods Associated with the Media Type

- abstract String getMediaType( ): return the media type information. This is a virtual method that is defined in the ORDAudio, ORDImage, and ORDVideo subclasses.

ORDMultiMedia Methods Associated with the Content Length

- public abstract int getContentLength( ): return the content length. This is a virtual method that is defined in the ORDAudio and ORDVideo subclasses.

ORDMultiMedia Methods Associated with Communication Between the Client and Server

- public abstract void refresh( ): send information from the server-side media object to the client-side media object with or without locking the database row. This is a virtual method that is defined in the ORDAudio, ORDImage, and ORDVideo subclasses.
- public abstract void flush( ): send information from the client-side media object to the server-side media object. This is a virtual method that is defined in the ORDAudio, ORDImage, and ORDVideo subclasses.
- String getUpdateStr( ): update the ORDMultiMedia object.

ORDMultiMedia Methods Associated with Locking

- void setLock( ): set the lock.
- boolean isLocked( ): get the lock value.

ORDMultiMedia Methods Associated with the SQL Type

- String getSQLConstructor( ): get the SQL type constructor for the ORDMultiMedia object; the type can be either ORDAudio, ORDImage, or ORDVideo.
- String defineSQLResults( ): define the SQL results that will be used to assign values to the ORDMultiMedia object attributes.
- int declareSQLResults( ): declare types for the SQL results corresponding to the different ORDMultiMedia object attributes.
- int getSQLResults( ): get the SQL results to assign to the attributes of the ORDMultiMedia object (mimeType or format, for example).
- String setSQLParams( ): set the values of the SQL type of the ORDMultiMedia object as parameters.

- int bindInSQLParams( ): bind the values of the SQL type of the ORDMultiMedia object.
- String getPlSqlStmtBlockTemplate( ): return a code template for executing a PL/SQL statement.

ORDMultiMedia Methods Associated with Generating SQL Queries

- abstract String getFormatStr( ): return the format String. This is a virtual method that is defined in the ORDAudio, ORDImage, and ORDVideo subclasses.
- abstract String getUpdStr( ): returns the SQL string that is used to update an instance. This is a virtual method that is defined in the ORDAudio, ORDImage, and ORDVideo subclasses.
- abstract String getSourceStr( ): returns the source String. This is a virtual method that is defined in the ORDAudio, ORDImage, and ORDVideo subclasses.
- abstract String getContentLengthAPI( ): returns the server-side contentLength API for this class. This is a virtual method that is defined in the ORDAudio, ORDImage, and ORDVideo subclasses.

ORDMultiMedia Methods Associated with the Media Data

- public byte[ ] getData(String, String, String): return the BLOB data of the server-side ORDMultiMedia object as a byte array, given tableName, columnName, and condition.
- public byte[ ] getData( ): return the BLOB data of the server-side ORDMultiMedia object as a byte array.
- public void getDataInFile( ): dump the server-side ORDMultiMedia object BLOB data into a file whose name is provided by the caller.
- public InputStream getDataInStream( ): return the server-side ORDMultiMedia object media data as an InputStream object.
- public boolean loadData(String, String, String): load the server-side ORDMultiMedia object media data into a BLOB identified by tableName, columnName, and condition.
- public boolean loadData(String): load the server-side ORDMultiMedia object media data from a given file into a media object designated by the corresponding binding parameters of the ORDMultiMedia object.

- boolean loadDataInChunks( ): load the server-side ORDMultiMedia object media data in chunks of 32K bytes at a time from a given file into a media object designated by the binding parameters.

3.2.1 ORDMultiMedia Methods Associated with the bindParams Attribute

This section presents reference information on the ORDMultiMedia methods associated with the bindParams attribute.

The methods described in this reference chapter show examples based on the instantiation of an ORDMultiMedia object. For the examples in Section 3.2.1 through Section 3.2.13, please consult the following code:

```
import oracle.ord.media.mediaClass;

public class clientProgram {
     public static void main(String[ ] args){
          The code in the examples appears here
     }
}
```

mediaClass can add features to audio, image, or video.

setBindParams( ) Method

Format

```
public void setBindParams(String tableName, String columnName,
                          String condition)
```

Scope

public

Description

Sets the data location parameters, also known as binding parameters, which will be used to bind the client-side proxy object to the server-side object.

The client-side proxy object is bound to a server-side object that resides in an object of a relational table. This method sets the binding parameters for the proxy object with the server-side object.

The parameters must point to a unique *inter*Media object in the database.

Parameters

tableName
The name of the table to which the object will be bound.

columnName
The name of the column in the table to which the object will be bound. The column must contain *inter*Media objects.

condition
The select condition on the table, used to identify the row to which the object will be bound.

Returns

None.

Exceptions

None.

Usage

```
mediaClass mediaObject = new mediaObject(connection);
.
.
.
mediaObject.setBindParams("audioTable", "song", "songID = 3");
```

where:

- audioTable: is the table to which the object will be bound.
- song: is the column to which the ORDAudio object will be bound.
- songID = 3: is the selection condition on the table, which identifies the row to which the object will be bound.

3.2.2 ORDMultiMedia Methods Associated with the connection Attribute

This section presents reference information on the ORDMultiMedia methods associated with the connection attribute.

See Section 3.2.1 for additional code required to run the example code.

setConnection( ) Method

Format

```
protected void setConnection(Connection the_connection)
```

Scope

protected

Description

Sets the connection with the database.

Parameter

the_connection
The connection instance.

Returns

None.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

getConnection( ) Method

Format

```
protected Connection getConnection( )
```

Scope

protected

Description

Gets the connection with the database.

Parameter

None.

Returns

This method returns the connection attribute.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

3.2.3 ORDMultiMedia Methods Associated with the source Attribute

This section presents reference information on the ORDMultiMedia methods associated with the source attribute.

See Section 3.2.1 for additional code required to run the example code.

getSourceObject( ) Method

Format

```
OrdSource getSourceObject( )
```

Scope

package

Description

Gets the client-side ORDSource object.

Parameter

None.

Returns

This method returns the ORDSource attribute.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

setSource( ) Method

Format

```
public void setSource(String type, String loc, String name)
```

Scope

public

Description

Sets the client-side ORDMultiMedia object source information, which is of the form srcType://srcLocation/srcName.

Parameters

type
The source type (for example, FILE or HTTP).

loc
The source location.

name
The source name.

Returns

None.

Exceptions

None.

Usage

```
mediaClass mediaObject = new mediaObject(connection);
.
.
.
mediaObject.setSource("file", "ORDIMGDIR", "jdoe.gif");
```

where:

- file: is the source type.
- ORDIMGDIR: is the source location.
- jdoe.gif: is the source name.

getSource( ) Method

Format

```
public String getSource( )
```

Scope

public

Description

Gets the client-side ORDMultiMedia object source information, which is of the form srcType://srcLocation/srcName.

Parameter

None.

Returns

This method returns the source information.

Exceptions

None.

Usage

```
mediaClass mediaObject = new mediaObject(connection);
.
.
.
String srcStr = mediaObject.getSource( );
```

setSourceInformation( ) Method

Format

```
public void setSourceInformation(String sourceType, String sourceLocation,
                                 String sourceName)
```

Scope

public

Description

Sets the source-related attribute values of the client-side ORDMultiMedia object.

Parameters

sourceType
The type of the source.

sourceLocation
The location of the source.

sourceName
The name of the source.

Returns

None.

Exceptions

None.

Usage

```
mediaClass mediaObject = new mediaObject(connection);
.
.
.
mediaObject.setSourceInformation("file", "ORDIMGDIR", "jdoe.gif");
```

where:

- file: is the type of the source.
- ORDIMGDIR: is the location of the source.
- jdoe.gif: is the name of the source.

getSourceType( ) Method

Format

```
public String getSourceType( )
```

Scope

public

Description

Gets the source type information of the client-side ORDMultiMedia object.

Parameter

None.

Returns

This method returns the source type information.

Exceptions

None.

Usage

```
mediaClass mediaObject = new mediaObject(connection);
.
.
.
String srcType = mediaObject.getSourceType( );
```

getSourceLocation( ) Method

Format

```
public String getSourceLocation( )
```

Scope

public

Description

Gets the source location information of the client-side ORDMultiMedia object.

Parameter

None.

Returns

This method returns the source location information.

Exceptions

None.

Usage

```
mediaClass mediaObject = new mediaObject(connection);
.
.
.
String str = mediaObject.getSourceLocation( );
```

getSourceName( ) Method

Format

```
public String getSourceName( )
```

Scope

public

Description

Gets the source name information of the client-side ORDMultiMedia object.

Parameter

None.

Returns

This method returns the source name information.

Exceptions

None.

Usage

```
mediaClass mediaObject = new mediaObject(connection);
.
.
.
String str = mediaObject.getSourceName( );
```

importData( ) Method

Format

```
public void importData(byte[ ] ctx)
throws SQLException
```

Scope

public

Description

Imports data from the source information into the server-side ORDMultiMedia object. The source information is provided in the attributes of the server-side ORD-MultiMedia object.

Parameter

ctx
The source plug-in context information.

Returns

None.

Exceptions

java.sql.SQLException

Usage

```
try{
     byte[ ] ctx = new byte[4000];
     mediaClass mediaObject = new mediaObject(connection);
     .
     .
     .
     mediaObject.importData(ctx);
}
catch(SQLException e){
     .
     .
```

.
}

where:

- ctx: contains the context information of the caller.

importFrom( ) Method

Format

```
public void importFrom(byte[ ] ctx, String type, String loc, String name)
throws SQLException
```

Scope

public

Description

Imports data from the source information into the server-side ORDMultiMedia object. The source information in provided in the parameters.

Parameters

ctx
The source plug-in context information.

type
The source type.

loc
The source location.

name
The source name.

Returns

None.

Exceptions

java.sql.SQLException

Usage

```
try{
     mediaClass mediaObject = new mediaObject(connection);
     byte[ ] ctx = new byte[4000];
```

```
      .
      .
      .
      mediaObject.importFrom(ctx, "HTTP", "directory1", "file");
}
catch(SQLException e){
      .
      .
      .
}
```

where:

- ctx: contains the context information of the caller.
- HTTP: is the source type.
- directory1: is the source location.
- file: is the source name.

export( ) Method

Format

```
public void export(byte[ ] ctx, String type, String loc, String name)
throws SQLException
```

Scope

public

Description

Exports the data in the server-side ORDMultiMedia object localData attribute to the target specified in the parameters.

Currently this method is not supported at the server side, so the client raises an exception.

Parameters

ctx
The source plug-in context information.

type
The source type.

loc
The source location.

name
The source name.

Returns

None.

Exceptions

java.sql.SQLException

Usage

```
try{
     mediaClass mediaObject = new mediaObject(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     mediaObject.export(ctx, "HTTP", "directory1", "file");
}
catch(SQLException e){
     .
     .
     .
}
```

where:

- ctx: contains the context information of the caller.
- HTTP: is the source type.
- directory1: is the source location.
- file: is the source name.

getContent( ) Method

Format

```
public BLOB getContent( )
throws SQLException
```

Scope

public

Description

Copy the BLOB from the server side to the client side and return the LOB locator from the client cache.

Parameter

None.

Returns

This method returns the LOB locator where the content of the multimedia object is stored.

Exceptions

java.sql.SQLException

Usage

```
try{
     mediaClass mediaObject = new mediaObject(connection);
     .
     .
     .
     BLOB blob = mediaObject.getContent( );
}
catch(SQLException e){
     .
     .
     .
}
```

getContentLength(byte[ ]) Method

Format

```
public int getContentLength(byte[ ] ctx)
throws SQLException
```

Scope

public

Description

Returns the content length of the media data.

Parameter

ctx
The source plug-in context information.

Returns

This method returns the length of the data content of the media object.

Exceptions

java.sql.SQLException

Usage

```
try{
     mediaClass mediaObject = new mediaObject(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     int length = mediaObject.getContentLength(ctx);
}
catch(SQLException e){
     .
     .
     .
}
```

where:

- ctx: contains the context information of the caller.

getContentLength( ) Method

Format

```
public int getContentLength( )
throws SQLException
```

Scope

public

Description

Gets the content length of the source localData attribute.

Parameter

None.

Returns

This method returns the content length of the source data.

Exceptions

java.sql.SQLException

Usage

```
try{
     mediaClass mediaObject = new mediaObject(connection);
     .
     .
     .
     int i = mediaObject.getContentLength( );
}
catch(SQLException e){
     .
     .
     .
}
```

deleteContent( ) Method

Format

```
public void deleteContent( )
throws SQLException
```

Scope

public

Description

Deletes the media data in the server-side ORDMultiMedia object.

Parameter

None.

Returns

None.

Exceptions

java.sql.SQLException

Usage

```
try{
     mediaClass mediaObject = new mediaObject(connection);
     .
     .
     .
     mediaObject.deleteContent( );
}
catch(SQLException e){
     .
     .
     .
}
```

setLocal( ) Method

Format

```
public void setLocal( )
```

Scope

public

Description

Sets the client-side ORDMultiMedia object source local attribute to *true*, which indicates that the localData attribute is valid.

Parameter

None.

Returns

None.

Exceptions

None.

Usage

```
mediaClass mediaObject = new mediaObject(connection);
.
.
.
mediaObject.setLocal( );
```

clearLocal( ) Method

Format

```
public void clearLocal( )
```

Scope

public

Description

Sets the client-side ORDMultiMedia object source local attribute to *false*.

Parameter

None.

Returns

None.

Exceptions

None.

Usage

```
mediaClass mediaObject = new mediaObject(connection);
.
.
.
mediaObject.clearLocal( );
```

isLocal( ) Method

Format

```
public boolean isLocal( )
```

Scope

public

Description

Gets the client-side ORDMultiMedia object source local attribute.

Parameter

None.

Returns

This method returns the value of the source local variable.

Exceptions

None.

Usage

```
mediaClass mediaObject = new mediaObject(connection);
.
.
.
if(mediaObject.isLocal( )){
     system.out.println("Source is local");
};
```

getBFILE( ) Method

Format

```
public BFILE getBFILE( )
throws SQLException
```

Scope

public

Description

Gets the server-side ORDMultiMedia object source BFILE attribute, assuming that srcType="file".

Parameter

None.

Returns

This method returns the BFILE.

Exceptions

java.sql.SQLException

Usage

```
try{
     mediaClass mediaObject = new mediaObject(connection);
     .
     .
     .
     BFILE bfile = mediaObject.getBFILE( );
}
catch(SQLException e){
     .
     .
     .
}
```

3.2.4 ORDMultiMedia Methods Associated with the mimeType Attribute

This section presents reference information on the ORDMultiMedia methods associated with the mimeType attribute.

See Section 3.2.1 for additional code required to run the example code.

setMimeType( ) Method

Format

```
public void setMimeType(String the_mimeType)
```

Scope

public

Description

Sets the MIME type of the client-side ORDMultiMedia object.

Parameter

the_mimeType
The value of the MIME type in the client-side object cache.

Returns

None.

Exceptions

None.

Usage

```
mediaClass mediaObject = new mediaObject(connection);
.
.
.
mediaObject.setMimeType("audio/basic");
```

where:

- audio/basic: is the MIME type stored in the cache on the client side.

getMimeType( ) Method

Format

```
public String getMimeType( )
```

Scope

public

Description

Gets the MIME type of the client-side ORDMultiMedia object.

Parameter

None.

Returns

This method returns the MIME type value.

Exceptions

None.

Usage

```
mediaClass mediaObject = new mediaObject(connection);
.
.
.
String mimeType = mediaObject.getMimeType( );
```

3.2.5 ORDMultiMedia Methods Associated with the format Attribute

This section presents reference information on the ORDMultiMedia methods associated with the format attribute.

See Section 3.2.1 for additional code required to run the example code.

setFormat( ) Method

Format

```
public void setFormat(String the_format)
```

Scope

public

Description

Sets the format of the client-side ORDMultiMedia object.

Parameter

the_format
The value of the format.

Returns

None.

Exceptions

None.

Usage

```
mediaClass mediaObject = new mediaObject(connection);
.
.
.
mediaObject.setFormat("AUFF");
```

where:

- AUFF: is the value of the format.

getFormat( ) Method

Format

```
public String getFormat( )
```

Scope

public

Description

Gets the format of the client-side ORDMultiMedia object.

Parameter

None.

Returns

This method returns the value of the format stored in the client-side object cache.

Exceptions

None.

Usage

```
mediaClass mediaObject = new mediaObject(connection);
.
.
.
String sqlStr = mediaObject.getFormat( );
```

3.2.6 ORDMultiMedia Methods Associated with the updateTime Attribute

This section presents reference information on the ORDMultiMedia methods associated with the updateTime attribute.

See Section 3.2.1 for additional code required to run the example code.

setUpdateTime( ) Method

Format

```
public void setUpdateTime( )
throws SQLException
```

Scope

public

Description

Sets the ORDMultiMedia source updateTime attribute to the current time in both the server-side and client-side ORDMultiMedia objects.

Parameter

None.

Returns

None.

Exceptions

java.sql.SQLException

Usage

```
try{
     mediaClass mediaObject = new mediaObject(connection);
     .
     .
     .
     mediaObject.setUpdateTime( );
}
catch(SQLException e){
     .
     .
     .
}
```

getUpdateTime( ) Method

Format

```
public Timestamp getUpdateTime( )
```

Scope

public

Description

Returns the value of the client-side ORDMultiMedia object source updateTime attribute.

Parameter

None.

Returns

This method returns the source updateTime attribute.

Exceptions

None.

Usage

```
mediaClass mediaObject = new mediaObject(connection);
.
.
.
String lastUpdateTime = mediaObject.getUpdateTime( );
```

3.2.7 ORDMultiMedia Methods Associated with the Media Type

This section presents reference information on the ORDMultiMedia methods associated with the media type.

See Section 3.2.1 for additional code required to run the example code.

getMediaType( ) Method

Format

```
abstract String getMediaType( )
```

Scope

packaged

Description

Returns the media type information.

This is a virtual method that is defined in the ORDAudio, ORDImage, and ORD-Video subclasses.

Parameter

None.

Returns

This method returns the media type.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

3.2.8 ORDVideo Methods Associated with the Content Length

This section presents reference information on the ORDVideo methods associated with the content length.

See Section 3.2.1 for additional code required to run the example code.

getContentLength( ) Method

Format

```
public abstract int getContentLength( )
throws SQLException
```

Scope

public

Description

Returns the content length.

This is a virtual method that is defined in the ORDAudio and ORDVideo subclasses.

Parameter

None.

Returns

This method returns the content length, in bytes.

Exceptions

java.sql.SQLException

Usage

See “getContentLength( ) Method” in Chapter 4; “getContentLength(byte[ ]) Method” or “getContentLength( ) Method” in Chapter 5; “getContentLength( ) Method” in Chapter 6; or “getContentLength(byte[ ]) Method” or “getContentLength( ) Method” in Chapter 7 for the usage of this method.

3.2.9 ORDMultiMedia Methods Associated with Communication Between the Client and Server

This section presents reference information on the ORDMultiMedia methods associated with communication between the client and server.

See Section 3.2.1 for additional code required to run the example code.

refresh( ) Method

Format

```
public abstract void refresh(boolean forUpdate)
throws SQLException
```

Scope

public

Description

Sends information from the server-side media object to the client-side media object with or without locking the database row. See Section 3.2.10 for information on locking.

This is a virtual method that is defined in the ORDAudio, ORDImage, and ORDVideo subclasses.

Parameter

forUpdate
The indicator for whether or not the option will be refreshed for update. If the object is refreshed for update, the value is *true*. Otherwise, the value is *false*.

Returns

None.

Exceptions

java.sql.SQLException

Usage

See "refresh( ) Method" in Chapter 5; "refresh( ) Method" in Chapter 6; or "refresh( ) Method" in Chapter 7 for the usage of this method.

flush( ) Method

Format

```
public abstract void flush( )
throws SQLException
```

Scope

public

Description

Sends information from the client-side media object to the server-side media object.

This is a virtual method that is defined in the ORDAudio, ORDImage, and ORD-Video subclasses.

Parameter

None.

Returns

None.

Exceptions

java.sql.SQLException

Usage

See “flush( ) Method” in Chapter 5; “flush( ) Method” in Chapter 6; or “flush( ) Method” in Chapter 7 for the usage of this method.

getUpdateStr( ) Method

Format

```
String getUpdateStr(String objName)
```

Scope

package

Description

Returns the SQL statement to update the ORDMultiMedia object.

Parameter

objName
The object variable name.

Returns

This method returns the SQL string that will update a database instance with the value of the object instance defined in objName.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

3.2.10 ORDMultiMedia Methods Associated with Locking

This section presents reference information on the ORDMultiMedia methods associated with locking.

See Section 3.2.1 for additional code required to run the example code.

setLock( ) Method

Format

```
void setLock(boolean lockOption)
```

Scope

package

Description

Sets the lock.

Parameter

lockOption
The lock value.

Returns

None.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

isLocked( ) Method

Format

```
boolean isLocked( )
```

Scope

package

Description

Gets the value of the lock.

Parameter

None.

Returns

This method returns the value of the lock.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

3.2.11 ORDMultiMedia Methods Associated with the SQL Type

This section presents reference information on the ORDMultiMedia methods associated with the SQL type.

See Section 3.2.1 for additional code required to run the example code.

getSQLConstructor( ) Method

Format

```
String getSQLConstructor(boolean updateOption, String objName)
```

Scope

package

Description

Gets the SQL type constructor for the ORDMultiMedia object, which can be either an ORDAudio, ORDImage, or ORDVideo object.

Parameters

updateOption
The indicator for whether or not the option is to be updated.

objName
The multimedia object variable name.

Returns

This method returns the SQL string that will bind an object instance variable to a database object instance.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

defineSQLResults( ) Method

Format

```
String defineSQLResults(String var)
```

Scope

package

Description

Defines the SQL results that will be used to assign values to the ORDMultiMedia object attributes.

This method is used by the refresh( ) method to assign the server-side object attribute values to the corresponding client-side object attributes.

Parameter

var
The name of the multimedia object variable.

Returns

This method returns the String that will define the SQL results.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

declareSQLResults( ) Method

Format

```
int declareSQLResults(int start, OracleCallableStatement stmt)
throws SQLException
```

Scope

package

Description

Declares types for the SQL results corresponding to the different ORDMultiMedia object attributes.

This method is used by the refresh( ) method to declare the types of the client-side object attributes needed for assigning values to those attributes corresponding to the server-side object attributes.

Parameters

start
The position in the statement where the binding begins.

stmt
The Oracle callable statement that will be executed on the server side.

Returns

This method returns an integer defined as position + 1, where position is the last index in the statement for which the type was declared.

Exceptions

java.sql.SQLException

Usage

Do not use this method unless you are extending the package.

getSQLResults( ) Method

Format

```
int getSQLResults(int start, OracleCallableStatement stmt)
throws SQLException
```

Scope

package

Description

Gets the SQL results to assign to the attributes of the ORDMultiMedia object (for example, mimeType or format).

This method is used by the refresh( ) method to assign the server-side object attribute values to the corresponding client-side object attributes.

Parameters

start
The initial position in the statement to get the results.

stmt
The Oracle callable statement that has been executed on the server side.

Returns

This method returns the position in the statement from which the next result can be fetched.

Exceptions

java.sql.SQLException

Usage

Do not use this method unless you are extending the package.

setSQLParams( ) Method

Format

```
String setSQLParams(String var)
```

Scope

package

Description

Sets the values of the SQL type of the ORDMultiMedia object as parameters.

This method is used by the flush( ) method to set the values of the different attributes of the server-side object with the client-side attributes.

Parameter

var
The name of the multimedia object variable.

Returns

This method returns the SQL string that will set the different attributes of the ORD-MultiMedia object.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

bindInSQLParams( ) Method

Format

```
int bindInSQLParams(int start, OracleCallableStatement stmt)
throws SQLException
```

Scope

package

Description

Binds the values of the SQL type of the ORDMultiMedia object.

This method is used by the flush( ) method to set the values of the different attributes of the server-side object with the client-side attributes.

Parameters

start
The position in the statement where the binding begins.

stmt
The Oracle callable statement which is to be executed on the server side.

Returns

This method returns an integer defined as position + 1, where position is the last index in the statement to which a value was bound.

Exceptions

java.sql.SQLException

Usage

Do not use this method unless you are extending the package.

getPlSqlStmtBlockTemplate( ) Method

Format

```
String getPlSqlStmtBlockTemplate(boolean updateOption,
                                                 String statement,
                                                 String objName)
```

Scope

package

Description

Returns a code template for executing a PL/SQL statement.

Parameters

updateOption
The indicator for whether or not the statement requires a lock.

statement
The PL/SQL statement to be embedded in the code template.

objName
The PL/SQL variable name for the multimedia object.

Returns

This method returns the SQL string that will contain the PL/SQL DECLARE... BEGIN... END statement block.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

3.2.12 ORDVideo Methods Associated with Generating SQL Queries

This section presents reference information on the ORDVideo methods associated with generating SQL queries.

See Section 3.2.1 for additional code required to run the example code.

getFormatStr( ) Method

Format

```
abstract String getFormatStr( )
```

Scope

package

Description

Returns the format String, which is used by other methods to generate SQL queries.

This is a virtual method that is defined in the ORDAudio, ORDImage, and ORD-Video subclasses.

Parameter

None.

Returns

This method returns the format String.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

getUpdStr( ) Method

Format

```
abstract String getUpdStr( )
```

Scope

package

Description

Returns the SQL string that is used to update an instance, which is used by other methods to generate SQL queries.

This is a virtual method that is defined in the ORDAudio, ORDImage, and ORD-Video subclasses.

Parameter

None.

Returns

This method returns the SQL string that is used to update an instance.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

getSourceStr( ) Method

Format

```
abstract String getSourceStr( )
```

Scope

package

Description

Returns the source String, which is used by other methods to generate SQL queries.

This is a virtual method that is defined in the ORDAudio, ORDImage, and ORD-Video subclasses.

Parameter

None.

Returns

This method returns the source String.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

getContentLengthAPI( ) Method

Format

```
abstract String getContentLengthAPI( )
```

Scope

package

Description

Returns the server-side contentLength API for this class, which is used by other methods to generate SQL queries.

This is a virtual method that is defined in the ORDAudio, ORDImage, and ORD-Video subclasses.

Parameter

None.

Returns

This method returns the server-side contentLength API for this class.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

3.2.13 ORDMultiMedia Methods Associated with the Media Data

This section presents reference information on the ORDMultiMedia methods associated with the media data.

See Section 3.2.1 for additional code required to run the example code.

getData(String, String, String) Method

Format

```
public byte[ ] getData(String tableName, String columnName,
                       String condition)
throws SQLException, OutOfMemoryError
```

Scope

public

Description

Returns the BLOB data of the server-side ORDMultiMedia object as a byte array, given the table name, the column name, and the data condition.

If the data cannot fit into the main memory at any one point in time, then this method raises an OutOfMemory error. In this case, the application program can catch the exception and call the getDataInFile( ) method, which will download the data in the BLOB into a file.

Parameters

tableName
The name of the table from which to get the data.

columnName
The name of the column from which to get the data.

condition
The selection condition to identify the row from which to get the data.

Returns

This method returns the byte array containing the BLOB data.

Exceptions

java.sql.SQLException

java.lang.OutOfMemoryError

Usage

```
try{
     mediaClass mediaObject = new mediaObject(connection);
     .
     .
     .
     byte[ ] data = mediaObject.getData("audioTable", "song", "songID = 3");
}
catch(SQLException e){
     .
     .
     .
}
catch(OutOfMemoryError oum){
     .
     .
     .
}
```

where:

- audioTable: is the name of the table from which to get the data.
- song: is the name of the column from which to get the data.
- songID = 3: is the selection condition to identify the row from which to get the data.

getData( ) Method

Format

```
public byte[ ] getData( )
throws SQLException, OutOfMemoryError
```

Scope

public

Description

Returns the BLOB data of the server-side ORDMultiMedia object as a byte array.

The bindParams attribute is used to locate the data in the database. It reads the length of the data content in the BLOB, reads the data in chunks of 32,300 bytes, and writes the data into a byte array. The method returns this byte array.

See “setBindParams( ) Method” for information on setting the binding parameters.

If the data cannot fit into the main memory at any one point in time, then this method raises an OutOfMemory error. In this case, the application program can call the getDataInFile( ) method, which will download the data in the BLOB into a file.

Parameter

None.

Returns

This method returns the byte array containing the BLOB data.

Exceptions

java.sql.SQLException

java.lang.OutOfMemoryError

Usage

```
try{
     mediaClass mediaObject = new mediaObject(connection);
     .
     .
```

```
      .
      byte[ ] data = mediaObject.getData( );
}
catch(SQLException e){
      .
      .
      .
}
catch(OutOfMemoryError oum){
      .
      .
      .
}
```

getDataInFile( ) Method

Format

```
public void getDataInFile(String fileName)
throws SQLException, IOException
```

Scope

public

Description

Puts the server-side ORDMultiMedia object BLOB data into a file whose name is provided in the parameter.

Parameter

fileName
The name of the file where the media data will be stored.

Returns

None.

Exceptions

java.sql.SQLException

java.io.IOException

Usage

```
try{
     mediaClass mediaObject = new mediaObject(connection);
     .
     .
     .
     mediaObject.getDataInFile(file);
}
catch(SQLException e){
     .
     .
     .
```

```
}
catch(IOException io){
      .
      .
      .
}
```

where:

- file: is the name of the file where the media data will be stored.

getDataInStream( ) Method

Format

```
public InputStream getDataInStream( )
throws SQLException
```

Scope

public

Description

Returns the server-side ORDMultiMedia object media data as an InputStream object.

Parameters

None.

Returns

This method returns the media data, as an InputStream object.

Exceptions

java.sql.SQLException

Usage

```
try{
     mediaClass mediaObject = new mediaObject(connection);
     .
     .
     .
     InputStream mediaStream = mediaObject.getDataInFile( );
}
catch(SQLException e){
     .
     .
     .
}
```

loadData(String, String, String, String) Method

Format

```
public boolean loadData(String fileName, String tableName,
                        String columnName, String condition)
throws SQLException, FileNotFoundException, IOException
```

Scope

public

Description

Loads the server-side ORDMultiMedia object media data into a BLOB that is identified by the parameters tableName, columnName, and condition.

Parameters

fileName
The file from which the data will be loaded.

tableName
The name of the table into which the data will be loaded.

columnName
The name of the column into which the data will be loaded.

condition
The selection condition to identify the row into which the data will be loaded.

Returns

This method returns *true* if successful; *false* otherwise.

Exceptions

java.sql.SQLException

java.io.FileNotFoundException

java.io.IOException

Usage

```
try{
     mediaClass mediaObject = new mediaObject(connection);
     .
     .
     .
     if(mediaObject.loadData("file", "audioTable", "song", "songID = 3")){
          system.out.println("Loading is successful");
     }
}
catch(SQLException e){
     .
     .
     .
}
catch(FileNotFoundException fnf){
     .
     .
     .
}
catch(IOException io){
     .
     .
     .
}
```

where:

- file: is the name of the file from which the data will be loaded.
- audioTable: is the name of the table into which the data will be loaded.
- song: is the name of the column into which the data will be loaded.
- songID = 3: is the selection condition to identify the row into which the data will be loaded.

loadData(String) Method

Format

```
public boolean loadData(String fileName)
throws SQLException, IOException, SecurityException
```

Scope

public

Description

Loads the server-side ORDMultiMedia object media data from a given file into a media object designated by the binding parameters.

Parameters

fileName
The file from which the data will be loaded.

Returns

This method returns *true* if the loading is successful; *false* otherwise.

Exceptions

java.sql.SQLException

java.io.IOException

java.lang.SecurityException

Usage

```
try{
     mediaClass mediaObject = new mediaObject(connection);
     .
     .
     .
     if(mediaObject.loadData("jdoe.gif")){
          system.out.println("Loading is successful.");
     }
}
```

```
catch(SQLException e){
     .
     .
     .
}
catch(OutOfMemoryError oum){
     .
     .
     .
}
catch(IOException io){
     .
     .
     .
}
catch(SecurityException s){
     .
     .
     .
}
```

where:

- jdoe.gif: is the name of the file from which the data will be loaded.

loadDataInChunks( ) Method

Format

```
boolean loadDataInChunks(String fileName, int fileLength)
throws SQLException, IOException
```

Scope

package

Description

Loads the server-side ORDMultiMedia object media data in chunks of 32K bytes at a time from a given file into a media object that is designated by the binding parameters.

This method is called by loadData( ) if the data file to be loaded is too large to fit into the memory.

Parameters

fileName
The file from which the data will be loaded.

fileLength
The total length of the file, in bytes.

Returns

This method returns *true* if loading is successful; *false* otherwise.

Exceptions

java.sql.SQLException

java.io.IOException

Usage

Do not use this method unless you are extending the package.

3.3 BindToTableParams Methods

This section presents reference information on the methods contained in the BindToTableParams class. These methods are described in the following groupings:

BindToTableParams Methods Associated with the tableName Attribute

- public void setTableName( ): set the table name.
- public String getTableName( ): gets the name of the table to which the client object is bound.

BindToTableParams Methods Associated with the columnName Attribute

- public void setColumnName( ): set the column name.
- public String getColumnName( ): gets the name of the column to which the client object is bound.

BindToTableParams Methods Associated with the dataCondition Attribute

- public void setDataCondition( ): set the data condition.
- public String getDataCondition( ): gets the select condition of the row of the server-side object to which the client-side object is bound.

3.3.1 BindToTableParams Methods Associated with the tableName Attribute

This section presents reference information on the BindToTableParams methods associated with the tableName attribute.

setTableName( ) Method

Format

```
public void setTableName(String the_tableName)
```

Scope

public

Description

Sets the table name.

Parameter

the_tableName
The table name.

Returns

None.

Exceptions

None.

Usage

None.

getTableName( ) Method

Format

```
public String getTableName( )
```

Scope

public

Description

Gets the table name to which the client object is bound.

Parameter

None.

Returns

This method returns the table name.

Exceptions

None.

Usage

None.

3.3.2 BindToTableParams Methods Associated with the columnName Attribute

This section presents reference information on the BindToTableParams methods associated with the columnName attribute.

setColumnName( ) Method

Format

```
public void setColumnName(String the_columnName)
```

Scope

public

Description

Sets the column name.

Parameter

the_columnName
The column name.

Returns

None.

Exceptions

None.

Usage

None.

getColumnName( ) Method

Format

```
public String getColumnName( )
```

Scope

public

Description

Gets the column name to which the client object is bound.

Parameter

None.

Returns

This method returns the column name.

Exceptions

None.

Usage

None.

3.3.3 BindToTableParams Methods Associated with the dataCondition Attribute

This section presents reference information on the BindToTableParams methods associated with the dataCondition attribute.

setDataCondition( ) Method

Format

```
public void setDataCondition(String the_dataCondition)
```

Scope

public

Description

Sets the data condition.

Parameter

the_dataCondition
The data condition.

Returns

None.

Exceptions

None.

Usage

None.

getDataCondition( ) Method

Format

```
public String getDataCondition( )
```

Scope

public

Description

Gets the select condition of the row on the server-side object to which the client-side object is bound.

Parameter

None.

Returns

This method returns the data condition.

Exceptions

None.

Usage

None.

4

ORDSource Reference Information

Oracle8*i* *inter*Media Audio, Image, and Video Java Client contains information about the ORDSource:

- Object type -- see "ORDSource Object Type" in Section 4.1.
- Methods -- see Section 4.2.

Methods invoked at the ORDSource level that are handed off for processing to the server-side source plug-in have `byte[ ] ctx` as a context parameter. The space for the parameter is created by the client (in the reference examples, 4000 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.

> **Note:** In the current release, not all source plug-ins will use or generate the context parameter, but if you include the parameter as previously described, your application should work with any current or future source plug-in.

See *Oracle8i interMedia Audio, Image, and Video User's Guide and Reference* for more information.

4.1 Object Types

Oracle8*i* *inter*Media Audio, Image, and Video Java Client describes the ORDSource object type, which supports access to a variety of sources of multimedia data.

ORDSource Object Type

The ORDSource object type supports access to data sources locally in a BLOB within an Oracle database, externally from a BFILE on a local file system, externally from a URL on an HTTP server, externally from streaming multimedia data on a media server, or externally from a user-defined source on another server.

> **Note:** In the following code, the methods have the default protection, which means that they can be called only by other methods in the package.

This object type is defined as follows:

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

public class OrdSource {

     int MAX_CHUNKSIZE = 32300;

     BindToTableParams bindParams = null;
     String containerType;
     Connection connection;
     boolean lock = false;

     BLOB localData = null;
     String srcType;
     String srcLocation;
     String srcName;
     Timestamp updateTime;
     int local;

     OrdSource( )
     { }

     void setLock(boolean lockOption)
     { }
```

```
boolean isLocked( )
{ }

void setConnection(Connection the_connecton)
{ }

Connection getConnection( )
{ }

String getSQLConstructor(boolean updateOption, String mmObjName,
     String srcObjName)
{ }

String getUpdateStr(String mmObjName, String srcObjName)
{ }

void flush( )
throws SQLException
{ }

void setBindParams(BindToTableParams bParams)
{ }

void setContainerType(String type)
{ }

String getContainerType( )
{ }

String defineSQLResults(String var)
{ }

int declareSQLResults(int start, OracleCallableStatement stmt)
throws SQLException
{ }

String setSQLParams(String var)
{ }

int bindInSQLParams(int start, OracleCallableStatement stmt)
throws SQLException
{ }

int getSQLResults(int start, OracleCallableStatement stmt)
throws SQLException
```

```
{ }

void refresh(boolean forUpdate)
throws SQLException
{ }

void setLocal( )
{ }

void clearLocal( )
{ }

boolean isLocal( )
{ }

Timestamp getUpdateTime( )
{ }

void setUpdateTime( )
throws SQLException
{ }

void setSourceInformation(String sourceType, String sourceLocation,
     String sourceName)
{ }

void setSourceType(String sourceType)
{ }

void setSourceLocation(String sourceLocation)
{ }

void setSourceName(String sourceName)
{ }

String getSourceInformation( )
{ }

String getSourceType( )
{ }

String getSourceLocation( )
{ }

String getSourceName( )
```

```
{ }

BLOB importData(byte[ ] ctx, BLOB dlob, StringBuffer mimetype,
     StringBuffer format)
throws SQLException
{ }

BLOB importFrom(byte[ ] ctx, BLOB dlob, StringBuffer mimetype,
     StringBuffer format, String sourceType, String sourceLocation,
     String sourceName)
throws SQLException
{ }

void export(byte[ ] ctx, String sourceType, String sourceLocation,
     String sourceName)
throws SQLException
{ }

int getContentLength( )
throws SQLException
{ }

String getSourceAddress( )
{ }

BLOB getLocalContent( )
throws SQLException
{ }

BLOB getContentInTempLob(byte[ ] ctx, BLOB blob,
     StringBuffer mimetype, StringBuffer format)
throws SQLException
{ }

BLOB getContentInTempLob(byte[ ] ctx, BLOB blob, StringBuffer mimetype,
     StringBuffer format, int duration, boolean cache)
throws SQLException
{ }

void deleteLocalContent( )
throws SQLException
{ }

int read(byte[ ] ctx, int startPos, int numBytes, byte[ ] buffer)
throws SQLException
```

```
    { }

    int write(byte[ ] ctx, int startPos, int numBytes, byte[ ] buffer)
    throws SQLException
    { }

    int open(byte[ ] userArg, byte[ ] ctx)
    throws SQLException
    { }

    int close(byte[ ] ctx)
    throws SQLException
    { }

    int trim(byte[ ] ctx, int newLen)
    throws SQLException
    { }

    BFILE getBFILE( )
    throws SQLException
    { }

    byte[ ] processCommand(byte[ ] ctx, String command, String argList,
         byte[ ] result)
    throws SQLException
    { }
}
```

where the class attributes are defined as:

- bindParams: contains the parameters used to bind the client-side object to a server-side object. For more information on the BindToTableParams object, see "BindToTableParams Object Type" and Section 3.3.
- containerType: indicates the type of object in which the source object is embedded (either ORDAudio, ORDImage, or ORDVideo).
- connection: is the connection to the database.
- lock: indicates whether or not the corresponding server-side object is to be updated. It is set to *true* if the server-side object is to be updated.
- localData: contains the locally stored multimedia data stored as a BLOB within the object. Up to 4GB of data can be stored as a BLOB within an Oracle data-

base and is protected by the Oracle security and transaction environment. This is obtained from the server-side ORDSource object.

- srcType: identifies the data source type. This is obtained from the server-side ORDSource object. Supported srcType values are the following:

srcType	Description
"unknown"	server access unknown or not provided
"file"	BFILE on a local file system
"http"	HTTP server
"<name>"	another server

- srcLocation: identifies the place where data can be found based on the srcType value. This is obtained from the server-side ORDSource object. Valid srcLocation values for corresponding srcType values are the following:

srcType	Location Value
"unknown"	NULL or not accessed
"file"	<DIR> or name of the directory object
"http"	<SourceBase> or URL needed to find the base directory
"<name>"	<iden> or identifier string required to access another server

- srcName: identifies the data object name. This is obtained from the server-side ORDSource object. Valid srcName values for corresponding srcType values are the following:

srcType	Name Value
"unknown"	NULL or not accessed
"file"	<FILE> or name is the name of the file
"http"	<Source> or name of the object
<name>	<object name> or name of the object

- updateTime: the time at which the data is updated. This is obtained from the server-side ORDSource object.
- local: a flag to determine whether the data is local or not. 1 or NULL means the data is in the LOB. 0 means the data is in external sources. This is obtained from the server-side ORDSource object.

4.2 Methods

This section presents ORDSource reference information on the methods used for source data manipulation. These methods are described in the following groupings:

ORDSource Methods Associated with the bindParams Attribute

- void setBindParams( ): set the data location parameters.

ORDSource Methods Associated with the containerType Attribute

- void setContainerType( ): set the object container type to ORDAudio, ORDImage, ORDVideo, or ORDVir.
- String getContainerType( ): get the container type of the ORDSource object.

ORDSource Methods Associated with the connection Attribute

- void setConnection( ): set the connection to the database.
- Connection getConnection( ): get the connection to the database.

ORDSource Methods Associated with the lock Attribute

- void setLock( ): set the lock value.
- boolean isLocked( ): get the lock value.

ORDSource Methods Associated with the srcName, srcType, and srcLocation Attributes

- void setSourceInformation( ): set information about the client-side ORDSource object data source.
- void setSourceType( ): set the client-side ORDSource object source type.
- void setSourceLocation( ): set the client-side ORDSource object source location.
- void setSourceName( ): set the client-side ORDSource object source name.
- String getSourceInformation( ): get the client-side ORDSource object source information.
- String getSourceType( ): get the client-side ORDSource object source type.
- String getSourceLocation( ): get the client-side ORDSource object source location.
- String getSourceName( ): get the client-side ORDSource object source name.

ORDSource Methods Associated with the updateTime Attribute

- void setUpdateTime( ): set the updated time of both the server-side and client-side ORDSource objects.
- Timestamp getUpdateTime( ): get the last updated time of the client-side ORDSource object.

ORDSource Methods Associated with the local Attribute

- void setLocal( ): set the client-side ORDSource object local attribute.
- boolean isLocal( ): get the client-side ORDSource object local attribute.
- void clearLocal( ): clear the client-side ORDSource object local attribute.

ORDSource Methods Associated with Communication Between Client and Server

- void refresh( ): send information from the server-side object to the client-side object with or without locking the database row.
- void flush( ): send information from the client-side object to the server-side object.
- String getUpdateStr( ): get the update String in order to update the database object.

ORDSource Methods Associated with the SQL Type

- String getSQLConstructor( ): get the SQL type constructor for the ORDSource object.
- String defineSQLResults( ): define the SQL results that will be used to assign values to the ORDSource object attributes.
- int declareSQLResults( ): declare the types for the SQL results corresponding to the different ORDSource object attributes.
- String setSQLParams( ): set the values of the SQL type of the ORDSource object as parameters.
- int bindInSQLParams( ): bind the values of the SQL type of the ORDSource object.
- int getSQLResults( ): get the SQL results to assign to the attributes of the ORDSource object.

ORDSource Methods Associated with Access Operations

- int open( ): open the server-side ORDSource object source.
- int close( ): close the server-side ORDSource object source.
- int trim( ): trim the server-side ORDSource object source to a given length.

ORDSource Methods Associated with Content Read/Write Operations

- int read( ): read from the server-side ORDSource object localData attribute into the buffer.
- int write( ): write the buffer contents into the server-side ORDSource object localData attribute.

ORDSource Methods Associated with Import and Export Operations

- BLOB importData( ): import data into the server-side BLOB from the source information in the server-side ORDSource object.
- BLOB importFrom( ): import data into the server-side BLOB from a data source specified by the parameters.
- void export( ): export data from the server-side ORDSource object to an external source.

ORDSource Methods Associated with Source Content Operations

- int getContentLength( ): get the content length of the server-side ORDSource object localData attribute.
- String getSourceAddress( ): get the client-side ORDSource object source address.
- BLOB getLocalContent( ): get the server-side ORDSource object LOB locator and copy the server-side LOB locator to the client side.
- BLOB getContentInTempLob(byte[ ], BLOB, StringBuffer, StringBuffer): get the server-side ORDSource object localData content in the temporary LOB with default values of duration = 10 and cache = true.
- BLOB getContentInTempLob(byte[ ], BLOB, StringBuffer, StringBuffer, int, boolean): get the server-side ORDSource object localData content in the temporary BLOB.
- void deleteLocalContent( ): delete the contents of the server-side ORDSource object localData attribute.

- BFILE getBFILE( ): get the server-side ORDSource object BFILE.

ORDSource Methods Associated with Processing Commands to the External Source

- byte[ ] processCommand( ): Calls the server-side processCommand( ) method. Currently raises exceptions.

4.2.1 ORDSource Methods Associated with the bindParams Attribute

This section presents reference information on the ORDSource methods associated with the bindParams attribute.

setBindParams( ) Method

Format

```
void setBindParams(BindToTableParams bParams)
```

Scope

package

Description

Sets the data location parameters.

Parameter

bParams
The binding parameter values. For more information on the BindToTableParams object, see "BindToTableParams Object Type" and Section 3.3.

Returns

None.

Exceptions

None.

Usage

None.

4.2.2 ORDSource Methods Associated with the containerType Attribute

This section presents reference information on the ORDSource methods associated with the containerType attribute.

setContainerType( ) Method

Format

```
void setContainerType(String type)
```

Scope

package

Description

Sets the object container type to ORDAudio, ORDImage, ORDVideo, or ORDVir.

The containerType attribute indicates the type of the object in which the ORD-Source object is embedded.

Parameter

type
The value of the object type.

Returns

None.

Exceptions

None.

Usage

None.

getContainerType( ) Method

Format

```
String getContainerType( )
```

Scope

package

Description

Gets the container type of the ORDSource object.

Parameter

None.

Returns

This method returns the object type.

Exceptions

None.

Usage

None.

4.2.3 ORDSource Methods Associated with the connection Attribute

This section presents reference information on the ORDSource methods associated with the connection attribute.

setConnection( ) Method

Format

```
void setConnection(Connection the_connection)
```

Scope

package

Description

Sets the connection to the database.

Parameter

the_connection
The connection object.

Returns

None.

Exceptions

None.

Usage

None.

getConnection( ) Method

Format

```
Connection getConnection( )
```

Scope

package

Description

Gets the connection to the database.

Parameter

None.

Returns

This method returns the connection.

Exceptions

None.

Usage

None.

4.2.4 ORDSource Methods Associated with the lock Attribute

This section presents reference information on the ORDSource methods associated with the lock attribute.

setLock( ) Method

Format

```
void setLock(boolean lockOption)
```

Scope

package

Description

Sets the lock value.

A value of *true* means the corresponding object on the server side can be updated. A value of *false* means the corresponding object on the server side cannot be updated.

Parameter

lockOption
The lock value.

Returns

None.

Exceptions

None.

Usage

None.

isLocked( ) Method

Format

```
boolean isLocked( )
```

Scope

package

Description

Gets the lock value.

A value of *true* means the corresponding object on the server side cannot be updated. A value of *false* means the corresponding object on the server side cannot be updated.

Parameter

None.

Returns

This method returns the value of the lock.

Exceptions

None.

Usage

None.

4.2.5 ORDSource Methods Associated with the srcName, srcType, and srcLocation Attributes

This section presents reference information on the ORDSource methods associated with the srcName, srcType, and srcLocation attributes.

setSourceInformation( ) Method

Format

```
void setSourceInformation(String sourceType, String sourceLocation,
                          String sourceName)
```

Scope

package

Description

Sets information about the client-side ORDSource object data source.

Parameters

sourceType
The source type.

sourceLocation
The source location.

sourceName
The source name.

Returns

None.

Exceptions

None.

Usage

None.

setSourceType( ) Method

Format

```
void SetSourceType(String sourceType)
```

Scope

package

Description

Sets the client-side ORDSource object source type.

Parameter

sourceType
The source type.

Returns

None.

Exceptions

None.

Usage

None.

setSourceLocation( ) Method

Format

```
void setSourceLocation(String sourceLocation)
```

Scope

package

Description

Sets the client-side ORDSource object source location.

Parameter

sourceLocation
The source location.

Returns

None.

Exceptions

None.

Usage

None.

setSourceName( ) Method

Format

```
void setSourceName(String sourceName)
```

Scope

package

Description

Sets the client-side ORDSource object source name.

Parameter

sourceName
The source name.

Returns

None.

Exceptions

None.

Usage

None.

getSourceInformation( ) Method

Format

```
String getSourceInformation( )
```

Scope

package

Description

Gets the client-side ORDSource object source information.

Parameter

None.

Returns

This method returns the source information in the following format: srcType: / / srcLocation / srcName.

Exceptions

None.

Usage

None.

getSourceType( ) Method

Format

```
String getSourceType( )
```

Scope

package

Description

Gets the client-side ORDSource object source type

Parameter

None.

Returns

This method returns the source type.

Exceptions

None.

Usage

None.

getSourceLocation( ) Method

Format

```
String getSourceLocation( )
```

Scope

package

Description

Gets the client-side ORDSource object source location.

Parameter

None.

Returns

This method returns the source location.

Exceptions

None.

Usage

None.

getSourceName( ) Method

Format

```
String getSourceName( )
```

Scope

package

Description

Gets the client-side ORDSource object source name.

Parameter

None.

Returns

This method returns the source name.

Exceptions

None.

Usage

None.

4.2.6 ORDSource Methods Associated with the updateTime Attribute

This section presents reference information on the ORDSource methods associated with the updateTime attribute.

setUpdateTime( ) Method

Format

```
void setUpdateTime( )
throws SQLException
```

Scope

package

Description

Sets the updated time of both the server-side and client-side ORDSource objects.

Parameter

None.

Returns

None.

Exceptions

java.sql.SQLException

Usage

None.

getUpdateTime( ) Method

Format

```
Timestamp getUpdateTime( )
```

Scope

package

Description

Gets the last updated time of the client-side ORDSource object.

Parameter

None.

Returns

This method returns the value of the last update time.

Exceptions

None.

Usage

None.

4.2.7 ORDSource Methods Associated with the local Attribute

This section presents reference information on the ORDSource methods associated with the local attribute.

setLocal( ) Method

Format

```
void setLocal( )
```

Scope

package

Description

Sets the client-side ORDSource local attribute to *true*, which indicates that the local-Data attribute is valid.

Parameter

None.

Returns

None.

Exceptions

None.

Usage

None.

isLocal( ) Method

Format

```
boolean isLocal( )
```

Scope

package

Description

Gets the client-side ORDSource object local attribute.

Parameter

None.

Returns

This method returns the value of the local variable.

Exceptions

None.

Usage

None.

clearLocal( ) Method

Format

```
void clearLocal( )
```

Scope

package

Description

Clears the client-side ORDSource object local attribute.

Parameter

None.

Returns

None.

Exceptions

None.

Usage

None.

4.2.8 ORDSource Methods Associated with Communication Between Client and Server

This section presents reference information on the ORDSource methods associated with communication between the client and the server.

refresh( ) Method

Format

```
void refresh(boolean forUpdate)
throws SQLException
```

Scope

package

Description

Sends information from the server-side object to the client-side object with or without locking the database row.

Parameter

forUpdate
The indicator for whether or not the option has been refreshed for update. A value of *true* indicates that the object has been refreshed for update; otherwise, the value is *false*.

Returns

None.

Exceptions

java.sql.SQLException

Usage

None.

flush( ) Method

Format

```
void flush( )
throws SQLException
```

Scope

package

Description

Copies the client-side object attribute values to the server-side object.

Parameter

None.

Returns

None.

Exceptions

java.sql.SQLException

Usage

None.

getUpdateStr( ) Method

Format

```
String getUpdateStr(String mmObjName, String srcObjName)
```

Scope

package

Description

Gets the update String in order to update the database object.

Parameters

mmObjName
The multimedia object variable.

srcObjName
The source object variable.

Returns

This method returns the SQL string to update an ORDSource instance.

Exceptions

None.

Usage

None.

4.2.9 ORDSource Methods Associated with the SQL Type

This section presents reference information on the ORDSource methods associated with the SQL type.

getSQLConstructor( ) Method

Format

```
String getSQLConstructor(boolean updateOption, String mmObjName,
                         String srcObjName)
```

Scope

package

Description

Gets the SQL type constructor for the ORDSource object.

Parameters

updateOption
A variable that specifies whether or not the lock is to be taken.

mmObjName
The multimedia object variable. It is set to ORDAudio, ORDImage, or ORDVideo.

srcObjName
The source object variable

Returns

This method returns the SQL string that will bind an object instance variable of ORDSource to a database ORDSource object instance.

Exceptions

None.

Usage

None.

defineSQLResults( ) Method

Format

```
String defineSQLResults(String var)
```

Scope

package

Description

Defines the SQL results that will be used to assign values to the ORDSource object attributes.

This method is used by the refresh( ) method to assign the server-side object attribute values to the corresponding server-side object attributes.

Parameter

var
The name of the source object variable.

Returns

This method returns the SQL string that will define the SQL results.

Exceptions

None.

Usage

None.

declareSQLResults( ) Method

Format

```
int declareSQLResults(int start, OracleCallableStatement stmt)
throws SQLException
```

Scope

package

Description

Declares the types for the SQL results corresponding to the different ORDSource object attributes.

This method is used by the refresh( ) method to declare the types of the client-side object attributes that are needed in order to assign values to the attributes corresponding to the server-side object attributes.

Parameters

start
The position in the statement where the binding begins.

stmt
The Oracle callable statement that will be executed on the server side.

Returns

This method returns an integer defined as position +1, where position is the last index in the statement for which the type was declared.

Exceptions

java.sql.SQLException

Usage

None.

setSQLParams( ) Method

Format

```
String setSQLParams(String var)
```

Scope

package

Description

Sets the values of the SQL type of the ORDSource object as parameters.

This method is used by the flush( ) method to set the values of the different attributes of the server-side object with the client-side attributes.

Parameter

var
The name of the source object variable.

Returns

This method returns the SQL string that will set the different attributes of the ORD-Source object.

Exceptions

None.

Usage

None.

bindInSQLParams( ) Method

Format

```
int bindInSQLParams(int start, OracleCallableStatement stmt)
throws SQLException
```

Scope

package

Description

Binds the values of the SQL type of the ORDSource object.

This method is used by the flush( ) method to set the values of the different attributes of the server-side object with the client-side attributes.

Parameters

start
The position in the statement where the binding begins.

stmt
The Oracle callable statement that will be executed on the server side.

Returns

This method returns an integer defined as position + 1, where position is the last index in the statement to which a value was bound.

Exceptions

java.sql.SQLException

Usage

None.

getSQLResults( ) Method

Format

```
int getSQLResults(int start, OracleCallableStatement stmt)
throws SQLException
```

Scope

package

Description

Gets the SQL results to assign to the attributes of the ORDSource object (srcLocation or srcType, for example).

This method is used by the refresh( ) method to assign the server-side object attribute values to the corresponding client-side object attributes.

Parameters

start
The starting position for getting results from the statement.

stmt
The Oracle callable statement that has been executed on the server side.

Returns

This method returns the SQL string that will bind a client object attribute to the corresponding database object attribute.

Exceptions

java.sql.SQLException

Usage

None.

4.2.10 ORDSource Methods Associated with Access Operations

This section presents reference information on the ORDSource methods associated with access operations.

open( ) Method

Format

```
int open(byte[ ] userArg, byte[ ] ctx)
throws SQLException
```

Scope

package

Description

Opens the server-side ORDSource object source.

Parameter

userArg
Permission-related parameters that are supplied by the user, such as READONLY.

ctx
The source plug-in context information.

Returns

This method returns 0 in case of success or an integer greater than 0 in case of failure.

Exceptions

java.sql.SQLException

Usage

None.

close( ) Method

Format

```
int close(byte[ ] ctx)
throws SQLException
```

Scope

package

Description

Closes the server-side ORDSource object source.

Parameter

ctx
The source plug-in context information.

Returns

This method returns 0 in case of success or an integer greater than 0 in case of failure.

Exceptions

java.sql.SQLException

Usage

None.

trim( ) Method

Format

```
int trim(byte[ ] ctx, int newLen)
throws SQLException
```

Scope

package

Description

Trims the server-side ORDSource object source to a given length.

Parameters

ctx
The source plug-in context information.

newLen
The new length of the content after trimming.

Returns

This method returns 0 in case of success or an integer greater than 0 in case of failure.

Exceptions

java.sql.SQLException

Usage

None.

4.2.11 ORDSource Methods Associated with Content Read/Write Operations

This section presents reference information on the ORDSource methods associated with content read and write operations.

read( ) Method

Format

```
int read(byte[ ] ctx, int startPos, int numBytes, byte[ ] buffer)
throws SQLException
```

Scope

package

Description

Reads from the server-side ORDSource object localData attribute into the buffer.

Parameters

ctx
The source plug-in context information.

startPos
The initial reading position in the BLOB.

numBytes
The number of bytes to be read.

buffer
The buffer into which the value is to be read.

Returns

This method returns the number of bytes read, or -1 in case of failure.

Exceptions

java.sql.SQLException

Usage

None.

write( ) Method

Format

```
int write (byte[ ] ctx, int startPos, int numBytes, byte[ ] buffer)
throws SQLException
```

Scope

package

Description

Writes the buffer contents into the server-side ORDSource object localData attribute.

Parameters

ctx
The source plug-in context information.

startPos
The start position for writing in the BLOB.

numBytes
The number of bytes to be written.

buffer
The buffer from which the value is to be written into the localData attribute.

Returns

This method returns the number of bytes written, or -1 in case of failure.

Exceptions

java.sql.SQLException

Usage

None.

4.2.12 ORDSource Methods Associated with Import and Export Operations

This section presents reference information on the ORDSource methods associated with import and export operations.

importData( ) Method

Format

```
BLOB importData(byte[ ] ctx, BLOB dlob, StringBuffer mimetype,
                StringBuffer format)
throws SQLException
```

Scope

package

Description

Imports data into the server-side BLOB from the source information in the server-side ORDSource object.

Parameters

ctx
The source plug-in context information.

dlob
The LOB locator into which the data is to be imported.

mimetype
The MIME type of the data.

format
The format of the data.

Returns

This method returns the BLOB locator that contains the imported data.

Exceptions

java.sql.SQLException

Usage

None.

importFrom( ) Method

Format

```
BLOB importFrom(byte[ ] ctx, BLOB dlob, StringBuffer mimetype,
                StringBuffer format, String sourceType,
                String sourceLocation, String sourceName)
throws SQLException
```

Scope

package

Description

Imports data into the server-side BLOB from a data source specified by the parameters.

Parameters

ctx
The source plug-in context information.

dlob
The LOB locator into which the data is to be imported.

mimetype
The MIME type of the data.

format
The format of the data.

sourceType
The type of the source from which the data is imported.

sourceLocation
The location of the source from which the data is imported.

sourceName
The name of the source from which the data is imported.

Returns

This method returns the LOB locator that contains the imported data.

Exceptions

java.sql.SQLException

Usage

None.

export( ) Method

Format

```
void export(byte[ ] ctx, String sourceType, String sourceLocation,
            String sourceName)
throws SQLException
```

Scope

package

Description

Exports data from the server-side ORDSource object to an external source.

Parameters

ctx
The source plug-in context information.

sourceType
The type of the source into which the data will be exported.

sourceLocation
The location of the source into which the data will be exported.

sourceName
The name of the source into which the data will be exported.

Returns

None.

Exceptions

java.sql.SQLException

Usage

None.

4.2.13 ORDSource Methods Associated with Source Content Operations

This section presents reference information on the ORDSource methods associated with source content operations.

getContentLength( ) Method

Format

```
int getContentLength( )
throws SQLException
```

Scope

package

Description

Gets the content length of the server-side ORDSource object localData attribute.

Parameter

None.

Returns

This method returns the content length of the source data, in bytes.

Exceptions

java.sql.SQLException

Usage

None.

getSourceAddress( ) Method

Format

```
String getSourceAddress( )
```

Scope

package

Description

Returns the client-side ORDSource object source address.

Parameter

None.

Returns

This method returns the source address in the format srcType://srcLocation/srcName.

Exceptions

None.

Usage

None.

getLocalContent( ) Method

Format

```
BLOB getLocalContent( )
throws SQLException
```

Scope

package

Description

Returns the server-side ORDSource object LOB locator and copies the server side LOB locator to the client side.

Parameter

None.

Returns

This method returns the LOB locator.

Exceptions

java.sql.SQLException

Usage

None.

getContentInTempLOB(byte[ ], BLOB, StringBuffer, StringBuffer) Method

Format

```
BLOB getContentInTempLob(byte[ ] ctx, BLOB blob, StringBuffer mimetype,
                         StringBuffer format)
throws SQLException
```

Scope

package

Description

Gets the server-side ORDSource object localData content in the temporary BLOB, using the default values of duration = 10 and cache = true.

Parameters

ctx
The source plug-in context information.

blob
The value of the temporary BLOB before the call.

mimetype
The MIME type of the data in the temporary BLOB.

format
The format of the data in the temporary BLOB.

Returns

This method returns the content of the temporary BLOB after the call.

Exceptions

java.sql.SQLException

Usage

None.

getContentInTempLOB(byte[ ], BLOB, StringBuffer, StringBuffer, int, boolean) Method

Format

```
BLOB getContentInTempLob(byte[ ] ctx, BLOB blob, StringBuffer mimetype,
                         StringBuffer format, int duration, boolean cache)
throws SQLException
```

Scope

package

Description

Gets the server-side ORDSource object localData content in the temporary LOB.

Parameters

ctx
The source plug-in context information.

blob
The value of the temporary BLOB before the call.

mimetype
The MIME type of the data in the temporary BLOB.

format
The format of the data in the temporary BLOB.

duration
The duration of the call.

cache
The indicator of whether or not to use caching.

Returns

This method returns the content of the temporary BLOB after the call.

Exceptions

java.sql.SQLException

Usage

None.

deleteLocalContent( ) Method

Format

```
void deleteLocalContent( )
throws SQLException
```

Scope

package

Description

Deletes the contents of the server-side ORDSource object localData attribute.

Parameter

None.

Returns

None.

Exceptions

java.sql.SQLException

Usage

None.

getBFILE( ) Method

Format

```
BFILE getBFILE( )
throws SQLException
```

Scope

package

Description

Gets the server-side ORDSource object BFILE.

This method calls the server-side method getBFILE( ) in the ORDSource object.

Parameter

None.

Returns

This method returns the BFILE.

Exceptions

java.sql.SQLException

Usage

None.

4.2.14 ORDSource Methods Associated with Processing Commands to the External Source

This section presents reference information on the ORDSource methods associated with processing commands to the external source.

processCommand( ) Method

Format

```
byte[ ] processCommand(byte[ ] ctx, String command, String arglist, byte[ ]
result)
throws SQLException
```

Scope

package

Description

Calls the server-side processCommand( ) method.

For more information on the commands that can be processed, see *Oracle8i interMedia Audio, Image, and Video User's Guide and Reference*.

Parameters

ctx
The source plug-in context information.

command
The command to be processed.

arglist
The argument list for the command to be processed.

result
The result of executing the command.

Returns

This method returns the result of executing the command.

Exceptions

java.sql.SQLException

Usage

None.

5

ORDAudio Reference Information

Oracle8*i* *inter*Media Audio, Image, and Video Java Client contains information about the ORDAudio:

- Object type -- see "ORDAudio Object Type" in Section 5.1.
- Methods -- see Section 5.2.

Methods invoked at the ORDAudio level that are handed off for processing to the server-side source plug-in or server-side format plug-in have `byte[ ] ctx` as a context parameter. The space for the parameter is created by the client (in the reference examples, 4000 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.

> **Note:** In the current release, not all source plug-ins or format plug-ins will use or generate the context parameter, but if you include the parameter as previously described, your application should work with any current or future source plug-ins or format plug-ins.

See *Oracle8i interMedia Audio, Image, and Video User's Guide and Reference* for more information.

5.1 Object Types

Oracle8*i* *inter*Media Audio, Image, and Video Java Client describes the ORDAudio object type, which supports the storage and management of audio data.

ORDAudio Object Type

The ORDAudio object type supports the storage and management of audio data.

> **Note:** In the following code, some methods will begin with *public*. This indicates that the method can be called by all classes in all packages. The methods that are not designated as public have the default protection, which means that they can be called only by other methods in the package.

This object type is defined as follows:

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

     public class ORDAudio extends ORDMultiMedia {

     String encoding;
     int numberOfChannels;
     int samplingRate;
     int sampleSize;
     String compressionType;
     int audioDuration;
     String description;
     CLOB comments;

     public ORDAudio( )
     { }

     public ORDAudio(Connection the_connection)
     { }

     String getMediaType( )
     { }

     String getFormatStr( )
     { }

     String getUpdStr( )
```

```
{ }

String getSourceStr( )
{ }

String getContentLengthAPI( )
{ }

public int getContentLength( )
throws SQLException
{ }

String getSQLConstructor(boolean updateOption, String objName)
{ }

String defineSQLResults(String var)
{ }

int declareSQLResults(int start, OracleCallableStatement stmt)
throws SQLException
{ }

String setSQLParams(String var)
{ }

int bindInSQLParams(int start, OracleCallableStatement stmt)
throws SQLException
{ }

int getSQLResults(int start, OracleCallableStatement stmt)
throws SQLException
{ }

public void refresh(boolean forUpdate)
throws SQLException
{ }

public void flush( )
throws SQLException
{ }

public void setEncoding(String the_encoding)
{ }

public void setNumberOfChannels(int the_numberOfChannels)
```

```
{ }

public void setSamplingRate(int the_samplingRate)
{ }

public void setSampleSize(int the_sampleSize)
{ }

public void setCompressionType(String the_compressionType)
{ }

public void setKnownAttributes(String knownFormat,
     String knownEncoding, int knownNumberOfChannels,
     int knownSampleSize, int knownSamplingRate,
     String knownCompressionType, int knownAudioDuration)
{ }

public void setAudioDuration(int the_audioDuration)
{ }

public String getFormat(byte[ ] ctx)
throws SQLException
{ }

public String getEncoding(byte[ ] ctx)
throws SQLException
{ }

public String getEncoding( )
{ }

public int getNumberOfChannels(byte[ ] ctx)
throws SQLException
{ }

public int getNumberOfChannels( )
{ }

public int getSamplingRate(byte[ ] ctx)
throws SQLException
{ }

public int getSamplingRate( )
{ }
```

```
public int getSampleSize(byte[ ] ctx)
throws SQLException
{ }

public int getSampleSize( )
{ }

public String getCompressionType(byte[ ] ctx)
throws SQLException
{ }

public String getCompressionType( )
{ }

public int getAudioDuration(byte[ ] ctx)
throws SQLException
{ }

public int getAudioDuration( )
{ }

public void setDescription(String the_description)
{ }

public String getDescription( )
{ }

public int getContentLength(byte[ ] ctx)
throws SQLException
{ }

public void setProperties(byte[ ] ctx)
throws SQLException
{ }

public boolean checkProperties(byte[ ] ctx)
throws SQLException
{ }

public void appendToComments(int amount, String buffer)
throws SQLException
{ }

public void writeToComments(int offset, int amount, String buffer)
throws SQLException
```

```
{ }

String readFromComments(int offset, int amount)
throws SQLException
{ }

public int locateInComment(String pattern, int offset, int occurrence)
throws SQLException
{ }

public void trimComments(int newlen)
throws SQLException
{ }

public int eraseFromComments(int amount, int offset)
throws SQLException
{ }

public void deleteComments( )
throws SQLException
{ }

public CLOB copyCommentsOut(CLOB dest, int amount, int from_loc,
     int to_loc)
throws SQLException
{ }

public int compareComments(CLOB dest, int amount,
     int start_in_comment, int start_in_compare_comment)
throws SQLException
{ }

public void loadCommentsFromFile(String loc, String fileName,
     int amount, int from_loc, int to_loc)
throws SQLException
{ }

public int getCommentLength( )
throws SQLException
{ }

public String getAttribute(byte[ ] ctx, String name)
throws SQLException
{ }
```

```
public CLOB getAllAttributes(byte[ ] ctx)
throws SQLException
{ }

public String getAllAttributesAsString(byte[ ] ctx)
throws SQLException
{ }

public BLOB getContentInLob(byte[ ] ctx, StringBuffer mimeType,
     StringBuffer format)
throws SQLException
{ }

public int openSource(byte[ ] userArg, byte[ ] ctx)
throws SQLException
{ }

public int closeSource(byte[ ] ctx)
throws SQLException
{ }

public int trimSource(byte[ ] ctx, int newLen)
throws SQLException
{ }

public int readFromSource(byte[ ] ctx, int startPos, int numBytes,
     byte[ ] buffer)
throws SQLException
{ }

public int writeToSource(byte[ ] ctx, int startPos, int numBytes,
     byte[ ] buffer)
throws SQLException
{ }

public byte[ ] processSourceCommand(byte[ ] ctx, String cmd,
     String args, byte[ ] result)
throws SQLException
{ }

public byte[ ] processAudioCommand(byte[ ] ctx, String cmd, String args,
     byte[ ] result)
throws SQLException
{ }
```

```
    public CLOB getComments( )
    throws SQLException
    { }

    public void setComments(CLOB the_comments)
    throws SQLException
    { }

    public String getCommentsAsString( )
    throws SQLException, OutOfMemoryError
    { }

    public boolean loadComments(String filename)
    throws SQLException, IOException, SecurityException
    { }

    boolean loadCommentsInChunks(String fileName, int fileLength)
    throws SQLException, IOException
    { }
}
```

where the class attributes are defined as:

- encoding: describes the encoding type of the audio data.
- numberOfChannels: describes the number of audio channels in the audio data.
- samplingRate: describes the rate in samples per second at which the audio data was recorded.
- sampleSize: describes the sample width or number of audio samples of audio in the data.
- compressionType: describes the compression type of the audio data.
- audioDuration: describes the audio duration of the audio data.
- description: identifies the description of the audio data.
- comments: describes the comment information of the audio data.

5.2 Methods

This section presents ORDAudio reference information on the methods used for audio data manipulation. These methods are described in the following groupings:

ORDAudio Methods Associated with Audio Attribute Accessors

- public void setEncoding( ): set the encoding of the client-side ORDAudio object.
- public String getEncoding(byte[ ]): get the encoding of the server-side ORDAudio object by calling the server-side method getEncoding(ctx RAW), which extracts the format from the BLOB data.
- public String getEncoding( ): get the encoding of the client-side ORDAudio object from the client cache.
- public void setNumberOfChannels( ): set the number of channels in the client-side ORDAudio object.
- public int getNumberOfChannels(byte[ ]): get the number of channels in the server-side ORDAudio object by calling the server-side method getNumberOfChannels(ctx RAW), which extracts the information from the BLOB data.
- public int getNumberOfChannels( ): get the number of channels in the client-side ORDAudio object from the client cache.
- public void setSamplingRate( ): set the sampling rate of the client-side ORDAudio object.
- public int getSamplingRate(byte[ ]): get the sampling rate of the server-side ORDAudio object by calling the server-side method getSamplingRate(ctx RAW), which extracts the information from the BLOB data.
- public int getSamplingRate( ): get the sampling rate of the client-side ORDAudio object from the client cache.
- public void setSampleSize( ): set the sample size of the client-side ORDAudio object.
- public int getSampleSize(byte[ ]): get the sample size of the server-side ORDAudio object by calling the server-side method getSampleSize(ctx RAW), which extracts the sample size from the BLOB data.
- public int getSampleSize( ): get the sample size of the client-side ORDAudio object from the client cache.
- public void setCompressionType: set the compression type of the client-side ORDAudio object.

- public String getCompressionType(byte[ ]): get the compression type of the server-side ORDAudio object by calling the server-side method getCompressionType(ctx RAW), which extracts the compression type from the BLOB data.
- public String getCompressionType( ): get the compression type of the client-side ORDAudio object from the client cache.
- public void setAudioDuration( ): set the audio duration of the client-side ORDAudio object.
- public int getAudioDuration(byte[ ]): get the audio duration of the server-side ORDAudio object by calling the server-side method getAudioDuration(ctx RAW), which extracts the audio duration from the BLOB data.
- public int getAudioDuration( ): get the audio duration of the client-side ORDAudio object from the client cache.
- public String getFormat( ): get the format of the server-side ORDAudio object.
- public void setProperties( ): set the properties of the specified audio data on both the client-side and server-side ORDAudio objects.
- public boolean checkProperties( ): check that the properties in the server-side ORDAudio object are consistent with those stored in the server-side raw media data.
- public void setKnownAttributes( ): set the known attributes in the client-side ORDAudio object.
- public String getAttribute( ): get the attribute value of the server-side ORDAudio object.
- public CLOB getAllAttributes( ): get all the attribute values of the server-side ORDAudio object and return them as a CLOB.
- public String getAllAttributesAsString( ): get all the attribute values of the server-side ORDAudio object and return them as a String.

ORDAudio Methods Associated with the description Attribute

- public void setDescription( ): set the description attribute in the client-side ORDAudio object.
- public String getDescription( ): get the description attribute with extension in the client-side ORDAudio object.

ORDAudio Methods Associated with the comments Attribute

- public void setComments( ): set the comments attribute on the server side and the client cache with a CLOB.
- public CLOB getComments( ): copy the Character LOB locator from the server side to the client side and return the CLOB locator from the client cache.
- public String getCommentsAsString( ): return the comments from the server-side ORDAudio object as a String.
- public void appendToComments( ): append data to the server-side ORDAudio object comments attribute.
- public void writeToComments( ): write data to the server-side ORDAudio object comments attribute.
- public String readFromComments( ): read data from the server-side ORDAudio object comments attribute.
- public int locateInComment( ): locate a pattern in the server-side ORDAudio object comments attribute starting from a certain offset for a certain number of occurrences.
- public void trimComments( ): trim the server-side ORDAudio object comments attribute to a given length.
- public int eraseFromComments( ): erase data from the server-side ORDAudio object comments attribute starting from a certain offset to a certain length.
- public void deleteComments( ): delete the server-side ORDAudio object comments.
- public CLOB copyCommentsOut( ): copy comments out from a server-side location to a separate location.
- public int compareComments( ): compare the comments of the server-side ORDAudio object with a separate CLOB.
- public void loadCommentsFromFile( ): load a given amount of comments data to the server-side ORDAudio object from a separate server-side file.
- public boolean loadComments( ): load comments data to the server-side ORDAudio object from a separate client-side file.
- boolean loadCommentsInChunks( ): load comments data to the server-side ORDAudio object from a separate client-side file in chunks of 32K bytes.
- public int getCommentLength( ): get the length of the server-side ORDAudio object comments attribute.

ORDAudio Methods Associated with the Media Type

- String getMediaType( ): return the media type information.

ORDAudio Methods Associated with the Content Length

- public int getContentLength(byte[ ]): return the content length of the media data in the server-side ORDAudio object.
- public int getContentLength( ): return the content length of the media data in the server-side ORDAudio object.

ORDAudio Methods Associated with Communication Between the Client and Server

- public void refresh( ): copy the server-side ORDAudio object attribute values to the client-side ORDAudio object with or without locking the database row.
- public void flush( ): copy the client-side ORDAudio object attribute values to the server-side ORDAudio object.

ORDAudio Methods Associated with the SQL Type

- String getSQLConstructor( ): get the SQL type constructor for the ORDAudio object.
- String defineSQLResults( ): define the SQL results that will be used to assign values to the ORDAudio object attributes.
- int declareSQLResults( ): declare types for the SQL results corresponding to the different ORDAudio object attributes.
- String setSQLParams( ): set the values of the SQL type of the ORDAudio object as parameters.
- int bindInSQLParams( ): bind the values of the SQL type of the ORDAudio object.
- int getSQLResults( ): get the SQL results to assign to the attributes of the ORD-MultiMedia Object (for example, encoding or numberOfChannels).

ORDAudio Methods Associated with Generating SQL Queries

- String getFormatStr( ): return the format String.
- String getUpdStr( ): returns the SQL string that is used to update an ORDAudio instance.
- String getSourceStr( ): returns the source String.

- String getContentLengthAPI( ): returns the server-side contentLength API for this class.

ORDAudio Methods Associated with File Operations

- public int openSource( ): open the file source of the server-side ORDAudio object.
- public int closeSource( ): close the file source of the server-side ORDAudio object.
- public int trimSource( ): trim the file source of the server-side ORDAudio object to a given length.
- public int readFromSource( ): read a buffer of a given number of bytes from the server-side ORDAudio object source, beginning at a given initial position.
- public int writeToSource( ): write a buffer of a given number of bytes to the server-side ORDAudio object source, beginning at a given initial position.

ORDAudio Methods Associated with Source Content Operations

- public BLOB getContentInLob( ): get the localData content from the server-side ORDAudio object in a temporary BLOB.

ORDAudio Methods Associated with Processing Audio Data

- public byte[ ] processSourceCommand( ): call the server-side processCommand( ) method.
- public byte[ ] processAudioCommand( ): call the server-side processAudio-Command( ) method.

5.2.1 ORDAudio Methods Associated with the Audio Attribute Accessors

This section presents reference information on the ORDAudio methods associated with audio attribute accessors.

The methods described in this reference chapter show examples based on the instantiation of an ORDAudio object. For the examples in Section 5.2.1 through Section 5.2.11, please consult the following code:

```
public class clientProgram{
     public static void main(String[ ] args){
          The code in the examples appears here
     }
}
```

In each example, the parameter connection is a connection to the Oracle database.

setEncoding( ) Method

Format

```
public void setEncoding(String the_encoding)
```

Scope

public

Description

Sets the encoding in the client-side ORDAudio object.

Parameter

the_encoding
The encoding value to be set.

Returns

None.

Exceptions

None.

Usage

```
OrdAudio audioObj = new OrdAudio(connection);
.
.
.
audioObj.setEncoding("MULAW");
```

where:

- MULAW: is the encoding value to be set.

getEncoding(byte[ ]) Method

Format

```
public String getEncoding(byte[ ] ctx)
throws SQLException
```

Scope

public

Description

Gets the encoding from the server-side ORDAudio object by calling the server-side method getEncoding(ctx RAW), which extracts the information from the BLOB data.

Parameter

ctx
The format plug-in context information.

Returns

This method returns the encoding.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     String audStr = audioObj.getEncoding(ctx);
}
catch(SQLException e){
     .
     .
```

.
}

where:

- ctx: contains the context information of the caller.

getEncoding( ) Method

Format

```
public String getEncoding( )
```

Scope

public

Description

Gets the encoding from the client-side ORDAudio object from the client cache.

Parameter

None.

Returns

This method returns the encoding value present in the client object cache.

Exceptions

None.

Usage

```
OrdAudio audioObj = new OrdAudio(connection);
.
.
.
String audStr = audioObj.getEncoding( );
```

setNumberOfChannels( ) Method

Format

```
public void setNumberOfChannels(int the_numberOfChannels)
```

Scope

public

Description

Sets the number of channels in the client-side ORDAudio object.

Parameter

the_numberOfChannels
The number of channels value to be set.

Returns

None.

Exceptions

None.

Usage

```
OrdAudio audioObj = new OrdAudio(connection);
.
.
.
audioObj.setNumberOfChannels(1);
```

where:

- 1: is the number of channels to be set.

getNumberOfChannels(byte[ ]) Method

Format

```
public int getNumberOfChannels(byte[ ] ctx)
throws SQLException
```

Scope

public

Description

Gets the number of channels in the server-side ORDAudio object by calling the server-side method getNumberOfChannels(ctx RAW), which extracts the number of channels from the BLOB data.

Parameter

ctx
The format plug-in context information.

Returns

This method returns the number of channels.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     int i = audioObj.getNumberOfChannels(ctx);
}
catch(SQLException e){
     .
     .
```

```
        .
}
```

where:

- ctx: contains the context information of the caller.

getNumberOfChannels( ) Method

Format

```
public int getNumberOfChannels( )
```

Scope

public

Description

Returns the number of channels in the client-side ORDAudio object from the client cache.

Parameter

None.

Returns

This method returns the number of channels present in the client object cache.

Exceptions

None.

Usage

```
OrdAudio audioObj = new OrdAudio(connection);
.
.
.
int i = audioObj.getNumberOfChannels( );
```

setSamplingRate( ) Method

Format

```
public void setSamplingRate(int the_samplingRate)
```

Scope

public

Description

Sets the sampling rate in the client-side ORDAudio object.

Parameter

the_samplingRate
The sampling rate to be set.

Returns

None.

Exceptions

None.

Usage

```
OrdAudio audioObj = new OrdAudio(connection);
.
.
.
audioObj.setSamplingRate(10);
```

where:

- 10: is the sampling rate to be set.

getSamplingRate(byte[ ]) Method

Format

```
public int getSamplingRate(byte[ ] ctx)
throws SQLException
```

Scope

public

Description

Gets the sampling rate of the server-side ORDAudio object. The method gets the sampling rate by calling the server-side method getSamplingRate(ctx RAW), which extracts the sampling rate from the BLOB data.

Parameter

ctx
The format plug-in context information.

Returns

This method returns the sampling rate.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     int i = audioObj.getSamplingRate(ctx);
}
catch(SQLException e){
     .
     .
```

```
        .
}
```

where:

- ctx: contains the context information of the caller.

getSamplingRate( ) Method

Format

```
public int getSamplingRate( )
```

Scope

public

Description

Gets the sampling rate of the client-side ORDAudio object from the client cache.

Parameter

None.

Returns

This method returns the sampling rate value in the client object cache.

Exceptions

None.

Usage

```
OrdAudio audioObj = new OrdAudio(connection);
.
.
.
int i = audioObj.getSamplingRate( );
```

setSampleSize( ) Method

Format

```
public void setSampleSize(int the_sampleSize)
```

Scope

public

Description

Sets the sample size of the client-side ORDAudio object.

Parameter

the_sampleSize
The sample size to be set.

Returns

None.

Exceptions

None.

Usage

```
OrdAudio audioObj = new OrdAudio(connection);
.
.
.
audioObj.setSampleSize(100);
```

where:

- 100: is the sample size to be set.

getSampleSize(byte[ ]) Method

Format

```
public int getSampleSize(byte[ ] ctx)
throws SQLException
```

Scope

public

Description

Gets the sample size of the server-side ORDAudio object. The method gets the sample size by calling the server-side method getSampleSize(ctx RAW), which extracts the sample size from the BLOB data.

Parameter

ctx
The format plug-in context information.

Returns

This method returns the sample size.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     int i = audioObj.getSampleSize(ctx);
}
catch(SQLException e){
     .
     .
```

```
        .
}
```

where:

- ctx: contains the context information of the caller.

getSampleSize( ) Method

Format

```
public int getSampleSize( )
```

Scope

public

Description

Gets the sample size of the client-side ORDAudio object from the client cache.

Parameter

None.

Returns

This method returns the sample size value from the client object cache.

Exceptions

None.

Usage

```
OrdAudio audioObj = new OrdAudio(connection);
.
.
.
int i = audioObj.getSampleSize( );
```

setCompressionType( ) Method

Format

```
public void setCompressionType(String the_compressionType)
```

Scope

public

Description

Sets the compression type of the client-side ORDAudio object.

Parameter

the_compressionType
The compression type to be set.

Returns

None.

Exceptions

None.

Usage

```
OrdAudio audioObj = new OrdAudio(connection);
.
.
.
audioObj.setCompressionType("8BITMONOAUDIO");
```

where:

- 8BITMONOAUDIO: is the compression type to be set.

getCompressionType(byte[ ]) Method

Format

```
public String getCompressionType(byte[ ] ctx)
throws SQLException
```

Scope

public

Description

Gets the compression type of the server-side ORDAudio object. The method gets the compression type by calling the server-side method getCompressionType(ctx RAW), which extracts the compression type from the BLOB data.

Parameter

ctx
The format plug-in context information.

Returns

This method returns the compression type.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     String audStr = audioObj.getCompressionType(ctx);
}
catch(SQLException e){
     .
     .
```

```
        .
}
```

where:

- ctx: contains the context information of the caller.

getCompressionType( ) Method

Format

```
public String getCompressionType( )
```

Scope

public

Description

Gets the compression type of the client-side ORDAudio object from the client cache.

Parameter

None.

Returns

This method returns the compression type stored in the client object cache.

Exceptions

None.

Usage

```
OrdAudio audioObj = new OrdAudio(connection);
.
.
.
String audStr = audioObj.getCompressionType( );
```

setAudioDuration( ) Method

Format

```
public void setAudioDuration(int the_audioDuration)
```

Scope

public

Description

Sets the audio duration in the client-side ORDAudio object.

Parameter

the_audioDuration
The audio duration to be set.

Returns

None.

Exceptions

None.

Usage

```
OrdAudio audioObj = new OrdAudio(connection);
.
.
.
audioObj.setAudioDuration(100);
```

where:

- 100: is the audio duration to be set.

getAudioDuration(byte[ ]) Method

Format

```
public int getAudioDuration(byte[ ] ctx)
throws SQLException
```

Scope

public

Description

Gets the audio duration of the server-side ORDAudio object. The method gets the audio duration by calling the server-side method getAudioDuration(ctx RAW), which extracts the audio duration from the BLOB data.

Parameter

ctx
The format plug-in context information.

Returns

This method returns the audio duration.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     int i = audioObj.getAudioDuration(ctx);
}
catch(SQLException e){
     .
     .
```

```
        .
}
```

where:

- ctx: contains the context information of the caller.

getAudioDuration( ) Method

Format

```
public int getAudioDuration( )
```

Scope

public

Description

Gets the audio duration of the client-side ORDAudio object from the client cache.

Parameter

None.

Returns

This method returns the audio duration value stored in the client object cache.

Exceptions

None.

Usage

```
OrdAudio audioObj = new OrdAudio(connection);
.
.
.
int i = audioObj.getAudioDuration( );
```

getFormat( ) Method

Format

```
public String getFormat(byte[ ] ctx)
throws SQLException
```

Scope

public

Description

Gets the format in the server-side ORDAudio object.

Parameter

ctx
The format plug-in context information.

Returns

This method returns the format.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     String audStr = audioObj.getFormat(ctx);
}
catch(SQLException e){
     .
     .
     .
}
```

where:

- ctx: contains the context information of the caller.

setProperties( ) Method

Format

```
public void setProperties(byte[ ] ctx)
throws SQLException
```

Scope

public

Description

Sets the properties of the specified audio data in both the client-side and server-side ORDAudio objects.

The audio attributes to be set include the following: format, encoding type, number of channels, sampling rate, and sample size.

Parameter

ctx
The format plug-in context information.

Returns

None.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     audioObj.setProperties(ctx);
}
catch(SQLException e){
     .
```

```
      .
      .
}
```

where:

- ctx: contains the context information of the caller.

checkProperties( ) Method

Format

```
public boolean checkProperties(byte[ ] ctx)
throws SQLException
```

Scope

public

Description

Checks whether or not the properties stored in the server-side ORDAudio object are consistent with those in the server-side raw media data.

Parameter

ctx
The format plug-in context information.

Returns

This method returns *true* if the object cache values of the attributes match those derived from the BLOB data content; *false* otherwise.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     if(audioObj.checkProperties(ctx)){
          system.out.println("Values matched.");
     }
}
catch(SQLException e){
```

```
        .
        .
        .
}
```

where:

- ctx: contains the context information of the caller.

setKnownAttributes( ) Method

Format

```
public void setKnownAttributes(String knownFormat, String knownEncoding,
                               int knownNumberOfChannels,
                               int knownSampleSize, int knownSamplingRate,
                               String knownCompressionType,
                               int knownAudioDuration)
```

Scope

public

Description

Sets the known attributes of the client-side ORDAudio object on the client side.

Parameters

knownFormat
The audio data format.

knownEncoding
The audio data encoding.

knownNumberOfChannels
The number of channels.

knownSampleSize
The sample size.

knownSamplingRate
The sampling rate.

knownCompressionType
The compression type.

knownAudioDuration
The audio duration.

Returns

None.

Exceptions

None.

Usage

```
OrdAudio audioObj = new OrdAudio( );
audioObj.setKnownAttributes("AUFF", "MULAW", 1, 100, 10, "8BITMONOAUDIO", 100);
```

where:

- AUFF: is the data format.
- MULAW: is the audio data encoding.
- 1: is the number of channels.
- 100: is the sample size.
- 10: is the sampling rate.
- 8BITMONOAUDIO: is the compression type.
- 100: is the audio duration.

getAttribute( ) Method

Format

```
public String getAttribute(byte[ ] ctx, String name)
throws SQLException
```

Scope

public

Description

Gets the attribute value of the server-side ORDAudio object.

Parameters

ctx
The format plug-in context information.

name
The name of the attribute.

Returns

This method returns the attribute value.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     String audStr = audioObj.getAttribute(ctx, "encoding");
}
catch(SQLException e){
     .
```

```
      .
      .
}
```

where:

- ctx: contains the context information of the caller.
- encoding: is the name of the attribute.

getAllAttributes( ) Method

Format

```
public CLOB getAllAttributes(byte[ ] ctx)
throws SQLException
```

Scope

public

Description

Gets all the attributes values of the server-side ORDAudio object and returns them as a CLOB.

Parameter

ctx
The format plug-in context information.

Returns

This method returns all of the attribute values, as a CLOB.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     CLOB clob = audioObj.getAllAttributes(ctx);
}
catch(SQLException e){
     .
     .
     .
```

```
}
```

where:

- ctx: contains the context information of the caller.

getAllAttributesAsString( ) Method

Format

```
public String getAllAttributesAsString(byte[ ] ctx)
throws SQLException
```

Scope

public

Description

Gets all the attributes values of the server-side ORDAudio object as a String.

Parameter

ctx
The format plug-in context information.

Returns

This method returns all the attribute values, as a String. It will be formatted similarly to the following String:

sample_size=8, sampling_rate=8, num_channels=1, file_format=AUFF, data_type=8BITMONOAUDIO, encoding=MULAW

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     String audStr = audioObj.getAllAttributesAsString(ctx);
}
catch(SQLException e){
     .
```

```
      .
      .
}
```

where:

- ctx: contains the context information of the caller.

5.2.2 ORDAudio Methods Associated with the description Attribute

This section presents reference information on the ORDAudio methods associated with the description attribute.

See Section 5.2.1 for additional code required to run the example code.

setDescription( ) Method

Format

```
public void setDescription(String the_description)
```

Scope

public

Description

Sets the description attribute of the client-side ORDAudio object.

Parameter

the_description
The description.

Returns

None.

Exceptions

None.

Usage

```
OrdAudio audioObj = new OrdAudio(connection);
.
.
.
audioObj.setDescription("audio1.wav");
```

where:

- audio1.wav: is the description.

getDescription( ) Method

Format

```
public String getDescription( )
```

Scope

public

Description

Gets the description attribute with extension of the client-side ORDAudio object.

If a lock has been acquired on the row containing the object, this method gets the title attribute locally. Otherwise, this method gets the value from the server side by refreshing the client-side object.

Parameter

None.

Returns

This method returns the description.

Exceptions

None.

Usage

```
OrdAudio audioObj = new OrdAudio(connection);
.
.
.
String audStr = audioObj.getDescription( );
```

5.2.3 ORDAudio Methods Associated with the comments Attribute

This section presents reference information on the ORDAudio methods associated with the comments attribute.

See Section 5.2.1 for additional code required to run the example code.

setComments( ) Method

Format

```
public void setComments(CLOB the_comments)
throws SQLException
```

Scope

public

Description

Sets the comments attribute on the server side and in the client cache with a CLOB.

Parameter

the_comments
The comments data.

Returns

None.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     .
     .
     .
     audioObj.setComments(commentsFile);
}
catch(SQLException e){
     .
     .
     .
}
```

where:

- commentsFile: is the CLOB that contains the comments.

getComments( ) Method

Format

```
public CLOB getComments( )
throws SQLException
```

Scope

public

Description

Copies the CLOB from the server side to the client side and returns the LOB locator from the client cache.

Parameter

None.

Returns

This method returns the comments attribute from the client cache.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     .
     .
     .
     CLOB clob = audioObj.getComments( );
}
catch(SQLException e){
     .
     .
     .
}
```

getCommentsAsString( ) Method

Format

```
public String getCommentsAsString( )
throws SQLException, OutOfMemoryError
```

Scope

public

Description

Returns the comments from the server-side ORDAudio object as a String.

Parameter

None.

Returns

This method returns the comments attribute as a String.

Exceptions

java.sql.SQLException

java.lang.OutOfMemoryError

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     .
     .
     .
     String comments = audioObj.getCommentsAsString( );
}
catch(SQLException e){
     .
     .
     .
}
```

appendToComments( ) Method

Format

```
public void appendToComments(int amount, String buffer)
throws SQLException
```

Scope

public

Description

Appends data to the server-side ORDAudio object comments attribute.

Parameters

amount
The amount of data to be appended, in bytes.

buffer
The content to be written from the buffer.

Returns

None.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     .
     .
     .
     audioObj.appendToComments(5, "Drama");
}
catch(SQLException e){
     .
     .
```

```
        .
}
```

where:

- 5: is the amount of data to be appended, in bytes.
- Drama: is the content to be written from the buffer.

writeToComments( ) Method

Format

```
public void writeToComments(int offset, int amount, String buffer)
throws SQLException
```

Scope

public

Description

Writes data to the server-side ORDAudio object comments attribute.

Parameters

offset
The offset from the beginning from which this method will start writing.

amount
The amount of data to be written, in bytes.

buffer
The content to be written from the buffer.

Returns

None.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     .
     .
     .
     audioObj.writeToComments(0, 5, "Drama");
}
```

```
catch(SQLException e){
      .
      .
      .
}
```

where:

- 0: is the offset from the beginning from which this method will start writing.
- 5: is the amount of data to be written, in bytes.
- Drama: is the content to be written from the buffer.

readFromComments( ) Method

Format

```
public String readFromComments(int offset, int amount)
throws SQLException
```

Scope

public

Description

Reads data from the server-side ORDAudio object comments attribute.

Parameters

offset
The offset from the beginning from which this method will start reading.

amount
The amount of data to be read, in bytes.

Returns

This method returns the String that is read.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     .
     .
     .
     String audStr = audioObj.readFromComments(0, 5);
}
catch(SQLException e){
     .
     .
```

.
}

where:

- 0: is the offset from the beginning from which this method will start reading.
- 5: is the amount of data to be read, in bytes.

locateInComment( ) Method

Format

```
public int locateInComment(String pattern, int offset, int occurrence)
throws SQLException
```

Scope

public

Description

Locates a pattern in the server-side ORDAudio object that appears a certain number of times starting from a certain offset in the comments attribute.

Parameters

pattern
The pattern to be matched.

offset
The offset from the beginning from which this method will start searching.

occurrence
The number of times that the pattern should appear in the comments.

Returns

This method returns *true* if the pattern occurs in the comments attribute for the specified number of times after the offset.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     .
     .
     .
```

```
     int i = audioObj.locateInComment("Drama", 0, 10);
}
catch(SQLException e){
     .
     .
     .
}
```

where:

- Drama: is the pattern to be matched.
- 0: is the offset from the beginning from which this method will start searching.
- 10: is the number of times that the pattern should appear in the comments.

trimComments( ) Method

Format

```
public void trimComments(int newlen)
throws SQLException
```

Scope

public

Description

Trims the server-side ORDAudio object comments to a given length.

Parameter

newlen
The new length of the comments attribute after trimming.

Returns

None.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     .
     .
     .
     audioObj.trimComments(100);
}
catch(SQLException e){
     .
     .
     .
}
```

where:

- 100: is the new length of the comments attribute after trimming.

eraseFromComments( ) Method

Format

```
public int eraseFromComments(int amount, int offset)
throws SQLException
```

Scope

public

Description

Erases data from the server-side ORDAudio object comments starting from a certain offset and continuing to a certain length.

Parameters

amount
The amount to be erased, in bytes.

offset
The offset from the beginning from which this method will start erasing.

Returns

This method returns the amount of data erased, in bytes.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     .
     .
     .
     audioObj.eraseFromComments(100, 0);
}
catch(SQLException e){
     .
```

```
      .
      .
}
```

where:

- 100: is the number of bytes to be erased.
- 0: is the offset from the beginning from which this method will start erasing.

deleteComments( ) Method

Format

```
public void deleteComments( )
throws SQLException
```

Scope

public

Description

Deletes the server-side ORDAudio object comments.

Parameter

None.

Returns

None.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     .
     .
     .
     audioObj.deleteComments( );
}
catch(SQLException e){
     .
     .
     .
}
```

copyCommentsOut( ) Method

Format

```
public CLOB copyCommentsOut(CLOB dest, int amount, in from_loc,
                            int to_loc)
throws SQLException
```

Scope

public

Description

Copies comments out from the a server-side location to a new location.

Parameters

dest
The CLOB into which the comments will be copied.

amount
The amount of data to be copied, in bytes.

from_loc
The position where the copying will begin.

to_loc
The position where the copying will end.

Returns

This method returns the CLOB into which the comments are copied.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     .
```

```
      .
      .
      CLOB clob = audioObj.copyCommentsOut(my_CLOB, 100, 0, 99);
}
catch(SQLException e){
      .
      .
      .
}
```

where:

- my_CLOB: is the CLOB into which the comments will be copied.
- 100: is the amount of data to be copied, in bytes.
- 0: is the position where this method begins copying.
- 99: is the position where this method ends copying.

compareComments( ) Method

Format

```
public int compareComments(CLOB dest, int amount,
                           int start_in_comment,
                           int start_in_compare_comment)
throws SQLException
```

Scope

public

Description

Compares the comments of the server-side ORDAudio object with a separate CLOB.

Parameters

dest
The location of the comments data that will be compared with the comments data of this object.

amount
The amount of data to be compared, in bytes.

start_in_comment
The starting position in the current object.

start_in_compare_comment
The starting position in the separate object.

Returns

This method returns 1 if the comments match; -1 if the comments do not match.

Exceptions

java.sql.SQLException

Usage

```
try{
```

```
     OrdAudio audioObj = new OrdAudio(connection);
     OrdAudio audioObj2 = new OrdAudio(connection);
     .
     .
     .
     int i = audioObj.compareComments(audioObj2.comments, 100, 0, 50);
}
catch(SQLException e){
     .
     .
     .
}
```

where:

- audioObj2.comments: is the location of the CLOB that will be compared to the comments of the current object.
- 100: is the amount of data to be compared, in bytes.
- 0: is the initial position in the current object.
- 50: is the initial position in the separate object.

loadCommentsFromFile( ) Method

Format

```
public void loadCommentsFromFile(String loc, String fileName,
                                 int amount, int from_loc, int to_loc)
throws SQLException
```

Scope

public

Description

Loads a given amount of comments data to the server-side ORDAudio object from a separate server-side file.

Parameters

loc
The location of the file from which the comments will be loaded.

fileName
The name of the file from which the comments will be loaded.

amount
The amount of data to be loaded, in bytes.

from_loc
The start location in the comments attribute into which the comments will be loaded.

to_loc
The end location in the comments attribute into which the comments will be loaded.

Returns

None.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     BFILE my_bfile;
     .
     .
     .
     audioObj.loadCommentsFromFile("commentsdir", "my_bfile", 100, 0, 99);
}
catch(SQLException e){
     .
     .
     .
}
```

where:

- commentsdir: is the location of the file from which the comments will be loaded.
- my_bfile: is the name of the file from which the comments will be loaded. The name is created by the CREATE or REPLACE DIRECTORY command in SQL*PLUS or svrmgr.
- 100: is the amount of data to be loaded, in bytes.
- 0: is the start location in the comments attribute into which the comments will be loaded.
- 99: is the end location in the comments attribute into which the comments will be loaded.

loadComments( ) Method

Format

```
public boolean loadComments(String fileName)
throws SQLException, IOException, SecurityException
```

Scope

public

Description

Loads comments data to the server-side ORDAudio object from a separate client-side file.

Parameter

fileName
The name of the file from which the comments will be loaded.

Returns

This method returns *true* upon successful loading; *false* otherwise.

Exceptions

java.sql.SQLException

java.io.IOException

java.lang.SecurityException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     .
     .
     .
     audioObj.loadCommentsFromFile(my_file);
}
catch(SQLException e){
     .
```

```
      .
      .
}
catch(IOException io){
      .
      .
      .
}
catch(SecurityException s){
      .
      .
      .
}
```

where:

- my_file: is the name of the file from which the comments will be loaded.

loadCommentsInChunks( ) Method

Format

```
boolean loadCommentsInChunks(String fileName, int fileLength)
throws SQLException, IOException
```

Scope

package

Description

Loads comments to the server-side ORDAudio object from a separate client-side file in chunks of 32K at a time.

Parameters

fileName
The name of the file from which the comments will be loaded.

fileLength
The total length of the file, in bytes.

Returns

This method returns *true* upon successful loading; *false* otherwise.

Exceptions

java.sql.SQLException

java.io.IOException

Usage

Do not use this method unless you are extending the package.

getCommentLength( ) Method

Format

```
public int getCommentLength( )
throws SQLException
```

Scope

public

Description

Gets the length of the server-side ORDAudio object comments attribute.

Parameter

None.

Returns

This method returns the length of the comments attribute.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     .
     .
     .
     int i = audioObj.getCommentLength( );
}
catch(SQLException e){
     .
     .
     .
}
```

5.2.4 ORDAudio Methods Associated with the Media Type

This section presents reference information on the ORDAudio methods associated with the media type.

See Section 5.2.1 for additional code required to run the example code.

getMediaType( ) Method

Format

```
String getMediaType( )
```

Scope

public

Description

Returns the media type information.

Parameter

None.

Returns

This method returns the media type, which is OrdAudio.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

5.2.5 ORDAudio Methods Associated with the Content Length

This section presents reference information on the ORDAudio methods associated with the content length.

See Section 5.2.1 for additional code required to run the example code.

getContentLength(byte[ ]) Method

Format

```
public int getContentLength(byte[ ] ctx)
throws SQLException
```

Scope

public

Description

Returns the content length of the media data in the server-side ORDAudio object.

Parameter

ctx
The source plug-in context information.

Returns

This method returns the length of the data content of the media data, in bytes.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     int i = audioObj.getContentLength(ctx);
}
catch(SQLException e){
     .
     .
     .
}
```

where

- ctx: contains the context information of the caller.

getContentLength( ) Method

Format

```
public int getContentLength( )
throws SQLException
```

Scope

public

Description

Returns the content length of the media data in the server-side ORDAudio object.

Parameter

None.

Returns

This method returns the content length, in bytes.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     .
     .
     .
     int i = audioObj.getContentLength( );
}
catch{
     .
     .
     .
}
```

5.2.6 ORDAudio Methods Associated with Communication Between the Client and Server

This section presents reference information on the ORDAudio methods associated with communication between the client and server.

See Section 5.2.1 for additional code required to run the example code.

refresh( ) Method

Format

```
public void refresh(boolean forUpdate)
throws SQLException
```

Scope

public

Description

Copies the server-side ORDAudio object attribute values to the client-side ORDAudio object with or without locking the database row.

Parameter

forUpdate
The indicator for whether or not the option has been refreshed for update. It is *true* if the object is refreshed for update; otherwise, it is *false*.

Returns

None.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     .
     .
     .
     audioObj.refresh(false);
}
catch(SQLException e){
     .
     .
     .
```

```
}
```

where

- false: indicates that the option has not been refreshed for update.

flush( ) Method

Format

```
public void flush( )
throws SQLException
```

Scope

public

Description

Sends information from the client-side ORDAudio object to the server-side ORDAudio object.

Parameter

None.

Returns

None.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     .
     .
     .
     audioObj.flush( );
}
catch(SQLException e){
     .
     .
     .
}
```

5.2.7 ORDAudio Methods Associated with the SQL Type

This section presents reference information on the ORDAudio methods associated with the SQL type.

See Section 5.2.1 for additional code required to run the example code.

getSQLConstructor( ) Method

Format

```
String getSQLConstructor(boolean updateOption, String objName)
```

Scope

package

Description

Gets the SQL type constructor for the ORDAudio object.

It calls the corresponding method in the ORDMultiMedia superclass.

Parameters

updateOption
The indicator for whether or not the option will be updated.

objName
The variable name of the audio object.

Returns

This method returns the SQL string that will bind an object instance variable to a database object instance.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

defineSQLResults( ) Method

Format

```
String defineSQLResults(String var)
```

Scope

package

Description

Defines the SQL results that will be used to assign the values of the ORDAudio object attributes.

This method is used by the refresh( ) method to assign the server-side object attribute values to the corresponding client-side object attributes.

Parameter

var
The name of the audio object variable.

Returns

This method returns the SQL string that defines the SQL results.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

declareSQLResults( ) Method

Format

```
int declareSQLResults(int start, OracleCallableStatement stmt)
throws SQLException
```

Scope

package

Description

Declares types for the SQL results corresponding to the different ORDAudio object attributes.

This method is used by the refresh( ) method to declare the types of the client-side object attributes needed in order to assign values to the attributes corresponding to the server-side object attributes.

Parameters

start
The position in the statement where the binding begins.

stmt
The Oracle callable statement that will be executed on the server side.

Returns

This method returns an integer defined as position + 1, where position is the last index in the statement for which the type was declared.

Exceptions

java.sql.SQLException

Usage

Do not use this method unless you are extending the package.

setSQLParams( ) Method

Format

```
String setSQLParams(String var)
```

Scope

package

Description

Sets the values of the SQL type of the ORDAudio object as parameters.

This method is used by the flush( ) method to set the values of the different attributes of the server-side object with the client-side attributes.

Parameter

var
The name of the audio object variable.

Returns

This method returns the SQL string that sets the different attributes of the audio object.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

bindInSQLParams( ) Method

Format

```
int bindInSQLParams(int start, OracleCallableStatement stmt)
throws SQLException
```

Scope

package

Description

Binds the values of the SQL type of the ORDAudio object.

This method is used by the flush( ) method to set the values of the different attributes of the server-side object with the client-side attributes.

Parameters

start
The position in the statement where the binding begins.

stmt
The Oracle callable statement that will be executed on the server side.

Returns

This method returns an integer defined as position + 1, where position is the index of the last position in the statement to which a value was bound.

Exceptions

java.sql.SQLException

Usage

Do not use this method unless you are extending the package.

getSQLResults( ) Method

Format

```
int getSQLResults(int start, OracleCallableStatement stmt)
throws SQLException
```

Scope

package

Description

Gets the SQL results to assign to the attributes of the ORDMultiMedia object (for example, encoding or numberOfChannels).

This method is used by the refresh( ) method to assign the server-side object attribute values to the corresponding client-side object attributes.

Parameters

start
The starting position for getting results from the statement.

stmt
The Oracle callable statement that has been executed on the server side.

Returns

This method returns the SQL string that contains the PL/SQL DECLARE... BEGIN... END statement block.

Exceptions

java.sql.SQLException

Usage

Do not use this method unless you are extending the package.

5.2.8 ORDAudio Methods Associated with Generating SQL Queries

This section presents reference information on the ORDAudio methods associated with generating SQL queries.

See Section 5.2.1 for additional code required to run the example code.

getFormatStr( ) Method

Format

```
String getFormatStr( )
```

Scope

package

Description

Returns the format String, which is used by other methods to generate SQL queries.

Parameters

None.

Returns

This method returns the format String.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

getUpdStr( ) Method

Format

```
String getUpdStr( )
```

Scope

package

Description

Returns the SQL string that is used to update an ORDAudio instance, which is used by other methods to generate SQL queries.

Parameters

None.

Returns

This method returns the SQL string that is used to update an ORDAudio instance.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

getSourceStr( ) Method

Format

```
String getSourceStr( )
```

Scope

package

Description

Returns the source String, which is used by other methods to generate SQL queries.

Parameters

None.

Returns

This method returns the source String.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

getContentLengthAPI( ) Method

Format

```
String getContentLengthAPI( )
```

Scope

package

Description

Returns the server-side contentLength API for this class, which is used by other methods to generate SQL queries.

Parameters

None.

Returns

This method returns the server-side contentLength API for this class.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

5.2.9 ORDAudio Methods Associated with File Operations

This section presents reference information on the ORDAudio methods associated with file operations.

See Section 5.2.1 for additional code required to run the example code.

openSource( ) Method

Format

```
public int openSource(byte[ ] userArg, byte[ ] ctx)
throws SQLException
```

Scope

public

Description

Opens the file source of the server-side ORDAudio object.

Parameters

userArg
Permission-related parameters that are supplied by the user, such as READONLY.

ctx
The source plug-in context information.

Returns

This method returns 0 in case of success or an integer greater than 0 in case of failure.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     .
     .
     .
     int i = audioObj.openSource(READONLY, ctx);
}
catch(SQLException e){
     .
```

```
      .
      .
}
```

where:

- READONLY: sets the permission.
- ctx: contains the context information of the caller.

closeSource( ) Method

Format

```
public int closeSource(byte[ ] ctx)
throws SQLException
```

Scope

public

Description

Closes the file source of the server-side ORDAudio object.

Parameter

ctx
The source plug-in context information.

Returns

This method returns 0 in case of success or an integer greater than 0 in case of failure.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     int i = audioObj.closeSource(ctx);
}
catch(SQLException e){
     .
     .
     .
```

```
}
```

where:

- ctx: contains the context information of the caller.

trimSource( ) Method

Format

```
public int trimSource(byte[ ] ctx, int newLen)
throws SQLException
```

Scope

public

Description

Trims the file source of the server-side ORDAudio object to the given length.

Parameters

ctx
The source plug-in context information.

newLen
The length, in bytes, to which the file source will be trimmed.

Returns

This method returns 0 in case of success or an integer greater than 0 in case of failure.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     int i = audioObj.trimSource(ctx, 100);
}
catch(SQLException e){
```

```
      .
      .
      .
}
```

where:

- ctx: contains the context information of the caller.
- 100: is the length, in bytes, to which the file source will be trimmed.

readFromSource( ) Method

Format

```
public int readFromSource(byte[ ] ctx, int startPos, int numBytes,
                          byte[ ] buffer)
throws SQLException
```

Scope

public

Description

Reads a buffer of a given number of bytes from the server-side ORDAudio object source, beginning at a given initial position.

Parameters

ctx
The source plug-in context information.

startPos
The position where the method begins reading.

numBytes
The amount that will be read, in bytes.

buffer
The buffer into which the data will be read.

Returns

This method returns the amount of data read, in bytes.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
```

```
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     byte[ ] my_buffer;
     audioObj.readFromSource(ctx, 0, 100, my_buffer);
}
catch(SQLException e){
     .
     .
     .
}
```

where:

- ctx: contains the context information of the caller.
- 0: is the position where the method begins reading.
- 100: is the amount of data to be read, in bytes.
- my_buffer: is the buffer into which the data will be read.

writeToSource( ) Method

Format

```
public int writeToSource(byte[ ] ctx, int startPos, int numBytes,
                         byte[ ] buffer)
throws SQLException
```

Scope

public

Description

Writes a buffer of a given number of bytes to the server-side ORDAudio object source, beginning at a given initial position.

Parameters

ctx
The source plug-in context information.

startPos
The initial position for writing the new data in the source.

numBytes
The number of bytes to be written.

buffer
The content to be written.

Returns

This method returns the amount of data written, in bytes.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
```

```
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     byte[ ] my_buffer;
     audioObj.writeToSource(ctx, 0, 100, my_buffer);
}
catch(SQLException e){
     .
     .
     .
}
```

where:

- ctx: contains the context information of the caller.
- 0: is the initial position for writing the new data.
- 100: is the number of bytes to be written.
- my_buffer: contains the content to be written.

5.2.10 ORDAudio Methods Associated with Source Content Operations

This section presents reference information on the ORDAudio methods associated with source content operations.

See Section 5.2.1 for additional code required to run the example code.

getContentInLob( ) Method

Format

```
public BLOB getContentInLob(byte[ ] ctx, StringBuffer mimeType,
                            StringBuffer format)
throws SQLException
```

Scope

public

Description

Gets the local data content from the server-side ORDAudio object in the temporary BLOB.

Parameters

ctx
The source plug-in context information.

mimeType
The MIME type of the temporary BLOB.

format
The format of the temporary BLOB.

Returns

This method returns a BLOB that contains the new contents.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
```

```
      .
      BLOB blob = audioObj.getContentInLob(ctx, mimeType, format);
}
catch(SQLException e){
      .
      .
      .
}
```

where:

- ctx: contains the context information of the caller.
- mimeType: contains the MIME type of the temporary BLOB.
- format: contains the format of the temporary BLOB.

5.2.11 ORDAudio Methods Associated with Processing Audio Data

This section presents reference information on the ORDAudio methods associated with processing audio data.

See Section 5.2.1 for additional code required to run the example code.

processSourceCommand( ) Method

Format

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

Scope

public

Description

Calls the server-side processSourceCommand( ) on the ORDAudio object.

For more information on the commands that can be processed, see *Oracle8i interMedia Audio, Image, and Video User's Guide and Reference*.

Parameters

ctx
The source plug-in context information.

cmd
The command that is to be executed.

args
The arguments for the command.

result
The result of the command execution.

Returns

This method returns the result of the command execution.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     byte[ ] ctx = new byte[4000];
     String cmd = "compress";
     String arg_list = "compression_type";
     byte[ ] the_result;
     .
     .
     .
     byte[ ] result = audioObj.processSourceCommand(ctx, cmd, arg_list,
          the_result);
}
catch(SQLException e){
     .
     .
     .
}
```

where:

- ctx: contains the context information of the caller.
- cmd: contains the command to be executed.
- arg_list: contains the list of arguments.
- the_result: is the result of the command execution.

processAudioCommand( ) Method

Format

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

Scope

public

Description

Calls the server-side processAudioCommand( ) method.

For more information on the commands that can be processed, see *Oracle8i interMedia Audio, Image, and Video User's Guide and Reference*.

Parameters

ctx
The format plug-in context information.

cmd
The command that is to be executed.

args
The arguments for the command.

result
The result of the command execution.

Returns

This method returns the result of the command execution.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     byte[ ] ctx = new byte[4000];
     String cmd = "compress";
     String arg_list = "compression_type";
     byte[ ] the_result;
     .
     .
     .
     byte[ ] result = audioObj.processAudioCommand(ctx, cmd, arg_list,
          the_result);
}
catch(SQLException e){
     .
     .
     .
}
```

where:

- ctx: contains the context information of the caller.
- cmd: contains the command that is to be executed.
- arg_list: contains the arguments for the command.
- the_result: is the result of the command execution.

6

ORDImage Reference Information

Oracle8*i* *inter*Media Audio, Image, and Video Java Client contains information about the ORDImage:

- Object type -- see “ORDImage Object Type” in Section 6.1.
- Methods -- see Section 6.2.

6.1 Object Types

Oracle8*i* *inter*Media Audio, Image, and Video Java Client describes the ORDImage object type, which supports access to a variety of sources of image data.

ORDImage Object Type

The ORDImage object type supports the storage and management of image data.

> **Note:** In the following code, some methods will begin with *public*. This indicates that the method can be called by all classes in all packages. Methods that begin with *protected* are visible to all classes in the package, plus all subclasses.

This object type is defined as follows:

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

public class OrdImage extends OrdMultiMedia {

     int height;
     int width;
     int contentLength;
     String contentFormat;
     String compressionFormat;

     public OrdImage( )
     { }

     public OrdImage(Connection connection)
     { }

     String getMediaType( )
     { }

     String getFormatStr( )
     { }

     String getUpdStr( )
     { }

     String getSourceStr( )
```

```
{ }

String getContentLengthAPI( )
{ }

int bindInSQLParams(int start, OracleCallableStatement stmt)
throws SQLException
{ }

String defineSQLResults(String var)
{ }

int declareSQLResults(int start, OracleCallableStatement stmt)
throws SQLException
{ }

String setSQLParams(String var)
{ }

int getSQLResults(int start, OracleCallableStatement stmt)
throws SQLException
{ }

public void refresh(boolean forUpdate)
throws SQLException
{ }

public void flush( )
throws SQLException
{ }

public void copy(OrdImage dest)
throws SQLException
{ }

public void setProperties( )
throws SQLException
{ }

public void setProperties(String description)
throws SQLException
{ }

public boolean checkProperties( )
throws SQLException
```

```
    { }

    public int getHeight( )
    throws SQLException
    { }

    public int getContentLength( )
    throws SQLException
    { }

    public int getWidth( )
    throws SQLException
    { }

    public String getContentFormat( )
    throws SQLException
    { }

    public String getCompressionFormat( )
    throws SQLException
    { }

    public String getAllAttributesAsString( )
    throws Exception
    { }

    public void process(String cmd)
    throws SQLException
    { }

    public void processCopy(String cmd, OrdImage dest)
    throws SQLException
    { }
}
```

where the class attributes are defined as:

- height: contains the height of the image in pixels.
- width: contains the width of the image in pixels.
- contentLength: size of the image data in bytes.
- fileFormat: describes the file type or format in which the image data is stored (TIFF, JIFF, and so forth).

- contentFormat: describes the type of image (monochrome, 8-bit grayscale, and so forth).
- compressionFormat: describes the compression algorithm used on the image.

6.2 Methods

This section presents ORDImage reference information on the methods used for image data manipulation. These methods are described in the following groupings:

ORDImage Methods Associated with Image Attribute Accessors

- public int getHeight( ): get the image height of the client-side ORDImage object from the client cache.
- public int getWidth( ): get the image width of the client-side ORDImage object from the client cache.
- public int getContentLength( ): get the length of the content of the client-side ORDImage object from the client cache.
- public String getContentFormat( ): get the content format of the client-side ORDImage object from the client cache.
- public String getCompressionFormat( ): get the compression format of the client-side ORDImage object from the client cache.
- public String getAllAttributesAsString( ): get all the image attributes of the client-side ORDImage object and return them as a String.

ORDImage Methods Associated with the Media Type

- String getMediaType( ): return the media type information.

ORDImage Methods Associated with Communication Between the Client and Server

- public void refresh( ): send information from the server-side ORDImage object to the client-side ORDImage object with or without locking the database row.
- public void flush( ): send information from the client-side ORDImage object to the server-side ORDImage object.

ORDImage Methods Associated with the SQL Type

- int bindInSQLParams( ): bind the local values of the ORDImage object attributes to the parameters of the given statement.
- String defineSQLResults( ): construct a SQL statement to transfer data from a fetched object to the local object attributes.
- int declareSQLResults( ): bind the output parameters in a statement produced by the defineSQLResults( ) method to the attributes of the ORDImage object.

- String setSQLParams( ): construct a SQL statement to set the attributes of a database ORDImage object to the values specified by the binding parameters.
- int getSQLResults( ): update the attributes of the client-side ORDImage object with values from the database object fetched by the executing statement.

ORDImage Methods Associated with Generating SQL Queries

- String getFormatStr( ): return the format String.
- String getUpdStr( ): return the SQL string that is used to update an ORDImage instance.
- String getSourceStr( ): return the source String.
- String getContentLengthAPI( ): return the server-side contentLength API for this class.

ORDImage Methods Associated with properties Operations

- public void setProperties( ): set the client-side and server-side ORDImage objects attributes according to the values stored in the content data header.
- public void setProperties(String): set the client-side and server-side ORDImage object attributes according to the values provided in the parameter.
- public boolean checkProperties( ): check that the properties in the server-side ORDImage object are consistent with those stored in the server-side raw media data.

ORDImage Methods Associated with Copy Operations

- public void copy( ): copy the image data from the client-side ORDImage object source to a server-side ORDImage object.

ORDImage Methods Associated with Processing Image Data

- public void process( ): execute a given command on the server-side ORDImage object and update that object.
- public void processCopy( ): copy image data from the server-side ORDImage object to a destination ORDImage object, and modify the copy according to the specified command.

6.2.1 ORDImage Methods Associated with Image Attribute Accessors

This section presents reference information on the ORDImage methods associated with the height attribute.

The methods described in this reference chapter show examples based on the instantiation of an ORDImage object. For the examples in Section 6.2.1 through Section 6.2.8, please consult the following Java code:

```
import oracle.ord.media.mediaClass;

public class clientProgram {
     public static void main(String[ ] args){
           The code in the examples appears here
     }
}
```

In each example, the parameter connection is a connection to the Oracle database.

getHeight( ) Method

Format

```
public int getHeight( )
throws SQLException
```

Scope

public

Description

Gets the image height of the client-side ORDImage object attribute from the client cache.

Parameter

None.

Returns

This method returns the height.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdImage imageObj = new OrdImage(connection);
     .
     .
     .
     int i = imageObj.getHeight( );
}
catch(SQLException e){
     .
     .
     .
}
```

getWidth( ) Method

Format

```
public int getWidth( )
throws SQLException
```

Scope

public

Description

Gets the image width attribute of the client-side ORDImage object attribute from the client cache.

Parameter

None.

Returns

This method returns the width.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdImage imageObj = new OrdImage(connection);
     .
     .
     .
     int i = imageObj.getWidth( );
}
catch(SQLException e){
     .
     .
     .
}
```

getContentLength( ) Method

Format

```
public int getContentLength( )
throws SQLException
```

Scope

public

Description

Gets the length attribute of the client-side ORDImage object attribute from the client cache.

Parameter

None.

Returns

This method returns the content length.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdImage imageObj = new OrdImage(connection);
     .
     .
     .
     int i = imageObj.getContentLength( );
}
catch(SQLException e){
     .
     .
     .
}
```

getContentFormat Method

Format

```
public String getContentFormat( )
throws SQLException
```

Scope

public

Description

Gets the image content format attribute of the client-side ORDImage object attribute from the client cache.

Parameter

None.

Returns

This method returns the content format.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdImage imageObj = new OrdImage(connection);
     .
     .
     .
     String imgStr = imageObj.getContentFormat( );
}
catch(SQLException e){
     .
     .
     .
}
```

getCompressionFormat( ) Method

Format

```
public String getCompressionFormat( )
throws SQLException
```

Scope

public

Description

Gets the compression format attribute of the client-side ORDImage object attribute from the client cache.

Parameter

None.

Returns

This method returns the compression format.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdImage imageObj = new OrdImage(connection);
     .
     .
     .
     String imgStr = imageObj.getCompressionFormat( );
}
catch(SQLException e){
     .
     .
     .
}
```

getAllAttributesAsString( ) Method

Format

```
public String getAllAttributesAsString( )
throws Exception
```

Scope

public

Description

Gets all the image attributes of the client-side ORDImage object attribute and returns them as a String.

Parameter

None.

Returns

This method returns all the ORDImage attributes.

Exceptions

java.lang.Exception

Usage

```
try{
     OrdImage imageObj = new OrdImage(connection);
     .
     .
     .
     String imgStr = imageObj.getAllAttributesAsString( );
}
catch(Exception e){
     .
     .
     .
}
```

6.2.2 ORDImage Methods Associated with the Media Type

This section presents reference information on the ORDImage methods associated with the media type.

See Section 6.2.1 for additional code required to run the example code.

getMediaType( ) Method

Format

```
package String getMediaType( )
```

Scope

package

Description

Returns the media type information.

Parameter

None.

Returns

This method returns the media type, which is OrdImage.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

6.2.3 ORDImage Methods Associated with Communication Between the Client and Server

This section presents reference information on the ORDImage methods associated with the communication between the client and server.

See Section 6.2.1 for additional code required to run the example code.

refresh( ) Method

Format

```
public void refresh(boolean forUpdate)
throws SQLException
```

Scope

public

Description

Sends information from the server-side ORDImage object to the client-side ORDImage object with or without locking the database row.

Parameter

forUpdate
The indicator for whether or not the option is selected for update. It is set to *true* if it is selected for update; *false* otherwise.

Returns

None.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdImage imageObj = new OrdImage(connection);
     .
     .
     .
     imageObj.refresh(false);
}
catch(SQLException e){
     .
     .
     .
```

```
}
```

where

- false: indicates that the option has not been refreshed for update.

flush( ) Method

Format

```
public void flush( )
throws SQLException
```

Scope

public

Description

Sends information from the client-side ORDImage object to the server-side ORDImage object.

Parameter

None.

Returns

None.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdImage imageObj = new OrdImage(connection);
     .
     .
     .
     imageObj.flush( );
}
catch(SQLException e){
     .
     .
     .
}
```

6.2.4 ORDImage Methods Associated with the SQL Type

This section presents reference information on the ORDImage methods associated with the SQL type.

See Section 6.2.1 for additional code required to run the example code.

bindInSQLParams Method

Format

```
int bindInSQLParams(int start, OracleCallableStatement stmt)
throws SQLException
```

Scope

package

Description

Binds the local values of the ORDImage object attributes to the parameters of the given statement.

Parameters

start
The initial position of the binding parameter index.

stmt
The Oracle callable statement to which the values will be bound.

Returns

This method returns an integer containing the next position to be bound.

Exceptions

java.sql.SQLException

Usage

Do not use this method unless you are extending the package.

defineSQLResults( ) Method

Format

```
String defineSQLResults(String var)
```

Scope

package

Description

Constructs a SQL statement to transfer data from a fetched object to the local object attributes.

Parameter

var
The name of the image object variable to be used in the PL/SQL block.

Returns

This method returns the SQL assignments from the fetched object to the binding attributes.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

declareSQLResults( ) Method

Format

```
int declareSQLResults(int start, OracleCallableStatement stmt)
throws SQLException
```

Scope

package

Description

Binds the output parameters in a statement produced by the defineSQLResults( ) method to the attributes of the ORDImage object.

Parameters

start
The initial position of the binding parameter index.

stmt
The Oracle callable statement to which the parameters will be bound.

Returns

This method returns an integer value indicating the next binding position.

Exceptions

java.sql.SQLException

Usage

Do not use this method unless you are extending the package.

setSQLParams( ) Method

Format

```
String setSQLParams(String var)
```

Scope

package

Description

Constructs a SQL statement to set the attributes of a database ORDImage object to the values specified by the binding parameters.

Parameter

var
The name of the image object variable in the PL/SQL block.

Returns

This method returns the SQL assignments from the binding parameters to the database object attributes.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

getSQLResults( ) Method

Format

```
int getSQLResults(int start, OracleCallableStatement stmt)
throws SQLException
```

Scope

package

Description

Updates the attributes of the client-side ORDImage object with values from the database object fetched by the executing statement.

Parameters

start
The initial position of the binding index.

stmt
The Oracle callable statement that will fetch the values from the database object.

Returns

This method returns an integer value indicating the next binding position for the statement.

Exceptions

java.sql.SQLException

Usage

Do not use this method unless you are extending the package.

6.2.5 ORDImage Methods Associated with Generating SQL Queries

This section presents reference information on the ORDImage methods associated with generating SQL queries.

See Section 6.2.1 for additional code required to run the example code.

getFormatStr( ) Method

Format

```
String getFormatStr( )
```

Scope

package

Description

Returns the format String, which is used by other methods to generate SQL queries.

Parameters

None.

Returns

This method returns the format String.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

getUpdStr( ) Method

Format

```
String getUpdStr( )
```

Scope

package

Description

Returns the SQL string that is used to update an ORDImage instance, which is used by other methods to generate SQL queries.

Parameter

None.

Returns

This method returns the SQL string that is used to update an ORDImage instance.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

getSourceStr( ) Method

Format

```
String getSourceStr( )
```

Scope

package

Description

Returns the source String, which is used by other methods to generate SQL queries.

Parameters

None.

Returns

This method returns the source String.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

getContentLengthAPI( ) Method

Format

```
String getContentLengthAPI( )
```

Scope

package

Description

Returns the server-side contentLength API for this class, which is used by other methods to generate SQL queries.

Parameters

None.

Returns

This method returns the server-side contentLength API for this class.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

6.2.6 ORDImage Methods Associated with properties Operations

This section presents reference information on the ORDImage methods associated with the properties attribute.

See Section 6.2.1 for additional code required to run the example code.

setProperties( ) Method

Format

```
public void setProperties( )
throws SQLException
```

Scope

public

Description

Sets the client-side and server-side ORDImage objects attributes according to the values stored in the content data header. The local member attributes will be updated. This method sets the following attributes: height, width, data size, file type, image type, and compression type.

Parameter

None.

Returns

None.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdImage imageObj = new OrdImage(connection);
     .
     .
     .
     imageObj.setProperties( );
}
catch(SQLException e){
     .
     .
     .
}
```

setProperties(String) Method

Format

```
public void setProperties(String description)
throws SQLException
```

Scope

public

Description

Sets the client-side and server-side ORDImage objects attributes according to the values provided in the parameter. This method sets the following attributes: height, width, data size, file type, image type, and compression type.

Passing an empty String for the parameter will cause a call to the server with no description parameter.

This object's member attributes will be refreshed from the database after the setProperties(String) method has been executed.

Parameter

description
The values to which the attributes will be set.

Returns

None.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdImage imageObj = new OrdImage(connection);
     .
     .
     .
```

```
     imageObj.setProperties(null);
}
catch(SQLException e){
     .
     .
     .
}
```

where:

- null: indicates that there is no description parameter.

checkProperties( ) Method

Format

```
public boolean checkProperties( )
throws SQLException
```

Scope

public

Description

Checks that the properties in the server-side ORDImage object are consistent with those stored in the server-side raw media data.

Parameter

None.

Returns

This method returns *true* if all database object attribute values are consistent with the values in the content header; *false* otherwise.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdImage imageObj = new OrdImage(connection);
     .
     .
     .
     if(imageObj.checkProperties( ))
          system.out.println("Properties are consistent.");
}
catch(SQLException e){
     .
     .
     .
}
```

6.2.7 ORDImage Methods Associated with Copy Operations

This section presents reference information on the ORDImage methods associated with source content operations.

See Section 6.2.1 for additional code required to run the example code.

copy( ) Method

Format

```
public void copy(OrdImage dest)
throws SQLException
```

Scope

public

Description

Copies the image data from the client-side ORDImage object source to a server-side ORDImage object.

Parameter

dest
The ORDImage that is the target of the copy operation.

Returns

None.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdImage imageObj = new OrdImage(connection);
     OrdImage imageObj2 = new OrdImage(connection);
     .
     .
     .
     imageObj.copy(imageObj2);
} catch(SQLException e){
     .
     .
     .
}
```

6.2.8 ORDImage Methods Associated with Processing Image Data

This section presents reference information on the ORDImage methods associated with processing commands to the external source.

See Section 6.2.1 for additional code required to run the example code.

process( ) Method

Format

```
public void process(String cmd)
throws SQLException
```

Scope

public

Description

Executes a given command on the server-side ORDImage object and updates the database object.

For more information on the commands that can be processed, see *Oracle8i interMedia Audio, Image, and Video User's Guide and Reference*.

Parameter

cmd
The command to be executed on the server side.

Returns

None.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdImage imageObj = new OrdImage(connection);
     String command = "FileFormat=JIFF maxScale = 32 32";
     .
     .
     .
     imageObj.process(command);
}
catch(SQLException e){
```

```
        .
        .
        .
}
```

where:

- command: contains the command to be processed.

processCopy( ) Method

Format

```
public void processCopy(String cmd, OrdImage dest)
throws SQLException
```

Scope

public

Description

Copies image data from the server-side ORDImage object to a destination ORDImage object and modifies the copy according to the specified command.

For more information on the commands that can be processed, see *Oracle8i interMedia Audio, Image, and Video User's Guide and Reference*.

Parameters

cmd
The command to be executed on the server side.

dest
The ORDImage object that will receive the results of the process operation.

Returns

None.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdImage imageObj = new OrdImage(connection);
     OrdImage imageObj2 = new OrdImage(connection);
     String command = "contentFormat = monochrome";
     .
     .
```

```
      .
      imageObj.processCopy(command, imageObj2);
}
catch(SQLException e){
      .
      .
      .
}
```

where:

- command: is the command to be processed.
- imageObj2: is the ORDImage object that will receive the results of the process operation.

7

ORDVideo and FrameDimension Reference Information

Oracle8*i* *inter*Media Audio, Image, and Video Java Client contains information about the ORDVideo:

- Object type -- see "ORDVideo Object Type" in Section 7.1.
- Methods -- see Section 7.2.

Oracle8*i* *inter*Media Audio, Image, and Video Java Client contains information about the FrameDimension:

- Object type -- see "FrameDimension Object Type" in Section 7.1.
- Methods -- see Section 7.3.

Methods invoked at the ORDVideo level that are handed off for processing to the server-side source plug-in or server-side format plug-in have `byte[ ] ctx` as a context parameter. The space for the parameter is created by the client (in the reference examples, 4000 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.

> **Note:** In the current release, not all source plug-ins or format plug-ins will use or generate the context parameter, but if you include the parameter as previously described, your application should work with any current or future source plug-ins or format plug-ins.

See *Oracle8i interMedia Audio, Image, and Video User's Guide and Reference* for more information.

7.1 Object Types

Oracle8*i* *inter*Media Audio, Image, and Video Java Client describes the ORDVideo object type, which supports the storage and management of video data.

ORDVideo Object Type

The ORDVideo object type supports the storage and management of video data.

> **Note:** In the following code, some methods will begin with *public*. This indicates that the method can be called by all classes in all packages. The methods that are not designated as public have the default protection, which means that they can be called only by other methods in the package.

This object type is defined as follows:

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

public class OrdVideo extends OrdMultiMedia {

     int width;
     int height;
     int frameResolution;
     int frameRate;
     int videoDuration;
     int numberOfFrames;
     String compressionType;
     int numberOfColors;
     int bitRate;
     String description;
     CLOB comments;

     public OrdVideo( )
     { }

     public OrdVideo(Connection the_connection)
     { }

     String getMediaType( )
     { }
```

```
String getFormatStr( )
{ }

String getUpdStr( )
{ }

String getSourceStr( )
{ }

String getContentLengthAPI( )
{ }

public int getContentLength( )
throws SQLException
{ }

String getSQLConstructor(boolean updateOption, String objName)
{ }

String defineSQLResults(String var)
{ }

int declareSQLResults(int start, OracleCallableStatement stmt)
throws SQLException
{ }

String setSQLParams(String var)
{ }

int bindInSQLParams(int start, OracleCallableStatement stmt)
throws SQLException
{ }

int getSQLResults(int start, OracleCallableStatement stmt)
throws SQLException
{ }

public void refresh(boolean forUpdate)
throws SQLException
{ }

public void flush( )
throws SQLException
{ }
```

```
public void setWidth(int the_width)
{ }

public void setHeight(int the_height)
{ }

public void setFrameResolution(int the_frameResolution)
{ }

public void setFrameRate(int the_frameRate)
{ }

public void setNumberOfFrames(int the_numberOfFrames)
{ }

public void setVideoDuration(int the_videoDuration)
{ }

public void setCompressionType(String the_compressionType)
{ }

public void setNumberOfColors(int the_numberOfColors)
{ }

public void setBitRate(int the_bitRate)
{ }

public void setKnownAttributes(String knownFormat, int knownWidth,
     int knownHeight, int knownFrameResolution, int knownFrameRate,
     int knownVideoDuration, int knownNumberOfFrames,
     String knownCompressionType, int knownNumberOfColors,
     int knownBitRate)
{ }

public String getFormat(byte[ ] ctx)
throws SQLException
{ }

public FrameDimension getFrameSize(byte[ ] ctx)
throws SQLException
{ }

public int getWidth( )
{ }
```

```
public int getHeight( )
{ }

public int getFrameResolution(byte[ ] ctx)
throws SQLException
{ }

public int getFrameResolution( )
{ }

public int getFrameRate(byte[ ] ctx)
throws SQLException
{ }

public int getFrameRate( )
{ }

public int getNumberOfFrames(byte[ ] ctx)
throws SQLException
{ }

public int getNumberOfFrames( )
{ }

public int getVideoDuration(byte[ ] ctx)
throws SQLException
{ }

public int getVideoDuration( )
{ }

public String getCompressionType(byte[ ] ctx)
throws SQLException
{ }

public String getCompressionType( )
{ }

public int getNumberOfColors(byte[ ] ctx)
throws SQLException
{ }

public int getNumberOfColors( )
{ }
```

```
public int getBitRate(byte[ ] ctx)
throws SQLException
{ }

public int getBitRate( )
{ }

public void setDescription(String the_description)
throws SQLException
{ }

public String getDescription( )
throws SQLException
{ }

public int getContentLength(byte[ ] ctx)
throws SQLException
{ }

public void setProperties(byte[ ] ctx)
throws SQLException
{ }

public boolean checkProperties(byte[ ] ctx)
throws SQLException
{ }

public void appendToComments(int amount, String buffer)
throws SQLException
{ }

public void writeToComments(int offset, int amount, String buffer)
throws SQLException
{ }

public String readFromComments(int offset, int amount)
throws SQLException
{ }

public int locateInComment(String pattern, int offset, int occurrence)
throws SQLException
{ }

public void trimComments(int newlen)
throws SQLException
```

```
{ }

public int eraseFromComments(int amount, int offset)
throws SQLException
{ }

public void deleteComments( )
throws SQLException
{ }

public CLOB copyCommentsOut(CLOB dest, int amount, int from_loc,
     int to_loc)
throws SQLException
{ }

public int compareComments(CLOB dest, int amount, int start_in_comment,
     int start_in_compare_comment)
throws SQLException
{ }

public void loadCommentsFromFile(String loc, String fileName,
     int amount, int from_loc, int to_loc)
throws SQLException
{ }

public int getCommentLength( )
throws SQLException
{ }

public String getAttribute(byte[ ] ctx, String name)
throws SQLException
{ }

public CLOB getAllAttributes(byte[ ] ctx)
throws SQLException
{ }

public String getAllAttributesAsString(byte[ ] ctx)
throws SQLException
{ }

public BLOB getContentInLob(byte[ ] ctx, StringBuffer mimeType,
     StringBuffer format)
throws SQLException
{ }
```

```
public int openSource (byte[ ] userArg, byte[ ] ctx)
throws SQLException
{ }

public int closeSource (byte[ ] ctx)
throws SQLException
{ }

public int trimSource (byte[ ] ctx, int newLen)
throws SQLException
{ }

public int readFromSource(byte[ ] ctx, int startPos, int numBytes,
     byte[ ] buffer)
throws SQLException
{ }

public int writeToSource(byte[ ] ctx, int startPos, int numBytes,
     byte[ ] buffer)
throws SQLException
{ }

public byte[ ] processSourceCommand(byte[ ] ctx, String cmd, String args,
     byte[ ] result)
throws SQLException
{ }

public byte[ ] processVideoCommand(byte[ ] ctx, String cmd, String args,
     byte[ ] result)
throws SQLException
{ }

public CLOB getComments( )
throws SQLException
{ }

public void setComments(CLOB the_comments)
throws SQLException
{ }
public String getCommentsAsString( )
throws SQLException, OutOfMemoryError
{ }

public boolean loadComments(String filename)
```

```
     throws SQLException, IOException, SecurityException
     { }

     boolean loadCommentsInChunks(String fileName, int fileLength)
     throws SQLException, IOException
     { }
}
```

where the class attributes are defined as:

- width: describes the width of each frame of the video data.
- height: describes the height of each frame of the video data.
- frameResolution: describes the frame resolution of the video data.
- frameRate: describes the frame rate of the video data.
- videoDuration: describes the time it takes to play the entirety of the video data.
- numberOfFrames: describes the number of frames in the video data.
- compressionType: describes the compression type of the video data.
- numberOfColors: describes the number of colors in the video data.
- bitRate: describes the bit rate of the video data.
- description: identifies the description of the video object.
- comments: describes the comment information of the video object.

FrameDimension Object Type

The FrameDimension object type defines the FrameDimension object, which is used to store video width and height information.

> **Note:** In the following code, the methods begin with *public*. This indicates that the method can be called by all classes in all packages.

This object type is defined as follows:

```
package oracle.ord.media;

public class FrameDimension {
     int width;
     int height;

     /**
      * Constructor
      */

     public FrameDimension( )
     { }

     public int getWidth( )
     { }

     public int getHeight( )
     { }

     public void setWidth(int the_width)
     { }

     public void setHeight(int the_height)
     { }

}
```

where the class attributes are defined as:

- width: is the width in an ORDVideo object.
- height: is the height in an ORDVideo object.

7.2 ORDVideo Methods

This section presents ORDVideo reference information on the methods used for video data manipulation. These methods are described in the following groupings:

ORDVideo Methods Associated with Video Attribute Accessors

- public void setWidth( ): set the width of the client-side ORDVideo object.
- public void setHeight( ): set the height of the client-side ORDVideo object.
- public FrameDimension getFrameSize( ): get the frame width and height from the server-side ORDVideo object.
- public int getWidth( ): get the width of the client-side ORDVideo object from the client cache.
- public int getHeight( ): get the height of the server-side ORDVideo object from the client cache.
- public void setFrameResolution( ): set the frame resolution of the client-side ORDVideo object.
- public int getFrameResolution(byte[ ]): get the frame resolution of the server-side ORDVideo object by calling the server-side method getFrameResolution(ctx RAW), which extracts the information from the BLOB data.
- public int getFrameResolution( ): get the frame resolution of the client-side ORDVideo object from the client cache.
- public void setFrameRate( ): set the frame rate of the client-side ORDVideo object.
- public int getFrameRate(byte[ ]): get the frame rate of the server-side ORDVideo object by calling the server-side method getFrameRate(ctx RAW), which extracts the information from the BLOB data.
- public int getFrameRate( ): get the frame rate of the client-side ORDVideo object from the client cache.
- public void setVideoDuration( ): set the video duration of the client-side ORDVideo object.
- public int getVideoDuration(byte[ ]): get the video duration of the server-side ORDVideo object by calling the server-side method getVideoDuration(ctx RAW), which extracts the information from the BLOB data.

- public int getVideoDuration( ): get the video duration of the client-side ORDVideo object from the client cache.
- public void setNumberOfFrames( ): set the number of frames in the client-side ORDVideo object.
- public int getNumberOfFrames(byte[ ]): get the number of frames in the server-side ORDVideo object by calling the server-side method getNumberOfFrames(ctx RAW), which extracts the information from the BLOB data.
- public int getNumberOfFrames( ): get the number of frames in the client-side ORDVideo object from the client cache.
- public void setCompressionType( ): set the compression type of the client-side ORDVideo object.
- public String getCompressionType(byte[ ]): get the compression of the server-side ORDVideo object by calling the server-side method getCompressionType(ctx RAW), which extracts the information from the BLOB data.
- public String getCompressionType( ): get the compression type of the client-side ORDVideo object from the client cache.
- public void setNumberOfColors( ): set the number of colors in the client-side ORDVideo object.
- public int getNumberOfColors(byte[ ]): get the number of colors in the server-side ORDVideo object by calling the server-side method getNumberOfColors(ctx RAW), which extracts the information from the BLOB data.
- public int getNumberOfColors( ): get the number of colors in the client-side ORDVideo object from the client cache.
- public void setBitRate( ): set the bit rate of the client-side ORDVideo object.
- public int getBitRate(byte[ ]): get the bit rate of the server-side ORDVideo object by calling the server-side method getBitRate(ctx RAW), which extracts the information from the BLOB data.
- public int getBitRate( ): get the bit rate of the client-side ORDVideo object from the client cache.
- public String getFormat( ): get the format of the server-side ORDVideo object.

- public void setProperties( ): set the properties of the specified video data in both the client-side and server-side ORDVideo object.
- public boolean checkProperties( ): check that the properties in the server-side ORDVideo object are consistent with those stored in the server-side raw media data.
- public void setKnownAttributes( ): set the known attributes of the client-side ORDVideo object.
- public String getAttribute( ): get the value of an attribute of the server-side ORDVideo object.
- public CLOB getAllAttributes( ): get the values of all the attributes of the server-side ORDVideo object and return them as a CLOB.
- public String getAllAttributesAsString( ): get the values of all the attributes of the server-side ORDVideo object and return them as a String.

ORDVideo Methods Associated with the description Attribute

- public void setDescription( ): set the description attribute of the client-side ORDVideo object from the client cache.
- public String getDescription( ): get the description attribute with extension of the client-side ORDVideo object from the client cache.

ORDVideo Methods Associated with the comments Attribute

- public void setComments( ): set the comments attribute in the client cache and on the server side with a CLOB.
- public CLOB getComments( ): copy the CLOB locator from the server-side ORDVideo object to the client-side ORDVideo object and return the LOB locator from the client cache.
- public String getCommentsAsString( ): return the server-side ORDVideo object comments as a String.
- public void appendToComments( ): append data to the server-side ORDVideo object comments attribute.
- public void writeToComments( ): write data to the server-side ORDVideo object comments attribute.
- public String readFromComments( ): read data from the server-side ORDVideo object comments attribute.

- public int locateInComment( ): locate a pattern that starts from a certain offset and appears a certain number of times in the server-side ORDVideo object comments attribute.
- public void trimComments( ): trim the server-side ORDVideo object comments to a given length.
- public int eraseFromComments( ): erase data from the server-side ORDVideo object comments starting from a given offset and continuing for a given length.
- public void deleteComments( ): delete the server-side ORDVideo object comments.
- public CLOB copyCommentsOut( ): copy comments out from a server-side location to a separate location.
- public int compareComments( ): compare the comments of a server-side ORDVideo object with the comments of a separate CLOB.
- public void loadCommentsFromFile( ): load a given amount of comments data to the server-side ORDVideo object comments attribute from a separate server-side file.
- public boolean loadComments( ): load comments to the server-side ORDVideo object comments attribute from a client-side file.
- boolean loadCommentsInChunks( ): load comments to the server-side ORDVideo object comments attribute from a client-side file in chunks of 32K bytes.
- public int getCommentLength( ): get the length of the server-side ORDVideo comments attribute.

ORDVideo Methods Associated with Communication Between the Client and Server

- public void refresh( ): copy the server-side ORDVideo object attribute values to the client-side ORDVideo object with or without locking the database row.
- public void flush( ): copy the client-side ORDVideo object attribute values to the server-side ORDVideo object.

ORDVideo Methods Associated with the Media Type

- String getMediaType( ): return the media type information.

ORDVideo Methods Associated with the Content Length

- public int getContentLength(byte[ ]): return the content length of the server-side ORDVideo object media data.
- public int getContentLength( ): return the content length of the server-side ORDVideo object media data.

ORDVideo Methods Associated with the SQL Type

- String getSQLConstructor( ): get the SQL type constructor for the ORDVideo object.
- String defineSQLResults( ): define the SQL results that will be used to assign values to the ORDVideo object attributes.
- int declareSQLResults( ): declare the types for the SQL results corresponding to the different ORDVideo object attributes.
- String setSQLParams( ): set the values of the SQL type of the ORDVideo object as parameters.
- int bindInSQLParams( ): bind the values of the SQL type of the ORDVideo object.
- int getSQLResults( ): get the SQL results in order to assign them to the attributes of the ORDMultiMedia object (for example, encoding or numberOfChannels).

ORDVideo Methods Associated with Generating SQL Queries

- String getFormatStr( ): return the format String.
- String getUpdStr( ): return the SQL string that is used to update an ORDVideo instance.
- String getSourceStr( ): return the source String.
- String getContentLengthAPI( ): return the server-side contentLength API for this class.

ORDVideo Methods Associated with File Operations

- public int openSource( ): open the server-side ORDVideo object file source.
- public int closeSource( ): close the server-side ORDVideo object file source.
- public int trimSource( ): trim the server-side ORDVideo object file source to the given length.

- public int readFromSource( ): read a buffer of a given number of bytes from a server-side ORDVideo object source, beginning at a given initial position.
- public int writeToSource( ): write a buffer of a given number of bytes to the server-side ORDVideo object source, beginning at a given initial position.

ORDVideo Methods Associated with Source Content Operations

- public BLOB getContentInLob( ): get the contents of the server-side ORDVideo object localData attribute and store them in a temporary BLOB.

ORDVideo Methods Associated with Processing Commands to the External Source

- public byte[ ] processSourceCommand( ): call the server-side processSourceCommand( ) method.
- public byte[ ] processVideoCommand( ): call the server-side processVideoCommand( ) method.

7.2.1 ORDVideo Methods Associated with Video Attribute Accessors

This section presents reference information on the ORDVideo methods associated with video attribute accessors.

The methods described in this reference chapter show examples based on the instantiation of an ORDVideo object. For the examples in Section 7.2.1 through Section 7.2.11, please consult the following Java code:

```
import oracle.ord.media.*;

public class clientProgram {
     public static void main(String[ ] args){
           The code in the examples appears here
     }
}
```

In each example, the parameter connection is a connection to the Oracle database.

setWidth( ) Method

Format

```
public void setWidth(int the_width)
```

Scope

public

Description

Sets the width of the client-side ORDVideo object.

Parameter

the_width
The width value to be set.

Returns

None.

Exceptions

None.

Usage

```
OrdVideo videoObj = new OrdVideo(connection);
.
.
.
videoObj.setWidth(300);
```

where:

- 300: is the video width in pixels.

setHeight( ) Method

Format

```
public void setHeight(int the_height)
```

Scope

public

Description

Sets the height of the client-side ORDVideo object.

Parameter

the_height
The height value to be set.

Returns

None.

Exceptions

None.

Usage

```
OrdVideo videoObj = new OrdVideo(connection);
.
.
.
videoObj.setHeight(300);
```

where:

- 300: is the video height in pixels.

getFrameSize( ) Method

Format

```
public FrameDimension getFrameSize(byte[ ] ctx)
throws SQLException
```

Scope

public

Description

Gets the frame width and height of the server-side ORDVideo object.

This method calls the server-side method getFrameSize(ctx RAW), which extracts the frame width and height information from the BLOB data.

Parameter

ctx
The format plug-in context information.

Returns

This method returns the frame size. For more information on the FrameDimension class, see "FrameDimension Object Type" and Section 7.3.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     Frame Dimension frameSpecs = videoObj.getFrameSize(ctx);
}
catch(SQLException e){
```

```
      .
      .
      .
}
```

where:

- ctx: contains the context information of the caller.

getWidth( ) Method

Format

```
public int getWidth( )
```

Scope

public

Description

Gets the width of the client-side ORDVideo object from the client cache.

Parameter

None.

Returns

This method returns the width value from the object attribute in the client cache.

Exceptions

None.

Usage

```
OrdVideo videoObj = new OrdVideo(connection);
.
.
.
int i = videoObj.getWidth( );
```

getHeight( ) Method

Format

```
public int getHeight( )
```

Scope

public

Description

Gets the height of the client-side ORDVideo object from the client cache.

Parameter

None.

Returns

This method returns the height value from the object attribute in the client cache.

Exceptions

None.

Usage

```
OrdVideo videoObj = new OrdVideo(connection);
.
.
.
int i = videoObj.getHeight( );
```

setFrameResolution( ) Method

Format

```
public void setFrameResolution(int the_frameResolution)
```

Scope

public

Description

Sets the frame resolution of the client-side ORDVideo object.

Parameter

the_frameResolution
The frame resolution value to be set.

Returns

None.

Exceptions

None.

Usage

```
OrdVideo videoObj = new OrdVideo(connection);
.
.
.
videoObj.setFrameResolution(4);
```

where:

- 4: is the number of pixels per inch of frames in the video data.

getFrameResolution(byte[ ]) Method

Format

```
public int getFrameResolution(byte[ ] ctx)
throws SQLException
```

Scope

public

Description

Gets the frame resolution of the server-side ORDVideo object.

This method calls the server-side method getFrameResolution(ctx RAW), which extracts the frame resolution from the BLOB data.

Parameter

ctx
The format plug-in context information.

Returns

This method returns the frame resolution.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     int i = videoObj.getFrameResolution(ctx);
}
catch(SQLException e){
     .
```

```
        .
        .
}
```

where:

- ctx: contains the context information of the caller.

getFrameResolution( ) Method

Format

```
public int getFrameResolution( )
```

Scope

public

Description

Gets the frame resolution of the client-side ORDVideo object from the client cache.

Parameter

None.

Returns

This method returns the frame resolution from the client cache.

Exceptions

None.

Usage

```
OrdVideo videoObj = new OrdVideo(connection);
.
.
.
int i = videoObj.getFrameResolution( );
```

setFrameRate( ) Method

Format

```
public void setFrameRate(int the_frameRate)
```

Scope

public

Description

Sets the frame rate of the client-side ORDVideo object.

Parameter

the_frameRate
The frame rate value to be set.

Returns

None.

Exceptions

None.

Usage

```
OrdVideo videoObj = new OrdVideo(connection);
.
.
.
videoObj.setFrameRate(5);
```

where:

- 5: is the number of frames displayed per second.

getFrameRate(byte[ ]) Method

Format

```
public int getFrameRate(byte[ ] ctx)
throws SQLException
```

Scope

public

Description

Gets the frame rate of the server-side ORDVideo object.

This method calls the server-side method getFrameRate(ctx RAW), which extracts the frame rate from the BLOB data.

Parameter

ctx
The format plug-in context information.

Returns

This method returns the frame rate.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     int i = videoObj.getFrameRate(ctx);
}
catch(SQLException e){
     .
```

```
        .
        .
}
```

where:

- ctx: contains the context information of the caller.

getFrameRate( ) Method

Format

```
public int getFrameRate( )
```

Scope

public

Description

Gets the frame rate of the client-side ORDVideo object from the client cache.

Parameter

None.

Returns

This method returns the frame rate from the client cache.

Exceptions

None.

Usage

```
OrdVideo videoObj = new OrdVideo(connection);
.
.
.
int i = videoObj.getFrameRate( );
```

setVideoDuration( ) Method

Format

```
public void setVideoDuration(int the_videoDuration)
```

Scope

public

Description

Sets the video duration of the client-side ORDVideo object.

Parameter

the_videoDuration
The value of the video duration to be set.

Returns

None.

Exceptions

None.

Usage

```
OrdVideo videoObj = new OrdVideo(connection);
.
.
.
videoObj.setVideoDuration(500);
```

where:

- 500: is the video duration in seconds.

getVideoDuration(byte[ ]) Method

Format

```
public int getVideoDuration(byte[ ] ctx)
throws SQLException
```

Scope

public

Description

Gets the video duration of the server-side ORDVideo object.

This method calls the server-side method getVideoDuration(ctx RAW), which extracts the video duration from the BLOB data.

Parameter

ctx
The format plug-in context information.

Returns

This method returns the video duration.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     int i = videoObj.getVideoDuration(ctx);
}
catch (SQLException e){
     .
```

```
        .
        .
}
```

where:

- ctx: contains the context information of the caller.

getVideoDuration( ) Method

Format

```
public int getVideoDuration
```

Scope

public

Description

Gets the video duration of the client-side ORDVideo object from the client cache.

Parameter

None.

Returns

This method returns the video duration from the client cache.

Exceptions

None.

Usage

```
OrdVideo videoObj = new OrdVideo(connection);
.
.
.
int i = videoObj.getVideoDuration( );
```

setNumberOfFrames( ) Method

Format

```
public void setNumberOfFrames(int the_numberOfFrames)
```

Scope

public

Description

Sets the number of frames in the client-side ORDVideo object.

Parameter

the_numberOfFrames
The value of the number of frames to be set.

Returns

None.

Exceptions

None.

Usage

```
OrdVideo videoObj = new OrdVideo(connection);
.
.
.
videoObj.setNumberOfFrames(2000);
```

where:

- 2000: is the number of frames in the video.

getNumberOfFrames(byte[ ]) Method

Format

```
public int getNumberOfFrames(byte[ ] ctx)
throws SQLException
```

Scope

public

Description

Gets the number of frames in the server-side ORDVideo object.

This method calls the server-side method getNumberOfFrames(ctx RAW), which extracts the number of frames from the BLOB data.

Parameter

ctx
The format plug-in context information.

Returns

This method returns the number of frames.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     int i = videoObj.getNumberOfFrames(ctx);
}
catch(SQLException e){
     .
```

```
        .
        .
}
```

where:

- ctx: contains the context information of the caller.

getNumberOfFrames( ) Method

Format

```
public int getNumberOfFrames( )
```

Scope

public

Description

Gets the number of frames in the client-side ORDVideo object from the client cache.

Parameter

None.

Returns

This method returns the number of frames from the client cache.

Exceptions

None.

Usage

```
OrdVideo videoObj = new OrdVideo(connection);
.
.
.
int i = videoObj.getNumberOfFrames( );
```

setCompressionType( ) Method

Format

```
public void setCompressionType(String the_compresionType)
```

Scope

public

Description

Sets the compression type of the client-side ORDVideo object.

Parameter

the_compressionType
The compression type value to be set.

Returns

None.

Exceptions

None.

Usage

```
OrdVideo videoObj = new OrdVideo(connection);
.
.
.
videoObj.setCompressionType("Cinepak");
```

where:

- Cinepak: is the compression type.

getCompressionType(byte[ ]) Method

Format

```
public String getCompressionType(byte[ ] ctx)
throws SQLException
```

Scope

public

Description

Gets the compression type of the server-side ORDVideo object.

This method calls the server-side method getCompressionType(ctx RAW), which extracts the compression type from the BLOB data.

Parameter

ctx
The format plug-in context information.

Returns

This method returns the compression type.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     String compType = videoObj.getCompressionType(ctx);
}
catch(SQLException e){
     .
```

```
        .
        .
}
```

where:

- ctx: contains the context information of the caller.

getCompressionType( ) Method

Format

```
public String getCompressionType( )
```

Scope

public

Description

Gets the compression type of the client-side ORDVideo object from the client cache.

Parameter

None.

Returns

This method returns the compression type from the client cache.

Exceptions

None.

Usage

```
OrdVideo videoObj = new OrdVideo(connection);
.
.
.
String compType = videoObj.getCompressionType( );
```

setNumberOfColors( ) Method

Format

```
public void setNumberOfColors(int the_numberOfColors)
```

Scope

public

Description

Sets the number of colors in the client-side ORDVideo object.

Parameter

the_numberOfColors
The value of the number of colors to be set.

Returns

None.

Exceptions

None.

Usage

```
OrdVideo videoObj = new OrdVideo(connection);
.
.
.
videoObj.setNumberOfColors(4);
```

where:

- 4: is the number of colors.

getNumberOfColors(byte[ ]) Method

Format

```
public int getNumberOfColors(byte[ ] ctx)
throws SQLException
```

Scope

public

Description

Gets the number of colors in the server-side ORDVideo object.

This method calls the server-side method getNumberOfColors(ctx RAW), which extracts the number of colors from the BLOB data.

Parameter

ctx
The format plug-in context information.

Returns

This method returns the number of colors.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     int i = videoObj.getNumberOfColors(ctx);
}
catch(SQLException e){
     .
```

```
        .
        .
}
```

where:

- ctx: contains the context information of the caller.

getNumberOfColors( ) Method

Format

```
public int getNumberOfColors( )
```

Scope

public

Description

Gets the number of colors in the client-side ORDVideo object from the client cache.

Parameter

None.

Returns

This method returns the number of colors from the client cache.

Exceptions

None.

Usage

```
OrdVideo videoObj = new OrdVideo(connection);
.
.
.
int i = videoObj.getNumberOfColors( );
```

setBitRate( ) Method

Format

```
public void setBitRate(int the_bitRate)
```

Scope

public

Description

Sets the bit rate in the client-side ORDVideo object.

Parameter

the_bitRate
The bit rate value to be set.

Returns

None.

Exceptions

None.

Usage

```
OrdVideo videoObj = new OrdVideo(connection);
.
.
.
videoObj.setBitRate(1500);
```

where:

- 1500: is the video bit rate.

getBitRate(byte[ ]) Method

Format

```
public int getBitRate(byte[ ] ctx)
throws SQLException
```

Scope

public

Description

Gets the bit rate in the server-side ORDVideo object.

This method calls the server-side method getBitRate(ctx RAW), which extracts the bit rate from the BLOB data.

Parameter

ctx
The format plug-in context information.

Returns

This method returns the bit rate.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     int i = videoObj.getBitRate(ctx);
}
catch(SQLException e){
     .
```

```
        .
        .
}
```

where:

- ctx: contains the context information of the caller.

getBitRate( ) Method

Format

```
public int getBitRate( )
```

Scope

public

Description

Gets the bit rate in the client-side ORDVideo object from the client cache.

Parameter

None.

Returns

This method returns the bit rate from the client cache.

Exceptions

None.

Usage

```
OrdVideo videoObj = new OrdVideo(connection);
.
.
.
int i = videoObj.getBitRate( );
```

getFormat( ) Method

Format

```
public String getFormat(byte[ ] ctx)
throws SQLException
```

Scope

public

Description

Gets the format of the server-side ORDVideo object.

This method calls the server-side method getFormat(ctx RAW), which extracts the format from the BLOB data.

Parameter

ctx
The format plug-in context information.

Returns

This method returns the format.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     String vidStr = videoObj.getFormat(ctx);
}
catch(SQLException e){
     .
```

```
      .
      .
}
```

where:

- ctx: contains the context information of the caller.

setProperties( ) Method

Format

```
public void setProperties(byte[ ] ctx)
throws SQLException
```

Scope

public

Description

Sets the properties of the specified video data in both the server-side and client-side ORDVideo objects.

The video attributes to be set include the following: format, frame size, frame resolution, frame rate, video duration, number of frames, compression type, number of colors, and bit rate.

Parameter

ctx
The format plug-in context information.

Returns

None.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     videoObj.setProperties(ctx);
}
```

```
catch(SQLException e){
     .
     .
     .
}
```

where:

- ctx: contains the context information of the caller.

checkProperties( ) Method

Format

```
public boolean checkProperties(byte[ ] ctx)
throws SQLException
```

Scope

public

Description

Checks that the properties stored in the server-side ORDVideo object are consistent with the properties stored in the server-side raw media data.

Parameter

ctx
The format plug-in context information.

Returns

This method returns *true* if the properties stored in the object cache are the same as the properties stored in the BLOB data; *false* otherwise.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     if(videoObj.checkProperties(ctx)){
          system.out.println("Properties are consistent.");
     }
}
catch(SQLException e){
```

```
      .
      .
      .
}
```

where:

- ctx: contains the context information of the caller.

setKnownAttributes( ) Method

Format

```
public void setKnownAttributes(String knownFormat, int knownWidth,
                               int knownHeight, int knownFrameResolution,
                               int knownFrameRate, int knownVideoDuration,
                               int knownNumberOfFrames,
                               String knownCompressionType,
                               int knownNumberOfColors, int knownBitRate)
```

Scope

public

Description

Sets the known attributes of the client-side ORDVideo object in the client cache.

Parameters

knownFormat
The video data format.

knownWidth
The width of the frame.

knownHeight
The height of the frame.

knownFrameResolution
The frame resolution.

knownFrameRate
The frame rate.

knownVideoDuration
The video duration.

knownNumberOfFrames
The number of frames.

knownCompressionType
The compression type.

knownNumberOfColors
The number of colors.

knownBitRate
The bit rate.

Returns

None.

Exceptions

None.

Usage

```
OrdVideo videoObj = new OrdVideo( );
videoObj.setKnownAttributes("MOOV", 1, 2, 4, 5, 500, 1000, "Cinepak", 256,
1500);
```

where:

- MOOV: is the data format.
- 1: is the video frame width.
- 2: is the video frame height.
- 4: is the frame resolution.
- 5: is the frame rate.
- 500: is the video duration in seconds.
- 1000: is the number of frames in the video.
- Cinepak: is the compression type.
- 256: is the number of colors.
- 1500: is the bit rate.

getAttribute( ) Method

Format

```
public String getAttribute(byte[ ] ctx, String name)
throws SQLException
```

Scope

public

Description

Gets an attribute value of the server-side ORDVideo object.

Parameters

ctx
The format plug-in context information.

name
The name of the attribute.

Returns

This method returns the attribute value.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     String vidStr = videoObj.getAttribute(ctx, "encoding");
}
catch(SQLException e){
     .
```

```
      .
      .
}
```

where:

- ctx: contains the context information of the caller.
- encoding: is the name of the attribute.

getAllAttributes( ) Method

Format

```
public CLOB getAllAttributes(byte[ ] ctx)
throws SQLException
```

Scope

public

Description

Gets the values of all the attributes of the server-side ORDVideo object and returns them as a CLOB.

Parameter

ctx
The format plug-in context information.

Returns

This method returns all the attribute values as a CLOB.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     CLOB clob = videoObj.getAllAttributes(ctx);
}
catch(SQLException e){
     .
     .
     .
```

}

where:

- ctx: contains the context information of the caller.

getAllAttributesAsString( ) Method

Format

```
public String getAllAttributesAsString(byte[ ] ctx)
throws SQLException
```

Scope

public

Description

Gets the values of all the attributes of the server-side ORDVideo object and returns them as a String.

Parameter

ctx
The format plug-in context information.

Returns

This method returns all the attribute values, as a String.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     String vidStr = videoObj.getAllAttributesAsString(ctx);
}
catch(SQLException e){
     .
     .
     .
```

```
}
```

where:

- ctx: contains the context information of the caller.

7.2.2 ORDVideo Methods Associated with the description Attribute

This section presents reference information on the ORDVideo methods associated with the description attribute.

See Section 7.2.1 for additional code required to run the example code.

setDescription( ) Method

Format

```
public void setDescription(String the_description)
throws SQLException
```

Scope

public

Description

Sets the description attribute of the client-side ORDVideo object in the client cache.

Parameter

the_description
The description.

Returns

None.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     .
     .
     .
     videoObj.setDescription("video1");
}
catch(SQLException e){
     .
     .
     .
}
```

where:

- video1: is the description.

getDescription( ) Method

Format

```
public String getDescription( )
throws SQLException
```

Scope

public

Description

Gets the description attribute with extension of the client-side ORDVideo object from the client cache.

Parameter

None.

Returns

This method returns the description.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     .
     .
     .
     String desc = videoObj.getDescription( );
}
catch(SQLException e){
     .
     .
     .
}
```

7.2.3 ORDVideo Methods Associated with the comments Attribute

This section presents reference information on the ORDVideo methods associated with the comments attribute.

See Section 7.2.1 for additional code required to run the example code.

setComments( ) Method

Format

```
public void setComments(CLOB the_comments)
throws SQLException
```

Scope

public

Description

Sets the comments attribute in the client cache and on the server side with a CLOB.

Parameter

the_comments
The comments to be set.

Returns

None.

Exceptions

java.sql.SQLException.

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     OrdVideo videoObj2 = new OrdVideo(connection);
     .
     .
     .
     videoObj.setComments(videoObj2.comments);
}
catch(SQLException e){
     .
     .
     .
}
```

where:

- videoObj2.comments: contains the comments to be set.

getComments( ) Method

Format

```
public CLOB getComments( )
throws SQLException
```

Scope

public

Description

Copies the CLOB locator from the server-side ORDVideo object to the client-side ORDVideo object and returns the LOB locator from the client cache.

Parameter

None.

Returns

This method returns the comments attribute from the client cache.

Exceptions

java.sql.SQLException.

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     .
     .
     .
     CLOB clob = videoObj.getComments( );
}
     catch(SQLException e){
     .
     .
     .
}
```

getCommentsAsString( ) Method

Format

```
public String getCommentsAsString( )
throws SQLException, OutOfMemoryError
```

Scope

public

Description

Returns the server-side ORDVideo object comments as a String.

Parameter

None.

Returns

This method returns the comments attribute as a String.

Exceptions

java.sql.SQLException

java.lang.OutOfMemoryError

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     .
     .
     .
     String comments = audioObj.getCommentsAsString( );
}
catch(SQLException e){
     .
     .
     .
}
```

appendToComments( ) Method

Format

```
public void appendToComments(int amount, String buffer)
throws SQLException
```

Scope

public

Description

Appends data to the server-side ORDVideo object comments attribute.

Parameters

amount
The amount of data to be appended, in bytes.

buffer
The content to be appended from the buffer.

Returns

None.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     .
     .
     .
     videoObj.appendToComments(5, "Drama");
}
catch(SQLException e){
     .
     .
```

```
        .
}
```

where:

- 5: is the amount of data to be appended, in bytes.
- Drama: is the data that will be appended.

writeToComments( ) Method

Format

```
public void writeToComments(int offset, int amount, String buffer)
throws SQLException
```

Scope

public

Description

Writes data to the server-side ORDVideo object comments attribute.

Parameters

offset
The offset from the beginning from which the method will start writing.

amount
The amount of data to be written, in bytes.

buffer
The content to be written from the buffer.

Returns

None.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     .
     .
     .
     videoObj.writeToComments(0, 5, "Drama");
}
```

```
catch(SQLException e){
      .
      .
      .
}
```

where:

- 0: is the offset from the beginning from which the method will start writing.
- 5: is the amount of data to be appended, in bytes.
- Drama: is the content to be written from the buffer.

readFromComments( ) Method

Format

```
public String readFromComments(int offset, int amount)
throws SQLException
```

Scope

public

Description

Reads data from the server-side ORDVideo object comments attribute.

Parameters

offset
The offset from the beginning from which the method will start reading.

amount
The amount of data to be read, in bytes.

Returns

This method returns the String that is read.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     .
     .
     .
     String videoStr = videoObj.readFromComments(0, 5);
}
catch(SQLException e){
     .
     .
```

```
        .
}
```

where:

- 0: is the offset from the beginning from which the method will start writing.
- 5: is the amount of data to be read, in bytes.

locateInComment( ) Method

Format

```
public int locateInComment(String pattern, int offset, int occurrence)
throws SQLException
```

Scope

public

Description

Locates a pattern in the server-side ORDVideo object that starts from a certain offset and appears a certain number of times in the comments attribute.

Parameters

pattern
The pattern to be matched.

offset
The offset from the beginning from which the method will start searching.

occurrence
The number of times that the pattern should appear in the comments.

Returns

This method returns *true* if the pattern occurs in the comments for the specified number of times; *false* otherwise.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     .
     .
     .
```

```
     int i = videoObj.locateInComment("Drama", 0, 10);
}
catch(SQLException e){
     .
     .
     .
}
```

where:

- Drama: is the pattern to be matched.
- 0: is the offset from the beginning from which the method will start searching.
- 10: is the number of times that the pattern should appear in the comments.

trimComments( ) Method

Format

```
public void trimComments(int newlen)
throws SQLException
```

Scope

public

Description

Trims the server-side ORDVideo object comments to a given length.

Parameter

newlen
The new length of the comments.

Returns

None.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     .
     .
     .
     videoObj.trimComments(100);
}
catch(SQLException e){
     .
     .
     .
}
```

where:

- 100: is the new length of the comments attribute after trimming.

eraseFromComments( ) Method

Format

```
public int eraseFromComments(int amount, int offset)
throws SQLException
```

Scope

public

Description

Erases a given amount of data from the server-side ORDVideo object comments starting from a given position.

Parameters

amount
The amount to be erased, in bytes.

offset
The offset from the beginning from which the method will start erasing.

Returns

The amount of data erased, in bytes.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     .
     .
     .
     videoObj.eraseFromComments(100, 0);
}
catch(SQLException e){
     .
```

```
        .
        .
}
```

where:

- 100: is the number of bytes to be erased.
- 0: is the offset from the beginning from which the method will start erasing.

deleteComments( ) Method

Format

```
public void deleteComments( )
throws SQLException
```

Scope

public

Description

Deletes the server-side ORDVideo object comments.

Parameter

None.

Returns

None.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     .
     .
     .
     videoObj.deleteComments( );
}
catch(SQLException e){
     .
     .
     .
}
```

copyCommentsOut( ) Method

Format

```
public CLOB copyCommentsOut(CLOB dest, int amount, int from_loc,
                            int to_loc)
throws SQLException
```

Scope

public

Description

Copies the comments out from a server-side location to a separate location.

Parameters

dest
The new location to which the data will be copied.

amount
The amount to be copied, in bytes.

from_loc
The position where the copying will begin.

to_loc
The position where the copying will end.

Returns

This method returns the CLOB into which the comments are copied.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     OrdVideo videoObj2 = new OrdVideo(connection);
```

```
      .
      .
      .
      CLOB clob = videoObj.copyCommentsOut(videoObj2.comments, 100, 0, 99);
}
catch(SQLException e){
      .
      .
      .
}
```

where:

- videoObj2.comments: is the CLOB into which the comments will be copied.
- 100: is the amount of data to be copied, in bytes.
- 0: is the position where the method begins copying.
- 99: is the position where the method ends copying.

compareComments( ) Method

Format

```
public int compareComments(CLOB dest, int amount, int start_in_comment,
                           int start_in_compare_comment)
throws SQLException
```

Scope

public

Description

Compares the comments of a server-side ORDVideo object with the comments of a separate CLOB.

Parameters

dest
The location of the comments data that will be compared with the comments data of the current object.

amount
The amount of data to be compared, in bytes.

start_in_comment
The starting position in the current object.

start_in_compare_comment
The starting position in the destination object.

Returns

This method returns 1 if there is a positive match; -1 otherwise.

Exceptions

java.sql.SQLException

Usage

```
try{
```

```
     OrdVideo videoObj = new OrdVideo(connection);
     OrdVideo videoObj2 = new OrdVideo(connection);
     .
     .
     .
     int i = videoObj.compareComments(videoObj2.comments, 100, 0, 50);
}
catch(SQLException e){
     .
     .
     .
}
```

where:

- videoObj2.comments: is the location of the CLOB that will be compared to the comments of the current object.
- 100: is the amount of data to be compared, in bytes.
- 0: is the initial position in the current object.
- 50: is the initial position in the separate object.

loadCommentsFromFile( ) Method

Format

```
public void loadCommentsFromFile(String loc, String fileName,
                                 int amount, int from_loc, int to_loc)
throws SQLException
```

Scope

public

Description

Loads a given amount of comments data from the server-side ORDVideo object from a separate server-side file.

Parameters

loc
The location of the file from which the comments will be loaded.

fileName
The name of the file from which the comments will be loaded.

amount
The amount of comments data to be loaded, in bytes.

from_loc
The start location in the comments attribute into which the comments will be loaded.

to_loc
The end location in the comments attribute into which the comments will be loaded.

Returns

None.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     .
     .
     .
     videoObj.loadCommentsFromFile("videodir", "jdoe.mov", 100, 0, 99);
}
catch(SQLException e){
     .
     .
     .
}
```

where:

- videodir: is the location of the file from which the comments will be loaded.
- jdoe.mov: is the name of the file from which the comments will be loaded.
- 100: is the amount of data to be loaded, in bytes.
- 0: is the start location in the comments attribute into which the comments will be loaded.
- 99: is the end location in the comments attribute into which the comments will be loaded.

loadComments( ) Method

Format

```
public boolean loadComments(String fileName)
throws SQLException, IOException, SecurityException
```

Scope

public

Description

Loads comments to the server-side ORDVideo object from a client-side file.

Parameter

fileName
The name of the file from which the comments will be loaded.

Returns

This method returns *true* upon successful loading; *false* otherwise.

Exceptions

java.sql.SQLException

java.io.IOException

java.lang.SecurityException

Usage

```
try{
     OrdAudio audioObj = new OrdAudio(connection);
     .
     .
     .
     audioObj.loadCommentsFromFile(my_file);
}
catch(SQLException e){
     .
     .
```

```
      .
}
catch(IOException io){
      .
      .
      .
}
catch(SecurityException s){
      .
      .
      .
}
```

where:

- my_file: is the name of the file from which the comments will be loaded.

loadCommentsInChunks( ) Method

Format

```
boolean loadCommentsInChunks(String fileName, int fileLength)
throws SQLException, IOException
```

Scope

package

Description

Loads comments to the server-side ORDVideo object from a client-side file in chunks of 32K bytes.

Parameters

fileName
The name of the file from which the comments will be loaded.

fileLength
The total length of the file, in bytes.

Returns

This method returns *true* upon successful loading; *false* otherwise.

Exceptions

java.sql.SQLException

java.io.IOException

Usage

Do not use this method unless you are extending the package.

getCommentLength( ) Method

Format

```
public int getCommentLength( )
throws SQLException
```

Scope

public

Description

Gets the length of the server-side ORDVideo object comments attribute.

Parameter

None.

Returns

This method returns the length of the comments attribute.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     .
     .
     .
     int i = videoObj.getCommentLength( );
}
catch(SQLException e){
     .
     .
     .
}
```

7.2.4 ORDVideo Methods Associated with Communication Between the Client and Server

This section presents reference information on the ORDVideo methods associated with communication between the client and server.

See Section 7.2.1 for additional code required to run the example code.

refresh( ) Method

Format

```
public void refresh(boolean forUpdate)
throws SQLException
```

Scope

public

Description

Copies the server-side ORDVideo object attribute values to the client-side ORDVideo object with or without locking the database row.

Parameter

forUpdate
The indicator for whether or not the object will be refreshed for update.

Returns

None.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     .
     .
     .
     videoObj.refresh(false);
}
catch(SQLException e){
     .
     .
     .
}
```

where

- false: indicates that the object has not been refreshed for update.

flush( ) Method

Format

```
public void flush( )
throws SQLException
```

Scope

public

Description

Copies the client-side ORDVideo object attribute values to the server-side ORDVideo object.

Parameter

None.

Returns

None.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     .
     .
     .
     videoObj.flush( );
}
catch(SQLException e){
     .
     .
     .
}
```

7.2.5 ORDVideo Methods Associated with the Media Type

This section presents reference information on the ORDVideo methods associated with the media type.

See Section 7.2.1 for additional code required to run the example code.

getMediaType( ) Method

Format

```
String getMediaType( )
```

Scope

package

Description

Returns the media type information.

Parameter

None.

Returns

This method returns the media type, which is OrdVideo.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

7.2.6 ORDVideo Methods Associated with the Content Length

This section presents reference information on the ORDVideo methods associated with the content length.

See Section 7.2.1 for additional code required to run the example code.

getContentLength(byte[ ]) Method

Format

```
public int getContentLength(byte[ ] ctx)
throws SQLException
```

Scope

public

Description

Returns the content length of the server-side ORDVideo object media data.

Parameter

ctx
The source plug-in context information.

Returns

This method returns the length of the data content of the media data, in bytes.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     int i = videoObj.getContentLength(ctx);
}
catch(SQLException e){
     .
     .
     .
}
```

where

- ctx: contains the context information of the caller.

getContentLength( ) Method

Format

```
public int getContentLength( )
throws SQLException
```

Scope

public

Description

Returns the content length of the server-side ORDVideo object media data.

Parameter

None.

Returns

This method returns the content length, in bytes.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     .
     .
     .
     int i = videoObj.getContentLength( );
}
catch{
     .
     .
     .
}
```

7.2.7 ORDVideo Methods Associated with the SQL Type

This section presents reference information on the ORDVideo methods associated with the SQL type.

See Section 7.2.1 for additional code required to run the example code.

getSQLConstructor( ) Method

Format

```
String getSQLConstructor(boolean updateOption, String objName)
```

Scope

package

Description

Gets the SQL type constructor for the ORDVideo object.

This method calls the corresponding getSQLConstructor( ) method in the superclass.

Parameters

updateOption
The indicator for whether or not the option is flagged to be updated.

objName
The variable name of the video object.

Returns

This method returns the SQL string that will bind an object instance variable to a database object instance.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

defineSQLResults( ) Method

Format

```
String defineSQLResults(String var)
```

Scope

package

Description

Defines the SQL results that will be used to assign values to the ORDVideo object attributes.

This method is used by the refresh( ) method to assign the server-side object attribute values to the corresponding client-side object attributes.

Parameter

var
The name of the video object variable.

Returns

This method returns the SQL string that will define the SQL results.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

declareSQLResults( ) Method

Format

```
int declareSQLResults(int start, OracleCallableStatement stmt)
throws SQLException
```

Scope

package

Description

Declares types for the SQL results corresponding to the different ORDVideo object attributes.

This method is used by the refresh( ) method to declare the types of the client-side object attributes needed to assign values to the corresponding server-side object attributes.

Parameters

start
The position in the statement where the binding begins.

stmt
The Oracle callable statement that will be executed on the server side.

Returns

This method returns an integer defined as position + 1, where position is the last index in the statement for which the type was declared.

Exceptions

java.sql.SQLException

Usage

Do not use this method unless you are extending the package.

setSQLParams( ) Method

Format

```
String setSQLParams(String var)
```

Scope

package

Description

Sets the values of the SQL type of the ORDVideo object as parameters.

This method is used by the flush( ) method to set the values of the different attributes of the server-side object to the same values as the attributes of the corresponding client-side object.

Parameter

var
The name of the video object variable.

Returns

This method returns the SQL string that will set the different attributes of the video object.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

bindInSQLParams( ) Method

Format

```
int bindInSQLParams(int start, OracleCallableStatement stmt)
throws SQLException
```

Scope

package

Description

Binds the values of the SQL type of the ORDVideo object.

This method is used by the flush( ) method to set the values of the different attributes of the server-side object to the values of the corresponding client-side object attributes.

Parameters

start
The position in the statement where the binding begins.

stmt
The Oracle callable statement that will be executed on the server side.

Returns

This method returns an integer defined as position + 1, where position is the last index in the statement to which a value was bound.

Exceptions

java.sql.SQLException

Usage

Do not use this method unless you are extending the package.

getSQLResults( ) Method

Format

```
int getSQLResults(int start, OracleCallableStatement stmt)
throws SQLException
```

Scope

package

Description

Gets the SQL results in order to assign them to the attributes of the OrdMultiMedia object (for example, encoding or numberOfChannels).

This method is used by the refresh( ) method to assign the server-side object attribute values to the corresponding client-side object attributes.

Parameters

start
The initial position for getting results from the statement.

stmt
The Oracle callable statement that has been executed on the server side.

Returns

This method returns an integer defined as position + 1, where position is the last index in the statement to which a value was bound.

Exceptions

java.sql.SQLException

Usage

Do not use this method unless you are extending the package.

7.2.8 ORDVideo Methods Associated with Generating SQL Queries

This section presents reference information on the ORDVideo methods associated with generating SQL queries.

See Section 7.2.1 for additional code required to run the example code.

getFormatStr( ) Method

Format

```
String getFormatStr( )
```

Scope

package

Description

Returns the format String, which is used by other methods to generate SQL queries.

Parameter

None.

Returns

This method returns the format String.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

getUpdStr( ) Method

Format

```
String getUpdStr( )
```

Scope

package

Description

Returns the SQL string that is used to update an ORDVideo instance, which is used by other methods to generate SQL queries.

Parameter

None.

Returns

This method returns the SQL string that is used to update an ORDVideo instance.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

getSourceStr( ) Method

Format

```
String getSourceStr( )
```

Scope

package

Description

Returns the source String, which is used by other methods to generate SQL queries.

Parameter

None.

Returns

This method returns the source String.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

getContentLengthAPI( ) Method

Format

```
String getContentLengthAPI( )
```

Scope

package

Description

Returns the server-side contentLength API for this class, which is used by other methods to generate SQL queries.

Parameter

None.

Returns

This method returns the server-side contentLength API for this class.

Exceptions

None.

Usage

Do not use this method unless you are extending the package.

7.2.9 ORDVideo Methods Associated with File Operations

This section presents reference information on the ORDVideo methods associated with file operations.

See Section 7.2.1 for additional code required to run the example code.

openSource( ) Method

Format

```
public int openSource(byte[ ] userArg, byte[ ] ctx)
throws SQLException
```

Scope

public

Description

Opens the server-side ORDVideo object file source.

Parameters

userArg
Permission-related parameters that are supplied by the user, such as READONLY.

ctx
The source plug-in context information.

Returns

This method returns 0 in case of success or an integer greater than 0 in case of failure.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     .
     .
     .
     int i = videoObj.openSource(READONLY, ctx);
}
catch(SQLException e){
     .
```

```
        .
        .
}
```

where:

- READONLY: sets the permission.
- ctx: contains the context information of the caller.

closeSource( ) Method

Format

```
public int closeSource(byte[ ] ctx)
throws SQLException
```

Scope

public

Description

Closes the server-side ORDVideo object file source.

Parameter

ctx
The source plug-in context information.

Returns

This method returns 0 in case of success or an integer greater than 0 in case of failure.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     int i = videoObj.closeSource(ctx);
}
catch(SQLException e){
     .
     .
     .
```

```
}
```

where:

- ctx: contains the context information of the caller.

trimSource( ) Method

Format

```
public int trimSource(byte[ ] ctx, int newLen)
throws SQLException
```

Scope

public

Description

Trims the server-side ORDVideo object file source to the given length.

Parameters

ctx
The source plug-in context information.

newLen
The length to which the source will be trimmed.

Returns

This method returns 0 in case of success or an integer greater than 0 in case of failure.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     byte[ ] ctx = new byte[4000];
     .
     .
     .
     int i = videoObj.trimSource(ctx, 100);
}
catch(SQLException e){
```

```
        .
        .
        .
}
```

where:

- ctx: contains the context information of the caller.
- 100: is the length, in bytes, to which the file source will be trimmed.

readFromSource( ) Method

Format

```
public int readFromSource(byte[ ] ctx, int startPos, int numBytes,
                          byte[ ] buffer)
throws SQLException
```

Scope

public

Description

Reads a buffer of a given number of bytes from a server-side ORDVideo object source, beginning at a given initial position.

Parameters

ctx
The source plug-in context information.

startPos
The position where the method begins reading.

numBytes
The amount that will be read, in bytes.

buffer
The buffer into which the data will be read.

Returns

This method returns the amount of data read, in bytes.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
```

```
     byte[ ] ctx = new byte[4000];
     byte[ ] my_buffer;
     .
     .
     .
     videoObj.readFromSource(ctx, 0, 100, my_buffer);
}
catch(SQLException e){
     .
     .
     .
}
```

where:

- ctx: contains the context information of the caller.
- 0: is the position where the method begins reading.
- 100: is the amount of data to be read, in bytes.
- my_buffer: is the buffer into which the data will be read.

writeToSource( ) Method

Format

```
public int writeToSource(byte[ ] ctx, int startPos, int numBytes,
                         byte[ ] buffer)
throws SQLException
```

Scope

public

Description

Writes a buffer of a given number of bytes to the server-side ORDVideo object source, beginning at a given initial position.

Parameters

ctx
The source plug-in context information.

startPos
The initial position where the written data will be inserted.

numBytes
The number of bytes to be written.

buffer
The content to be written.

Returns

This method returns the amount of data written, in bytes.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
```

```
      byte[ ] ctx = new byte[4000];
      byte[ ] my_buffer;
      .
      .
      .
      videoObj.writeToSource(ctx, 0, 100, my_buffer);
}
catch(SQLException e){
      .
      .
      .
}
```

where:

- ctx: contains the context information of the caller.
- 0: is the initial position for writing the new data.
- 100: is the number of bytes to be written.
- my_buffer: contains the content to be written.

7.2.10 ORDVideo Methods Associated with Source Content Operations

This section presents reference information on the ORDVideo methods associated with source content operations.

See Section 7.2.1 for additional code required to run the example code.

getContentInLob( ) Method

Format

```
public BLOB getContentInLob(byte[ ] ctx, StringBuffer mimeType,
                            StringBuffer format)
throws SQLException
```

Scope

public

Description

Gets the contents of the server-side ORDVideo object localData attribute and stores them in a temporary BLOB.

Parameters

ctx
The source plug-in context information.

mimeType
The MIME type of the temporary BLOB.

format
The format of the temporary BLOB.

Returns

This method returns the BLOB into which the contents have been copied.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     byte[ ] ctx = new byte[4000];
     String mimeType = "video/mpi";
     String format = "mpi";
```

```
      .
      .
      .
      BLOB blob = videoObj.getContentInLob(ctx, mimeType, format);
}
catch(SQLException e){
      .
      .
      .
}
```

where:

- ctx: contains the context information of the caller.
- mimeType: contains the MIME type of the temporary BLOB.
- format: contains the format of the temporary BLOB.

7.2.11 ORDVideo Methods Associated with Processing Video Data

This section presents reference information on the ORDVideo methods associated with processing video data.

See Section 7.2.1 for additional code required to run the example code.

processSourceCommand( ) Method

Format

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

Scope

public

Description

Calls the server-side processSourceCommand( ) method.

For more information on the commands that can be processed, see *Oracle8i interMedia Audio, Image, and Video User's Guide and Reference*.

Parameters

ctx
The source plug-in context information.

cmd
The command that is to be executed.

args
The arguments for the command that is to be executed.

result
The result of the command that is to be executed.

Returns

This method returns the results of the command execution.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     byte[ ] ctx = new byte[4000];
     String cmd = "compress";
     String arg_list = "compression_type";
     .
     .
     .
     byte[ ] result = videoObj.processSourceCommand(ctx, cmd, arg_list,
          the_result);
}
catch(SQLException e){
     .
     .
     .
}
```

where:

- ctx: contains the context information of the caller.
- cmd: contains the command to be executed.
- arg_list: contains the list of arguments.
- the_result: is the result of the command execution.

processVideoCommand( ) Method

Format

```
public byte[ ] processVideoCommand(byte[ ] ctx, String cmd, String args,
                                   byte[ ] result)
throws SQLException
```

Scope

public

Description

Calls the server-side processVideoCommand( ) method.

For more information on the commands that can be processed, see *Oracle8i interMedia Audio, Image, and Video User's Guide and Reference*.

Parameters

ctx
The format plug-in context information.

cmd
The command that is to be executed.

args
The arguments for the command that is to be executed.

result
The result of the command that is to be executed. The memory for this variable is allocated by the client.

Returns

This method returns the results of the execution of the command.

Exceptions

java.sql.SQLException

Usage

```
try{
     OrdVideo videoObj = new OrdVideo(connection);
     byte[ ] ctx = new byte[4000];
     String cmd = "compress";
     String arg_list = "compression_type";
     .
     .
     .
     byte[ ] result = videoObj.processVideoCommand(ctx, cmd, arg_list,
          the_result);
}
catch(SQLException e){
     .
     .
     .
}
```

where:

- ctx: contains the context information of the caller.
- cmd: contains the command that is to be executed.
- arg_list: contains the arguments for the command.
- the_result: is the result of the command execution.

7.3 FrameDimension Methods

This section presents FrameDimension reference information on the methods used for setting and returning video width and height information. These methods are described in the following groupings:

FrameDimension Methods Associated with the width Attribute

- public void setWidth( ): set the width.
- public int getWidth( ): get the width.

FrameDimension Methods Associated with the height Attribute

- public void setHeight( ): set the height.
- public int getHeight( ): get the height.

7.3.1 FrameDimension Methods Associated with the width Attribute

This section presents reference information on the FrameDimension methods associated with the width attribute.

setWidth( ) Method

Format

```
public void setWidth(int the_width)
```

Scope

public

Description

Sets the width in the ORDVideo object.

Parameter

the_width
The width value to be set.

Returns

None.

Exceptions

None.

getWidth( ) Method

Format

```
public int getWidth( )
```

Scope

public

Description

Gets the width in the ORDVideo object.

Parameter

None.

Returns

This method returns the width value.

Exceptions

None.

7.3.2 FrameDimension Methods Associated with the height Attribute

This section presents reference information on the FrameDimension methods associated with the height attribute.

setHeight( ) Method

Format

```
public void setHeight(int the_height)
```

Scope

public

Description

Sets the height in the ORDVideo object.

Parameter

the_height
The height value to be set.

Returns

None.

Exceptions

None.

getHeight( ) Method

Format

```
public int getHeight( )
```

Scope

public

Description

Gets the height in the ORDVideo object.

Parameter

None.

Returns

This method returns the height value.

Exceptions

None.

8

DataSource Reference Information

The Oracle8*i* *inter*Media Audio, Image, and Video Java Client library contains information about the DataSource:

- Object type -- see “Object Types” in Section 8.1.
- Methods -- see Section 8.2.

8.1 Object Types

Oracle8*i* *inter*Media Audio, Image, and Video Java Client describes the DataSource object type, which contains the interaction of the front end with the database server. Through use of Java Database Connectivity (JDBC), this object type facilitates the storage and retrieval of data from the Oracle database.

DataSource Object Type

The DataSource object type contains the interaction of the front end with the database server and facilitates the storage and retrieval of data from the Oracle database.

> **Note:** In the following code, the methods begin with *public*. This indicates that the method can be called by all classes in all packages.

This object type is defines as follows:

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

public class DataSource {
     OrdMultiMedia mediaObject;

     public DataSource (OrdMultiMedia mObject)
     { }

     public boolean initCheck( )
     { }

     public void setMediaLocator(String tableName, String columnName,
          String condition)
     throws NullPointerException
     { }

     public void setConnection(Connection the_connection)
     throws NullPointerException
     { }

     public String getContentType( )
     throws SQLException, NullPointerException
     { }

     public int getContentLength( )
     throws SQLException, NullPointerException
```

```
    { }

    public void getDataInFile(String fileName)
    throws SQLException, FileNotFoundException, IOException,
         NullPointerException
    { }

    public byte[ ] getData( )
    throws SQLException, NullPointerException, OutOfMemoryError
    { }

    public InputStream getDataInStream( )
    throws SQLException, NullPointerException
    { }

    public String getSourceInformation( )
    throws NullPointerException
    { }

    public void setSourceInformation(String srcType, String srcLocation,
         String srcName)
    throws NullPointerException
    { }

    public boolean loadData(String fileName)
    throws SQLException, FileNotFoundException, IOException,
         NullPointerException
    { }
}
```

where the class attribute is defined as:

- mediaObject: is an object of the ORDMultiMedia class.

8.2 Methods

This section presents reference information on the DataSource object type methods. These methods are as follows:

DataSource Methods

- public boolean initCheck( ): check whether or not the data source is initialized.
- public void setMediaLocator( ): set the media locator parameters of the data source.
- public void setConnection( ): set the connection to the data source.
- public String getContentType( ): get the content type of the data in the data source (that is, the MIME type of the media data).
- public int getContentLength( ): get the content length of the server-side media data in the data source.
- public void getDataInFile( ): place the server-side media data into the file and return the file name to be passed into the media player.
- public byte[ ] getData( ): get the server-side media data in a byte array.
- public InputStream getDataInStream( ): return the server-side media data as an InputStream object.
- public String getSourceInformation( ): return the corresponding data source type information for the media data in the format protocol://location/name.
- public void setSourceInformation( ): set the source information attribute in the ORDSource object of the media object, given the source type, location, and name.
- public boolean loadData( ): load media data from a given file into a server-side media object designated by the corresponding media locator parameters.

8.2.1 DataSource Methods

This section presents reference information on the DataSource methods.

The methods described in this reference chapter show examples based on the instantiation of a DataSource object. For the examples in the following section, assume that file includes the following Java code:

```
import oracle.ord.media.*;

public class clientProgram{
     public static void main(String[ ] args){
           The code in the examples appears here
     }
}
```

In each example, the parameter audObj is a previously created ORDAudio object.

initCheck( ) Method

Format

```
public boolean initCheck( )
```

Scope

public

Description

Checks whether or not the DataSource object is initialized.

Parameter

None.

Returns

This method returns *true* if the object is initialized; *false* otherwise.

Exceptions

None.

Usage

```
DataSource dataSource = new DataSource(audioObj);
.
.
.
if(dataSource.initCheck( )){
     system,out.println("The object is initialized.");
}
```

setMediaLocator( ) Method

Format

```
public void setMediaLocator(String tableName, String columnName,
                            String condition)
throws NullPointerException
```

Scope

public

Description

Sets the media locator parameters of the data source.

Parameters

tableName
The name of the table to which the data source will be bound.

columnName
The name of the column to which the data source will be bound.

condition
The select condition on the table, used to identify the row to which the data source will be bound.

Returns

None.

Exceptions

java.lang.NullPointerException

Usage

```
try{
     DataSource dataSource = new DataSource(audioObj);
     .
     .
     .
```

```
     dataSource.setMediaLocator("audioTable", "song", "songID = 3");
}
catch(NullPointerException np){
     .
     .
     .
}
```

where:

- audioTable: is the table to which the object will be bound.
- song: is the column to which the object will be bound.
- songID = 3: is the selection condition on the table, which identifies the row to which the object will be bound.

setConnection( ) Method

Format

```
public void setConnection(Connection the_connection)
throws NullPointerException
```

Scope

public

Description

Sets the connection to the data source.

Parameter

the_connection
The connection parameter.

Returns

None.

Exceptions

java.lang.NullPointerException

Usage

```
try{
     DataSource dataSource = new DataSource(audioObj);
     .
     .
     .
     dataSource.setConnection(connection);
}
catch(NullPointerException np){
     .
     .
     .
}
```

where:

- connection: is a connection to the database.

getContentType( ) Method

Format

```
public String getContentType( )
throws SQLException, NullPointerException
```

Scope

public

Description

Gets the content type of the data in the data source (that is, the MIME type of the media data).

Parameter

None.

Returns

This method returns the MIME type or content type of the data in the data source.

Exceptions

java.sql.SQLException

java.lang.NullPointerException

Usage

```
try{
     DataSource dataSource = new DataSource(audioObj);
     .
     .
     .
     String dataStr = dataSource.getContentType( );
}
catch(SQLException e){
     .
     .
     .
}
```

```
catch(NullPointerException np){
     .
     .
     .
}
```

getContentLength( ) Method

Format

```
public int getContentLength( )
throws SQLException, NullPointerException
```

Scope

public

Description

Gets the content length of the server-side media data in the data source.

Parameter

None.

Returns

This method returns the content length of the data in the data source.

Exceptions

java.sql.SQLException

java.lanf.NullPointerException

Usage

```
try{
     DataSource dataSource = new DataSource(audioObj);
     .
     .
     .
     int i = dataSource.getContentLength( );
}
catch(SQLException e){
     .
     .
     .
}
```

```
catch(NullPointerException np){
     .
     .
     .
}
```

getDataInFile( ) Method

Format

```
public void getDataInFile(String fileName)
throws SQLException, FileNotFoundException, IOException, NullPointerException
```

Scope

public

Description

Places the server-side media data into the file and returns the file name to be passed into the media player.

The file name passed should be without extension; the extension will be automatically added depending on the format.

Parameter

fileName
The file name where the media data will be stored.

Returns

None.

Exceptions

java.sql.SQLException

java.io.FileNotFoundException

java.io.IOException

java.lanf.NullPointerException

Usage

```
try{
     DataSource dataSource = new DataSource(audioObj);
     .
     .
```

```
      .
      dataSource.getDataInFile("ds.dat");
}
catch(SQLException e){
      .
      .
      .
}
catch(FileNotFoundException fnf){
      .
      .
      .
}
catch(IOException io){
      .
      .
      .
}
catch(NullPointerException np){
      .
      .
      .
}
```

where:

- ds.dat: is the file where the media data will be stored.

getData( ) Method

Format

```
public byte[ ] getData( )
throws SQLException, NullPointerException, OutOfMemoryError
```

Scope

public

Description

Gets the server-side media data as a byte array.

Parameter

None.

Returns

This method returns the media data as a byte array.

Exceptions

java.sql.SQLException

java.lang.NullPointerException

java.lang.OutOfMemoryError

Usage

```
try{
     DataSource dataSource = new DataSource(audioObj);
     .
     .
     .
     byte[ ] mediaData = dataSource.getData( );
}
catch(SQLException e){
     .
     .
     .
}
```

```
catch(NullPointerException np){
     .
     .
     .
}
```

getDataInStream( ) Method

Format

```
public InputStream getDataInStream( )
throws SQLException, NullPointerException
```

Scope

public

Description

Returns the server-side media data as an InputStream object.

Parameter

None.

Returns

This method returns the media data as a byte array.

Exceptions

java.sql.SQLException

java.lang.NullPointerException

Usage

```
try{
     DataSource dataSource = new DataSource(audioObj);
     .
     .
     .
     InputStream mediaStream = dataSource.getDataInStream( );
}
catch(SQLException e){
     .
     .
     .
}
```

```
catch(NullPointerException np){
     .
     .
     .
}
```

getSourceInformation( ) Method

Format

```
public String getSourceInformation( )
throws NullPointerException
```

Scope

public

Description

Returns the corresponding data source type information for the media data in the format protocol://location/name.

Parameter

None.

Returns

This method returns the source information in the format protocol://location/name.

Exceptions

java.lang.NullPointerException

Usage

```
try{
     DataSource dataSource = new DataSource(audioObj);
     .
     .
     .
     String sourceStr = dataSource.getSourceInformation( );
}
catch(nullPointerException np){
     .
     .
     .
}
```

setSourceInformation( ) Method

Format

```
public void setSourceInformation(String srcType, String srcLocation,
                                 String srcName)
throws NullPointerException
```

Scope

public

Description

Sets the source information attribute in the ORDSource object of the media object, given the source type, location, and name.

Parameters

srcType
The type of the source (FILE or HTTP, for example).

srcLocation
The location of the source.

srcName
The name of the source.

Returns

None.

Exceptions

java.lang.NullPointerException

Usage

```
try{
     DataSource dataSource = new DataSource(audioObj);
     .
     .
     .
```

```
     dataSource.setSourceInformation("HTTP", "directory1", "file");
}
catch(NullPointerException np){
     .
     .
     .
}
```

where:

- HTTP: is the type of the source.
- directory1: is the location of the source.
- file: is the name of the source.

loadData( ) Method

Format

```
public boolean loadData(String fileName)
throws SQLException, FileNotFoundException, IOException, NullPointerException
```

Scope

public

Description

Loads media data from a given file into a server-side media object designated by the corresponding media locator parameters.

Parameter

fileName
The name of the file from which to load the data.

Returns

This method returns *true* if loading is successful; *false* otherwise.

Exceptions

java.sql.SQLException

java.io.FileNotFoundException

java.io.IOException

java.lang.NullPointerException

Usage

```
try{
     DataSource dataSource = new DataSource(audioObj);
     .
     .
     .
     if(dataSource.loadData("file")){
          system.out.println("Loading is successful");
```

```
      }
}
catch(SQLException e){
      .
      .
      .
}
catch(FileNotFoundException fnf){
      .
      .
      .
}
catch(IOException io){
      .
      .
      .
}
catch(NullPointerException np){
      .
      .
      .
}
```

where:

- file: is the name of the file from which to load the data.

A

Sample Program Information

A.1 MediaAnnotator

The MediaAnnotator program is not contained on the Oracle8*i* *inter*Media Audio, Image, and Video Java Client CD. Instead, it (along with other free Oracle software) may be found at the following URL:

http://www.oracle.com/products/free_software/

A.2 Java Demonstration

Java demonstration programs has been provided to help you learn to use both the audio and video client-side Java classes so you can build your own applications. In these two programs, the audio or video object is instantiated at the client side and a number of accessor methods are invoked. The audio Java files are located in the *$ORACLE_HOME*/ord/aud/demo directory, and the video Java files are located in the *$ORACLE_HOME*/ord/vid/demo directory. See the README.txt file in each directory for requirements and instructions on how to run the programs.

B

Exceptions and Errors

This appendix contains information on the exceptions and errors that can be raised by the classes that extend the ORDMultiMedia superclass.

B.1 Exception

The Exception class (and its subclasses, including SQLException) indicates conditions of which the application will make the user aware.

```
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 FileNotFoundException

The FileNotFoundException class signals that a file could not be found.

```
public class java.io.FileNotFoundException extends java.io.IOException {
     //Constructs a FileNotFoundException with no detailed message
     public FileNotFoundException();

     //Constructs a FileNotFoundException with the specified detailed message
     public FileNotFoundException(String s);
}
```

B.3 IOException

The IOException class signals that an I/O exception of some sort has occurred.

```
public class java.io.IOException extends java.lang.Exception {
     //Constructs a FileNotFoundException with no detailed message
     public IOException();

     //Constructs a FileNotFoundException with the specified detailed message
     public IOException(String s);
}
```

B.4 NullPointerException

The NullPointerException class signals that an application has attempted to use *null* in a case where an object is required. These cases include:

- Calling an instance method of a null object
- Accessing or modifying a field in a null object
- Taking a length of *null* as if it were an array
- Accessing or modifying the slots of *null* as if it were an array
- Throwing *null* as if it were a Throwable value

Applications should throw instances of this class to indicate other invalid uses of the null object.

```
public class java.lang.NullPointerException extends java.lang.RuntimeException {
     //Constructs a NullPointerException with no detailed message
     public NullPointerException();

     //Constructs a NullPointerException with the specified detailed message
     public NullPointerException(String s);
}
```

B.5 OutOfMemoryError

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 SecurityException

The SecurityException is thrown by an instance of the SecurityManager class to indicate a security violation.

```
public class java.lang.SecurityException extends java.lang.RuntimeException {
    //Constructs a SecurityException with no detailed message
    public SecurityException();

    //Constructs a SecurityException with the specified detailed message
    public SecurityException(String s);
}
```

B.7 SQLException

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);

    //Construct a SQLException with venderCode value of 0
    public SQLException(String reason, String SQLState);

    //Construct a SQLException with vendorCode value of 0 and a null SQLState
    public SQLException(String reason);

    //Construct a SQLException with vendorCode of 0, a null SQLState. and a
    //null reason
    public SQLException();

    //The following four methods are public instance methods

    //Get the vendor-specific exception code
    public int getErrorCode();
```

```
        //Get the exception connected to this one
        public SQLException getNextException();

        //Get the SQL state
        public String getSQLState();

        //Add a SQLException to the end
        public synchronized void setNextException(SQLException ex);
    }
```

C

Reference Information

The following appendix contains additional reference information on the methods that operate on the server-side object, the client-side object, or both.

C.1 Client-Side/Server-Side Chart

The following chart illustrates which ORDAudio, ORDImage, and ORDVideo methods operate on the server-side object, the client-side object, or both.

In the chart, **S** indicates that the method operates on the server-side object, **C** indicates that the method operates on the client-side object, **S, C** indicates that the method operates on both the server-side and client-side object, and **NA** indicates that the method does not apply to the object.

This chart includes methods inherited from the ORDMultiMedia superclass.

Table C–1 Methods That Operate on the Client-Side or Server-Side Object

Method	ORDImage	ORDAudio	ORDVideo
appendToComments( )	NA	S	S
checkProperties( )	S	NA	NA
checkProperties(byte[ ])	NA	S	S
clearLocal( )	C	C	C
closeSource( )	NA	S	S
compareComments( )	NA	S	S
copy( )	S	NA	NA
copyCommentsOut( )	NA	S	S
deleteComments( )	NA	S	S

Table C–1 Methods That Operate on the Client-Side or Server-Side Object(Cont.)

deleteContent( )	S	S	S
eraseFromComments( )	NA	S	S
export( )	S	S	S
getAllAttributes( )	NA	S	S
getAllAttributesAsString( )	C	S	S
getAttribute( )	NA	S	S
getAudioDuration( )	NA	C	NA
getAudioDuration(byte[ ])	NA	S	NA
getBFILE( )	S	S	S
getBitRate( )	NA	NA	C
getBitRate(byte[ ])	NA	NA	S
getCommentLength( )	NA	S	S
getComments( )	NA	C,S	C,S
getCommentsAsString( )	NA	S	S
getCompressionFormat( )	C	NA	NA
getCompressionType( )	NA	C	C
getCompressionType(byte[ ])	NA	S	S
getContent( )	C,S	C,S	C,S
getContentFormat( )	C	NA	NA
getContentInLob( )	NA	S	S
getContentLength( )	C	S	S
getContentLength(byte[ ])	NA	S	S
getData( )	S	S	S
getData(String, String, String)	S	S	S
getDataInFile( )	S	S	S
getDataInStream( )	S	S	S
getDescription( )	NA	C	C
getEncoding( )	NA	C	NA
getEncoding(byte[ ])	NA	S	S
getFormat( )	C	C	C

Table C–1 Methods That Operate on the Client-Side or Server-Side Object(Cont.)

getFormat(byte[ ])	NA	S	S
getFrameRate( )	NA	NA	C
getFrameRate(byte[ ])	NA	NA	S
getFrameResolution( )	NA	NA	C
getFrameResolution(byte[ ])	NA	NA	S
getFrameSize(byte[ ])	NA	NA	S
getHeight( )	C	NA	C
getMimeType( )	C	C	C
getNumberOfChannels( )	NA	C	NA
getNumberOfChannels(byte[ ])	NA	S	NA
getNumberOfColors( )	NA	NA	C
getNumberOfColors(byte[ ])	NA	NA	S
getNumberOfFrames( )	NA	NA	C
getNumberOfFrames(byte[ ])	NA	NA	S
getSampleSize( )	NA	C	NA
getSampleSize(byte[ ])	NA	S	NA
getSamplingRate( )	NA	C	NA
getSamplingRate(byte[ ])	NA	S	NA
getSource( )	C	C	C
getSourceLocation( )	C	C	C
getSourceName( )	C	C	C
getSourceType( )	C	C	C
getUpdateTime( )	C	C	C
getVideoDuration( )	NA	NA	C
getVideoDuration(byte[ ])	NA	NA	S
getWidth( )	C	NA	C
importData( )	S	S	S
importFrom( )	S	S	S
isLocal( )	C	C	C
loadComments( )	NA	S	S

Table C–1 Methods That Operate on the Client-Side or Server-Side Object(Cont.)

loadCommentsFromFile( )	NA	S	S
loadData(String, String, String, String)	S	S	S
loadData(String)	S	S	S
locateInComment( )	NA	S	S
openSource( )	NA	S	S
process( )	S	NA	NA
processAudioCommand( )	NA	S	NA
processCopy( )	S	NA	NA
processSourceCommand( )	NA	S	S
processVideoCommand( )	NA	NA	S
readFromComments( )	NA	S	S
readFromSource( )	NA	S	S
setAudioDuration( )	NA	C	NA
setBitRate( )	NA	NA	C
setComments( )	NA	C,S	C,S
setCompressionType( )	NA	C	C
setDescription( )	NA	C	C
setEncoding( )	NA	C	NA
setFormat( )	C	C	C
setFrameRate( )	NA	NA	C
setFrameResolution( )	NA	NA	C
setHeight( )	NA	NA	C
setKnownAttributes( )	NA	C	C
setLocal( )	C	C	C
setMimeType( )	C	C	C
setNumberOfChannels( )	NA	C	NA
setNumberOfColors( )	NA	NA	C
setNumberOfFrames( )	NA	NA	C
setProperties( )	C,S	NA	NA

Table C–1 Methods That Operate on the Client-Side or Server-Side Object(Cont.)

setProperties(byte[ ])	NA	C,S	C,S
setProperties(String)	C,S	NA	NA
setSampleSize( )	NA	C	NA
setSamplingRate( )	NA	C	NA
setSource( )	C	C	C
setSourceInformation( )	C	C	C
setUpdateTime( )	C,S	C,S	C,S
setVideoDuration( )	NA	NA	C
setWidth( )	NA	NA	C
trimComments( )	NA	S	S
trimSource	NA	S	S
writeToComments( )	NA	S	S
writeToSource( )	NA	S	S

Glossary

Some glossary entries are from the free online Dictionary of Computing located at: http://wombat.doc.ic.ac.uk/foldoc/index.html.

AAF

Advanced Authoring Format, a professional multimedia authoring format defined by Microsoft Corporation for the Windows operating system to succeed the AVI file format. AAF is an open, industry-developed format that enables the exchange of media among digital production tools and content creation applications.

A-Law

The European and worldwide (not North America and Japan) standard for nonuniform quantifying logarithmic compression. A-Law is used as the European telephony standard.

AC-3 (MPEG-2 Audio)/MP3

Surround sound audio.

ADPCM

Adaptive Differential Pulse Code Modulation. A compression technique that records only the difference between samples and adjusts the coding scale dynamically to accommodate large and small differences. ADPCM is simple to implement, but introduces much white noise.

ADT

Abstract data type. *See also* object type.

AIFF

Audio Interchange File Format. A file format developed by Apple Computer, Inc., for storing high-quality sampled audio and musical instrument information.

AIFF-C

Audio Interchange File Format with Compression. A file format developed by Apple Computer, Inc., for storing high-quality sampled audio and musical instrument information that uses compression.

ASF

Advanced Streaming Format. A multimedia streaming format defined by Microsoft Corporation for the Windows operating system to succeed the AVI file format. ASF is an open, industry-developed format specifically tuned for streaming media distribution.

aspect ratio

Proportions of height versus width.

attributes

Components of object types that are built-in data types or other user-defined types that model the structure of the real world. See *Oracle8i Concepts* for more information.

AU

Audio. Audio file format common to the UNIX platform, especially Sun and NeXT computing platforms.

audio

Sound. Computers (and audio compact discs and digital audio tape) handle sound by storing a sequence of discrete samples. The continuous sound waveform from the original source is sampled tens of thousands of times a second. Each sample represents the intensity of the sound pressure wave at that instant. Apart from the sampling frequency, the other parameter is the digital encoding of each sample including the number of bits used. The encoding may be linear, logarithmic or mu-law. The source could be a piece of music, voice recording, or generated audio. *See also* mu-law or u-law.

audio processing

A change to the properties of audio data, such as format, encoding type, number of channels, sampling rate, or sample size.

AVI

Audio Video Interleaved. A file container format defined by Microsoft Corporation. An AVI file typically includes both an audio and a video sequence. The audio and video data can be in several formats, such as Intel Indeo, Iterated Systems ClearVideo (fractal), or Motion JPEG. *See also* AAF *and* ASF.

bandwidth

The difference between the highest and lowest frequencies of a transmission channel.

BFILE

A large object whose value is composed of binary data that is stored outside the database in an operating system file. The file itself is not stored in the database, but a pointer to the file is stored in the database. Because they are outside of the database, BFILEs are read-only.

B-frame

Bidirectionally predictive-coded frame. In MPEG video, a B-frame is encoded using data describing how its picture has changed from the picture in the closest two I-frames or P-frames, one in the past and one in the future.

binary large object

See BLOB.

BLOB

Binary large object. Large objects are stored in the database tablespace in a way that optimizes space and provides efficient access. Only a pointer to the object is actually stored in the row.

client side

Used to indicate that a file or application is available only to a local system.

CLOB

A large object whose value is composed of single-byte fixed-width character data that corresponds to the database character set defined for the Oracle8 database.

codec

COmpressor DECompressor. Software modules that perform compression and decompression of audio and video data. Some common audio codecs or encoding types are ADPCM, Mu-Law, Two's Compliment, AC-3, and others. Some common video codecs or encoding types are MPEG-1, MPEG-2, Intel Indeo, Iterated Systems ClearVideo (fractal), Radius Cinepak, Motion JPEG, and others.

compression

The coding of data to save storage space or transmission time. Compressed data must be decompressed before it can be used.

compression format

The format of an encoded audio, image, or video file produced by a codec. The Oracle Video Server, for example, supports several compression formats including MPEG-1, MPEG-2, Intel Indeo, Iterated Systems ClearVideo (fractal), Radius Cinepak, and Motion JPEG. Compression formats for image files include JPEG and GIF.

container format

The way in which an encoder multiplexes video and audio together. For example, the container formats supported by the Oracle Video Server include MPEG-1 system, MPEG-2 transport, Oracle Video Stream, and any container format that meets the Raw Key Frame requirements.

content format

A description of the image data, such as the pixel or color format.

cropping

Selecting the portion of an image within a specified rectangle and removing everything outside of that rectangle. *See also* cutting.

cutting

Selecting the portion of an image within a specified rectangle to create a subimage. If the subimage is copied to a new image, the effect is a cut. If the subimage replaces the original, the effect is a crop.

data option

The mechanism by which clients, application-specific servers, and database servers can be easily and reliably extended.

default constructor

A method that creates an empty instance of an object.

duration

The time it takes to play the entire video data object.

encoding type

The particular codec algorithm used to encode the audio and video data. Some common audio codecs are ADPCM, Mu-Law, Two's Compliment, AC-3, and others. Some common video codecs are Indeo, MPEG, and others. For audio and video data, usually the encoding type is specific to the computer platform.

export

To move video data from the Oracle database to external storage as files, such as to a local file system as a BFILE, or to an HTTP server.

file format

The format of the audio data, such as AIFF, AIFF-C, AU, WAV, MIDI, RealAudio, and SND. The format of the video data, such as QuickTime, AVI, MPEG1, and Real Video.

flush

A process that sends data from the client-side object to the corresponding server-side object and updates the server-side object based on the data.

format

A specific data structure or organization; usually an industry standard such as AIFF-C for audio data or QuickTime, AVI, or MPEG1 for video data.

frame rate

The number of frames displayed every second. Thirty frames / second is considered a good rate for the human eye. A lower frame rate requires less bandwidth but reduces the quality of the video data. A higher frame rate produces a smoother, higher quality video, but requires more processing power and a greater bandwidth.

frame resolution

The number of pixels per inch of frames in the video data. Frame resolution can be adjusted to fit bandwidth requirements. Reducing frame resolution reduces video quality.

frame size

The height and width in pixels of each frame in the video data. This information is useful in estimating bandwidth requirements and determining appropriate display size for a given display screen size.

HTTP header

A body of information that a browser sends along with a URL when requesting a Web page. It includes such information as the browser type and MIME types.

I-frame

Intra-coded frames. In MPEG video, an I-frame is a frame coded as a still image.

I-frame regularity

To ensure good rate control performance, I-frames must occur at regular intervals throughout a video stream. For the US and Japan, there are usually 11 frames (B-frames and P-frames) between each pair of I-frames in a sequence such as: IBBPBBPBBPBBIBBPBBP.... This is based on a random access requirement that an I-frame (starting point) is needed at least once every 0.4 seconds or so. MPEG encoding software often enables you to configure the I-frame frequency.

image

A graphic picture. The source could be a photograph, drawing, or generated image.

image processing

Changing the properties of an image, such as through scaling, rotation, or compression.

import

To move video data into an Oracle database from external storage, such as external file servers, media servers, HTTP servers, FTP servers, and so forth.

JDBC

Java Database Connectivity. Allows Java programs to send SQL queries to databases and retrieve the results of those queries.

JPEG

Joint Photographic Experts Group. The original name of the committee that designed the standard image compression algorithm. The filename extension is .jpg or .jpeg. JPEG is a static-image compression format.

LOB

Large object. LOBs are used to hold large amounts of raw, binary data. Oracle8*i* *inter*Media Audio, Image, and Video Options support LOBs.

localData

A BLOB attribute that can keep up to 4GB of video data stored within the Oracle database. All data within localData is protected by Oracle's security and transaction support.

local file system

A file system that allows local access where, for example, BFILES can be stored.

local source

The location of audio, image, or video data within the Oracle database.

lossless compression

A term describing a data compression algorithm that retains all the information in the data, allowing it to be recovered perfectly by decompression.

A means for reducing the storage space required for an image. The decompressed image is bit-for-bit identical to the original.

Unix compress and GNU gzip perform lossless compression.

lossy compression

A term describing a data compression algorithm that actually reduces the amount of information in the data, rather than just the number of bits used to represent that information. Information is removed because it is subjectively less important to the quality of the data (usually an image or sound) or because it can be recovered reasonably by interpolation from the remaining data.

A means for reducing the storage space required for an image. The decompressed image is not bit-for-bit identical to the original. Some details are lost or changed, although this might not be noticeable to the naked eye.

MPEG and JPEG are examples of lossy compression techniques.

methods

Components of an object type that are functions or procedures written in PL/SQL and stored in the database or written in a language like C or Java and stored externally. Methods implement specific operations that an application can perform on the data. See *Oracle8i Concepts* for more information.

MIME

Multipurpose Internet Mail Extensions. *See* Multipurpose Internet Mail Extensions (MIME).

MIME Type

A file format defined by the Multipurpose Internet Mail Extensions standard. This unique identifier is used for different file types when conveyed across a MIME-based protocol such as MIME e-mail or HTTP.

moving JPEG, motion JPEG, M-JPEG

A compression technique for moving images that applies JPEG still image compression to each frame of a moving picture sequence.

Play-back requires a machine capable of decompressing and displaying each JPEG image quickly enough to sustain the required frame rate of the picture sequence.

There is no standard for Moving JPEG as with JPEG, but there are JPEG compression chips, which are designed to work at television frame rates and resolutions.

MPEG

Moving Picture Expert's Group, an ISO body focused on developing compression formats for full-motion video and audio signals and the synchronization of these signals during playback.

MPEG-1

The first version of the MPEG format, optimized for CD-ROM. It uses discrete cosine transform (DCT) and Huffman coding to remove spatially redundant data within a frame and block-based motion compensated prediction (MCP) to remove data which is temporally redundant between frames. Audio is compressed using subband encoding.

MPEG-2

A variant of the MPEG video and audio compression algorithm and file format, optimised for broadcast quality video for digital storage media up to 4.0Mbits/second. The file extension is .MP2.

MPEG-2 has been approved as International Standard IS-13818.

MPEG-3

A variant of the MPEG video and audio compression algorithm and file format. The file extension is .MP3. This variant no longer exists and has been merged into MPEG-2.

MPEG-4

A variant of the MPEG video and audio compression algorithm and file format used for low bandwidth video telephony. The file extension is .MP4. The International Standards Organization (ISO) has adopted the QuickTime 3 file format to form the starting point for a unified digital media storage format for the MPEG-4 specification. *See also* QuickTime 3.

mu-law or u-law

The North American and Japanese standard for nonuniform quantising logarithmic compression.

Multipurpose Internet Mail Extensions (MIME)

An Internet specification describing file types. Servers and browsers read the MIME type in the file header and decide what to do with the file, such as displaying it with a viewer or playing it as an audio file. MIME is used by HTTP servers to describe the type of file being delivered.

number of channels

Number of audio channels present in the formatted audio data; also known as the channel count. Number of channels may range from 1 for mono audio data to 6 for AC3 encoded surround sound audio data.

object relational database

A database having both object-oriented and relational characteristics. Objects can be defined and stored, and then retrieved using standard relational methods.

object type

A data type that includes attributes of the data and methods (functions and methods) for operating on the data.

An object type is sometimes referred to as an abstract data type (ADT).

OCI

Oracle Call Interface.

Oracle Video Server

See OVS.

Oracle Web Application Server

An HTTP server that enables you to create and deploy distributed, cross-platform, dynamic applications. It provides a framework that encompasses a modular distributed architecture, an open API that enables you to create portable applications, and different application models or paradigms. Oracle Web Application Server consists of three components: HTTP Daemons (UNIX) or Listeners, Web Request Broker (WRB), and options or server-side applications.

OVS

Oracle Video Server. An Oracle product that provides an end-to-end software solution for networked client and server computers that store, manage, deliver, and display digital video and audio on demand. Client applications run a wide variety of consumer and corporate platforms. OVS is supported on a variety of server platforms and scales to serve many concurrent users.

P-frame

Predictive-coded frames. In MPEG video, a P-frame is encoded using data describing how its picture has changed from the picture in a previous I-frame or P-frame.

QuickTime

A file container format defined by Apple Computer, Inc., for integrating full-motion video and digitized sound into application programs.

QuickTime 3

An industry standard file format defined by Apple Computer, Inc., to create and publish digital media for the Mac OS, Java, and Windows operating environments. The International Standards Organization (ISO) has adopted the QuickTime 3 file format to form the starting point for a unified digital media storage format for the MPEG-4 specification. This will allow the vast majority of existing hardware, software, and digital content to work seamlessly with this next generation version of MPEG. *See also* MPEG-4.

Raw Key Frame (RKF)

A set of rules to which the content of video container format can conform. Some video servers, such as the Oracle Video Server, can only play content in container formats that conform to the rules of RKF. The rules of RKF are:

- **stateless** - all the data for displaying the picture in a video frame is contained entirely in that frame, rather than in previous frames.
- **contiguous** - all the data for a frame is stored together in the video file and no other data is mixed with it.

raw pixel format

An uncompressed image format with a simple, fixed-size header.

refresh

A process that sends data from the server-side object to the client-side object and updates the client-side object based on the data.

sampling rate

The rate in samples per second at which the audio data was recorded. The sampling rate ranges from 5,500 (one fourth of the Mac sampling rate) to 48,000 (Digital Audio Tape (DAT) sampling rate).

sample size

The number of samples of audio data present in the audio data, or, stated another way, the number of bits per sample. The number of bits per sample is either 8 (8-bit) or 16 (16-bit).

scaling

Changing the proportions of an image in one or both dimensions. To enlarge an image, scale by a factor greater than one. To shrink an image, scale by a factor between zero and one.

server side

Used to indicate that files or applications are accessible by a number of different computers over a network.

SND

Sound. Audio file format in use on the UNIX platform, especially Sun and NeXT computing platforms.

uniform resource identifier (URI)

A compact string representation of a location (URL) for use in identifying an abstract or physical resource. URI is one of many addressing schemes, or protocols, invented for the Internet for the purpose of accessing objects using an encoded address string.

uniform resource locator (URL)

A form of URI. A compact string representation of the location for a resource that is available on the Internet. It is also the text-string format that client-side applications use to encode requests to Oracle Application Server.

video

Streaming images. The source could be a video camera, a recording, or a generated animation.

video compression

Compression of a sequences of images.

WAV

Waveform-audio. WAV is a standard file container format developed by Microsoft Corporation for storing digital audio files. Conversion tools are available to allow most other operating systems to play WAV files.

Index

F

G

I

J

L

M

O

P

R

S

T

W