Oracle® *inter*Media Annotator

User's Guide

Release 9.0.1

June 2001

Part No. A88784-01

Oracle *inter*Media Annotator is a utility that extracts metadata from audio, image, and video sources of certain formats and inserts the metadata, along with the media source file, into an Oracle database.

ORACLE

*Oracle inter*Media Annotator User's Guide, Release 9.0.1

Part No. A88784-01

Primary Author: Max Chittister

Contributors: Marco Carrer, Kan Deng, Cheng Han, Sam Lee, Paul Lin, Susan Mavris, Susan Shepard, Alok Srivastava, Rod Ward

Contents

6 Annotator Engine API Reference Information

7 Creating PL/SQL Upload Templates

List of Examples

List of Figures

List of Tables

Send Us Your Comments

Oracle *inter*Media Annotator User's Guide, Release 9.0.1

Part No. A88784-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. 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?

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

- Electronic mail: nedc-doc_us@oracle.com
- FAX: 603.897.3825 Attn: Oracle *inter*Media Annotator Documentation
- Postal service:
  Oracle Corporation
  Oracle *inter*Media Annotator Documentation
  One Oracle Drive
  Nashua, NH 03062-2804
  USA

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

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

Preface

Oracle *inter*Media Annotator is a utility that extracts metadata from audio, image, and video sources of certain formats and inserts the metadata, along with the media source file, into an Oracle database.

Audience

This guide is intended for anyone who is interested in extracting metadata from a multimedia file and storing both the metadata and the multimedia file in an Oracle database. Users who want to integrate the Annotator engine with their applications should be familiar with Java and JDBC. Advanced users who want to write their own PL/SQL Upload Templates should be familiar with PL/SQL. Advanced users who want to write their own annotation types or parsers should be familiar with Java and XML.

Organization

This guide contains 10 chapters and 4 appendixes:

Chapter 1	Contains a general introduction.
Chapter 2	Contains an overview and configuration information for the Annotator GUI.
Chapter 3	Contains information on using the Annotator GUI to create and manipulate annotations.
Chapter 4	Contains instructions for using the PL/SQL Template Wizard to generate PL/SQL Upload Templates.

Chapter 5	Contains a full-length example of a Java application using the Annotator engine.
Chapter 6	Contains reference information on the Java APIs associated with the Annotator engine.
Chapter 7	Contains instructions for writing a PL/SQL Upload Template to upload your annotation to an Oracle database.
Chapter 8	Contains a full-length example of a custom parser.
Chapter 9	Contains reference information on the Java APIs associated with writing custom parsers.
Chapter 10	Contains information on creating your own annotation types.
Appendix A	Contains information on using Oracle9*i* Text to query stored annotations.
Appendix B	Contains reference information on supported formats.
Appendix C	Contains reference information on annotation attributes.
Appendix D	Contains answers to frequently asked questions.

Accessibility

Oracle's goal is to make our products, services, and supporting documentation accessible to the disabled community with good usability. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For additional information, visit the Oracle Accessibility Program Web site at `http://www.oracle.com/accessibility/`.

JAWS, a Windows screen reader, may not always correctly read the Java code examples in this document. The conventions for writing Java code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

Conventions

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

Although Boolean is a proper noun, it is presented as boolean in this guide because Java is case-sensitive.

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

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 either a term defined in the text, the glossary, or in both locations; or a window name, button name, menu name, or menu item.
`monospace font`	Monospace font in text indicates a code example, a URL, or an absolute path name.
< >	Angle brackets enclose user-supplied names.
[ ]	Brackets enclose optional clauses from which you can choose one or none.

1
Introduction

This chapter supplies information on Oracle *inter*Media Annotator, which extracts information (or **metadata**) from media sources of certain formats and inserts the metadata, along with the media source, into an Oracle database.

*inter*Media Annotator uses Oracle *inter*Media.

1.1 Purpose

When managing multimedia data in an object-relational database system, you will likely face the problem of how to extract, process, and manage metadata associated with your media sources. Metadata, which typically consists of text-based information that describes the media source, is usually embedded within the media source using a proprietary format, and is therefore not always easily accessible. To be able to efficiently manage and use metadata, you must be able to extract it from many different types of media sources. After extraction, you must have a consistent, accurate representation of the metadata, regardless of the original media source.

Oracle *inter*Media Annotator is a Java-based engine that is used to organize a set of multimedia content and metadata and upload it to an Oracle database.

You can use Annotator to parse a media source, extract its metadata, and group the metadata into an organized structure called a **logical annotation** (or **annotation**). Every annotation is organized as a set of text attributes and optional samples. An **attribute** provides information about the media source, either its data format (such as MIME type or format) or data content (such as song title or movie director). **Samples** are multimedia data (such as audio clips or closed captions) extracted from the media source.

You can use *inter*Media Annotator to parse your audio, image, or video files (see Appendix B for a list of supported file formats) and extract attributes to build an annotation.

*inter*Media Annotator also creates a separate annotation for each track of the media source. For example, for a media source containing a movie, *inter*Media Annotator can create separate annotations for the video data and audio data; those annotations would be sub-annotations of the movie annotation.

You can use *inter*Media Annotator to insert the annotation along with the media source into an Oracle database. Once the annotation is in the database, you can use Oracle9*i* Text to query the annotation.

You can use the *inter*Media Annotator functions in one of two ways. If you want to become more familiar with *inter*Media Annotator, or if you have simple needs that do not require writing a Java application, you can use the graphical user interface (GUI) of the *inter*Media Annotator utility to generate annotations. The Annotator GUI is built on the Annotator Java-based engine.

For example, users who are responsible for creating and storing movie trailers can use the Annotator GUI to upload the movie trailers to an Oracle database. The Annotator GUI can automate the process of extracting metadata and uploading both the metadata and the movie trailer to an Oracle database in a few clicks. Once the content is uploaded, the movie trailers can be searched, published, or streamed through an application, such as the MediaFinder sample application.

See Part I, "interMedia Annotator GUI" for more information on the *inter*Media Annotator GUI.

Additionally, if you are an application developer who needs to organize related multimedia files, you can use the *inter*Media Annotator engine Java APIs to integrate the Annotator functions into your application. You can also use the *inter*Media Annotator engine APIs as a tool for bulk loading many multimedia files into the database.

For example, in the scenario presented previously, if you have a large number of movie trailers to store, you might not want to click through the GUI for every movie trailer. Instead, you can write a custom Web-based application to parse the movie trailers, generate annotations for each trailer, and upload the movie trailers to an Oracle database automatically.

See Part II, "interMedia Annotator Java-Based Engine" for more information on the *inter*Media Annotator Java engine.

*inter*Media Annotator is extensible; you can use the Annotator parser Java APIs to write a custom parser for your media source files, or create your own annotation types.

For example, a real-estate company might maintain a list of properties to be sold, with each entry containing a picture of the exterior, a short movie of the interior, a technical document, selling price, and other information.

Using the extensibility features of Annotator, a developer can easily create a Web-based application to allow selling agents to supply the entry, including the multimedia files, and upload them to Oracle as a property content unit. The developer can define a new annotation type named *property*, containing the name, selling price, closing date, and other text information. The new annotation type can include an image sub-annotation describing the picture of the exterior or a movie sub-annotation describing the movie of the interior of the house. The developer can also write a new parser to create the property annotation.

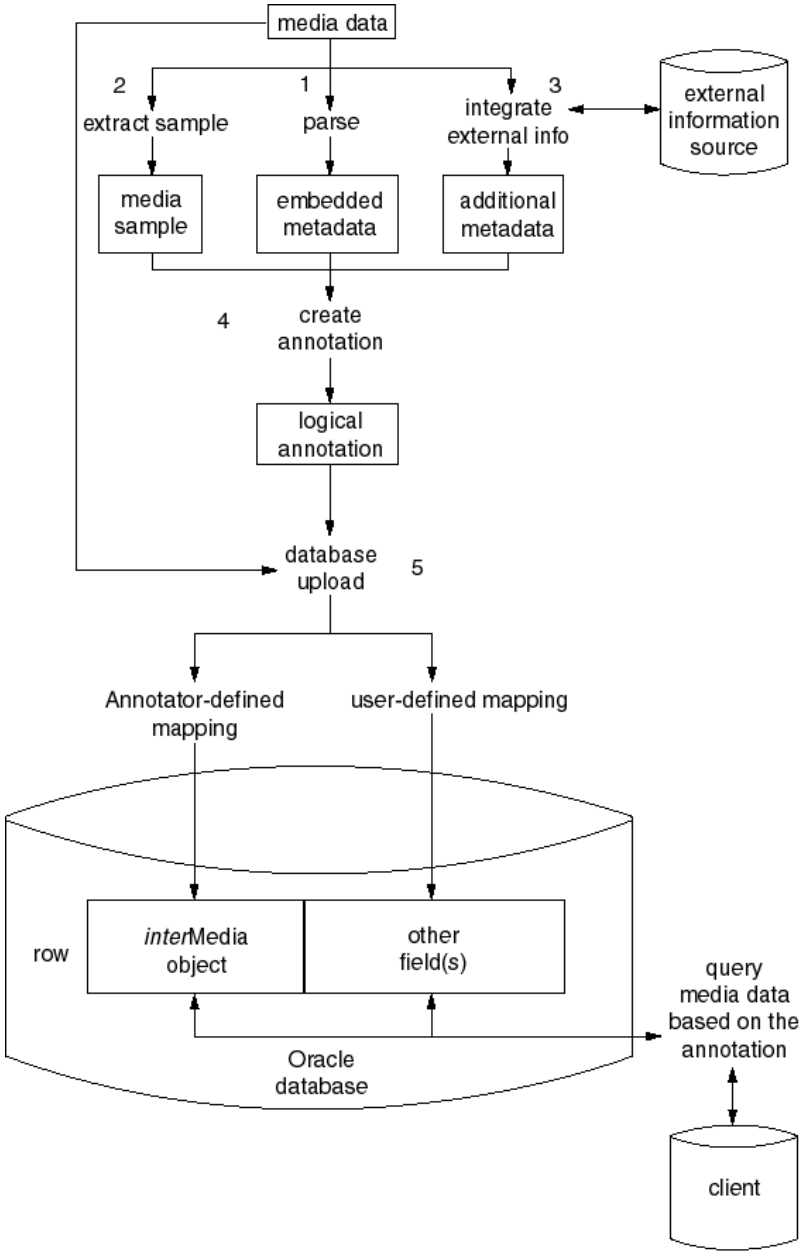
See Part III, "interMedia Annotator Extensibility" for more information on extending *inter*Media Annotator.

1.2 *inter*Media Annotator Operations Overview

The main functions of *inter*Media Annotator are to build a logical annotation from a media source and to then upload both the annotation and the source file to an Oracle database. Users can then query the media data in the database based on information in the annotation.

Figure 1–1 provides an overview of this process. This process can be performed either by the Annotator GUI or a custom-written Java application using the Annotator Java APIs.

Figure 1–1 Overview of interMedia Annotator Operations



You can use *inter*Media Annotator to perform the following operations, in this order:

1. Parse the media source. *inter*Media Annotator extracts the metadata from the source file.
2. Extract samples from the media source. *inter*Media Annotator extracts a sample from the media data (such as a text track from a movie file).
3. Integrate information from additional sources. Some information that would be useful in an annotation is not necessarily included in the metadata. For example, you could import data from a previously generated annotation.
4. Create a logical annotation. *inter*Media Annotator combines the extracted samples and the metadata, and builds a logical annotation.

   Applications can further customize the annotation at this point.
5. Upload the annotation and the media source to an Oracle database.

   *inter*Media Annotator will upload the media source and the annotation (in XML format) into an *inter*Media object in the database. *inter*Media Annotator can also upload individual attributes from the annotation into other columns of the database. You specify the *inter*Media object to which you will upload, along with the rest of the information to be uploaded, in a PL/SQL Upload Template. You can create a template using a text editor or the PL/SQL Template Wizard.

   See Chapter 4 and Chapter 7 for more information on the upload process.

Once you have completed these steps, you will be able to query the information in the annotation in order to use information about the media source that cannot be directly extracted. You can also build indexes on the information in the annotation using Oracle9*i* Text.

1.3 Prerequisites

To use *inter*Media Annotator, you must have access (either local or remote) to an Oracle database with Oracle *inter*Media, and an Oracle JDBC driver (either Thin or OCI) for Oracle 8.1.5 or later.

To use *inter*Media functions in your Java applications, you should use the Java Development Kit 1.1.7 or later.

Part I

*inter*Media Annotator GUI

Part I provides information about the Oracle *inter*Media Annotator GUI and contains the following chapters:

- Chapter 2, "GUI Overview and Configuration"
- Chapter 3, "Generating and Manipulating Annotations with the Annotator GUI"
- Chapter 4, "PL/SQL Template Wizard"

2

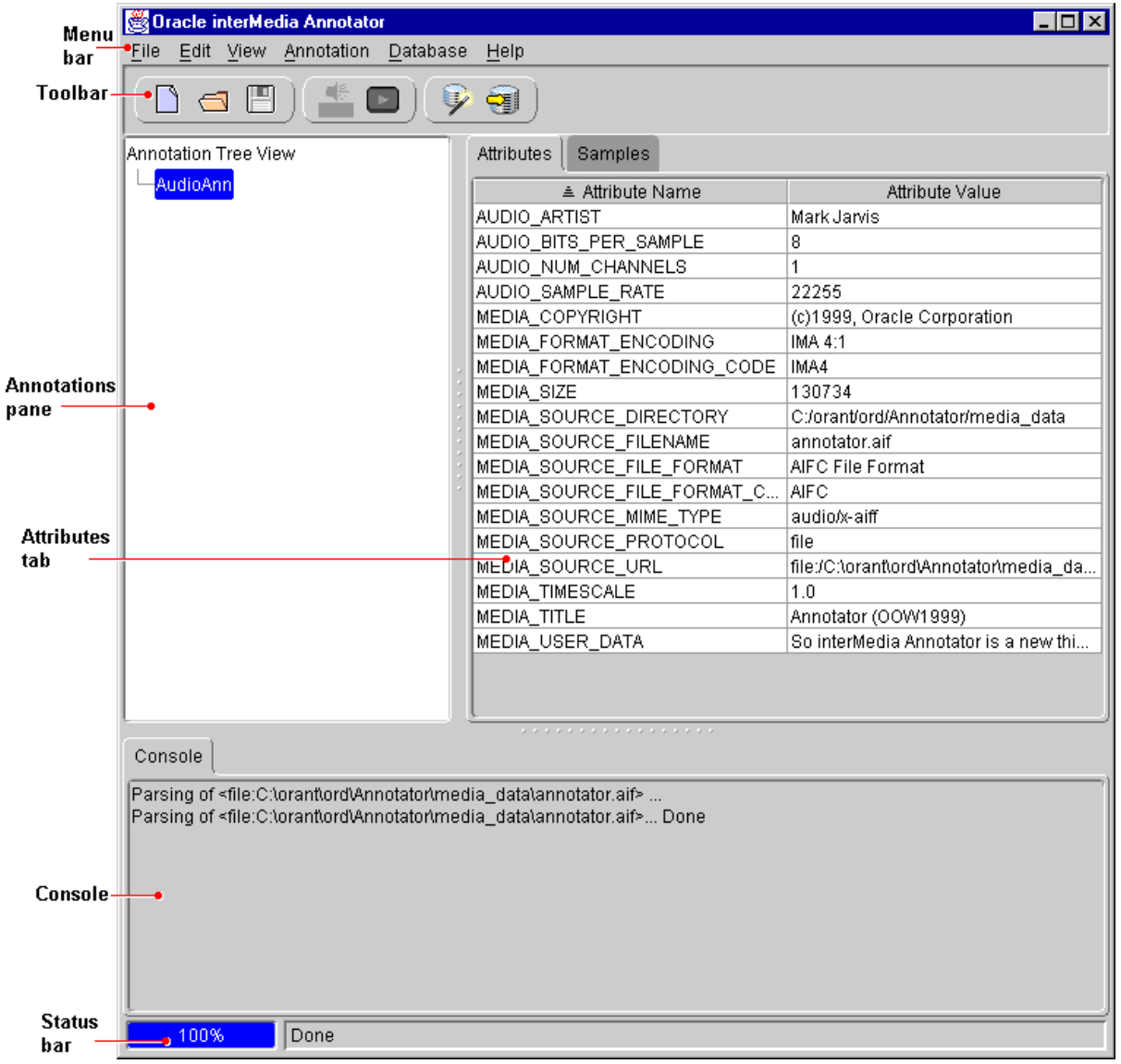
GUI Overview and Configuration

This chapter provides an overview of the Oracle *inter*Media Annotator GUI. This chapter also provides instructions on the configuration of Annotator and the installation and configuration of additional helper applications.

2.1 GUI Overview

After you install and configure *inter*Media Annotator, run *inter*Media Annotator by opening either `Annotator.bat` (on Windows NT systems), or `Annotator.sh` (on Unix and Linux systems). The **Oracle *inter*Media Annotator** window appears (Figure 2–1).

Figure 2–1 Oracle interMedia Annotator Window



The main features of the Oracle *inter*Media Annotator window are the following:

- **Menu bar**: The menu bar contains six menus: **File**, **Edit**, **View**, **Annotation**, **Database**, and **Help**.

  The **File** menu contains commands to create a new annotation, open an annotation, close an annotation, save an annotation, save the contents of the **Console** window, and exit *inter*Media Annotator.

  For more information on saving annotations, see Section 3.6.

  The **Edit** menu contains a command to open the **Preferences** window.

  For more information on the *inter*Media Annotator **Preferences** window, see Section 2.2.1 and Section 3.5.3.

  The **View** menu contains check boxes that let you choose whether or not to view the three toolbars or the **Console**.

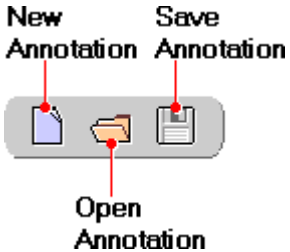
  The **Annotation** menu contains commands to extract samples from the media source, play the media source in the appropriate media player, insert or delete an attribute, and insert or delete an annotation. For more information on extracting samples, see Section 3.5. For more information on playing media files and samples, see Section 3.8. For more information on defining helper applications, see Section 2.2.5.

  The **Database** menu contains commands to create and run PL/SQL Upload Templates. See Chapter 4 for more information.

  The **Help** menu contains a command to show more information about *inter*Media Annotator.

- **Toolbar**: The toolbar consists of three smaller toolbars, the **Standard** toolbar (Figure 2–2), the **Annotation** toolbar (Figure 2–3), and the **Oracle** toolbar (Figure 2–4).

Figure 2–2 Standard Toolbar



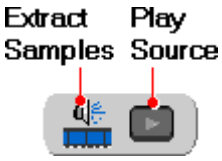
The **Standard** toolbar contains three buttons: **New Annotation**, **Open Annotation**, and **Save Annotation**.

Clicking the **New Annotation** button opens a dialog box in which you choose the media source to parse. *inter*Media Annotator then parses the media source and creates a new annotation. See Section 3.1 for more information.

Clicking the **Open Annotation** button opens a previously extracted annotation.

Clicking the **Save Annotation** button saves your annotation as an XML file.

Figure 2–3 Annotation Toolbar

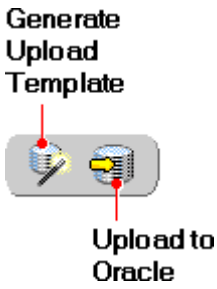


The **Annotation** toolbar contains two buttons: **Extract Samples** and **Play Source**.

The **Extract Samples** button extracts samples from the media source. See Section 3.5 for more information.

The **Play Source** button plays the media source in the appropriate media player. For more information on playing media files and samples, see Section 3.8. For more information on defining helper applications, see Section 2.2.5.

Figure 2–4 Oracle Toolbar



The Oracle toolbar contains two buttons: **Generate Upload Template** and **Upload to Oracle**.

The **Generate Upload Template** button starts the PL/SQL Template Wizard, which takes you through a step-by-step process to define a PL/SQL Upload Template.

The **Upload to Oracle** button opens a dialog box in which you select a PL/SQL Upload Template, which uploads your media source and annotation to a selected Oracle database.

See Chapter 4 for more information.

- **Annotations pane**: The **Annotations** pane contains an expandable list containing the hierarchy of annotations and sub-annotations. The types of the annotations and sub-annotations are shown in the **Annotations** pane.

  For more information, see Section 3.1.

- **Attributes tab**: The **Attributes** tab shows the annotation attributes and their values. You can use the **Attributes** tab to change attribute values.

  View the **Attributes** tab by clicking the gray **Attributes** box when the **Samples** tab is visible.

  For more information on the **Attributes** tab, see Section 3.2 and Section 3.3.

- **Samples tab**: The **Samples** tab shows the contents of an extracted sample. It shows either the text of a text sample laid out on a time line, or images extracted from a QuickTime movie file. It appears in the same panel of the **Oracle *inter*Media Annotator** window as the **Attributes** tab.

  View the **Samples** tab by clicking the gray **Samples** box when the **Attributes** tab is visible.

  The **Samples** tab is not shown in Figure 2–1. See Figure 3–8 for an illustration of the **Samples** tab.

- **Console**: The **Console** window displays messages pertaining to the status of *inter*Media Annotator operations. If an error occurs, notification is printed to the **Console**, along with notification of any action that is taken by *inter*Media Annotator.

- **Status bar**: The status bar shows how much of an operation is completed, from 0% to 100%.

2.2 Configuration

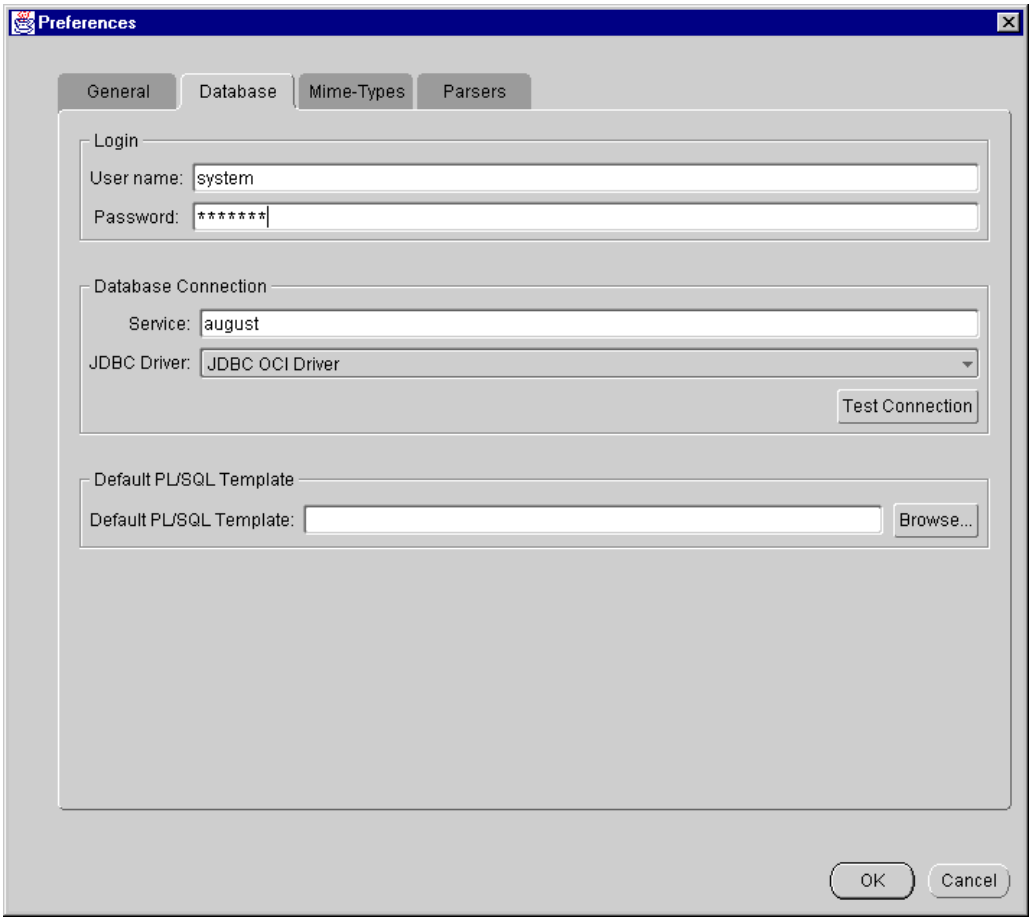
To configure the *inter*Media Annotator GUI to best use its features, perform the steps listed in Section 2.2.1 through Section 2.2.5.

2.2.1 Set Connection to the Database

One of the most useful and powerful features of *inter*Media Annotator is the ability to upload media sources and annotations to an Oracle database. To connect to an Oracle database, you must correctly specify the database connection parameters by performing the following operations:

1. From the **Edit** menu, select **Preferences** and click the **Database** tab. The **Database** tab of the **Preferences** window appears (Figure 2–5).

Figure 2–5 Database Tab of the Preferences Window



2. Enter the necessary information to ensure that you will connect to the correct database:
   - If you are using the JDBC Thin driver, your service name must follow the syntax <machine-name>:<port-name>:<oracle-sid>.
   - If you are using the JDBC OCI driver, your service name must follow the syntax used by Oracle Net. See *Oracle Net Services Administrator's Guide* for more information.

   Note: All changes that you make in the **Preferences** window will be saved from session to session except the database password. For security reasons, you must re-enter the password every session.

3. Click **OK** to confirm and save the changes.

2.2.2 Create Tables

Before you can upload annotations and media files, your database must contain tables that can be used for the storage of audio and video data. If it does not, you will need to create these tables.

The following SQL statements will create two sample tables, one for the storage of the video data and one for the storage of the audio data. These statements will create the tables that are used in the example in Chapter 4:

```
CREATE TYPE VideoType AS OBJECT (ID    NUMBER,
                                 title VARCHAR2(256),
                                 vsrc  ORDSYS.ORDVIDEO);
CREATE TABLE VideoStorage OF VideoType (ID PRIMARY KEY)
            LOB(vsrc.source.localdata) STORE AS (NOCACHE NOLOGGING);

CREATE TYPE AudioType AS OBJECT (ID    NUMBER,
                                 title VARCHAR2(256),
                                 asrc ORDSYS.ORDAUDIO);
CREATE TABLE AudioStorage OF AudioType (ID PRIMARY KEY)
            LOB(asrc.source.localdata) STORE AS (NOCACHE NOLOGGING);
```

Although object types and object tables are used in the examples, any relational tables with *inter*Media Audio or Video types can be used for the storage of audio or video data.

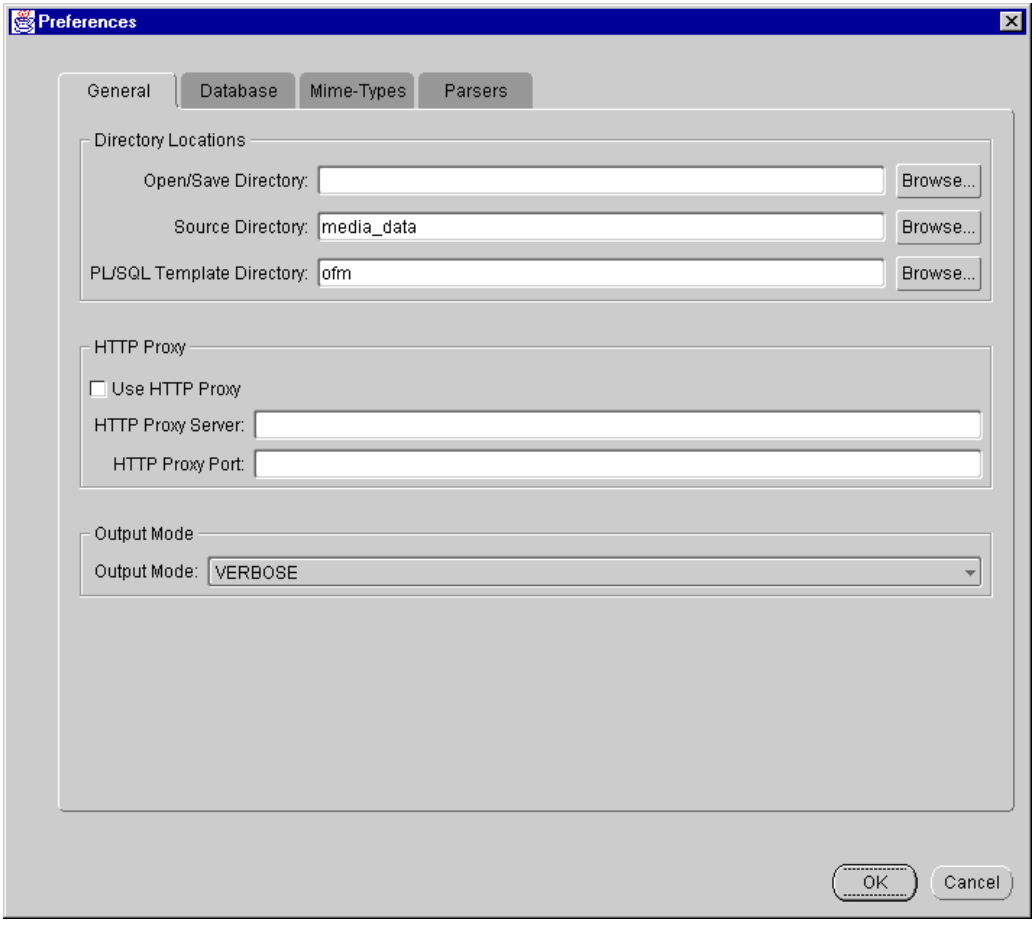
2.2.3 Check the Proxy Settings

*inter*Media Annotator can annotate media sources that are available remotely over the Internet through the HTTP protocol.

If you are running in a secure environment, you will need to configure *inter*Media Annotator to use your proxy server before you can access the Internet. Configure the proxy server by performing the following operations:

1. From the **Edit** menu, select **Preferences** and click the **General** tab. The **General** tab of the **Preferences** window appears (Figure 2–6).

Figure 2–6 General Tab of the Preferences Window



2. Enter your proxy server settings in the HTTP Proxy Server: and HTTP Proxy Port: text fields.
3. Click **OK**.

2.2.4 Connect to a CDDB

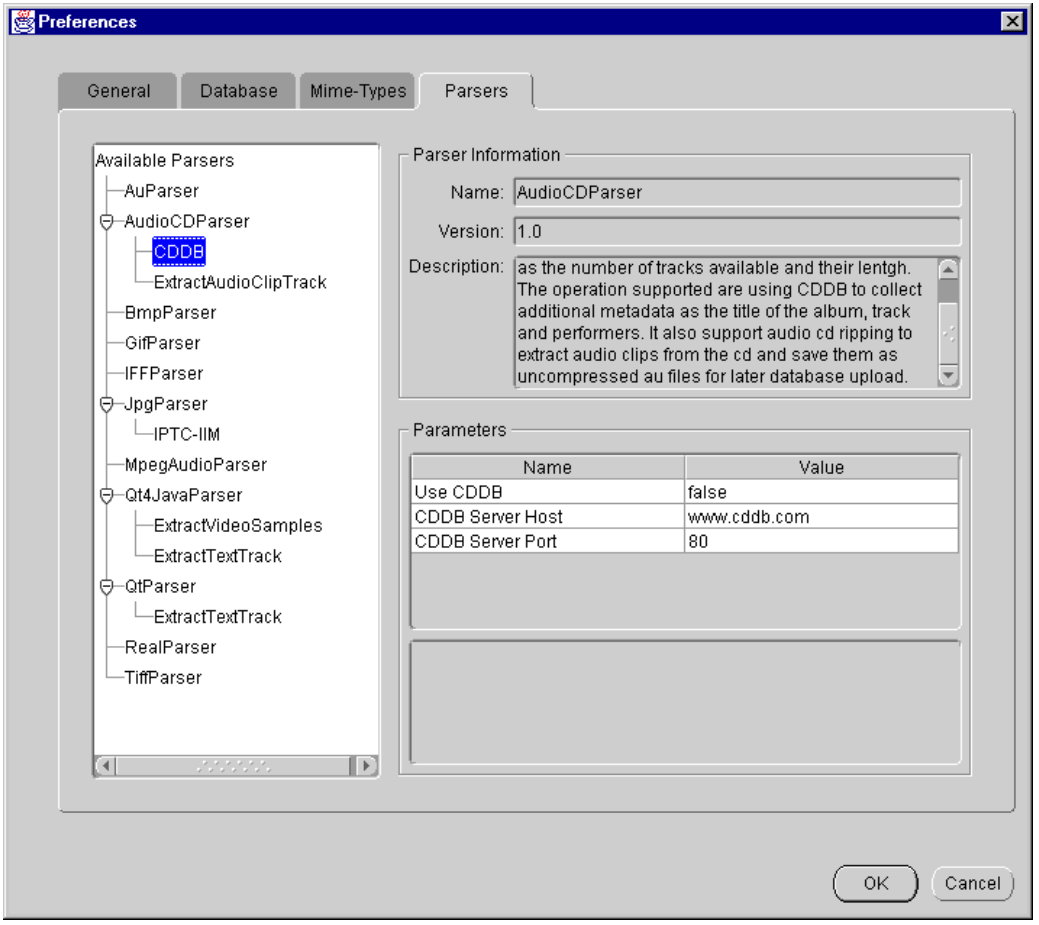
*inter*Media Annotator can connect to a CDDB to add information to audio CD annotations. However, you must contact the administrators of the CDDB in order to obtain a license before using CDDB functions; contact them for a license at:
`http://www.cddb.com/developers.html`

To change the CDDB URL that Annotator queries for information, perform the following operations:

1. From the **Edit** menu, select **Preferences** and click the **Parsers** tab.

   The **Parsers** tab of the **Preferences** window appears (Figure 2–7).

Figure 2–7 Parsers Tab of the Preferences Window



2. If CDDB is not visible under AudioCDParser, click the plus sign (+) next to AudioCDParser to expose it.
3. Enter the host name and port number in the CDDB Server Host: and CDDB Server Port: text fields.

Note: You must obtain a license from the CDDB server before you can use its functions.

---

4. Click **OK** to confirm and save the changes.

2.2.5 Install Helper Applications

*inter*Media Annotator is capable of playing media sources and extracted media samples. However, in order to play them, you may need to install some additional helper applications.

Recommended Windows NT applications are:

- Microsoft Windows Media Player
- QuickTime Player
- RealPlayer
- WinAmp

Recommended Macintosh applications are:

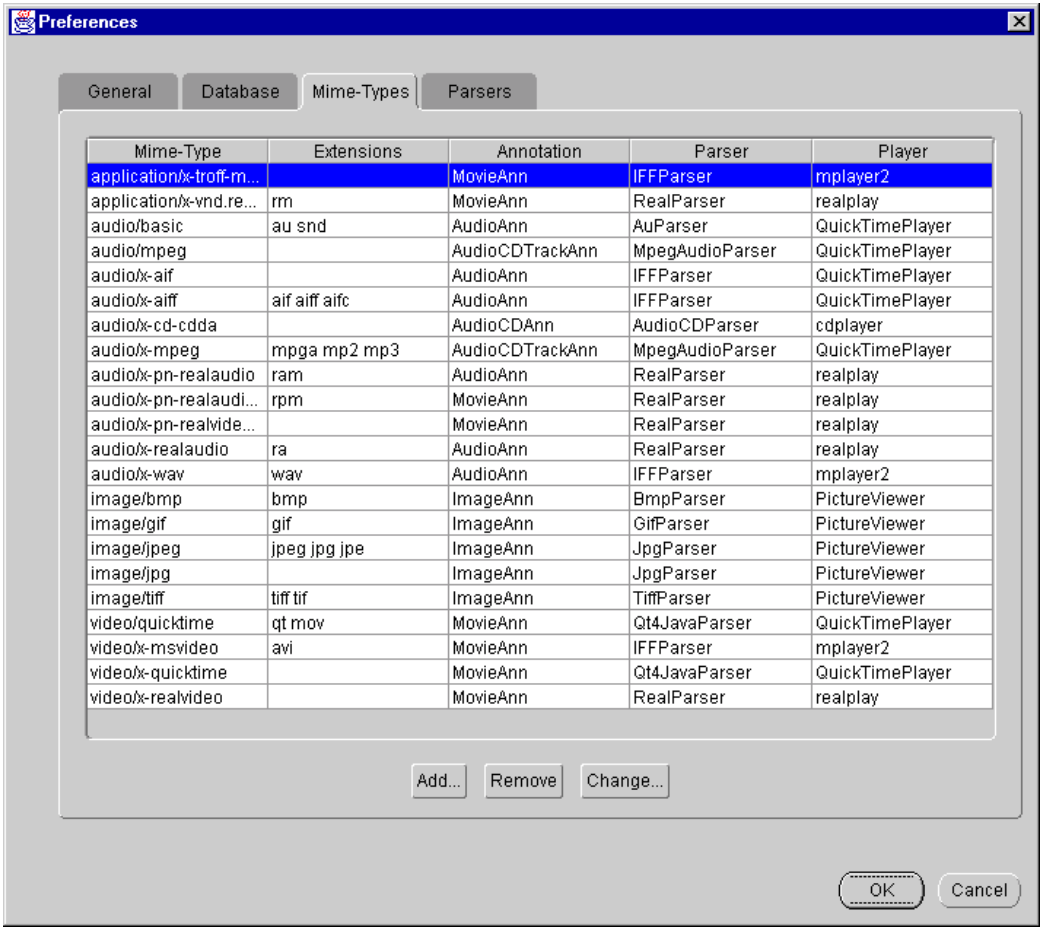
- QuickTime 4.0 Player
- RealPlayer

Make sure that each MIME type is paired up with the correct file extension and player by performing the following operations:

1. From the **Edit** menu, select **Preferences** and select the **Mime-Types** tab.

   The **Mime-Types** tab of the **Preferences** window appears (Figure 2–8).

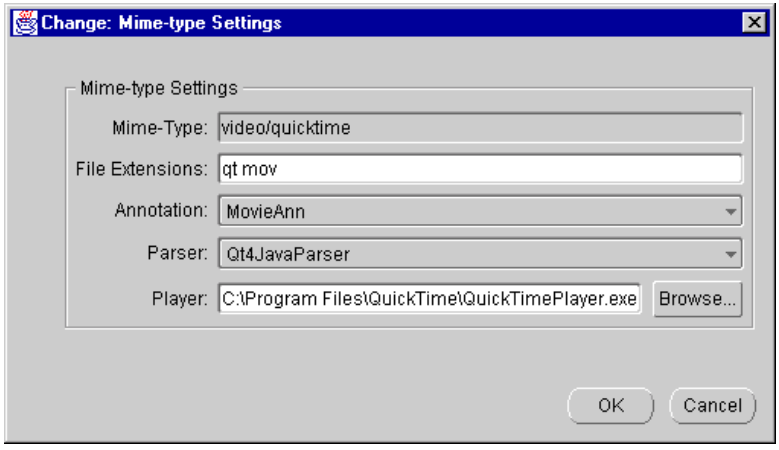
Figure 2–8 Mime-Types Tab of the Preferences Window



2. Double-click on the row of the MIME type that you want to edit.

   The **Change: Mime-type Settings** window appears (Figure 2–9).

Figure 2–9 Change: Mime-type Settings Window



3. Edit the contents of the window and click **OK**.

   The **Change: Mime-type Settings** window closes.

4. Click **OK** to confirm and save the changes.

3

Generating and Manipulating Annotations with the Annotator GUI

*inter*Media Annotator is packaged along with several sample multimedia files. They are included in the <*ORACLE_HOME*>\ord\Annotator\media_data directory. You can use *inter*Media Annotator on these files (or on files of your own choosing) to perform a number of operations. These operations include:

- Creating an annotation
- Editing attributes
- Adding and deleting attributes to the annotation
- Adding and deleting an annotation
- Extracting samples
- Saving an annotation
- Opening an annotation
- Playing media sources or samples

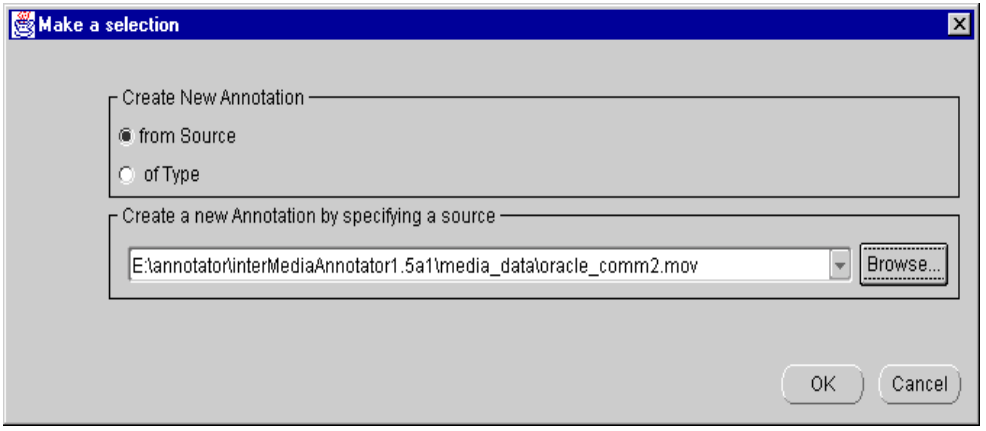
3.1 Creating an Annotation

To create an annotation, perform the following operations:

1. Either select **New** from the **File** menu or click the **New Annotation** button.

   The **Make a Selection** window appears (Figure 3–1).

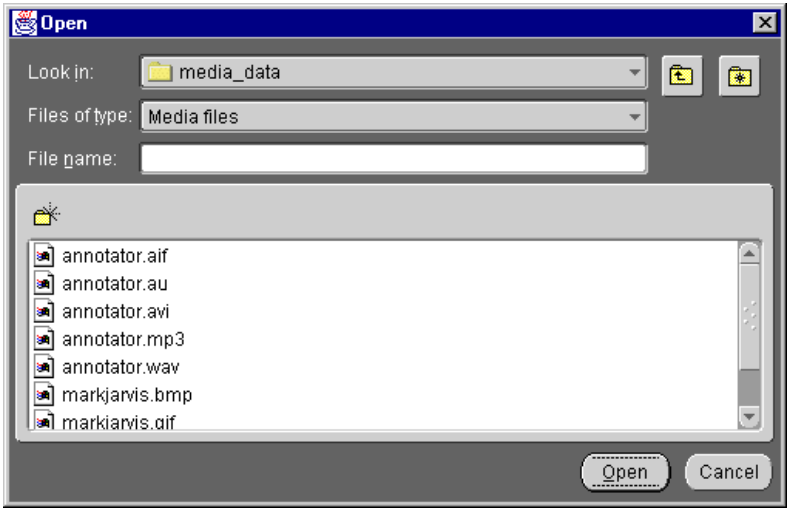
Figure 3–1 Make a Selection Window



2. To create an annotation populated with metadata from a media file, click the **from Source** button and select a media file from the pull-down menu.

   If the media file does not appear in the pull-down menu, click the **Browse** button and select a media file in the **Open** window (Figure 3–2).

Figure 3–2 Open Window



3. To create an empty annotation, click the **Files of Type** pull-down menu and select an annotation type from the pull-down menu. (This option will be used most often in conjunction with a user-defined annotation type. See Chapter 10 for more information on creating your own annotation types.)
4. Click **OK**.

See Figure 1–1 for more information on how *inter*Media Annotator builds an annotation.

*inter*Media Annotator can parse media sources accessible through the URL protocols shown in Table 3–1.

Table 3–1 Available URL Protocols

URL Protocol	Description
`file`	Access all the files on local or remotely mounted disks in your computer.
`http`	Access media available through an Internet Web server.
`cd`	Access audio compact discs in your local CDROM drive.

Note: The URL used to extract annotations from a compact disc is not a standard URL. The URL is defined as follows:

- Windows NT: `cd:<your Windows CD-ROM drive>#cdda`
- Macintosh: `cd:#cdda`

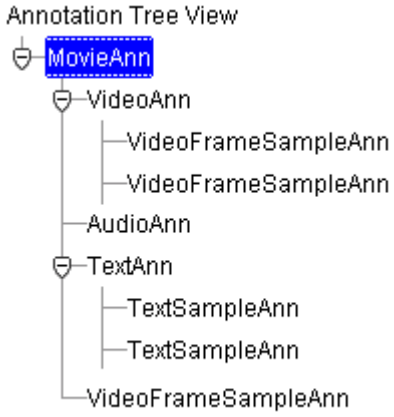
If you are parsing a local file or a file available over the Internet through the `http` protocol, *inter*Media Annotator extracts the time-independent attributes from the media file and inserts them into a logical annotation.

If you are parsing an audio CD, *inter*Media Annotator can connect to a CDDB, find the entry corresponding to your CD, and create a logical annotation.

If you are parsing a media source with multiple tracks, such as a video source or audio CD, an annotation is created for each track.

When the parsing is complete, the annotation type appears in the Annotation Tree View of the **Annotations** pane. The attribute names and values for the currently selected annotation are displayed in the **Attributes** tab of the ***inter*Media Annotator** window (Figure 2–1). The **Annotations** pane contains an expandable list, which shows the hierarchy of annotations and sub-annotations (Figure 3–3).

Figure 3–3 Annotations Pane with Expanded List



In order to display the attributes of another annotation, select it in the Annotation Tree View.

Some JPEG files contain additional metadata in the Information Interchange Model (IIM) format. This metadata can optionally be extracted into an IpctIimAnn sub-annotation, which appears as a sub-annotation of your main annotation. In order to create an IpctIimAnn sub-annotation, perform the following operations:

1. From the **Edit** menu, select **Preferences** and click the **Parsers** tab.

   The **Parsers** tab of the **Preferences** window appears (Figure 2–7).

2. Click the plus sign (**+**) next to the JpgParser option.
3. Select **IPTC-IIM** to create the IpctIimAnn sub-annotation.
4. Click **OK** to confirm and save the changes.

3.2 Editing Attribute Values

You can edit the value that appears for each attribute by performing the following operations:

1. Double-click the text in the right-hand column of the attribute.

   A solid outline will appear around the table cell and an I-beam cursor will appear.

2. Edit the text.

Note: *inter*Media Annotator cannot change the attribute values in the media itself; it can change only the attribute values in the extracted annotation. If you parse the media file again, your annotation will be overwritten and any attributes that you have edited will revert to their original values.

You can save your changes to the annotation. See Section 3.6 for more information.

3.3 Adding and Deleting Attributes to the Annotation

*inter*Media Annotator defines a given number of attributes (see Appendix C for a complete list of attributes). However, not all media sources will provide values for every attribute. You can use *inter*Media Annotator to add a value to your annotation for any attribute that does not have a value.

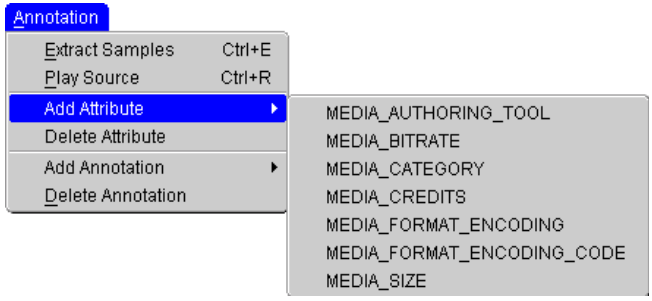
*inter*Media Annotator cannot write any new attribute values back to the media sources. The current annotation will contain the value, but any annotations created later by *inter*Media Annotator will not contain the new value.

To add a value for an attribute that has not been automatically set, perform the following operations:

1. From the **Annotation** menu, select **Add Attribute**.

   The **Add Attribute** submenu appears (Figure 3–4), listing the attributes that have no values.

Figure 3–4 Add Attribute Submenu



2. Select an attribute from the **Add Attribute** submenu.

The new attribute appears in the **Attributes** tab with no value in the right-hand column.

3. Enter a value in the right-hand column.

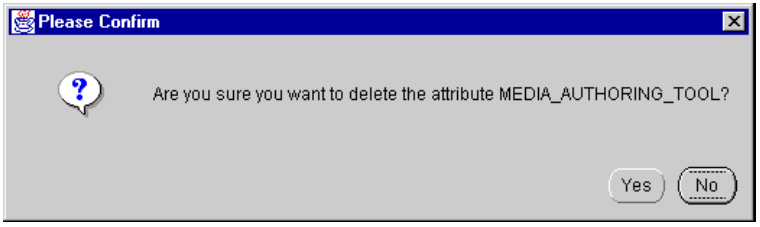
   Validation of the annotation type is done at this time.

To delete an attribute from your annotation, perform the following operations:

1. Select the attribute to delete.
2. From the **Annotation** menu, select **Delete Attribute**.

   The **Please Confirm** window opens (Figure 3–5).

Figure 3–5 Please Confirm Window for Deleting Attributes



3. Click **Yes**.

The attribute and its value are deleted from the annotation.

3.4 Adding and Deleting an Annotation

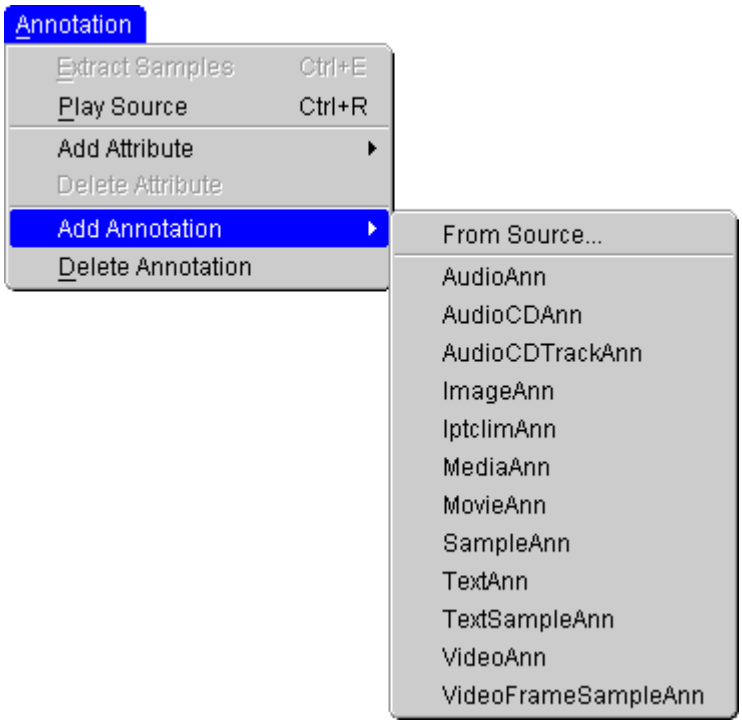
An annotation will usually contain one or more sub-annotations, which contain the metadata associated with a portion of the media source, such as a text track or an audio track. In addition to these populated sub-annotations, you can create your own sub-annotations by adding an empty annotation and then populating it with your own values.

To create an empty annotation, perform the following operations:

1. In the Annotations pane, select the annotation under which you will add a sub-annotation.
2. From the **Attribute** menu, select **Add Annotation**.

   The **Add Annotation** submenu appears (Figure 3–6).

Figure 3–6 Add Annotation Submenu



3. Select the type of annotation to create.

   Your new annotation appears in the Annotation Tree View.

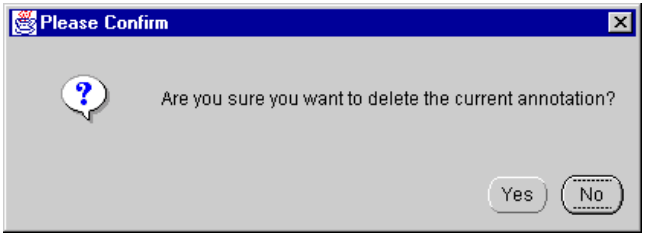
You can populate an empty annotation with attribute values in the same way you would add attribute values to a populated annotation. See Section 3.3 for more information.

You can delete annotations and sub-annotations. You can delete sub-annotations that you have created and sub-annotations that were created automatically in the parsing process. To delete an annotation, perform the following operations:

1. Select the annotation or sub-annotation to be deleted.
2. From the **Attribute** menu, select **Delete Annotation**.

   The **Please Confirm** window opens (Figure 3–7).

Figure 3–7 Please Confirm Window for Deleting Annotations



3. Click **Yes**.

   The selected annotation is removed from the Annotation Tree View.

3.5 Extracting a Sample

You can use *inter*Media Annotator to extract three types of samples: text selections (or **tracks**) from a QuickTime movie file, images from a QuickTime movie file, or audio tracks from a compact disc.

3.5.1 Text Tracks from a QuickTime Movie

To extract a text track from a QuickTime movie, perform the following operations:

1. Annotate a QuickTime movie that contains a text track, such as `oow_annotator.mov` (included with *inter*Media Annotator). See Section 3.1 for more information on creating an annotation.
2. Select **TextAnn** in the left-hand window pane.
3. Either select **Extract Samples** from the **Annotation** menu or click the **Extract Samples** button on the toolbar.

View the extracted text information by clicking the **Samples** tab when the text annotation is selected (Figure 3–8).

Figure 3–8 Samples Tab with Audio Samples

3.5.2 Video Tracks from a QuickTime Movie

To extract a video track from a QuickTime movie, perform the following operations:

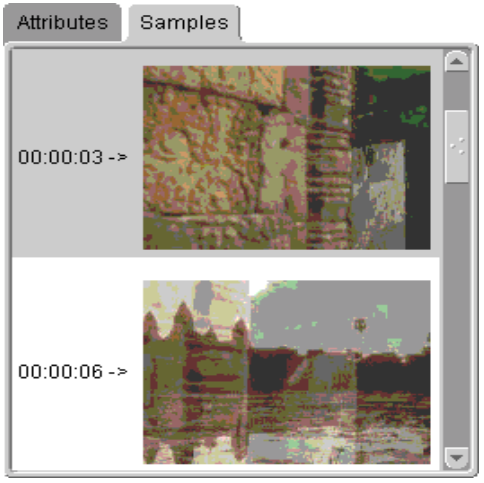
1. Annotate a QuickTime movie. See Section 3.1 for more information on creating an annotation.
2. Select **VideoAnn** in the left-hand window pane.
3. Set the amount of video data to be extracted by performing the following steps:
   a. From the **Edit** menu, select **Preferences** and click the **Parsers** tab.

      The **Parsers** tab of the **Preferences** window appears (Figure 2–7).

   b. Click the plus sign (**+**) next to the QT4JavaParser option.
   c. In the list that appears, select **ExtractVideoSamples**.
   d. If you want to specify the time interval to pass between frame extractions, then enter *true* in the Extraction by Time Interval field and enter the interval (in seconds) in the Extraction Parameter field.
   e. If you want to specify the number of frames to be extracted, then enter *false* in the Extraction by Time Interval field and enter the number of frames to be extracted in the Extraction Parameter field.
   f. Click **OK** to confirm and save the changes.
4. Either select **Extract Samples** from the **Annotation** menu or click the **Extract Samples** button on the toolbar.

View the extracted video information by clicking the **Samples** tab when the video annotation is selected (Figure 3–9).

Figure 3–9 Samples Tab with Video Samples



3.5.3 Audio Tracks from a CD

*inter*Media Annotator can extract audio data from a CD track. You can specify the start and end points of your extracted sample.

To extract samples from a CD, perform the following operations:

1. Annotate an audio CD. See Section 3.1 for more information on creating an annotation.
2. Select an audio CD track annotation in the Annotations pane.
3. Set the amount of audio data to be extracted by performing the following steps:
   a. From the **Edit** menu, select **Preferences** and click the **Parsers** tab.

      The **Parsers** tab of the **Preferences** window appears (Figure 2–7).
   b. Click the plus sign (**+**) next to the Audio CD Parser option.
   c. In the list that appears, select **ExtractAudioClipTrack**.
   d. In the Audio Sample Start Time field, enter the time in seconds from which you want to start extracting.
   e. In the Audio Sample Length field, enter the duration of the sample to be extracted, in seconds.
   f. Click **OK** to confirm and save the changes.

4. Click the **Extract Media Samples** button on the toolbar.

The audio sample is extracted and stored in the Sun AU sound file format.

After extraction, media source attributes will be modified in order to refer to the extracted sample. These attributes include file format, MIME type, file name, directory, and URL, among others.

See Section 3.8 for more information on playing back the extracted sample.

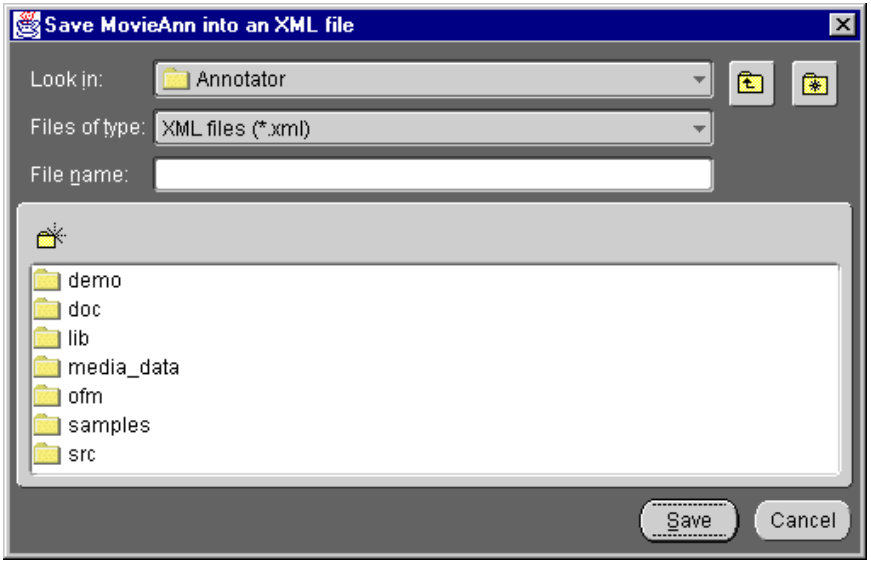
3.6 Saving an Annotation

Once you have parsed a media source, you can save the annotation as an XML document by performing the following operations:

1. From the **File** menu, select **Save**.

   The **Save** window opens (Figure 3–10).

Figure 3–10 Save Window



2. Enter a file name in the File name: field and click **Save**.

Note: *inter*Media Annotator does not automatically append the suffix *.xml* to the provided file name. You must manually enter the suffix.

The XML document can be viewed through any text editor or through *inter*Media Annotator. See Section 3.7 for more information.

The default folder where *inter*Media Annotator will save annotations is set in the **General** tab of the **Preferences** window. To change the default folder, perform the following operations:

1. From the **Edit** menu, select **Preferences** and click the **General** tab. The **General** tab of the **Preferences** window appears (Figure 2–6).
2. Enter the name of the new default folder in the Open / Save Directory: field.
3. Click **OK** to confirm and save the changes.

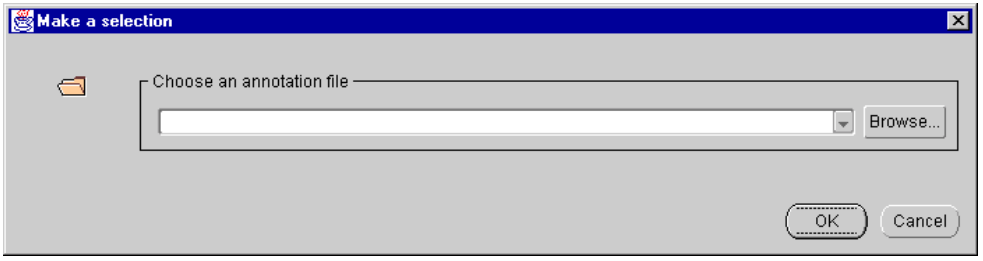
3.7 Opening a Saved Annotation

If you have a saved annotation, you can open it in *inter*Media Annotator by performing the following operations:

1. Either click the **Open** button on the toolbar or select **Open** from the **File** menu.

   The **Make a Selection** window opens (Figure 3–11).

Figure 3–11 Make a Selection Window



2. Click the **Browse** button.

   The **Open** window appears (Figure 3–2).
3. Select an annotation file and click **Open**.

The annotation and its sub-annotations appear in the Annotation Tree View.

3.8 Playing Media Sources or Viewing Extracted Samples

You can use *inter*Media Annotator to play your media source and any text or video sample that you have extracted.

Before playing a media source or sample, check the **Mime-Types** tab of the **Preferences** window to ensure that each MIME type is paired with the correct path to the appropriate helper application. See Section 2.2.5 for more information.

3.8.1 Media Source

Play a media source or an extracted sample by performing the following operations:

1. Select the annotation at the root of the Annotation Tree View.
2. Either click the **Play Source** button on the **Annotation** toolbar or select **Play Source** from the **Annotation** menu.

   The appropriate media player opens and plays the media source.

3.8.2 Media Sample

Play an extracted media sample by performing the following operations:

1. Extract a media sample. See Section 3.5 for more information.
2. Select the sub-annotation associated with the sample you want to play.
3. Either click the **Play Source** button on the **Annotation** toolbar or select **Play Source** from the **Annotation** menu.

   The appropriate media player opens and plays the media source.

3.8.3 Text Sample

You can view a text sample without opening a separate viewer.

After extracting the text sample, click the **Samples** tab in the right-hand window pane. The text sample appears, along with a time line indicating roughly where each piece of text appears in the video or song.

4

PL/SQL Template Wizard

*inter*Media Annotator can upload media data and an associated annotation into an Oracle database where Oracle *inter*Media has been installed. It does so through an Oracle PL/SQL Upload Template, which contains both PL/SQL calls and Annotator-specific keywords.

You create your own PL/SQL Upload Templates. Novice users can use the PL/SQL Template Wizard, which is a graphical user interface that progresses through each step of PL/SQL Upload Template creation.

Before proceeding, read Steps 1 and 2 of Section 2.2.1 in order to configure the preferences for your database connection.

4.1 Using the PL/SQL Template Wizard to Generate Upload Templates

Users with limited PL/SQL experience should use the PL/SQL Template Wizard to generate a PL/SQL Upload Template. The PL/SQL Template Wizard takes you through a step-by-step process to create a PL/SQL Upload Template using a graphical user interface; you do not need to understand the Annotator-specific keywords involved.

To start the PL/SQL Template Wizard, perform one of the following operations:

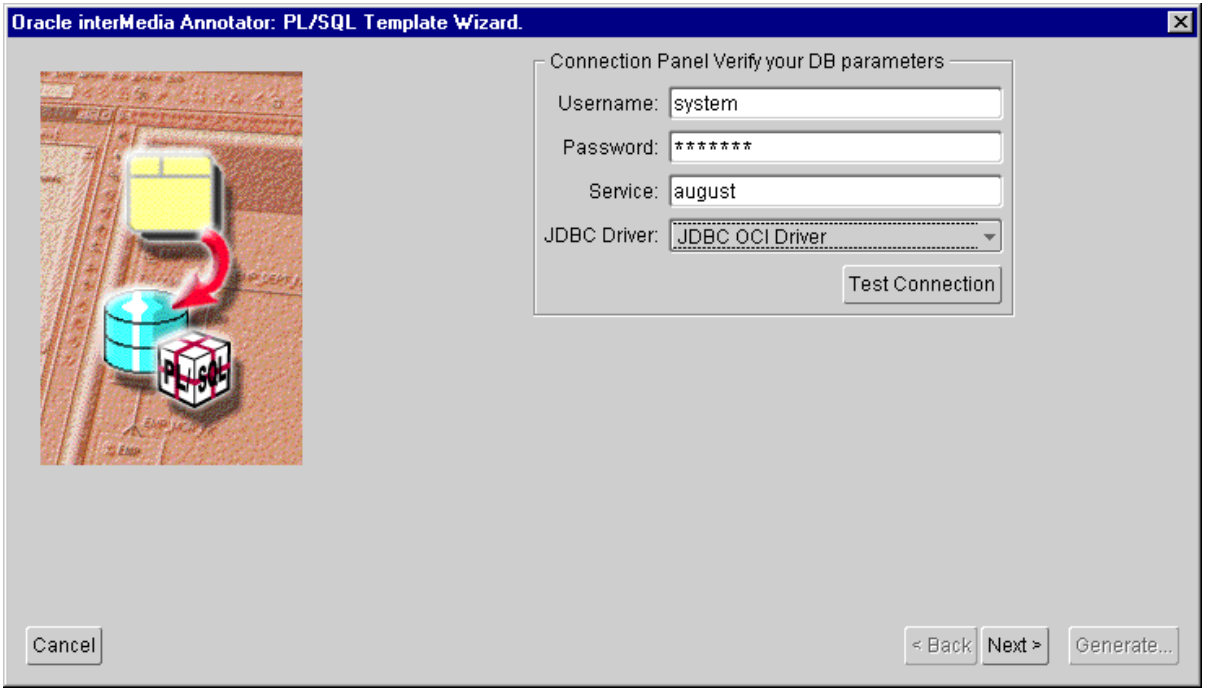
- Click the **Generate Upload Template** button on the Oracle toolbar (Figure 2–4).
- From the **Database** menu, select **PL/SQL Template Wizard**.

The following sections go through each window of the PL/SQL Template Wizard.

4.1.1 Connection Panel Window

The **Connection Panel** window (Figure 4–1) asks you to specify your database parameters.

Figure 4–1 PL/SQL Template Wizard Connection Panel Window

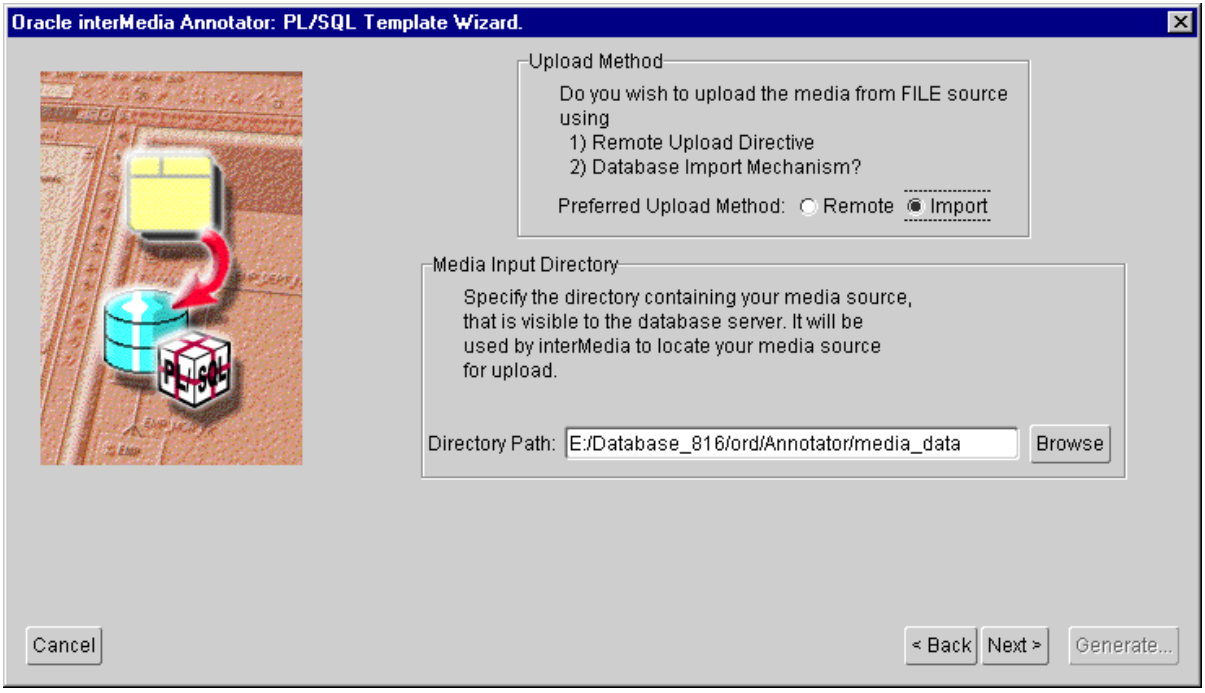


This window contains the fields and values from the **Database** tab of the **Preferences** window. Set the database parameters of the database to which you will upload your annotation if you have not already done so.

4.1.2 Upload Method Window

The **Upload Method** window (Figure 4–2) asks you to specify a method to upload the media source.

Figure 4–2 PL/SQL Template Wizard Upload Method Window



There are two different methods that Annotator can use to upload the media source and annotation to your database: import and remote. In the import upload method, the media source must be visible to the database server (either in a file system or through an HTTP stream), and the media source will be loaded directly from the file system to the database. In the remote upload method, the media source does not have to be visible to the database; the file is loaded into Annotator, and Annotator loads the file into the database through JDBC calls.

The import upload method uses the Oracle *inter*Media import( ) method, while the remote upload method uses the ${MANN_UPLOAD_SRC} Annotator-specific keyword.

If you select **Import** and the media source is in a file system, you must specify the path to the directory where the media file resides. The directory path should be specified from the point of view of the Oracle database server to which you are uploading. For example, if you are running *inter*Media Annotator on Windows NT and you want to upload data to an Oracle database that is running on a UNIX platform, the media data must reside in a directory that can be accessed by both

machines. You can do this by mounting a UNIX directory on the server into a Windows NT network drive. *Prior* to entering the PL/SQL Template Wizard, you would refer to the media file using the mapped Windows drive/directory name. In the PL/SQL Template Wizard, however, you must specify the directory using the UNIX directory name that Oracle will use to access the media.

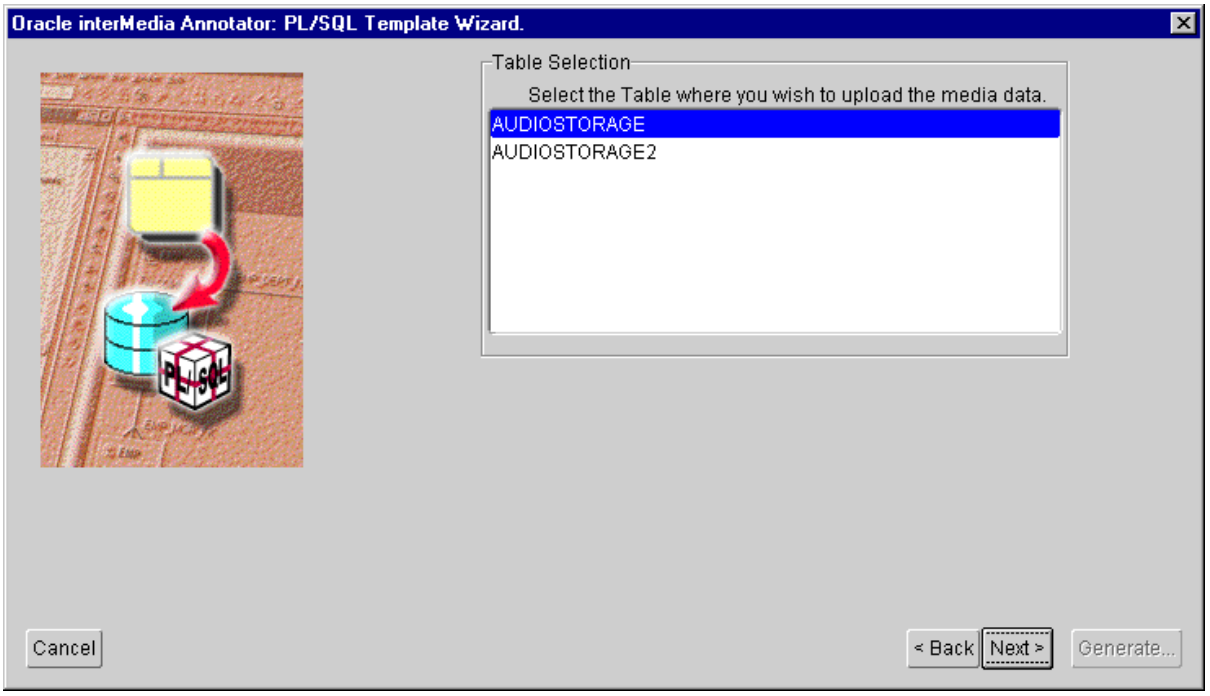
If you select **Import** and the media source is an HTTP stream, you can choose either to import the media data into the database, or to store the URL in the database.

If you select **Remote** and you are using the JDBC Thin driver, your upload performance may be poor, especially if you are uploading large files.

4.1.3 Table Selection Window

The **Table Selection** window (Figure 4–3) asks you to choose the table into which the *inter*Media object will be uploaded.

Figure 4–3 PL/SQL Template Wizard Table Selection Window

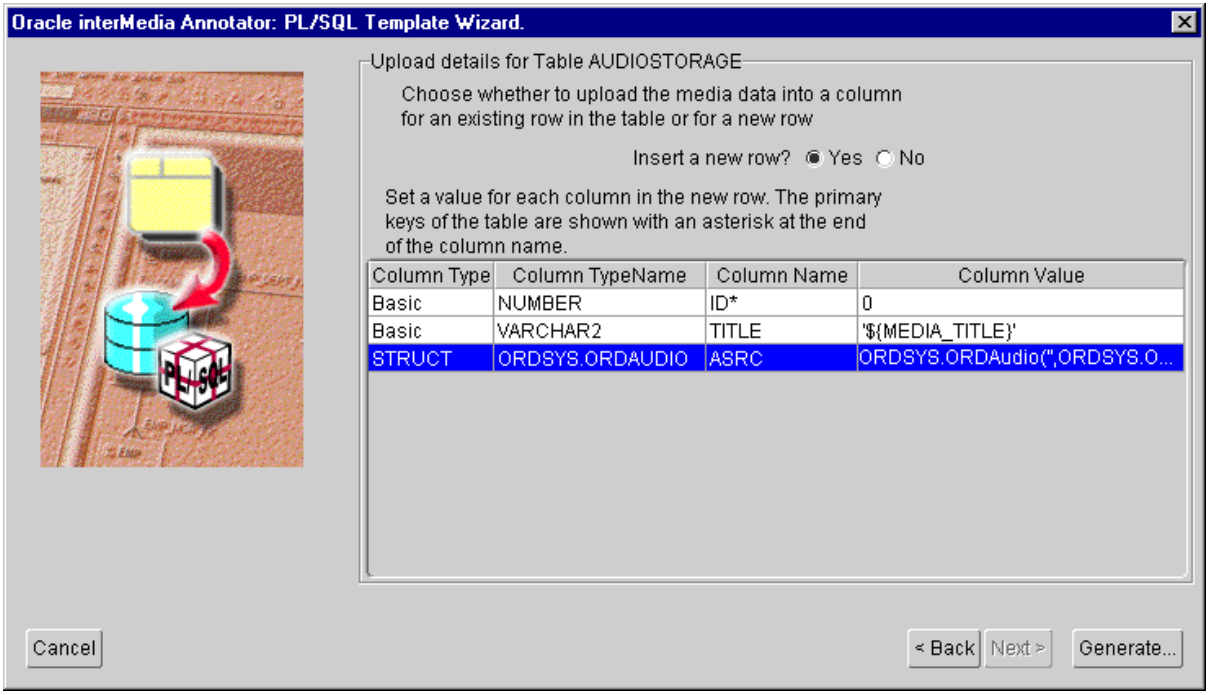


This table must have at least one column of the appropriate *inter*Media object type (ORDSYS.ORDAudio for annotations for audio files or songs from a CD, ORDSYS.ORDImage for annotations for image files, or ORDSYS.ORDVideo for annotations for video files) in order to proceed. The PL/SQL Template Wizard will check for this condition and notify you if it is not met.

4.1.4 Upload Details Window

The **Upload Details** window (Figure 4–4) asks if you want to upload the object into an existing row of the table or into a new row.

Figure 4–4 PL/SQL Template Wizard Upload Details Window



If you choose **No** (that is, not to insert a new row), you will be taken to the **Row Selection** window. See Section 4.1.6 for more information.

If you choose **Yes** (that is, to insert a new row), you have to specify a value for each table column. This is necessary for the primary keys, which will be marked with an asterisk. For the remaining columns, you are given several possible input methods:

- Use the default values provided by *inter*Media Annotator.
- Manually fill in the values, observing the usual SQL syntax according to the column type.
- From the pull-down menu, choose an attribute of the annotation that you are inserting. *inter*Media Annotator will insert the value of this attribute into the new row. This option is recommended if you want to build an index on a specific column that represents an annotation attribute.

Figure 4–4 shows the ID column (primary key) having its value manually typed in, the ASRC column (ORDSYS.ORDAudio) using the default value provided by *inter*Media Annotator, and the TITLE column being mapped to the annotation's MEDIA_TITLE attribute.

If your table has multiple columns containing *inter*Media objects, clicking the **Next** button will take you to the **Column Selection** window. See Section 4.1.5 for more information.

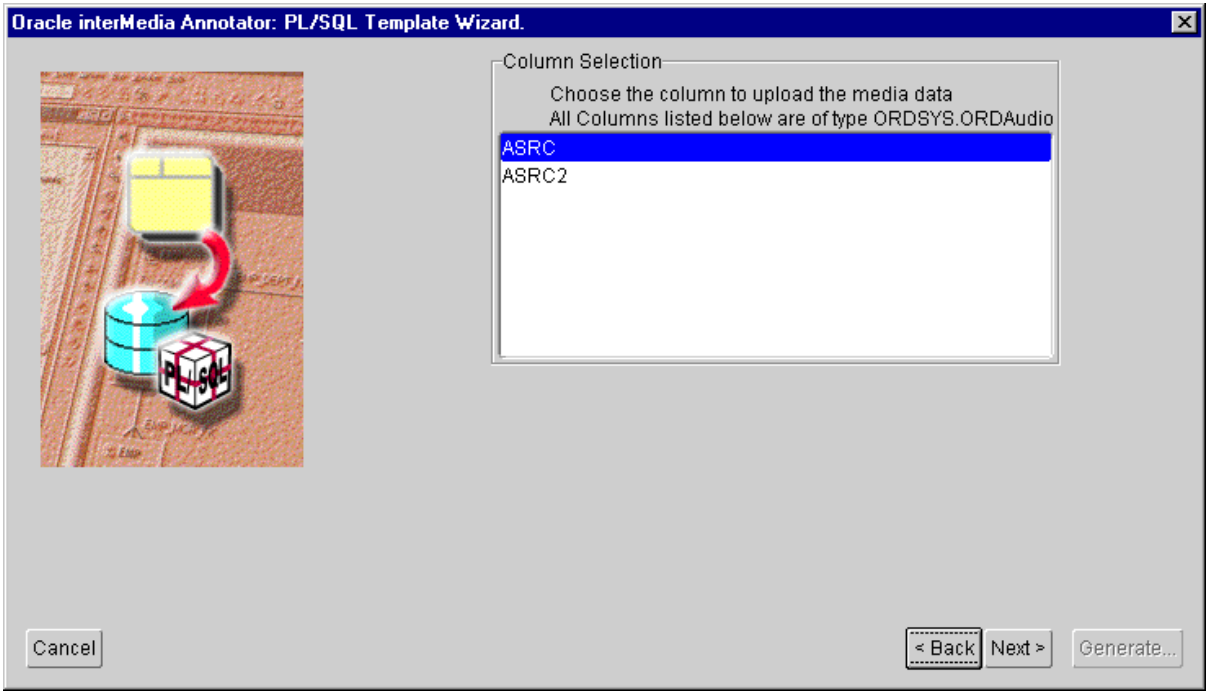
If you are uploading to an existing row, clicking the **Next** button will take you to the **Row Selection** window. See Section 4.1.6 for more information.

If your table has only one column of *inter*Media objects and you are uploading to a new row, the **Generate** button will be enabled. See Section 4.1.7 for more information.

4.1.5 Column Selection Window

The media data and the annotation in XML form will be uploaded only to a table that contains an *inter*Media object whose type can be mapped to the annotation. If your table contains two or more such columns, the **Column Selection** window (Figure 4–5) will prompt you to select the column into which the *inter*Media object will be uploaded.

Figure 4–5 PL/SQL Template Wizard Column Selection Window



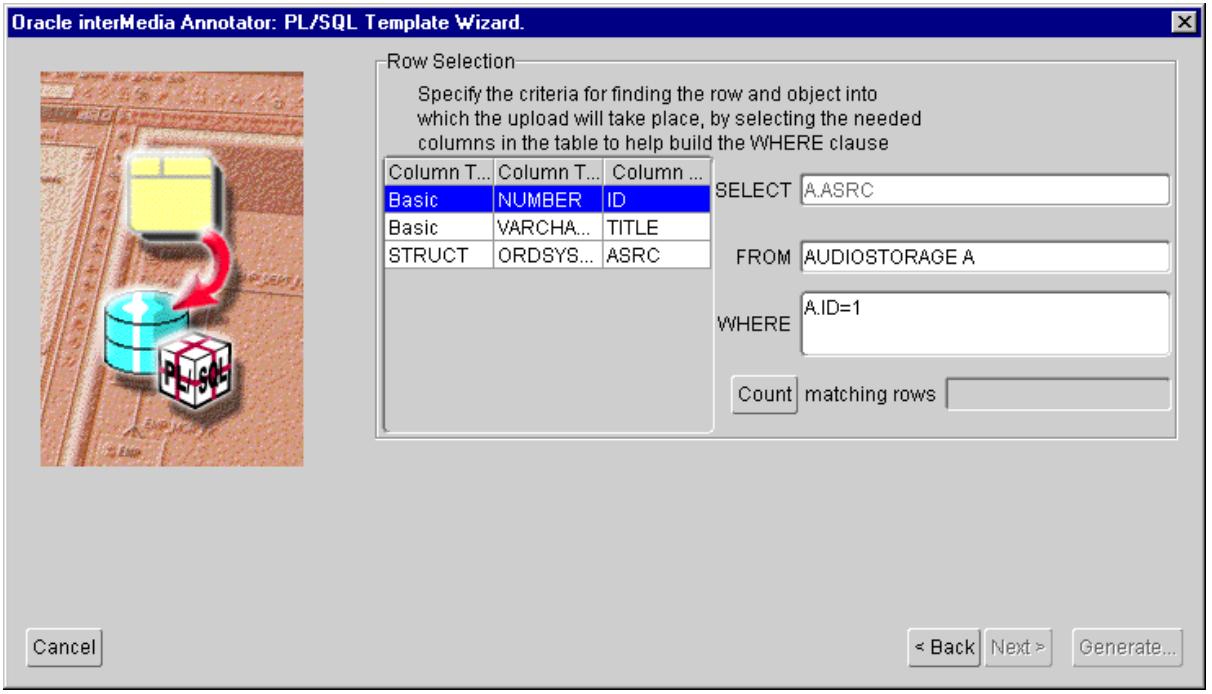
If you previously chose to insert a *new* row, the **Generate** button is enabled after you select a column. See Section 4.1.7 for more information.

If you previously chose to upload to an existing row, the **Next** button will take you to the **Row Selection** window. See Section 4.1.6 for more information.

4.1.6 Row Selection Window

The **Row Selection** window (Figure 4–6) asks you to enter the criteria to be used for querying a row that contains the column into which the *inter*Media object will be uploaded.

Figure 4–6 PL/SQL Template Wizard Row Selection Window



The table name was specified earlier, so you need to specify the WHERE clause of a SQL SELECT statement that will return the *inter*Media destination object. You can use the **Count** button to ensure that only one row is returned.

Once you complete this step, the **Generate** button is enabled. See Section 4.1.7 for more information.

4.1.7 Generate Button

When you have entered all the necessary information, the **Generate** button will become active. Click the **Generate** button to close the PL/SQL Template Wizard and generate your PL/SQL Upload Template. Possible SQL errors will be reported through the **Console** window.

Your PL/SQL Upload Template will be saved to a default directory that is defined in the **Preferences** window. Initially, the directory is set to <*ORACLE_HOME*>\ord\Annotator\ofm. To change this directory, perform the following operations:

1. From the **Edit** menu, select **Preferences** and click the **General** tab. The **General** tab of the **Preferences** window appears (Figure 2–6).
2. Perform only *one* of the following operations:
   - Enter the path to the new directory in the PL/SQL Template Directory field.
   - Click the **Browse** button next to the PL/SQL Template Directory and select the new directory in the dialog box.
3. Click **OK** to apply and save the changes.

See Section 4.2 for information on how to run a PL/SQL Upload Template.

4.2 Using Upload Templates to Upload to the Database

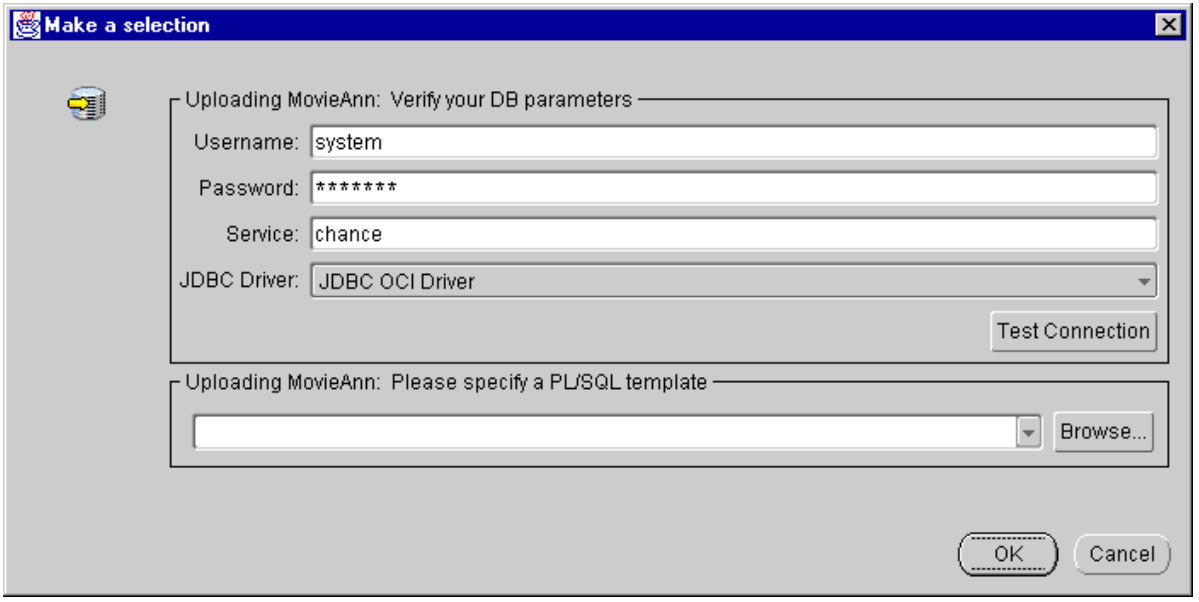
If you use the PL/SQL Template Wizard or a text editor to create a new PL/SQL Upload Template, or if you use one of the files provided with *inter*Media Annotator, you have to run the PL/SQL Upload Template from within *inter*Media Annotator in order to actually upload your media source and annotation to your database.

To run a PL/SQL Upload Template, perform the following operations:

1. Perform only *one* of the following operations:
   - Click the **Upload to Oracle** button on the Oracle toolbar (Figure 2–4).
   - From the **Database** menu, select **Upload**.

The **Make a Selection** window opens (Figure 4–7).

Figure 4–7 Make a Selection Window



2. Perform only *one* of the following operations:
   - Enter the path to the PL/SQL Upload Template in the appropriate field.
   - Click the **Browse** button, select the PL/SQL Upload Template in the dialog box, and click **Open**.
3. Click **OK** to run the PL/SQL Upload Template.

Errors encountered during the upload process will appear in the **Console** window.

4.3 Editing Existing PL/SQL Upload Templates

You cannot use the PL/SQL Template Wizard to edit an existing PL/SQL Upload Template; it can create only new PL/SQL Upload Templates. To edit a PL/SQL Upload Template that you created with the PL/SQL Template Wizard, you must use a text editor. See Chapter 7 for more information on writing and editing PL/SQL Upload Templates.

Part II

*inter*Media Annotator Java-Based Engine

Part II discusses the *inter*Media Annotator Java-based engine and contains the following chapters:

- Chapter 5, "interMedia Annotator Engine Example"
- Chapter 6, "Annotator Engine API Reference Information"
- Chapter 7, "Creating PL/SQL Upload Templates"

5

*inter*Media Annotator Engine Example

This chapter provides a description of SimpleAnnotator.java, which is an example of a user-developed application that was written using the *inter*Media Annotator engine APIs for use in a synchronous environment.

You can find the source code at the following location:

`<ORACLE_HOME>/ord/Annotator/demo/examples/src/SimpleAnnotator.java`

The code that appears in this chapter will not necessarily match the code shipped as SimpleAnnotator.java. If you want to run this example on your system, use the file provided with the *inter*Media Annotator installation; do not attempt to compile and run the code presented in this chapter.

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

The Annotator client example in SimpleAnnotator.java contains user-defined methods that use Java and *inter*Media Annotator APIs to perform the following operations:

- Initialize an instance of the Annotator client
- Set preferences
- Parse a media source file
- Get and set attributes of the annotation
- Define sub-annotations

- Extract media samples from the media source file
- Upload the annotation to an Oracle database

5.1 Import Statements

Example 5–1 shows the import statements that must be included to properly run an Annotator client.

Example 5–1 Import Statements

```
import java.net.*;
import java.io.*;
import java.util.*;
import java.text.*;
import java.sql.*;

import oracle.ord.media.annotator.handlers.annotation.*;
import oracle.ord.media.annotator.annotations.Attribute;
import oracle.ord.media.annotator.annotations.Annotation;
import oracle.ord.media.annotator.listeners.*;
import oracle.ord.media.annotator.handlers.*;
import oracle.ord.media.annotator.handlers.db.*;
import oracle.ord.media.annotator.utils.*;
import oracle.ord.media.annotator.AnnotatorException;
```

5.2 Class Definition and Instance Variables

Example 5–2 shows the class definition and instance variables for the sample Annotator client.

Example 5–2 Class Definition and Instance Variables

```
public class SimpleAnnotator implements AnnListener, OutputListener{
     private Status m_st;
     private AnnotationHandler m_ah;
```

An Annotator client must implement the AnnListener interface to have access to the callback methods used for Annotator engine operations. See Section 6.7 for more information.

An Annotator client must implement OutputListener to get access to traces from the engine. See Section 6.8 for more information.

The class contains two instance variables:

- m_st is the Status object that will be used to update the status in the GUI of the application. See Section 6.10 for more information.
- m_ah is the AnnotationHandler object that will actually produce the annotation for the given content source. See Section 6.5 for more information.

5.3 main( ) Method

Example 5–3 shows the main( ) method.

Example 5–3 main( ) Method (SimpleAnnotator)

```
public static void main(String[ ] args){
     [1] if(args.length == 0){
          System.err.println("Usage: java SimpleAnnotator mediaURL");
          System.err.println("mediaURL: URL of the media you want to parse
               (for example, file:/myjpeg.jpg"));
          return;
     }

     [2] SimpleAnnotator sa = new SimpleAnnotator( );
     [3] sa.init( );

     [4] String szURL = args[0];
     [5] sa.parse(szURL);
}
```

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

1. Tests to make sure the URL of the media source file to be parsed was passed as an argument. If there are no arguments, an error message is printed.

   The URL of the media source file to be operated upon should be the first argument.

2. Creates an empty instance of SimpleAnnotator.
3. Calls the init( ) method to initialize the newly created SimpleAnnotator instance. See Section 5.4 for more information on the init( ) method.

   Once the SimpleAnnotator instance is initialized, the client can invoke the Annotator engine operations, such as parsing, extraction, and insertion.

4. Creates a String named szURL and sets its value to the first argument (that is, the URL of the media source file).
5. Calls the parse( ) method to parse the media source file.

   See Section 5.5 for more information on the parse( ) method.

5.4 init( ) Method

Example 5–4 shows the contents of the init( ) method, which initializes the Annotator client. This method is specific to this example.

Example 5–4 init( ) Method

```
public void init( ){
     [1] report("Initializing Annotator Engine...");

     [2] Status.initStatus(this);
     [3] m_st = Status.getStatus( );
     [4] m_st.SetOutputMode(Status.OUTPUT_MODE_VERBOSE);

     [5] try {
          m_ah = new AnnotationHandler(AnnotationHandler.OP_MODE_SYNCH);
     }
     [6] catch(Exception e) {
          report("Initializing... Failed.");
          reportError(e);
     }

     [7] Preferences prefs = Preferences.getPrefs( );
     [8] prefs.setProperty(SZ_CONN_PASSWORD, "mypassword");

     [9] report("Initializing Annotator Engine... Done");
}
```

The code in the init( ) method performs the following actions:

1. Prints a message that the initialization is beginning. See Section 5.12 for more information on the report(String) method.
2. Initializes the Status object. Because SimpleAnnotator implements the OutputListener class, the current instance of SimpleAnnotator can receive the status messages. See initStatus( ) in Section 6.10 for more information.
3. Sets the initialized Status object to the m_st instance variable.

4. Sets the status output mode to VERBOSE.
5. Creates a new AnnotationHandler instance in synchronous mode and sets it to the m_ah instance variable.
6. Catches any exceptions that were raised in the previous step and prints the error to the screen with the report( ) and reportError( ) methods. See Section 5.14 for more information on the reportError( ) method.

   If the Status and AnnotationHandler objects were both created with no errors, you will be able to set any necessary preferences.
7. Creates a Preferences object and initialize it.
8. Sets a new preference named SZ_CONN_PASSWORD.
9. Prints a message that initialization was successful.

5.5 parse( ) Method

Example 5–5 shows the contents of the parse( ) method. This method is specific to this example.

Example 5–5 parse( ) Method

```
public void parse(String szURL){
     if(m_ah != null){
          AnnTaskMonitor atm = m_ah.parseMedia(szURL, this);
     }
}
```

The code in the parse( ) method calls the AnnotationHandler.parseMedia( ) method with the URL and the AnnotationListener as parameters. Because SimpleAnnotator implements AnnListener, the current instance of SimpleAnnotator can be used.

See Section 6.5 for more information about the parseMedia( ) method.

AnnotationHandler.parseMedia( ) is called only if the annotation handler has been initialized properly in the init( ) method.

When AnnotationHandler.parseMedia( ) has finished parsing the source file and building the annotation, it calls the AnnListener.parsePerformed( ) call-back function. This method is overridden in the SimpleAnnotator class, so the actual method called is SimpleAnnotator.parsePerformed( ). See Section 5.6 for more information.

5.6 parsePerformed( ) Method

Example 5–6 shows the parsePerformed( ) method. Because your application must implement AnnListener, this method is required.

Example 5–6 parsePerformed( ) Method

```
public void parsePerformed(Annotation ann){
     [1] if(ann != null){
          [2] String szMimeType = (String) ann.getAttribute
               ("MEDIA_SOURCE_MIME_TYPE");
          [3] Enumeration eAttrs = ann.getAttributes( );
          while(eAttrs.hasMoreElements( )){
               [4] Attribute attr = (Attribute)eAttrs.nextElement( );
               Object oAttrValue = attr.getValue( );
          }
          [5] Enumeration eSubAnns = ann.getSubAnnotations( );
          while (eSubAnns.hasMoreElements( )){
               [6] Annotation subAnn = (Annotation)eSubAnns.nextElement( );
          }
          /**
           * Example: (Advanced)
           */
          try {
               [7] Annotation inventoryAnn = m_ah.createAnnotationByName
                    ("InventoryAnn");
               [8] ann.addSubAnnotation(inventoryAnn);
               [9] inventoryAnn.setAttribute("SALES_PRICE", new Float(19.99));
          }
          [10] catch (AnnotatorException ae){
               errorOccured(ann, ae);
               return;
          }
          [11] report(ann);
     }
     [12] if(m_ah.isExtractable(ann)){
          [13] m_ah.extractMedia(ann);
     }
}
```

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

1. Executes the code in the next block only if a valid annotation was passed by the caller.

If the caller did pass a valid annotation, you would typically use the parsePerformed( ) method to manipulate the annotation before it is uploaded to the database. Steps 2 through 11 show examples of the kinds of operations you may perform. The tasks in steps 7 through 10 should be used only by advanced programmers.

2. Gets the value of the MEDIA_SOURCE_MIME_TYPE attribute of the annotation and casts it into a String object.
3. Gets a list of all attributes that have a valid value and stores their names in an Enumeration object.
4. Accesses the values stored in the Enumeration.
5. Gets all sub-annotations of the annotation and stores them in an Enumeration object.
6. Accesses the values stored in the Enumeration.
7. Creates an empty annotation named inventoryAnn.
8. Adds inventoryAnn to ann as a sub-annotation.
9. Sets the SALES_PRICE attribute in inventoryAnn to 19.99.
10. Catches any errors raised in steps 7 through 9 and reports them with the errorOccured( ) method. See Section 5.10 for more information on the errorOccured( ) method.
11. Uses the report(Annotation) method to print the annotation as an XML file. See Section 5.13 for more information.
12. Checks to see if it is possible to extract samples from the annotation or any of its subannotations. If it is possible, the code in step 13 is executed. If not, the code in step 13 is skipped.
13. Extracts the media samples from the annotation.

    When AnnotationHandler.extractMedia( ) has finished, it calls the call-back function AnnListener.extractionPerformed( ). This method is overridden in the SimpleAnnotator class, so the actual method called is SimpleAnnotator.extractionPerformed( ). See Section 5.7 for more information.

5.7 extractionPerformed( ) Method

Example 5–7 shows the extractionPerformed( ) method. Because your application must implement AnnListener, this method is required.

Example 5–7 extractionPerformed( ) Method

```
public void extractionPerformed(Annotation ann){
     [1] report(ann);
     [2] OrdFileMapping ofm = new OrdFileMapping("e:\\mylogic.ofm");
     [3] m_ah.insertMedia(ann, ofm, this);
}
```

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

1. Uses the report(Annotation) method to print the annotation as an XML file. See Section 5.13 for more information.
2. Creates a new OrdFileMapping object based on the mapping file located at "e:\\mylogic.ofm."
3. Uploads the annotation to the database, using the OrdFileMapping object to map the contents of the annotation to the proper locations on the database.

   Alternatively, you could specify a Connection object that represents the JDBC connection to be used in the upload process. The same JDBC connection can be used for multiple upload operations. See Section 6.5, "insertMedia(Annotation,OrdMapping,AnnListener,Connection)" for more information.

   When AnnotationHandler.insertMedia( ) has finished, it calls the call-back function AnnListener.insertionPerformed( ). This method is overridden in the SimpleAnnotator class, so the actual method called is SimpleAnnotator.insertionPerformed( ). See Section 5.8 for more information.

5.8 insertionPerformed( ) Method

Example 5–8 shows the insertionPerformed( ) method. Because your application must implement AnnListener, this method is required.

Example 5–8 insertionPerformed( ) Method

```
public void insertionPerformed(Annotation ann, Connection conn){
     try {
          [1] conn.commit( );
          [2] conn.close( );
     }
     [3] catch (SQLException sqle){
          errorOccured(ann, sqle);
     }
};
```

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

1. Commits all changes made to the database.
2. Closes the connection to the database.

   Instead of closing the connection, the client could reuse the connection by passing it to another AnnotationHandler call.
3. Catches any errors raised in steps 1 and 2 and reports them with the errorOccured( ) method. See Section 5.10 for more information on the errorOccured( ) method.

5.9 warningOccured( ) Method

Example 5–9 shows the warningOccured( ) method. Because your application must implement AnnListener, this method is required.

Example 5–9 warningOccured( ) Method

```
public void warningOccured(Annotation ann, Exception e){
     reportError(e);
}
```

The code in the warningOccured( ) method implements the AnnListener.warningOccured( ) method. This method uses the reportError( ) method to capture the warning and report it. If a warning occurs and this method is called, the Annotator engine continues to operate.

See Section 5.15 for more information on the reportError( ) method.

5.10 errorOccured( ) Method

Example 5–10 shows the errorOccured( ) method. Because your application must implement AnnListener, this method is required.

Example 5–10 errorOccured( ) Method

```
public void errorOccured(Annotation ann, Exception e){
     reportError(e);
}
```

The code in the errorOccured( ) method implements the AnnListener.errorOccured( ) method. This method uses the reportError( ) method to capture the error and report it. If an error occurs and this method is called, the Annotator engine will not continue to operate.

See Section 5.15 for more information on the reportError( ) method.

5.11 ConsoleOutput( ) Method

Example 5–11 shows the ConsoleOutput( ) method. Because your application must implement OutputListener, this method is required.

Example 5–11 ConsoleOutput( ) method

```
public void ConsoleOutput(String sz){
     report(sz);
}
```

The code in the ConsoleOutput( ) method implements the OutputListener.ConsoleOutput( ) method. This method uses the report(String) method to print messages during execution.

See Section 5.12 for more information on the report(String) method.

5.12 report(String) Method

Example 5–12 shows the report(String) method. This method is specific to this example.

Example 5–12 report(String) Method

```
public void report(String szValue){
     System.err.println(szValue);
}
```

The code in the report(String) method prints the given stream to the error stream.

5.13 report(Annotation) Method

Example 5–13 shows the report(Annotation) method. This method is specific to this example.

Example 5–13 report(Annotation) Method

```
public void report(Annotation ann){
     [1] StringWriter sw = new StringWriter( );
     [2] m_ah.exportToXML(sw, ann);
     [3] report(sw.toString( ));
}
```

The code in the report(Annotation) method performs the following operations:

1. Creates a new StringWriter object.
2. Creates an XML representation of the given annotation and uses the StringWriter object to write the contents to XML.
3. Casts the generated XML into a String object and uses the report(String) method to print the annotation to the error stream.

5.14 reportWarning( ) Method

Example 5–14 shows the reportWarning( ) method. This method is specific to this example.

Example 5–14 reportWarning( ) Method

```
public void reportWarning(Exception e){
     report("WARNING:");
     reportError(e);
}
```

This method uses the reportError( ) method to report the given error.

5.15 reportError( ) Method

Example 5–15 shows the reportError( ) method. This method is specific to this example.

Example 5–15 reportError( ) Method

```
public void reportError(Exception e){
     StringWriter sw = new StringWriter( );
     PrintWriter  pw = new PrintWriter(sw);
     e.printStackTrace(pw);
     report(sw.toString( ));
}
```

The code in the reportError( ) method captures the contents of the exception, casts them into a String object, and uses the report(String) method to print the exception to the error stream.

6

Annotator Engine API Reference Information

This chapter contains reference material for the classes and methods that beginning users will need to write a Java application that uses the Annotator engine. See the Javadoc included with the Annotator installation for complete reference information.

6.1 Class oracle.ord.media.annotator.annotations.Annotation

This section presents reference information on the methods of the Annotation class. This class is the superclass for all annotations; it offers the necessary data structure to hold logical annotations, their modifiers, and their accessor methods.

The attribute codes defined in this class are contained in Annotation.xml.

This class extends java.lang.Object.

addSubAnnotation( )

Format

public void addSubAnnotation(Annotation annChild)

Description

Adds the given annotation as a sub-annotation of the current annotation.

Parameters

annChild
The annotation to be added as a sub-annotation.

Return Value

None.

Exceptions

None.

Example

See Section 5.6 for an example of this method.

getAttribute( )

Format

public java.lang.Object getAttribute(java.lang.String szAttrCode)

Description

Gets the value of the given attribute as an Object. The client is responsible for casting the Object appropriately to access the returned value.

Parameters

szAttrCode
The attribute code of the attribute to be retrieved, as a String.

Return Value

This method returns the value of the given attribute, as an Object. If the given attribute has no value, null is returned.

Exceptions

None.

Example

See Section 5.6 for an example of this method.

getAttributes( )

Format

public java.util.Enumeration getAttributes( )

Description

Returns the list of attribute codes where a value has been set. This list does not include any attribute whose value is null.

Parameters

None.

Return Value

An Enumeration object that contains a list of attribute codes whose values have been set. Each code is returned as an Integer.

Exceptions

None.

Example

See Section 5.6 for an example of this method.

getDescriptor( )

Format

public AnnotationDesc getDescriptor( )

Description

Returns the AnnotationDesc object that is needed by the XML exporter.

For more information about the AnnotationDesc object, see Section 6.2.

Parameters

None.

Return Value

This method returns the AnnotationDesc object that is needed by the XML exporter.

Exceptions

None.

Example

None; only advanced users should call this method directly.

getName( )

Format

public java.lang.String getName( )

Description

Returns the name of the current annotation.

Parameters

None.

Return Value

This method returns the name of the current annotation.

Exceptions

None.

Example

```
String name = ann.getName( );
```

getNumSubAnnotations( )

Format

public int getNumSubAnnotations( )

Description

Returns the number of sub-annotations of the current annotation.

Parameters

None.

Return Value

This method returns the number of sub-annotations, as an integer.

Exceptions

None.

Example

```
int i = ann.getNumSubAnnotations( );
```

getParent( )

Format

public Annotation getParent( )

Description

Returns the parent object of the annotation.

Parameters

None.

Return Value

This method returns the parent object of this annotation.

Exceptions

None.

Example

```
Annotation parent = ann.getParent( );
```

getSampleAnns( )

Format

public java.util.Enumeration getSampleAnns( )

Description

Gets a list of the sub-annotations of the current annotation.

Parameters

None.

Return Value

This method returns an Enumeration object that contains a list of the sub-annotations of the current annotation.

Exceptions

None.

Example

```
Enumeration eSubAnns = ann.getSampleAnns( );
```

getSubAnnotations( )

Format

public java.util.Enumeration getSubAnnotations( )

Description

Gets an Enumeration object of the vector of sub-annotations.

Parameters

None.

Return Value

This method returns an Enumeration object of the vector of sub-annotations.

Exceptions

None.

Example

See Section 5.6 for an example of this method.

getURL( )

Format

public java.net.URL getURL( )

Description

Returns the URL of the annotation.

Parameters

None.

Return Value

This method returns the URL of the annotation.

Exceptions

None.

Example

```
java.net.URL location = ann.getURL( );
```

isDescendantOf( )

Format

public boolean isDescendantOf(java.lang.String szAncestor)

Description

Checks if the current annotation is a sub-annotation of the given annotation.

Parameters

szAncestor
The annotation of which the current annotation may be a sub-annotation.

Return Value

This method returns true if the current annotation is a sub-annotation of the given annotation; false otherwise.

Exceptions

None.

Example

```
if(subAnn.isDescendantOf("ann")
     boolean removedSuccessfully = ann.removeSubAnnotation(subAnn);
```

removeAttribute( )

Format

public void removeAttribute(java.lang.Object key)

Description

Removes an attribute and its value from the current annotation.

Parameters

key
The attribute that will be removed.

Return Value

None.

Exceptions

None.

Example

```
ann.removeAttribute(SALES_PRICE);
```

removeSampleAnns( )

Format

public void removeSampleAnns( )

Description

Removes all sub-annotations of the current annotation.

Parameters

None.

Return Value

None.

Exceptions

None.

Example

```
ann.removeSampleAnns( );
```

removeSubAnnotation( )

Format

public boolean removeSubAnnotation(Annotation ann)

Description

Removes the given sub-annotation from the current annotation.

Parameters

ann
The sub-annotation to be removed.

Return Value

This method returns true if the sub-annotation was removed successfully; false otherwise.

Exceptions

None.

Example

See the isDescendantOf( ) method for an example of this method.

setAttribute( )

Format

public void setAttribute(java.labg.String szAttrCode, java.lang.Object oValue)

Description

Inserts a new attribute into the current annotation.

Parameters

szAttrCode
The attribute code of the attribute whose value is to be changed, as a String.

oValue
The new value of the attribute, as an Object.

Return Value

None.

Exceptions

None.

Example

See Section 5.6 for an example of this method.

6.2 Class oracle.ord.media.annotator.descriptors.AnnotationDesc

This section presents reference information on the methods of the AnnotationDesc class, which creates annotation descriptor objects. This class provides the attribute definitions of the annotation.

This class extends oracle.ord.media.annotator.descriptors.Descriptor.

getAncestors( )

Format

public java.util.Vector getAncestors( )

Description

Gets the parent annotations of the current annotation.

Parameters

None.

Return Value

This method returns a Vector object that contains the parent annotations of the current annotation.

Exceptions

None.

Example

None; only advanced users should call this method directly.

getAttributeDesc( )

Format

public AttributeDesc getAttributeDesc(java.lang.String szAttributeName)

Description

Gets the attribute descriptor for the given attribute.

Parameters

szAttributeName
The name of the attribute for which you want to get the attribute descriptor.

Return Value

This method returns the attribute descriptor of the given attribute, as an AttributeDesc object.

Exceptions

None.

Example

None; only advanced users should call this method directly.

getSuppAttributes( )

Format

public java.util.Enumeration getSuppAttributes( )

Description

Gets the supported attribute descriptions defined in the annotation type.

Parameters

None.

Return Value

This method returns an Enumeration object that contains the supported attribute descriptions defined in the annotation type, as AttributeDesc objects.

Exceptions

None.

Example

None; only advanced users should call this method directly.

6.3 Class oracle.ord.media.annotator.descriptors.ParserDesc

This section presents reference information on the methods of the ParserDesc class, which creates parser descriptor objects. This class provides the definitions of the operations defined by the parsers, their parameters, their options, and the option parameters.

This class extends oracle.ord.media.annotator.descriptors.Descriptor.

getOperationDesc( )

Format

public OperationDesc getOperationDesc(java.lang.String szOpName)

Description

Gets the operation descriptor of the given operation.

Parameters

szOpName
The name of the operation whose descriptor will be returned.

Return Value

This method returns the operation descriptor of the given operation, as an OperationDesc object.

Exceptions

None.

Example

None; only advanced users should call this method directly.

getOperations( )

Format

public java.util.Enumeration getOperations( )

Description

Gets the descriptions of the operations supported by the parser, as defined in the parser descriptor.

Parameters

None.

Return Value

This method returns an Enumeration object that contains the descriptions of the operations supported by the parser, as OperationDesc objects.

Exceptions

None.

Example

None; only advanced users should call this method directly.

isEnabledAndExecutable( )

Format

public boolean isEnabledAndExecutable(java.lang.szOpName)

Description

Checks that the given operation is enabled and executable.

Parameters

szOpName
The name of the operation to check.

Return Value

This method returns true if the method is enabled and executable; false otherwise.

Exceptions

oracle.ord.media.annotator.descriptors.DescriptorException

Example

None; only advanced users should call this method directly.

6.4 Class oracle.ord.media.annotator.handlers.AnnTaskMonitor

This section presents reference information on the methods of the AnnTaskMonitor class, which creates an annotation task monitor. The **annotation task monitor** object is one of the components involved in monitoring tasks as they are being performed by an AnnotationHandler object (or annotation handler). Whenever a task is started by an annotation handler, an annotation task manager and an annotation task monitor are created. The **annotation task manager** runs on the server side; it tracks the progress of the task on the database server. The annotation task monitor runs on the client side; it tracks the progress value and messages from the returned annotation task monitor instance through a **task progress monitor**.

For more information on the annotation task manager, see Section 9.1.

This class extends java.lang.Object.

getMessage( )

Format

public java.lang.String getMessage( )

Description

Gets the current message from the task progress monitor.

Parameters

None.

Return Value

This method returns the current message of the task progress monitor, as a String.

Exceptions

None.

Example

```
String message = atm.getMessage( );
```

getTaskCurrent( )

Format

public int getTaskCurrent( )

Description

Gets the current value of the task progress monitor.

Parameters

None.

Return Value

This method returns the current value of the task progress monitor.

Exceptions

None.

Example

See the isDone( ) method for an example of this method.

getTaskEnd( )

Format

public int getTaskEnd( )

Description

Gets the end value of the task progress monitor.

Parameters

None.

Return Value

This method returns the end value of the task progress monitor.

Exceptions

None.

Example

See the isInitialized( ) method for an example of this method.

getTaskStart( )

Format

public int getTaskStart( )

Description

Gets the starting value of the task progress monitor.

Parameters

None.

Return Value

This method returns the initial value of the task progress monitor.

Exceptions

None.

Example

```
int i = atm.getTaskStart( );
```

isDone( )

Format

public boolean isDone( )

Description

Determines if the task has been completed.

Parameters

None.

Return Value

This method returns true if the task has been completed; false otherwise.

Exceptions

None.

Example

```
if(atm.isDone == false)
     int i = atm.getTaskCurrent( );
```

isInitialized( )

Format

public boolean isInitialized( )

Description

Checks if the annotation task monitor has been initialized. If it has, the getStartTask( ) and getEndTask( ) methods can be called to find the starting and ending times of the task.

Parameters

None.

Return Value

This method returns true if the annotation task monitor is initialized; false otherwise.

Exceptions

None.

Example

```
if(atm.isInitialized( ))
     int i = atm.getTaskEnd( );
```

6.5 Class oracle.ord.media.annotator.handlers.AnnotationHandler

This section presents reference information on the methods of the AnnotationHandler class, which creates an annotation handler. This class provides methods that produce an annotation for a given content source. An application that calls AnnotationHandler should implement the AnnListener interface to listen to the various responses to the handler. See Section 6.7 for more information.

You should create and use only one AnnotationHandler instance in your application. AnnotationHandler is stateless and thread-safe; you can have multiple threads calling the same AnnotationHandler instance.

This class extends java.lang.Object.

This class contains the following fields:

- public static final int OP_MODE_ASYNCH

  This signifies asynchronous mode.

- public static final int OP_MODE_SYNCH

  This signifies synchronous mode.

The examples in this section are based on the assumption that an AnnotationHandler object named handler has been created. See AnnotationHandler( ) and AnnotationHandler(int) for examples of creating an AnnotationHandler object.

AnnotationHandler( )

Format

public AnotationHandler( )

Description

Creates an AnnotationHandler object. As a default, the constructor uses the asynchronous mode of operations.

To ensure all engine traces are handled, the caller must create a Status instance before creating an annotation handler. See Section 6.10 for more information about Status.

Parameters

None.

Return Value

None.

Exceptions

oracle.ord.media.annotator.handlers.AnnotationHandlerException

Example

```
private AnnotationHandler handler = new AnnotationHandler( );
```

AnnotationHandler(int)

Format

public AnnotationHandler(int iOperationMode)

Description

Creates an AnnotationHandler object. As a default, the constructor uses the asynchronous mode of operations.

The AnnotationHandler class contains two static integers named OP_MODE_ASYNCH and OP_MODE_SYNCH. To create an annotation handler that runs in asynchronous mode, set iOperationMode to OP_MODE_ASYNCH. To create an annotation handler that runs in synchronous mode, set iOperationMode to OP_MODE_SYNCH.

To ensure all engine status messages are handled, the caller must create a Status instance before creating an annotation handler. See Section 6.10 for more information about Status.

Parameters

iOperationMode
The mode (either synchronous or asynchronous) that the annotation handler will use.

Return Value

None.

Exceptions

oracle.ord.media.annotator.handlers.AnnotationHandlerException

Example

See Section 5.4 for an example of this method.

createAnnotationByName( )

Format

public Annotation createAnnotationByname(java.lang.String szAnnName)

Description

Creates a new instance of an annotation, given the annotation type.

Parameters

szAnnName
The annotation type of the annotation to be created.

Return Value

This method returns the newly created annotation.

Exceptions

AnnotatorException

Example

See Section 5.6 for an example of this method.

exportToXML( )

Format

public void exportToXML(java.io.Writer w, Annotation ann)

Description

Builds an XML representation of an annotation and its sub-annotations and exports the representation to an XML file.

Parameters

w
The Writer object that will write the content to XML.

ann
The annotation to be exported.

Return Value

None.

Exceptions

None.

Example

See Section 5.13 for an example of this method.

extractMedia( )

Format

public AnnTaskMonitor extractMedia(Annotation ann, AnnListener annListener)

Description

Extracts media samples from an annotation. After the extraction is complete, the method calls the call-back function AnnListener.extractionPerformed( ).

Parameters

ann
The annotation from which samples will be extracted.

annListener
The listener that will be notified upon the completion of the parsing.

Return Value

This method returns the AnnTaskMonitor object associated with this task.

Exceptions

None.

Example

See Section 5.6 for an example of this method.

getAnnotationNames( )

Format

public java.util.Enumeration getAnnotationNames( )

Description

Returns a list of String objects with the names of the annotation types that are defined in the resource file.

Parameters

None.

Return Value

This method returns a list of String objects with the names of the annotation types that are defined in the resource file.

Exceptions

AnnotatorException

Example

```
Enumeration annTypes = handler.getAnnotationNames( );
```

getParserNames( )

Format

public java.util.Enumeration getParserNames( )

Description

Returns a list of the parser types defined in the resource file.

Parameters

None.

Return Value

This method returns a list of the parser types defined in the resource file.

Exceptions

AnnotatorException

Example

```
Enumeration parserTypes = handler.getParserNames( );
```

getRelVersion( )

Format

public final java.lang.String getRelVersion( )

Description

Returns the version of the *inter*Media Annotator release.

Parameters

None.

Return Value

This method returns the version of the *inter*Media Annotator release.

Exceptions

None.

Example

```
String release = handler.getRelVersion( )
```

importFromXML( )

Format

public Annotation importFromXML(java.io.Reader r)

Description

Creates a new Annotation object whose content is read from an XML file.

Parameters

r
The Reader object that will read the content from the XML file.

Return Value

This method returns a new Annotation object.

Exceptions

oracle.ord.media.annotator.annotations.AnnotationException

oracle.ord.media.annotator.handlers.annotation.AnnotationFactoryException

Example

```
java.io.FileReader reader = new FileReader("e:\\myAnnotation.xml");
Annotation ann = new Annotation(handler.importFromXML(reader));
```

insertMedia(Annotation,OrdMapping,AnnListener)

Format

public AnnTaskMonitor insertMedia(Annotation ann, OrdMapping om, AnnListener annListener)

Description

Creates a new connection to the database and inserts the annotation into an Oracle *inter*Media object on the database server.

Parameters

ann
The annotation from which samples will be extracted.

om
The mapping between the annotation and an Oracle *inter*Media object on the database server.

annListener
The listener that will be notified upon the completion of the parsing.

Return Value

This method returns the AnnTaskMonitor object associated with this task.

Exceptions

None.

Example

See Section 5.7 for an example of this method.

insertMedia(Annotation,OrdMapping,AnnListener,Connection)

Format

public AnnTaskMonitor insertMedia(Annotation ann, OrdMapping om, AnnListener annListener,
java.sql.Connection conn)

Description

Creates a new connection to the database and inserts the annotation into an Oracle *inter*Media object. After the parsing is complete, the method calls the call-back method AnnListener.insertionPerformed( ).

Parameters

ann
The annotation from which samples will be extracted.

om
The mapping between the annotation and an Oracle *inter*Media object on the database server. See the Annotator Javadoc for more information about the OrdMapping object.

annListener
The listener that will be notified upon the completion of the operation.

conn
The connection to the database. If this parameter is set to null, a new connection will be created.

Return Value

This method returns the AnnTaskMonitor object associated with this task.

Exceptions

None.

Example

```
handler.insertMedia(ann, ofm, listener, null);
```

where:

- ann: is the annotation to be inserted into the database.
- ofm: is the mapping object used to map to the database object.
- listener: is the listener that will be notified upon the completion of the operation.
- null: indicates that a new connection will be created.

isExtractable( )

Format

public boolean isExtractable (Annotation ann)

Description

Determines if it is possible to extract samples from the given annotation or any of its sub-annotations.

Parameters

ann
The annotation from which you want to extract samples.

Return Value

This method returns true if it is possible to extract samples; false otherwise.

Exceptions

None.

Example

See Section 5.6 for an example of this method.

isPlayable( )

Format

public boolean isPlayable(Annotation ann)

Description

Determines if it is possible to play the media content represented by the given annotation.

Parameters

ann
The annotation from which you want to play the content.

Return Value

This method returns true if it is possible to play the media content; false otherwise.

Exceptions

None.

Example

```
if(handler.isPlayable(ann)){
     handler.playMedia(ann,listener)
}
```

where:

- ann: is the annotation from which you will play the content.
- listener: is a reference to the AnnListener object that will be notified upon completion.

parseMedia(InputStream,String,AnnListener)

Format

public AnnTaskMonitor parseMedia(java.io.InputStream is, java.lang.String sURL,
AnnListener annListener)

Description

Parses the source associated with the given InputStream and creates an annotation of the given URL. After the parsing is complete, the method performs the following operations:

- Attempts to set the MEDIA_SOURCE_MIME_TYPE attribute in the annotation
- Invokes the call-back function AnnListener.parsePerformed( )

Parameters

is
The InputStream of the media file to be parsed.

sURL
The URL of the media file to be parsed.

annListener
The listener that will be notified upon the completion of the parsing.

Return Value

This method returns the AnnTaskMonitor object associated with this task.

Exceptions

None.

Example

```
//Assign the URL to a string named szURL
//The current client (represented by this) implements the AnnListener interface
FileInputStream fStream = new FileInputStream("test.mpg");
AnnTaskMonitor atm = handler.parseMedia(fStream, szURL, this);
```

parseMedia(String,AnnListener)

Format

public AnnTaskMonitor parseMedia(java.lang.String sURL, AnnListener annListener)

Description

Parses the source and creates an annotation of the given URL. After the parsing is complete, the method performs the following operations:

- Attempts to set the following attributes in the annotation
  - MEDIA_SIZE
  - MEDIA_SOURCE_DIRECTORY
  - MEDIA_SOURCE_FILENAME
  - MEDIA_SOURCE_PROTOCOL
  - MEDIA_SOURCE_URL
  - MEDIA_SOURCE_MIME_TYPE
- Invokes the call-back function AnnListener.parsePerformed( )

Parameters

sURL
The URL of the media file to be parsed.

annListener
The listener that will be notified upon the completion of the parsing.

Return Value

This method returns the AnnTaskMonitor object associated with this task.

Exceptions

None.

Example

See Section 5.5 for an example of this method.

playMedia( )

Format

public void playMedia(Annotation ann, AnnListener annListener)

Description

Plays the content represented by the named annotation. This method is synchronous; it does not return an AnnTaskMonitor object.

Parameters

ann
The annotation from which you want to play the content.

annListener
The listener that will be notified upon the completion of the parsing.

Return Value

None.

Exceptions

None.

Example

See the isPlayable( ) method for an example of this method.

6.6 Class oracle.ord.media.annotator.handlers.db.OrdFileMapping

This section presents reference information on the methods associated with the OrdFileMapping object, which maps the contents of an annotation instance to specific tables and specific rows in the database.

This class extends oracle.ord.media.annotator.handlers.db.OrdMapping.

generateStatement( )

Format

public java.lang.String generateStatement(Annotation ann)

Description

Returns the PL/SQL statement that is used to insert the annotation into the database. This statement is processed by the Annotator pre-processor to insert Annotator-specific directives.

This method overrides OrdMapping.generateStatement( ).

Parameters

ann
The annotation to be inserted.

Return Value

This method returns the PL/SQL statement that will be used to insert the annotation into the database.

Exceptions

java.io.IOException

Example

```
String sqlStatement = ofm.generateStatement(ann);
```

OrdFileMapping( )

Format

public OrdFileMapping(java.lang.String szFileName)

Description

Creates an OrdFileMapping object, which contains the mapping of the contents of the annotation to the database.

Parameters

szFileName
The name of the file that contains the mapping.

Return Value

None.

Exceptions

None.

Example

See Section 5.7 for an example of this method.

6.7 Class oracle.ord.media.annotator.listener.AnnListener

This section presents reference information on the methods of the AnnListener interface. The client must implement this interface in order to invoke the Annotator engine.

This class extends java.util.EventListener.

errorOccured( )

Format

public void errorOccured(Annotation ann, java.lang.Exception e)

Description

Returns an exception in the case of fatal errors.

If an error is generated by AnnotationHandler.insertMedia( ), the JDBC connection is automatically rolled back and closed.

Parameters

ann
The annotation instance.

e
An exception that explains why the failure occurred.

Return Value

None.

Exceptions

None.

Example

See Section 5.10 for an example of this method.

extractionPerformed( )

Format

public void extractionPerformed(Annotation ann)

Description

Performs any necessary operations after the completion of media sample extraction. This method is the call-back function of AnnotationHandler.extractMedia( ).

After the extraction is completed, new attributes are defined in the annotation. The new attributes are relative to the extracted sample; a refresh on the client is probably required.

Parameters

ann
The annotation instance from which the extraction was performed.

Return Value

None.

Exceptions

None.

Example

See Section 5.7 for an example of this method.

insertionPerformed( )

Format

public void insertionPerformed(Annotation ann, java.sql.Connection conn)

Description

Performs any necessary operations after the completion of the insertion of the annotation into the database. These operations include explicitly committing or rolling back the changes to the database and closing the connection to the database.

You can keep the connection to the database open and pass it to another call of AnnotationHandler.insertMedia( ); however, it is your responsibility to check the thread-safety of the connection.

This method is the call-back function of AnnotationHandler.extractMedia( ).

Parameters

ann
The annotation instance that has been inserted into the database.

conn
The JDBC connection used to perform the insertion.

Return Value

None.

Exceptions

None.

Example

See Section 5.8 for an example of this method.

parsePerformed( )

Format

public void parsePerformed(Annotation ann)

Description

Performs any necessary operations on the annotation after it is created and before it is uploaded to the database. This method is the call-back function of AnnotationHandler.parseMedia( ).

Parameters

ann
The newly created media annotation.

Return Value

None.

Exceptions

None.

Example

See Section 5.6 for an example of this method.

warningOccured( )

Format

public void warningOccured(Annotation ann, java.lang.Exception e)

Description

Returns an exception in the case of non-fatal errors.

Parameters

ann
The annotation instance.

e
An exception that explains why the failure occurred.

Return Value

None.

Exceptions

None.

Example

See Section 5.9 for an example of this method.

6.8 Class oracle.ord.media.annotator.listener.OutputListener

This section presents reference information on the methods of the OutputListener interface. The client invokes this method to process status output from the engine.

This class extends java.util.EventListener.

ConsoleOutput( )

Format

public void ConsoleOutputd(java.lang.String szOutput)

Description

Prints status messages while the engine is running.

Parameters

szOutput
The status message to be printed.

Return Value

None.

Exceptions

None.

Example

See Section 5.11 for an example of this method.

6.9 Class oracle.ord.media.annotator.utils.Preferences

This section presents reference information on the methods associated with the Preferences class. This class is primarily used by the engine. It supports the loading of preferences, the dynamic changing of preferences, and saving preferences to a file. The implementation of this class is independent of the other Annotator classes.

This class extends java.lang.Object and implements oracle.ord.media.annotator.utils.PreferenceConstants and java.lang.Cloneable.

clone( )

Format

public java.lang.Object clone( )

Description

Creates and returns a copy of this object. For more information, see the Java 1.2 documentation for the java.lang.Object.clone( ) method.

Parameters

None.

Return Value

This method returns a copy of the current Preferences object, as an Object.

Exceptions

java.lang.CloneNotSupportedException

Example

None; this method should be called only by advanced programmers who want to manually access the Annotator preferences.

getPrefs( )

Format

public static Preferences getPrefs( )

Description

Gets the Preferences object of the current annotation.

Parameters

None.

Return Value

This method returns the Preferences object of the current annotation.

Exceptions

None.

Example

See Section 5.4 for an example of this method.

getProperty( )

Format

public java.lang.String getProperty(java.lang.String s)

Description

Gets the value of the given property from the preferences of the current annotation.

Parameters

s
The name of the property for which you will get the value.

Return Value

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

Exceptions

None.

Example

None; this method should be called only by advanced programmers who want to manually access the Annotator preferences.

Preferences( )

Format

public Preferences( )

Description

Creates a Preferences object.

Parameters

None.

Return Value

None.

Exceptions

None.

Example

None; this method should be called only by advanced programmers who want to manually access the Annotator preferences.

saveToFile( )

Format

public void saveToFile( )

Description

Saves the preferences to a file.

Parameters

None.

Return Value

None.

Exceptions

None.

Example

None; this method should be called only by advanced programmers who want to manually access the Annotator preferences.

setPreferences( )

Format

public static void setPreferences(Preferences prefs)

Description

Sets the preferences of the current annotation to match the given Preferences object.

Parameters

prefs
The preferences to be set in the annotation.

Return Value

None.

Exceptions

None.

Example

None; this method should be called only by advanced programmers who want to manually access the Annotator preferences.

setProperty( )

Format

public void setProperty(java.lang.String s, java.lang.Object o)

Description

Sets the given property to the given value.

Parameters

s
The name of the property that you will set.

o
The value to set.

Return Value

None.

Exceptions

None.

Example

See Section 5.4 for an example of this method.

6.10 Class oracle.ord.media.annotator.utils.Status

This section presents reference information on the methods associated with the Status class. This class updates the current status in the GUI of the application. The user can choose from three supported status modes. In order, from least output to most output, they are STATUS (or TERSE), VERBOSE, and TRACE.

The Status class follows a singleton pattern, so only one instance is needed for all instances of the Annotator engine in the Java virtual machine.

This class extends java.lang.Object.

The class contains the following fields that are used to set the error level:

- public static final short ERR_LEVEL_WARNING
- public static final short ERR_LEVEL_ERROR
- public static final short ERR_LEVEL_FATALERROR

The class contains the following fields that are used to set the output mode:

- public static final short OUTPUT_MODE_STATUS
- public static final short OUTPUT_MODE_TERSE
- public static final short OUTPUT_MODE_TRACE
- public static final short OUTPUT_MODE_VERBOSE

GetOutputMode( )

Format

public short GetOutputMode( )

Description

Returns the current output mode of the Status object.

Parameters

None.

Return Value

This method returns the current output mode of the Status object. The possible values are OUTPUT_MODE_STATUS, OUTPUT_MODE_TERSE, OUTPUT_MODE_TRACE, or OUTPUT_MODE_VERBOSE.

Exceptions

None.

Example

```
short outputMode = m_st.GetOutputMode( );
```

getStatus( )

Format

public static Status getStatus( )

Description

Gets the Status object of the current annotation.

Parameters

None.

Return Value

This method returns the Status object.

Exceptions

None.

Example

See Section 5.4 for an example of this method.

initStatus( )

Format

public static void initStatus(OutputListener ol)

Description

Initializes the Status object. This method should be invoked before initializing the AnnotationHandler object.

Parameters

ol
The instance of the OutputListener class that will receive the status messages from the AnnotationHandler object.

Return Value

None.

Exceptions

None.

Example

See Section 5.4 for an example of this method.

Report( )

Format

public void Report(short omDesignated, java.lang.String szStatus)

Description

Prints the given message to the appropriate output source. The output source is set internally when the Status object is instantiated.

This method should be used by parser developers only.

Parameters

omDesignated
The output mode. If the output mode given here is of a lower priority than the output mode that has been set for the engine, the message will not be reported.

szStatus
The message to be reported.

Return Value

None.

Exceptions

None.

Example

See Section 5.4 for an example of this method.

ReportError(short, Object, String, int, String)

Format

public void ReportError(short sErrLevel, java.lang.Object oInstance,
java.lang.String szMethodName, int iLineNumber, java.lang.String szDesc)

Description

Reports errors through the System.err stream. Multiple error levels can be given to specify consequences.

Parameters

sErrLevel
The 16-bit error level (ERR_LEVEL_WARNING, ERR_LEVEL_ERROR, or ERR_LEVEL_FATALERROR).

oInstance
The object pointer of the source of the error.

szMethodName
The name of the method where the error occurred, as a String.

iLineNumber
The line number where the error occurred.

szDesc
A lengthy description of the error.

Return Value

None.

Exceptions

None.

Example

```
status.ReportError(Status.ERR_LEVEL_WARNING, this,
     "name_of_current_method", iCurrentLineNum, "error description");
```

ReportError(short,Throwable)

Format

public void ReportError(short sErrLevel, java.lang.Throwable sException)

Description

Reports errors through the System.err stream. Multiple error levels can be given to specify consequences.

Parameters

sErrLevel
The 16-bit error level (ERR_LEVEL_WARNING, ERR_LEVEL_ERROR, or ERR_LEVEL_FATALERROR).

sException
The exception that was raised, as a Throwable object.

Return Value

None.

Exceptions

None.

Example

```
status.ReportError(Status.ERR_LEVEL_WARNING, myExceptionInstance);
```

SetOutputMode( )

Format

public void SetOutputMode(short omNew)

Description

Sets the status output mode to STATUS, TERSE, TRACE, or VERBOSE.

Parameters

omNew
The output mode to be set; the value should be OUTPUT_MODE_STATUS, OUTPUT_MODE_TERSE, OUTPUT_MODE_TRACE, or OUTPUT_MODE_VERBOSE.

Return Value

None.

Exceptions

None.

Example

See Section 5.4 for an example of this method.

7

Creating PL/SQL Upload Templates

*inter*Media Annotator can upload media data and an associated annotation into an Oracle database where Oracle *inter*Media has been installed. It does so through an Oracle PL/SQL Upload Template, which contains both PL/SQL calls and Annotator-specific keywords.

You create your own PL/SQL Upload Templates. Advanced users with PL/SQL experience can write PL/SQL Upload Templates using a text editor.

7.1 Creating Upload Templates Manually

*inter*Media Annotator users with experience in PL/SQL and JDBC may want to create their own PL/SQL Upload Templates instead of using the PL/SQL Template Wizard in the Annotator GUI (see Chapter 4 for more information). You can create a PL/SQL Upload Template using any text editor.

7.1.1 Structure of Upload Templates

The PL/SQL Upload Template begins with a list of DML and DDL statements. Using this list is optional, depending on your needs.

One anonymous PL/SQL block follows the list. You cannot have more than one anonymous PL/SQL block, and nothing should appear in the PL/SQL Upload Template after you end the block.

The anonymous PL/SQL block contains both standard PL/SQL code and *inter*Media Annotator-specific keywords. For more information on the keywords, see Section 7.1.2. For more information on writing PL/SQL code, see *PL/SQL User's Guide and Reference*.

Depending on the platform of the database server, there may be a limit on the maximum size of the anonymous PL/SQL block. If you encounter this problem,

you can work around it by packaging some of your statements into PL/SQL procedures in order to reduce the size of your PL/SQL block.

7.1.2 Annotator-Specific Keywords

In addition to standard PL/SQL calls, the PL/SQL Upload Templates contain Annotator-specific keywords. The keywords are delimited by a dollar sign and a left brace at the beginning of a keyword and a right brace at the end of the keyword (*${* and *}*). These keywords are interpreted by the Annotator preprocessor, which interprets the keywords and generates the appropriate PL/SQL code.

> **Note:** An Annotator-specific keyword must appear on its own line in the PL/SQL Upload Template. You cannot have multiple keywords on the same line.

The following sections provide more information on the keywords.

7.1.2.1 Attribute Values

Instead of hard-coding values for specific attributes in your PL/SQL Upload Template, you provide the name of the attribute, enclosed by the *${* and *}* characters. This tells the preprocessor to get the actual value of the attribute from the current annotation, and to use that value to replace the keyword in the PL/SQL Upload Template. This simple replacement lets you use the same PL/SQL Upload Template for multiple annotations.

Example 7–1 shows keywords that will later be replaced with attribute values.

Example 7–1 Attribute Values as Keywords

```
audioObj.setMimeType('${MEDIA_SOURCE_MIME_TYPE}');
INSERT INTO SongsTable VALUES('${AUDIO_CD_TRACK_CDID}');
```

7.1.2.2 ${MANN_BEGIN_ITERATE} and ${MANN_END_ITERATE}

The ${MANN_BEGIN_ITERATE} and ${MANN_END_ITERATE} keywords indicate that the code enclosed by the keywords should be repeated for each sub-annotation of the given type. The name of the annotation type follows the ${MANN_BEGIN_ITERATE} keyword.

Example 7–2 shows a block of code that will be run for each AudioCDTrackAnn annotation that exists as a sub-annotation of the current annotation.

Example 7–2 ${MANN_BEGIN_ITERATE} and ${MANN_END_ITERATE}

```
${MANN_BEGIN_ITERATE} AudioCDTrackAnn
INSERT INTO SongsTable VALUES('${AUDIO_CD_TRACK_CDID}',
                              '${AUDIO_CD_TRACK_ID}');
${MANN_END_ITERATE}
```

7.1.2.3 ${MANN_BEGIN_TRACK} and ${MANN_END_TRACK}

The ${MANN_BEGIN_TRACK} and ${MANN_END_TRACK} keywords indicate that the code enclosed by the keywords should be run on the first instance of a sub-annotation of the given class. The name of the annotation type follows the ${MANN_BEGIN_TRACK} keyword.

Example 7–3 shows a block of code that will be run upon the first AudioCDTrackAnn annotation that exists as a sub-annotation of the current annotation.

Example 7–3 ${MANN_BEGIN_TRACK} and ${MANN_END_TRACK}

```
${MANN_BEGIN_TRACK} AudioCDTrackAnn
INSERT INTO SongsTable VALUES('${AUDIO_CD_TRACK_CDID}',
                              '${AUDIO_CD_TRACK_ID}');
${MANN_END_TRACK}
```

7.1.2.4 ${MANN_BEGIN_IFDEF} and ${MANN_END_IFDEF}

The ${MANN_BEGIN_IFDEF} and ${MANN_END_IFDEF} keywords indicate that the code enclosed by the keywords should be run only if the current annotation has a defined value for a given attribute. The name of the attribute follows the ${MANN_BEGIN_IFDEF} keyword.

Example 7–4 shows a block of code that will be run only if the MEDIA_SOURCE_MIME_TYPE attribute is defined in the current annotation.

Example 7–4 ${MANN_BEGIN_IFDEF} and ${MANN_END_IFDEF}

```
${MANN_BEGIN_IFDEF} MEDIA_SOURCE_MIME_TYPE
audioObj.setMimeType('${MEDIA_SOURCE_MIME_TYPE}');
${MANN_END_IFDEF}
```

7.1.2.5 ${MANN_BEGIN_IFEQUALS} and ${MANN_END_IFEQUALS}

The ${MANN_BEGIN_IFEQUALS} and ${MANN_END_IFEQUALS} keywords indicate that the code enclosed by the keywords should be run only if the current annotation contains a given attribute of a given value. The name of the attribute and

the value follow the ${MANN_BEGIN_IFEQUALS} keyword. The string comparison is case-sensitive.

Example 7–5 shows a block of code that will be run only if the MEDIA_SOURCE_MIME_TYPE attribute is defined as *audio/basic* in the current annotation.

Example 7–5 ${MANN_BEGIN_IFEQUALS} and ${MANN_END_IFEQUALS}

```
${MANN_BEGIN_IFEQUALS} MEDIA_SOURCE_MIME_TYPE audio/basic
audioObj.setMimeType('${MEDIA_SOURCE_MIME_TYPE}');
${MANN_END_IFEQUALS}
```

7.1.2.6 ${MANN_UPLOAD_SRC}

The ${MANN_UPLOAD_SRC} keyword indicates that the media source data associated with the current annotation should be uploaded to the current Oracle database table using JDBC; the file is loaded into Annotator, and Annotator loads the file into the database. The name of the server-side object and attribute (of the BLOB type) follows the ${MANN_UPLOAD_SRC} keyword.

Upload performance with the ${MANN_UPLOAD_SRC} keyword may be slow if you are using the JDBC Thin driver to upload a large media source, or if you have a slow network connection. You may get better results by using the *inter*Media import( ) method. See Section 4.1.2 for more information on the differences between the two upload options. See *Oracle interMedia User's Guide and Reference* for more information on the import( ) method.

Example 7–6 shows a block of code that will upload the current media source data to the source.localData attribute of the server-side *inter*Media object videoObj.

Example 7–6 ${MANN_UPLOAD_SRC}

```
${MANN_UPLOAD_SRC} videoObj.source.localData
```

7.1.2.7 ${MANN_UPLOAD_XML}

The ${MANN_UPLOAD_XML} keyword indicates that the current annotation should be uploaded to the current Oracle database table. The annotation should be uploaded to a CLOB in an Oracle *inter*Media object. The name of the server-side object and CLOB attribute follows the ${MANN_UPLOAD_XML} keyword.

Example 7–7 shows a block of code that will upload the current annotation to the comments attribute of the server-side *inter*Media object videoObj.

Example 7–7 ${MANN_UPLOAD_XML}

```
${MANN_UPLOAD_XML} videoObj.comments
```

For more information on Oracle *inter*Media APIs, see *Oracle interMedia User's Guide and Reference*.

7.1.3 Saved Files

Once you have written your PL/SQL Upload Template, save it with the suffix *.ofm*. The default directory that Annotator uses for PL/SQL Upload Templates is <*ORACLE_HOME*>\ord\Annotator\ofm. To change the default directory, see Section 4.1.7.

See Section 4.2 for information on how to run a PL/SQL Upload Template through the Annotator GUI.

7.1.4 Complete PL/SQL Upload Template Example

Example 7–8 contains a sample PL/SQL Upload Template. It will upload a video object and its associated annotation to an Oracle table named MediaTable. The sample contains one anonymous PL/SQL block containing a mix of PL/SQL calls and Annotator-specific keywords.

Example 7–8 PL/SQL Upload Template Sample

```
DECLARE
   videoObj       ORDSYS.ORDVIDEO;
   ctx            RAW(4000) := NULL;
BEGIN
   INSERT INTO MediaTable VALUES (
      1,
      ORDSYS.ORDVIDEO(
         '${MEDIA_TITLE}',
         ORDSYS.ORDSource(EMPTY_BLOB(),
            NULL,
            '${MEDIA_SOURCE_DIRECTORY}',
            '${MEDIA_SOURCE_FILENAME}',
            NULL,NULL),
         '${MEDIA_SOURCE_FILE_FORMAT}',
         '${MEDIA_SOURCE_MIME_TYPE}',
         EMPTY_CLOB(), NULL, NULL, NULL, NULL, NULL, NULL, '',
         NULL, NULL)
   );
```

```
   SELECT M.mediaSource INTO videoObj
   FROM   MediaTable M
   WHERE  M.MediaId = 1
   FOR UPDATE;

   ${MANN_UPLOAD_SRC} videoObj.source.localData
   ${MANN_UPLOAD_XML} videoObj.comments

   UPDATE MediaTable M SET M.mediaSource = videoObj
   WHERE  M.mediaId = 1;

END;
```

7.2 Editing Existing PL/SQL Upload Templates

Whether you use the PL/SQL Template Wizard or you write your own PL/SQL Upload Templates, you can edit your PL/SQL Upload Templates using a text editor.

You cannot use the PL/SQL Template Wizard to edit an existing PL/SQL Upload Template; it can create only new PL/SQL Upload Templates.

Part III

*inter*Media Annotator Extensibility

Part III provides information about the extensibility of Oracle *inter*Media Annotator and contains the following chapters:

- Chapter 8, "interMedia Annotator Custom Parser Example"
- Chapter 9, "Annotator Parser API Reference Information"
- Chapter 10, "Creating New Annotation Types"

8

*inter*Media Annotator Custom Parser Example

This chapter provides a description of AuParser.java, which is an example of a user-developed parser for a custom content format that was written using the *inter*Media Annotator parser APIs. AuParser.java contains user-defined methods that use Java and *inter*Media Annotator APIs to define a parser for the Next/Sun AU file format. The purpose of the parser is to extract the format encoding information and the associated user data from the file.

The contents of AuParser.java are included in a zip file at the following location:

`<ORACLE_HOME>/ord/Annotator/src/parsers.zip`

This chapter provides an example of creating a new parser for a custom content format.

This code will not necessarily match the code shipped as AuParser.java with the *inter*Media Annotator installation. If you want to run this example on your system, use the file provided with the *inter*Media Annotator installation; do not attempt to compile and run the code presented in this chapter.

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

8.1 Parser Creation Overview

To define a new parser, perform the following operations:

1. Create a new Java class that inherits from oracle.ord.media.annotator.parsers.Parser. In this example, the Java class is defined in AuParser.java.

   Your Java class must implement the following methods:

   - parse( ): parses the content from the InputStream object
   - saveToAnnotation( ): saves the results of the parsing as an annotation named m_annInst
   - extractSamples( ): extracts samples from the media source file. You must implement this method whether or not you want to support sample extraction.

   Additionally, you can add other methods depending on the operations that you want your parser to perform.

   Section 8.3 through Section 8.10 provide examples of the contents of AuParser.java.

2. Write a parser descriptor XML file and add it to the *<ORACLE_HOME>*/ord/Annotator/lib/descriptors/parsers directory. For an example of a parser descriptor XML file, see *<ORACLE_HOME>*/ord/Annotator/lib/descriptors/parsers/AuParser.xml.

3. Optionally, modify the following file to set your parser to be used for a specific MIME type:

   <ORACLE_HOME>`/ord/Annotator/lib/conf/Annotator.mime`

   You can also set a parser for a specific MIME type with the Annotator GUI. See Section 2.2.5 for more information.

8.2 AU File Structure

Example 8–1 shows the basic structure of an AU file.

Example 8–1 Basic Structure of an AU File

```
<pre><code>
typedef struct {
     int magic;              // magic number SND_MAGIC
     int dataLocation;       // offset or pointer to the data
     int dataSize;           // number of bytes of data
     int dataFormat;         // the data format code
```

```
     int samplingRate;        // the sampling rate
     int channelCount;        // the number of channels
     char info[4];            // optional text information
} SNDSoundStruct;
</code></pre>
```

The magic number is equal to ((int)0x2e736e64), which is a representation of .snd". The info parameter will be associated with the user data attribute of the annotation.

8.3 Package and Import Statements

Example 8–2 shows the import statements that must be included to properly run an Annotator parser, and the package statements that set the package of this class.

Example 8–2 Package and Import Statements

```
package oracle.ord.media.annotator.parsers.au;

import java.io.*;
import java.util.*;
import java.net.*;

import oracle.ord.media.annotator.parsers.*;
import oracle.ord.media.annotator.annotations.*;
import oracle.ord.media.annotator.utils.*;
```

8.4 Class Definition and Instance Variables

Example 8–3 shows the class definition and instance variables for AuParser.java.

Example 8–3 Class Definition and Instance Variables

```
public class AuParser extends Parser{
     private static final Integer SND_FORMAT_UNSPECIFIED = new Integer(0);
     private static final Integer SND_FORMAT_MULAW_8 = new Integer(1);
     private static final Integer SND_FORMAT_LINEAR_8 = new Integer(2);
     private static final Integer SND_FORMAT_LINEAR_16 = new Integer(3);
     private static final Integer SND_FORMAT_LINEAR_24 = new Integer(4);
     private static final Integer SND_FORMAT_LINEAR_32 = new Integer(5);
     private static final Integer SND_FORMAT_FLOAT = new Integer(6);
     private static final Integer SND_FORMAT_DOUBLE = new Integer(7);
     private static final Integer SND_FORMAT_INDIRECT = new Integer(8);
```

```
private static final Integer SND_FORMAT_NESTED = new Integer(9);
private static final Integer SND_FORMAT_DSP_CORE = new Integer(10);
private static final Integer SND_FORMAT_DSP_DATA_8 = new Integer(11);
private static final Integer SND_FORMAT_DSP_DATA_16 = new Integer(12);
private static final Integer SND_FORMAT_DSP_DATA_24 = new Integer(13);
private static final Integer SND_FORMAT_DSP_DATA_32 = new Integer(14);
private static final Integer SND_FORMAT_UNKNOWN = new Integer(15);
private static final Integer SND_FORMAT_DISPLAY = new Integer(16);
private static final Integer SND_FORMAT_MULAW_SQUELCH = new Integer(17);
private static final Integer SND_FORMAT_EMPHASIZED = new Integer(18);
private static final Integer SND_FORMAT_COMPRESSED = new Integer(19);
private static final Integer SND_FORMAT_COMPRESSED_EMPHASIZED = new
     Integer(20);
private static final Integer SND_FORMAT_DSP_COMMANDS = new Integer(21);
private static final Integer SND_FORMAT_DSP_COMMANDS_SAMPLES = new
     Integer(22);
private static final Integer SND_FORMAT_ADPCM_G721 = new Integer(23);
private static final Integer SND_FORMAT_ADPCM_G722 = new Integer(24);
private static final Integer SND_FORMAT_ADPCM_G723_3 = new Integer(25);
private static final Integer SND_FORMAT_ADPCM_G723_5 = new Integer(26);
private static final Integer SND_FORMAT_ALAW_8 = new Integer(27);

private Hashtable m_htFormatInfo;
private int m_iBitsPerSample;
private int m_iSampleRate;
private int m_iChannelCount;
private String m_szUserData;
private FormatInfo m_fiFormatInfo;
```

A parser must extend the Parser class. See Section 9.3 for more information on the fields and methods of the Parser class.

The AU parser contains a Hashtable object named m_htFormatInfo, which will map keys to values. The keys are instantiated as private static Integer objects and given sequential values. These are specific to AuParser.java and may not be necessary for your parser.

AuParser.java contains a FormatInfo object named m_fiFormatInfo, which is used to encapsulate information related to a specific data format. See Section 8.5 for more information. These are specific to AuParser.java and may not be necessary for your parser.

AuParser.java also contains the following instance variables:

- m_iBitsPerSample is the number of audio bits per sample, as an integer.

- m_iSampleRate is the sampling rate of the AU file, as an integer.
- m_iChannelCount is the number of channels in the AU file, as an integer.
- m_szUserData is optional text information related to the AU file.

8.5 FormatInfo Class

Example 8–4 shows the contents of the FormatInfo class, which is used to encapsulate information related to a specific data format.

Example 8–4 FormatInfo Class

```
private class FormatInfo{
     [1] private String m_szFormatString;
     private String m_szFormatCode;

     [2] public FormatInfo(String szFormatString, String szFormatCode){
          m_szFormatString = szFormatString;
          m_szFormatCode = szFormatCode;
     }

     [3] public String getFormatString( ){
          return m_szFormatString;
     }

     [4] public String getFormatCode( ){
          return m_szFormatCode;
     }
}
```

The FormatInfo class contains the following code:

1. Two instance variables that store values for the format string and format code.
2. A constructor with two parameters that set the format string and the format code.
3. A method that returns the format string to the caller.
4. A method that returns the format code to the caller.

8.6 AuParser( ) Method

Example 8–5 shows the contents of the AuParser( ) method, which is the constructor of this class.

Example 8–5 AuParser( ) Method

```
public AuParser( ){
     [1] m_htFormatInfo = new Hashtable( );
     [2] FillFormatHashTable( );
}
```

A parser must have a constructor with no parameters.

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

1. Instantiates the m_htFormatInfo variable.
2. Populates m_htFormatInfo with data by calling the FillFormatHashTable( ) method. See Example 8–9 for more information.

8.7 parse( ) Method

Example 8–6 shows the parse( ) method, which parses an AU file and extracts some of its metadata.

Example 8–6 parse( ) Method

```
public void parse( ) throws ParserException{
     try {
          [1] int iMagicNumber = m_madisResource.readInt( );

          [2] if(iMagicNumber != ((int)0x2e736e64))
               throw new ParserException("Format Exception. Expecting a
                    Next/Sun au formatted file");

          [3] int iDataLocation = m_madisResource.readInt( );

          [4] int iIntCounter = 2;

          [5] m_annTaskMan.setTask(0, iDataLocation);
          [6] m_annTaskMan.setTaskCurrent(iIntCounter*4,
               "Parsing AU Header...");

          [7] int iDataSize = m_madisResource.readInt( );
```

```
        m_annTaskMan.setTaskCurrent((++iIntCounter)*4);

        [8] int iDataFormat = m_madisResource.readInt( );
        m_annTaskMan.setTaskCurrent((++iIntCounter)*4);

        [9] m_iSampleRate = m_madisResource.readInt( );
        m_annTaskMan.setTaskCurrent((++iIntCounter)*4);

        [10] m_iChannelCount = m_madisResource.readInt( );
        m_annTaskMan.setTaskCurrent((++iIntCounter)*4);

        [11] int iInfoLength = iDataLocation - 24;
        m_szUserData = m_madisResource.readString(iInfoLength);
        [12] m_annTaskMan.setTaskCurrent(iDataLocation);
        [13] m_annTaskMan.done( );

        [14] m_fiFormatInfo = (FormatInfo) m_htFormatInfo.get
             (new Integer(iDataFormat));

        [15] m_iBitsPerSample = 0;
        [16] if((iDataFormat == SND_FORMAT_MULAW_8.intValue( )) ||
             (iDataFormat == SND_FORMAT_LINEAR_8.intValue( )) ||
             (iDataFormat == SND_FORMAT_DSP_DATA_8.intValue( )) ||
             (iDataFormat == SND_FORMAT_ALAW_8.intValue( )))
             m_iBitsPerSample = 8;
        else
        if((iDataFormat == SND_FORMAT_LINEAR_16.intValue( )) ||
             (iDataFormat == SND_FORMAT_DSP_DATA_16.intValue( )) ||
             (iDataFormat == SND_FORMAT_EMPHASIZED.intValue( )) ||
             (iDataFormat == SND_FORMAT_COMPRESSED.intValue( )) ||
             (iDataFormat == SND_FORMAT_COMPRESSED_EMPHASIZED.intValue( )))
             m_iBitsPerSample = 16;
        else
        if((iDataFormat == SND_FORMAT_LINEAR_24.intValue( )) ||
             (iDataFormat == SND_FORMAT_DSP_DATA_24.intValue( )))
             m_iBitsPerSample = 24;
        else
        if((iDataFormat == SND_FORMAT_LINEAR_32.intValue( ))   ||
             (iDataFormat == SND_FORMAT_DSP_DATA_32.intValue( )) ||
             (iDataFormat == SND_FORMAT_FLOAT.intValue( )) ||
             (iDataFormat == SND_FORMAT_DOUBLE.intValue( )))
             m_iBitsPerSample = 32;
        else
             m_iBitsPerSample = -1;
    }
```

```
     [17] catch(IOException ioExc) {
          throw new ParserException("IOException raised while
               reading from input stream");
     }

     [18] saveToAnnotation( );
}
```

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

1. Reads the first integer in the AU file (the magic number) and sets it to iMagicNumber.

   m_madisResource is the MADataInputStream that contains the AU file to be processed. See Section 9.4 for more information.

2. Tests to see if the magic number that was just read matches the magic number of an AU file. If it does not, then the file is not an AU file and an exception is thrown.

3. Reads the next integer in m_madisResource, which is the offset from the beginning of the stream where the media data begins, and sets it to iDataLocation.

4. Sets a counter. Because 2 integers (8 bytes total) have already been read, the counter is set to 2.

5. Sets the start and end value of the AnnTaskMonitor object. The end value is the value of iDataLocation, which is where the media data begins. This value also represents the length of the header information.

6. Sets the current task in the AnnTaskMonitor object. The current value is set to 8 (the number of bytes read), and the message is set to "Parsing AU Header..."

7. Reads the next integer in m_madisResource, which is the number of bytes of media data in the file. Sets the value to iDataSize. Sets the task progress monitor to reflect the 4 bytes that were read.

8. Reads the next integer in m_madisResource, which is the data format code. Sets the value to iDataFormat. Sets the task progress monitor to reflect the 4 bytes that were read.

9. Reads the next integer in m_madisResource, which is the sampling rate of the media data in the file. Sets the value to iSampleRate. Sets the task progress monitor to reflect the 4 bytes that were read.

10. Reads the next integer in m_madisResource, which is the number of channels in the media data in the file. Sets the value to iChannelCount. Sets the task progress monitor to reflect the 4 bytes that were read.
11. Reads the rest of the header information and sets the value to m_szUserData.

    The length of the rest of the header information is determined by subtracting the number of bytes already read (24) from the total length of the header information.
12. Sets the value of the task progress monitor to show that all header information has been read.
13. Ends the current task in the AnnTaskManager object.
14. Sets the value of m_fiFormatInfo by getting the appropriate value from the Hashtable object named m_htFormatInfo.
15. Sets the value of m_iBitsPerSample to 0 as a default.
16. Checks the value of iDataFormat against a series of values in the m_ htFormatInfo Hashtable object and sets the value of m_iBitsPerSample to the appropriate value.
17. Catches any errors or exceptions that may have been raised in the previous steps.
18. Calls the saveToAnnotation( ) method to save the annotation. See Section 8.8 for more information. This method should be called at the end of any implementation of the parse( ) method.

8.8 saveToAnnotation( ) Method

Example 8–7 shows the saveToAnnotation( ) method. This method should be called after the parse( ) method has successfully finished.

Example 8–7 saveToAnnotation( ) Method

```
public void saveToAnnotation( ){
     [1] m_annInst.setAttribute("MEDIA_SOURCE_FILE_FORMAT_CODE", "AUFF");
     m_annInst.setAttribute("MEDIA_SOURCE_FILE_FORMAT",
          "Next/Sun audio file format");

     [2] if (m_fiFormatInfo != null) {
          m_annInst.setAttribute("MEDIA_FORMAT_ENCODING",
               m_fiFormatInfo.getFormatString( ));
```

```
            m_annInst.setAttribute("MEDIA_FORMAT_ENCODING_CODE",
                 m_fiFormatInfo.getFormatCode( ));
        }

        [3] if(m_szUserData.trim( ).length( ) != 0)
            m_annInst.setAttribute("MEDIA_USER_DATA", m_szUserData);

        [4] m_annInst.setAttribute("AUDIO_BITS_PER_SAMPLE",
            new Integer(m_iBitsPerSample));
        m_annInst.setAttribute("AUDIO_SAMPLE_RATE",
            new Integer(m_iSampleRate));
        m_annInst.setAttribute("AUDIO_NUM_CHANNELS",
            new Integer(m_iChannelCount));
}
```

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

1. Sets the MEDIA_SOURCE_FILE_FORMAT_CODE and MEDIA_SOURCE_FILE_FORMAT attributes in the annotation to the values for an AU file.
2. If m_fiFileFormat has a value, sets its format string and format code to the MEDIA_FORMAT_ENCODING and MEDIA_FORMAT_ENCODING_CODE attributes in the annotation, respectively.
3. If m_szUserData has a value, sets it to the MEDIA_USER_DATA attribute in the annotation.
4. Sets the values of m_iBitsPerSample, m_iSampleRate, and m_iChannelCount to the AUDIO_BITS_PER_SAMPLE, AUDIO_SAMPLE_RATE, and AUDIO_NUM_CHANNELS attributes in the annotation, respectively.

To create a sub-annotation, a parser uses the AnnotationFactory to create a sub-annotation and attach it to m_annInst. However, AuParser does not create sub-annotations. See QtParser.java, which is the QuickTime parser included with Annotator (located in <*ORACLE_HOME*>/ord/Annotator/src/parsers.zip) for an example of a parser that creates sub-annotations.

8.9 extractSamples( ) Method

Example 8–8 shows the extractSamples( ) method. This method is invoked by AnnotationHandler.extractMedia( ).

Example 8–8 extractSamples( ) Method

```
public void extractSamples( ) throws ParserException{
```

```
     [1] m_sStatus.Report(Status.OUTPUT_MODE_STATUS, "AuParser does not
          support any sample extraction.");
     [2] m_annTaskMan.done( );
}
```

Annotator does not support sample extraction from an AU file. Instead of throwing an error or exception, this method performs the following operations:

1. Uses the Status object to print a message stating that AuParser.java does not support sample extraction. See Section 6.10 for more information.
2. Ends the current task with the AnnTaskManager.done( ) method. See Section 9.1 for more information.

See the QuickTime parser for an example of a parser that does support sample extraction.

8.10 FillFormatHashTable( ) Method

Example 8–9 shows the FillFormatHashTable( ) method, which uses the Hashtable.put( ) method to assign a value to each key in the m_htFormatString Hashtable object. See the Java 1.2 Javadoc for more information. This method is specific to AuParser.java and may not be needed for your parser.

Example 8–9 FillFormatHashTable( ) Method

```
private void FillFormatHashTable( ){
     m_htFormatInfo.put(SND_FORMAT_UNSPECIFIED, new FormatInfo("unspecified
          format", "UNSPECIFIED"));
     m_htFormatInfo.put(SND_FORMAT_MULAW_8, new FormatInfo("8-bit mu-law
          samples", "MULAW"));
     m_htFormatInfo.put(SND_FORMAT_LINEAR_8, new FormatInfo("8-bit linear
          samples", "LINEAR"));
     m_htFormatInfo.put(SND_FORMAT_LINEAR_16, new FormatInfo("16-bit linear
          samples", "LINEAR"));
     m_htFormatInfo.put(SND_FORMAT_LINEAR_24, new FormatInfo("24-bit linear
          samples", "LINEAR"));
     m_htFormatInfo.put(SND_FORMAT_LINEAR_32, new FormatInfo("32-bit linear
          samples", "LINEAR"));
     m_htFormatInfo.put(SND_FORMAT_FLOAT, new FormatInfo("floating-point
          samples", "FLOAT"));
     m_htFormatInfo.put(SND_FORMAT_DOUBLE, new FormatInfo("double-precision
          float samples", "DOUBLE"));
     m_htFormatInfo.put(SND_FORMAT_INDIRECT, new FormatInfo("fragmented sampled
          data", "FRAGMENTED"));
```

```
        m_htFormatInfo.put(SND_FORMAT_NESTED, new FormatInfo("nested format",
             "NESTED"));
        m_htFormatInfo.put(SND_FORMAT_DSP_CORE, new FormatInfo("DSP program", "DSP_
             CORE"));
        m_htFormatInfo.put(SND_FORMAT_DSP_DATA_8, new FormatInfo("8-bit fixed-point
             samples", "DSP_DATA"));
        m_htFormatInfo.put(SND_FORMAT_DSP_DATA_16, new FormatInfo("16-bit
             fixed-point samples", "DSP_DATA"));
        m_htFormatInfo.put(SND_FORMAT_DSP_DATA_24, new FormatInfo("24-bit
             fixed-point samples", "DSP_DATA"));
        m_htFormatInfo.put(SND_FORMAT_DSP_DATA_32, new FormatInfo("32-bit
             fixed-point samples", "DSP_DATA"));
        m_htFormatInfo.put(SND_FORMAT_UNKNOWN, new FormatInfo("unknown au
             format", "UNKNOWN"));
        m_htFormatInfo.put(SND_FORMAT_DISPLAY, new FormatInfo("non-audio display
             data", "DISPLAY"));
        m_htFormatInfo.put(SND_FORMAT_MULAW_SQUELCH, new FormatInfo("squelch
             format", "MULAW_SQUELCH"));
        m_htFormatInfo.put(SND_FORMAT_EMPHASIZED, new FormatInfo("16-bit linear
             with emphasis", "EMPHASIZED"));
        m_htFormatInfo.put(SND_FORMAT_COMPRESSED, new FormatInfo("16-bit linear
             with compression", "COMPRESSED"));
        m_htFormatInfo.put(SND_FORMAT_COMPRESSED_EMPHASIZED, new FormatInfo("16-bit
             linear with emphasis and compression", "COMPRESSED_EMPHASIZED"));
        m_htFormatInfo.put(SND_FORMAT_DSP_COMMANDS, new FormatInfo("Music Kit DSP
             commands", "DSP_COMMANDS"));
        m_htFormatInfo.put(SND_FORMAT_DSP_COMMANDS_SAMPLES, new FormatInfo("DSP
             commands samples", "DSP_COMMANDS_SAMPLES"));
        m_htFormatInfo.put(SND_FORMAT_ADPCM_G721, new FormatInfo("adpcm G721",
             "ADPCM_G721"));
        m_htFormatInfo.put(SND_FORMAT_ADPCM_G722, new FormatInfo("adpcm G722",
             "ADPCM_G722"));
        m_htFormatInfo.put(SND_FORMAT_ADPCM_G723_3, new FormatInfo("adpcm G723_3",
             "ADPCM_G723_3"));
        m_htFormatInfo.put(SND_FORMAT_ADPCM_G723_5, new FormatInfo("adpcm G723_5",
             "ADPCM_G723_5"));
        m_htFormatInfo.put(SND_FORMAT_ALAW_8, new FormatInfo("8-bit a-law samples",
             "ALAW"));
    }
```

9

Annotator Parser API Reference Information

This chapter contains reference material for the classes and methods that inexperienced users will need to write a custom Annotator parser. See the Javadoc included with the Annotator installation for complete reference information.

To create a custom parser, in addition to using these APIs to create a Java class, you must write a parser descriptor XML file and add it to the <*ORACLE_HOME*>\ord\Annotator/lib/descriptors/parsers directory. For an example of a parser descriptor XML file, see <*ORACLE_HOME*>\ord\Annotator/lib/descriptors/parsers/AuParser.xml.

9.1 Class oracle.ord.media.annotator.handlers.AnnTaskManager

This section presents reference information on the methods of the AnnTaskManager class, which creates an annotation task manager. The **annotation task manager** object is one of the components involved in monitoring tasks as they are being performed by an AnnotationHandler object (or annotation handler). Whenever a task is started by an annotation handler, an annotation task manager and an annotation task monitor are created. The **annotation task manager** runs on the server side; it tracks the progress of the task on the database server. The annotation task monitor runs on the client side; it tracks the progress value and messages from the returned annotation task monitor instance through a **task progress monitor**.

For more information on the annotation task monitor, see Section 6.4.

This class extends java.lang.Object.

This class contains the following fields:

- protected boolean m_bInitialized

- protected int m_iIterCounter
- protected int m_iTaskCurrent
- protected int m_iTaskEnd
- protected int m_iTaskStart
- protected java.lang.String m_szMessage

addIterCounter( )

Format

public void addIterCounter(int iIterCounter)

Description

Adds the given number to the counter.

Parameters

iIterCounter
The value to be added to the counter.

Return Value

None.

Exceptions

None.

Example

```
m_annTaskMan.addIterCounter(4);
```

decrIterCounter( )

Format

public void decrIterCounter( )

Description

Decreases the value of the counter by one.

Parameters

None.

Return Value

None.

Exceptions

None.

Example

```
m_annTaskMan.decrIterCounter( );
```

done( )

Format

public void done( )

Description

Signifies that the current task is complete.

Parameters

None.

Return Value

None.

Exceptions

None.

Example

See Section 8.9 for an example of this method.

getIterCounter( )

Format

public int getIterCounter( )

Description

Gets the current value of the counter.

Parameters

None.

Return Value

This method returns the current value of the counter.

Exceptions

None.

Example

```
int counter = m_annTaskMan.getIterCounter( );
```

getMessage( )

Format

public java.lang.String getMessage( )

Description

Gets the current message of the task progress monitor.

Parameters

None.

Return Value

This method returns the current message of the task progress monitor.

Exceptions

None.

Example

```
String message = m_annTaskMan.getMessage( );
```

getTaskCurrent( )

Format

public int getTaskCurrent( )

Description

Gets the current value of the task progress monitor.

Parameters

None.

Return Value

This method returns the current value of the task progress monitor.

Exceptions

None.

Example

```
int progress = m_annTaskMan.getTaskCurrent( );
```

getTaskEnd( )

Format

public int getTaskEnd( )

Description

Gets the ending value of the task progress monitor.

Parameters

None.

Return Value

This method returns the the ending value of the task progress monitor.

Exceptions

None.

Example

```
int end = m_annTaskMan.getTaskEnd( );
```

getTaskStart( )

Format

public int getTaskStart( )

Description

Gets the starting value of the task progress monitor.

Parameters

None.

Return Value

This method returns the starting value of the task progress monitor.

Exceptions

None.

Example

See the isInitialized( ) method for an example of this method.

incrIterCounter( )

Format

public void incrIterCounter( )

Description

Increases the value of the counter by one.

Parameters

None.

Return Value

None.

Exceptions

None.

Example

```
m_annTaskMan.incrIterCounter( );
```

incrTaskCurrent( )

Format

public void incrTaskCurrent(int iTaskToAdd)

Description

Adds the given value to the current value of the task progress monitor.

Parameters

iTaskToAdd
The amount to add to the current value of the task progress monitor.

Return Value

None.

Exceptions

None.

Example

```
m_annTaskMan.incrTaskCurrent(4);
```

isDone( )

Format

public boolean isDone( )

Description

Determines if the current task has been completed.

Parameters

None.

Return Value

This method returns true if the current task has been completed; false otherwise.

Exceptions

None.

Example

```
if(m_annTaskMan.isDone( ) == false)
     m_annTaskMan.setIterCounter(0);
```

isInitialized( )

Format

public boolean isInitialized( )

Description

Determines if the annotation task monitor has been initialized. If it has been initialized, then you will be able to use the getTaskStart( ) and getTaskEnd( ) methods.

Parameters

None.

Return Value

This method returns true if the annotation task monitor has been initialized; false otherwise.

Exceptions

None.

Example

```
if(m_annTaskMan.isInitialized( ))
     m_annTaskMan.getTaskStart( );
```

setIterCounter( )

Format

public void setIterCounter(int iIterCounter)

Description

Sets the counter to keep track of an iterative process. When the done( ) method is called, the counter decreases by one. The isDone( ) method returns true if the counter is zero.

Parameters

iIterCounter
The initial value of the counter. The default value is 1.

Return Value

None.

Exceptions

None.

Example

See the isDone( ) method for an example of this method.

setMessage( )

Format

public void setMessage(java.lang.String szMessage)

Description

Sets the message of the task progress monitor.

Parameters

szMessage
The message to be set.

Return Value

None.

Exceptions

None.

Example

```
m_annTaskMan.setMessage("Parsing AU Header...");
```

setTask( )

Format

public void setTask(int iTaskStart, int iTaskEnd)

Description

Sets the start and end values of the task progress monitor.

Parameters

iTaskStart
The starting value of the task progress monitor.

iTaskEnd
The ending value of the task progress monitor.

Return Value

None.

Exceptions

None.

Example

See Section 8.7 for an example of this method.

setTaskCurrent(int)

Format

public void setTaskCurrent(int iTaskCurrent)

Description

Sets the current value of the task progress monitor.

Parameters

iTaskCurrent
The value to be set for the task progress monitor.

Return Value

None.

Exceptions

None.

Example

See Section 8.7 for an example of this method.

setTaskCurrent(int,String)

Format

public void setTaskCurrent(int iTaskCurrent, java.lang.String szMessage)

Description

Sets the current value and the message of the task progress monitor.

Parameters

iTaskCurrent
The value to be set for the task progress monitor.

szMessage
The message to be set for the task progress monitor.

Return Value

None.

Exceptions

None.

Example

See Section 8.7 for an example of this method.

9.2 Class oracle.ord.media.annotator.handlers.annotation.AnnotationFactory

This section presents reference information on the methods of the AnnotationFactory class. This class is the factory class for annotations; it contains two sub-factories (for parser descriptors and annotation descriptors), which are used to create parsers and annotations. The AnnotationFactory class can also create annotations by name.

This class extends java.lang.Object.

createAnnotationByName( )

Format

public oracle.ord.media.annotator.annotations.Annotation createAnnotationByName(java.lang.String szAnnName)

Description

Instantiates an annotation by getting the annotation descriptor from the annotation descriptor factory.

Parameters

szAnnName
The name of the new annotation.

Return Value

This method returns a newly created annotation.

Exceptions

oracle.ord.media.annotator.handlers.annotation.AnnotationFactoryException

Example

See Section 5.6 for an example of this method.

9.3 Class oracle.ord.media.annotator.parsers.Parser

This section presents reference information on the methods of the Parser class. This class is the base class for all parsers; you must extend this class to write your own parser. The Parser class is an abstract class that defines the functions that are expected from a parser: parsing metadata, extracting samples, and saving metadata to annotations. You must implement the methods that provide these functions in all subclasses of the Parser class.

The Parser class operates on the basis of an underlying wrapper of a DataInputStream object, which is used to read objects during the parsing process. The Parser object is associated with an annotation instance that is populated with the metadata that the parser finds in the stream. The Parser object is associated with an instance of the AnnTaskManager class that provides the GUI with progress information related to the parsing of the media data. The Parser object is associated with an AnnotationFactory object to create a sub-annotation of the associated annotation instance.

This class extends java.lang.Object and contains the following fields:

- protected AnnotationFactory m_annFactory

  This is the AnnotationFactory object that is used to create sub-annotations.

- protected oracle.ord.media.annotator.annotations.Annotation m_annInst

  This is the annotation that is processed by the parser.

- protected AnnTaskManager m_annTaskMan

  This is the AnnTaskManager object that is used to produce progress information.

- protected boolean m_bExtractable

  This determines if the parser can extract samples from the annotation. The default is false.

- protected MADataInputStream m_madisResource

  This is the media source to be processed.

- protected oracle.ord.media.annotator.descriptors.ParserDesc m_pd

  This is the in-memory representation of the parser descriptor XML file.

- protected oracle.ord.media.annotator.utils.Status m_sStatus

  This is the Status object that is used to produce trace information for the GUI.

extractSamples( )

Format

public abstract void extractSamples( )

Description

Extracts samples from the current media source and sets the samples in the current annotation instance. The AnnotatorEngine object sets the m_madisResource field and the m_annInst field automatically with the setSource( ) and setAnnotation( ) methods, respectively.

Parameters

None.

Return Value

None.

Exceptions

ParserException

See the Annotator Javadoc for more information.

Example

See Section 8.9 for an example of this method.

parse( )

Format

public abstract void parse( )

Description

Parses the source and extracts the metadata. The AnnotatorEngine object sets the m_madisResource field and the m_annInst field automatically with the setSource( ) and setAnnotation( ) methods, respectively.

After running this method, you should call the saveToAnnotation( ) method to set the metadata in the annotation.

Parameters

None.

Return Value

None.

Exceptions

ParserException

See the Annotator Javadoc for more information.

Example

See Section 8.7 for an example of this method.

saveToAnnotation( )

Format

public abstract void saveToAnnotation( )

Description

Sets the extracted metadata in the annotation in the m_annInst field. The annotation is set by the setAnnotation( ) method automatically before parsing.

You should call this method immediately after parsing the media source; this ensures that the annotation is modified only if parsing is successful.

Parameters

None.

Return Value

None.

Exceptions

None.

Example

See Section 8.7 for an example of this method.

9.4 Class oracle.ord.media.annotator.utils.MADataInputStream

This section presents reference information on the methods of the MADataInputStream class. This class provides methods to read the following types from an input stream:

- 16-bit fixed-point numbers
- 32-bit fixed-point numbers
- 80-bit extended floating-point numbers
- Audio Video Interleaved (AVI) language codes
- Date (both as an int and as a formatted string)
- Four Character Codes (FourCC)
- int (both big-endian and little-endian)
- long (both big-endian and little-endian)
- Pascal string (both variable-sized and fixed-size)
- QuickTime language codes
- short (both big-endian and little-endian)
- unsigned int (both big-endian and little-endian)
- unsigned long (both big-endian and little-endian)
- unsigned short (both big-endian and little-endian)

This class extends java.lang.Object.

available( )

Format

public int available( )

Description

Returns the number of bytes that can be read or skipped in this input stream without blocking by the next caller of a method for this input stream.

Parameters

None.

Return Value

This method returns the number of bytes that can be read or skipped without blocking by the next caller.

Exceptions

java.io.IOException

Example

```
int availbytes = m_madisResource.available( );
```

close( )

Format

public void close( )

Description

Closes the input stream and releases any system resources associated with the stream.

Parameters

None.

Return Value

None.

Exceptions

None.

Example

```
if(m_madisResource.getLeft( ) == 0)
     m_madisResource.close( );
```

endBlock( )

Format

public void endBlock(FourCC fccChunk)

Description

Skips all unread bytes in the marked block.

Parameters

fccChunk
The Four Character Code used to identify the block; it is used for reading purposes only.

Return Value

None.

Exceptions

java.io.IOException

Example

```
FourCC code = m_madisResource.readFourCC( );
long size = m_madisResource.readLong( );
m_madisResource.startBlock(size);
...
m_madisResource.endBlock(code);
```

getBytesRead( )

Format

public long getBytesRead( )

Description

Gets the total number of bytes read from the input stream.

Parameters

None.

Return Value

This method returns the total number of bytes read.

Exceptions

None.

Example

```
long bytesRead = m_madisResource.getBytesRead( );
```

getLeft( )

Format

public long getLeft( )

Description

Gets the number of unread bytes in the input stream.

Parameters

None.

Return Value

This method returns the number of unread bytes in the input stream.

Exceptions

None.

Example

See the close( ) method for an example of this method.

isLittleEndian( )

Format

public boolean isLittleEndian( )

Description

Checks to see whether or not the input stream is able to read data that is in the little-endian format.

Parameters

None.

Return Value

This method returns true if the stream is able to read little-endian data; false otherwise.

Exceptions

None.

Example

See the setLittleEndian( ) method for an example of this method.

mark( )

Format

public void mark(int readLimit)

Description

Marks the current position in the input stream. A subsequent call to the reset( ) method repositions the stream at the last marked position.

Parameters

readLimit
The maximum number of bytes that can be read before the mark position becomes invalid.

Return Value

None.

Exceptions

None.

Example

```
m_madisResource.mark(5000);
int i = 128;
if(i == m_madisResource(skipBytes(i))
     int data = m_madisResource.readInt( );
m_madisResource.reset( );
```

read(byte[ ])

Format

public final int read(byte[ ] b)

Description

Reads some number of bytes from the input stream and stores them in the buffer array b. The number of bytes actually read is returned as an integer. To ensure that some data will always be read into the buffer, this method blocks until input data is available, end-of-file is detected, or an exception is thrown.

If b is null, a NullPointerException is thrown. If the length of b is 0, then no bytes are read and 0 is returned; otherwise, there is an attempt to read at least 1 byte. If no byte is available because the stream is at end-of-file, the value -1 is returned; otherwise, at least 1 byte is read and stored in b.

Parameters

b
The buffer into which the data will be read.

Return Value

This method returns the number of bytes that were read.

Exceptions

java.io.IOException

Example

```
byte[ ] buffer = new byte[4000];
m_madisResource.read(buffer);
```

read(byte[ ], int, int)

Format

public final int read(byte[ ] b, int off, int len)

Description

Reads a number of bytes (up to the value of len) of data from the input stream into an array of bytes. An attempt is made to read as many as len bytes, but a smaller number may be read, possibly 0. The number of bytes actually read is returned as an integer.

If len is 0, then no bytes are read and 0 is returned; otherwise, there is an attempt to read at least 1 byte. If no byte is available because the stream is at end-of-file, the value -1 is returned; otherwise, at least 1 byte is read and stored into the array.

To ensure that some data will always be read into the buffer, this method blocks until input data is available, end-of-file is detected, or an exception is thrown.

If b is null, a NullPointerException is thrown. If the value of off is negative, or len is negative, or off+len is greater than the length of the array b, then an IndexOutOfBoundsException is thrown. If the first byte cannot be read for any reason other than end-of-file, then an IOException is thrown. In particular, an IOException is thrown if the input stream has been closed.

Parameters

b
The buffer into which the data will be read.

off
The offset from the beginning of the buffer at which the data will be read.

len
The maximum number of bytes to be read.

Return Value

This method returns the number of bytes that were read.

Exceptions

java.io.IOException

Example

```
byte[ ] buffer = new byte[4000];
m_madisResource.read(buffer,64,128);
```

readAVILanguage( )

Format

public AVILanguage readAVILanguage( )

Description

Reads the next AVI language code in the underlying input stream.

See the Javadoc for more information about the AVILanguage class.

Parameters

None.

Return Value

This method returns the AVI language code that was read from the input stream.

Exceptions

java.io.IOException

Example

```
AVILanguageCode code = m_madisResource.readAVILanguage( );
```

readByte( )

Format

public byte readByte( )

Description

Reads the next byte from the input stream.

Parameters

None.

Return Value

This method returns the byte that was read from the input stream.

Exceptions

java.io.IOException

Example

```
byte b = m_madisResource.readByte( );
```

readByteArray(byte[ ],int)

Format

public int readByteArray(byte[ ] bBuffer, int iNumBytesToRead)

Description

Reads the given number of bytes from the underlying data input stream into the given byte array.

Parameters

bBuffer
The byte array into which the data will be read.

iNumBytesToRead
The number of bytes to be read.

Return Value

This method returns the number of bytes that were read.

Exceptions

java.io.IOException

Example

```
int i = 256;
byte[ ] b = new byte[i];
if(i == m_madisResource.readByteArray(b,i))
     System.out.println("Read successful");
```

readByteArray(int)

Format

public byte[ ] readByteArray(int iNumBytesToRead)

Description

Reads the given number of bytes from the underlying data input stream into a byte array.

Parameters

iNumBytesToRead
The number of bytes to be read.

Return Value

This method returns the byte array that was read.

Exceptions

java.io.IOException

Example

```
int i = 256;
byte[ ] b = new byte[i];
b = m_madisResource.readByteArray(i);
```

readColor48( )

Format

public long readColor48( )

Description

Reads 6 bytes from the data stream and returns a value in the primitive type long that represents the color in RGB format.

This is used for the QuickTime file format.

Parameters

None.

Return Value

This method returns the value of the color in RGB format in the primitive type long.

Exceptions

java.io.IOException

Example

```
long RGB = m_madisResource.readColor48( );
```

readDate( )

Format

public java.util.Date readDate( )

Description

Reads the next java.util.Date object from the underlying input stream.

Parameters

None.

Return Value

This method returns the java.util.Date object that was read from the input stream.

Exceptions

java.io.IOException

Example

```
java.util.Date date = m_madisResource.readDate( );
```

readDate(int,String)

Format

public java.util.Date readDate(int iLen, java.lang.String szPattern)

Description

Returns the bytes read from the data stream as a java.util.Date object, using the given named pattern.

Parameters

iLen
The number of bytes to be read.

szPattern
The date pattern of the bytes, following the specification of java.text.SimpleDateFormat.

Return Value

This method returns a java.util.Date object that was read.

Exceptions

java.io.IOException

Example

```
java.util.Date date = m_madisResource.readDate(13,"yyyy.MM.dd hh:mm a")
```

readExtended( )

Format

public Extended readExtended( )

Description

Reads the next 80-bit, extended floating-point number in the underlying input stream.

See the Javadoc for more information about the Extended class.

Parameters

None.

Return Value

This method returns the 80-bit, extended floating-point number that was read from the input stream.

Exceptions

java.io.IOException

Example

```
Extended number = m_madisResource.readExtended( );
```

readFixedPoint16( )

Format

public FixedPoint16 readFixedPoint16( )

Description

Reads the next 16-bit, fixed-point number in the underlying input stream.

See the Javadoc for more information about the FixedPoint16 class.

Parameters

None.

Return Value

This method returns the 16-bit, fixed-point number that was read from the input stream.

Exceptions

java.io.IOException

Example

```
FixedPoint16 number = m_madisResource.readFixedPoint16( );
```

readFixedPoint32( )

Format

public FixedPoint32 readFixedPoint32( )

Description

Reads the next 32-bit, fixed-point number in the underlying input stream.

See the Javadoc for more information about the FixedPoint32 class.

Parameters

None.

Return Value

This method returns the 32-bit, fixed-point number that was read from the input stream.

Exceptions

java.io.IOException

Example

```
FixedPoint32 number = m_madisResource.readFixedPoint32( );
```

readFourCC( )

Format

public FourCC readFourCC( )

Description

Reads the next Four Character Code in the underlying input stream.

See the javadoc for more information about the FourCC class.

Parameters

None.

Return Value

This method returns the Four Character Code that was read from the input stream.

Exceptions

java.io.IOException

Example

```
FourCC code = m_madisResource.readFourCC( );
```

readInt( )

Format

public int readInt( )

Description

Reads the next int from the input stream.

Parameters

None.

Return Value

This method returns the int that was read from the input stream.

Exceptions

None.

Example

See the mark( ) method for an example of this method.

readLong( )

Format

public long readLong( )

Description

Reads the next value of the primitive type long from the underlying input stream.

Parameters

None.

Return Value

This method returns the value of the primitive type long that was read from the input stream.

Exceptions

java.io.IOException

Example

```
long number = m_madisResource.readLong( );
```

readPascalString( )

Format

public java.lang.String readPascalString( )

Description

Reads the next Pascal string from the underlying input stream and returns the contents of the Pascal string as a Java String object.

Parameters

None.

Return Value

This method returns the contents of the Pascal string, as a Java String object.

Exceptions

java.io.IOException

Example

```
String pascal = m_madisResource.readPascalString( );
```

readPascalString(int)

Format

public java.lang.String readPascalString(int iNumBytesToRead)

Description

Reads the next Pascal string from the underlying input stream and returns the contents of the Pascal string as a Java String object.

Parameters

iNumBytesToRead
The number of bytes to read from the input stream.

Return Value

This method returns the contents of the Pascal string, as a Java String object.

Exceptions

java.io.IOException

Example

```
String pascal = m_madisResource.readPascalString(32);
```

readPascalString(Short)

Format

public java.lang.String readPascalString(java.lang.Short sLengthSize)

Description

Reads the next enhanced Pascal string from the underlying input stream and returns the contents of the Pascal string as a Java String object.

An enhanced Pascal string is a Pascal string with a string length of 8, 16, or 32 bits set at the beginning, followed by the contents of the string. The length must be 8, 16, or 32 bits.

Parameters

sLengthSize
The number of bits in the string. It must be 8, 16, or 32.

Return Value

This method returns the contents of the Pascal string, as a Java String object.

Exceptions

java.io.IOException

Example

```
String pascal = m_madisResource.readPascalString(32);
```

readQTLanguage( )

Format

public QTLanguage readQTLanguage( )

Description

Reads the next QuickTime language code from the underlying input stream.

See the Javadoc for more information about the QTLanguage object.

Parameters

None.

Return Value

This method returns the QuickTime language code that was read from the input stream.

Exceptions

java.io.IOException

Example

```
QTLanguage code = m_madisResource.readQTLanguage( );
```

readRectangle( )

Format

public Rectangle readRectangle( )

Description

Reads 8 bytes of data from the input stream, which are interpreted as the coordinates of a rectangle.

This is used for the QuickTime and RIFF data formats.

Parameters

None.

Return Value

This method returns the 8 bytes that were read from the input stream, as a Rectangle object.

Exceptions

java.io.IOException

Example

```
Rectangle size = m_madisResource.readRectangle( );
```

readShort( )

Format

public short readShort( )

Description

Reads the next short in the input stream.

Parameters

None.

Return Value

This method returns the short that was read from the input stream.

Exceptions

None.

Example

```
short number = m_madisResource.readShort( );
```

readString( )

Format

public java.lang.String readString(int iNumBytesToRead)

Description

Reads the given number of bytes from the underlying input stream and formats them as a String.

Parameters

iNumBytesToread
The number of bytes to be read.

Return Value

This method returns the String that was read from the input stream.

Exceptions

java.io.IOException

Example

```
String info = m_madisResource.readString(24);
```

readUnsignedByte( )

Format

public int readUnsignedByte( )

Description

Reads the next unsigned byte from the underlying input stream.

Parameters

None.

Return Value

This method returns the unsigned byte that was read from the input stream object.

Exceptions

java.io.IOException

Example

```
int number = m_madisResource.readUnsignedByte( );
```

readUnsignedInt( )

Format

public long readUnsignedInt( )

Description

Reads the next unsigned int from the underlying input stream.

Parameters

None.

Return Value

This method returns the unsigned int that was read from the input stream.

Exceptions

java.io.IOException

Example

```
long number = m_madisResource.readUnsignedInt( );
```

readUnsignedShort( )

Format

public int readUnsignedShort( )

Description

Reads the next unsigned short from the underlying input stream.

Parameters

None.

Return Value

This method returns the short that was read from the input stream.

Exceptions

java.io.IOException

Example

```
int number = m_madisResource.readUnsignedShort( );
```

reset( )

Format

public void reset( )

Description

Repositions the underlying input stream to the point at which the last mark was placed.

Parameters

None.

Return Value

None.

Exceptions

java.io.IOException

Example

See the mark( ) method for an example of this method.

setLittleEndian( )

Format

public void setLittleEndian(boolean bLittleEndian)

Description

Sets whether or not the input stream can read data that is stored in little-endian format.

Parameters

bLittleEndian
Determines whether or not the stream can read data that is stored in little-endian format.

Return Value

None.

Exceptions

None.

Example

```
if(m_madisResource.isLittleEndian( ))
     m_madisResource.setLittleEndian(false);
```

skipBytes(int)

Format

public int skipBytes(int iNum)

Description

Skips the given number of bytes in the underlying input stream.

Parameters

iNum
The number of bytes to be skipped.

Return Value

This method returns the number of bytes to be skipped.

Exceptions

java.io.IOException

Example

See the mark( ) method for an example of this method.

skipBytes(long)

Format

public long skipBytes(long lNum)

Description

Skips the given number of bytes in the underlying input stream.

Parameters

lNum
The number of bytes to be skipped.

Return Value

This method returns the number of bytes that was skipped.

Exceptions

java.io.IOException

Example

```
long number = 256;
if(number == m_madisResource.skipBytes(number)
     int data = m_madisResource.readInt( );
```

startBlock( )

Format

public void startBlock(long lSizeLeft)

Description

Sets a block that begins at the current point in the stream and contains the given number of bytes.

This construct is useful for file formats where a Four Character Code serves as the identifier of a block, followed by the number of bytes in the block.

Parameters

lSizeLeft
The number of bytes in the block.

Return Value

None.

Exceptions

None.

Example

```
m_madisResource.startBlock(128);
```

10

Creating New Annotation Types

In addition to the supplied annotation types, you can use *inter*Media Annotator to create your own annotation types in order to better meet the needs of your applications. For example, the owner of an online sales database can define annotations containing inventory and price information alongside the media data and the extracted metadata.

For a complete example of a user-defined annotation type, see <*ORACLE_HOME*>\ord\Annotator\demo\examples\SampleInventoryAnn.xml, which is included in the Annotator installation.

10.1 Writing a New Annotation Type

You define a new annotation type in an XML file. The XML file must follow the AnnotationDescriptor document type definition (DTD), found at<*ORACLE_HOME*>\ord\Annotator\lib\descriptors\annotation\AnnotationDescriptor.dtd.

When you finish writing the XML file, you should save it to <*ORACLE_HOME*>\ord\Annotator\lib\descriptors\annotations.

The DTD describes the AnnotationDescriptor DTD, which contains two elements: AnnotationProperties and AttributeDescriptors.

10.1.1 AnnotationProperties Element

The AnnotationProperties element contains elements that provide information about the annotation as a whole. These elements are the following:

- Name
- Version
- Description

- Extends
- Contains
- ClassName
- IconFileName.

Name and Version are required elements. They contain the name of the new annotation type and the version number, respectively.

Description is an optional element that contains a brief description of the annotation as a whole.

Extends is an optional element. It contains the name of another annotation type, which your new annotation type will extend; that is, your new annotation type will include all the attribute definitions from the given annotation type, as well as any additional attributes that you define. However, you cannot write over the attributes that are inherited from the existing annotation type; you can create only new attributes. If you want to create an annotation type that is not related to another annotation type, do not include the Extends element.

Contains, ClassName, and IconFileName are reserved elements; do not assign values to these elements.

10.1.2 AttributeDescriptors Element

The AttributeDescriptors element contains one or more AttributeDescriptor elements. An AttributeDescriptor element contains one AttributeProperties element.

An AttributeProperties element contains elements that provide information about the specific attributes of your new annotation type. These elements are the following:

- AttributeName
- AttributeType
- AttributeTypePattern
- AttributeAlias
- AttributeDescription
- AttributeDefaultValue.

AttributeName is a required element that contains the name of your new attribute.

AttributeType is a required element that contains the Java object type of the attribute value. AttributeType must be a Java object type; Java primitives are not allowed in your XML document. For example, if you want to use an integer, do not use int, but rather java.lang.Integer.

Almost any Java object type can be used as the AttributeType, as long as the Java object type defines two valid methods: public String toString( ) and public static *Object* valueOf(String), where *Object* is the Java object type. These methods return the contents of the object as a valid String and returns the contents of a given String as a valid object of type *Object*, respectively.

The class java.util.Date is a special case; it does not use the previous methods to provide a String representation of the contents of the object. Instead, it uses the AttributeTypePattern element. This element (which should be used only if the AttributeType is java.lang.Date) specifies the String pattern that should be used when displaying the date. The pattern follows the syntax in java.text.SimpleDateFormat.

AttributeTypePattern is an optional element.

AttributeAlias, AttributeDescription, and AttributeDefaultValue are optional elements that define a shorter attribute name for display purposes, a short description of the attribute, and the default value of the attribute to be inserted in the annotation, respectively.

10.1.3 Element Hierarchy

In general, the structure of your XML document should be similar to the following:

```
<?xml version="1.0">
<!DOCTYPE AnnotationDescriptor SYSTEM "AnnotationDescriptor.dtd>
<AnnotationDescriptor>
   <AnnotationProperties>
      <Name>...</Name>
      <Version>...</Version>
      <Description>...</Description>
      <Extends>...</Extends>
   </AnnotationProperties>
   <AttributeDescriptors>
      <AttributeDescriptor>
         <AttributeProperties>
            <AttributeName>...</AttributeName>
            <AttributeType>...</AttributeType>
            <AttributeTypePattern>...</AttributeTypePattern>
            <AttributeAlias>...</AttributeAlias>
```

```
            <AttributeDescription>...</AttributeDescription>
            <AttributeDefaultValue>...</AttributeDefaultValue>
          </AttributeProperties>
        </AttributeDescriptor>
        <AttributeDescriptor>
        .
        .
        .
        </AttributeDescriptor>
    </AttributeDescriptors>
</AnnotationDescriptor>
```

10.2 Using a New Annotation Type

Once you have written your new XML file and included it in the <*ORACLE_HOME*>\ord\Annotator\lib\descriptors\annotations directory, you will be able to use your new annotation type in the same way that you use the predefined annotation types. See Section 3.1 and Section 3.4 for more information on creating new annotations.

Note: An XML file is space-sensitive; "java.lang.Double" is valid, while "java.lang.Double " is invalid. Be careful that your XML file does not contain extraneous spaces because it could lead to errors.

Part IV

Appendixes

Part IV contains four appendixes:

- Appendix A, "Querying Stored Annotations"
- Appendix B, "Supported Formats"
- Appendix C, "Defined Annotation Attributes"
- Appendix D, "Frequently Asked Questions"

A

Querying Stored Annotations

After the media data and the annotation are uploaded into an Oracle database, you can use Oracle9*i* Text to perform a query on the annotation (which is saved in the database in XML form).

The following PL/SQL code excerpt is an example of how to build an Oracle9*i* Text index on the VideoStorage table. This code excerpt generates an Oracle9*i* Text index on the comments field of the vsrc column of the VideoStorage table, with the list preference, as well as the XML tags, defined in the section group:

```
-- create a preference
execute ctx_ddl.create_preference('ANNOT_WORDLIST', 'BASIC_WORDLIST');
execute ctx_ddl.set_attribute('ANNOT_WORDLIST', 'stemmer', 'ENGLISH');
execute ctx_ddl.set_attribute('ANNOT_WORDLIST', 'fuzzy_match', 'ENGLISH');
...

-- section group
execute ctx_ddl.create_section_group('MOVIEANN_TAGS',
                                     'xml_section_group');
execute ctx_ddl.add_zone_section('MOVIEANN_TAGS', 'MOVIECASTTAG',
                                 'MOVIE_CAST');
execute ctx_ddl.add_zone_section('MOVIEANN_TAGS',
                                 'MEDIACOPYRIGHTTAG',
                                 'MEDIA_COPYRIGHT');
execute ctx_ddl.add_zone_section('MOVIEANN_TAGS',
                                 'MEDIASOURCEFILEFORMATTAG',
                                 'MEDIA_SOURCE_FILE_FORMAT');
...
CREATE INDEX videoIdx ON VideoStorage(vsrc.comments) INDEXTYPE IS
     CTXSYS.CONTEXT PARAMETERS('stoplist CTXSYS.EMPTY_STOPLIST wordlist
     ANN_WORDLIST filter CTXSYS.NULL_FILTER section group MOVIEANN_TAGS');
```

See Section 2.2.2 for more information on creating the VideoStorage table.

The following PL/SQL code excerpt is an example of how to query the VideoStorage table:

```
-- Perform a query
select id, score(99)
from VideoStorage V
where
     CONTAINS(V.vsrc.comments, '(John Doe) WITHIN MOVIECASTTAG',
     99) > 0;
```

The preceding query returns the clip identification number and the relevancy score (generated by Oracle9*i* Text) of the video clips that contain John Doe in the MOVIE_CAST attribute of the associated annotation.

A copy of the preceding PL/SQL statements is available in:

`<`*`ORACLE_HOME`*`>\ord\Annotator\demo\examples\SampleCode.sql`

For more information, see the Oracle9*i* Text documentation.

B

Supported Formats

*inter*Media Annotator supports the following file formats:

- Apple QuickTime 4.0
- Microsoft AVI
- AIFF / AIFC
- AU
- WAV
- Audio MPEG I / II, all layers
- RealMedia
- Audio compact disc
- JPEG / JFIF
- GIF87a / GIF89a
- BMP
- TIFF

Table B–1 shows the media source formats that have built-in support from *inter*Media Annotator and what, if any, extraction capabilities are offered by the corresponding built-in parser.

Table B–1 Built-in Parsers

File Format	MIME type	Extraction
Apple QuickTime 4.0	video/quicktime	Text track
		Video frame (with Apple QuickTime Library only)
RIFF	video/x-msvideo	None
	application/x-troff-msvideo	
	audio/x-wav	
AIFF	audio/x-aiff	None
	audio/x-aif	
RealMedia	video/x-realvideo	None
	video/x-realaudio	
Sun Audio	audio/basic	None
MPEG I/II Audio	audio/x-mpeg	None
Audio CD	audio/x-cd-cdda	Audio sample
JPEG/JFIF	image/jpeg	None
	image/jpg	
GIF	image/gif	None
BMP	image/bmp	None
TIFF	image/tiff	None

C

Defined Annotation Attributes

The tables in this appendix show the defined annotation attributes that are included with *inter*Media Annotator.

Table C–1 MediaAnn Codes

Attribute	Description
MEDIA_TITLE	Title of the media
MEDIA_DESCRIPTION	Description of the media
MEDIA_INFORMATION	Information on the media
MEDIA_COPYRIGHT	Copyright information of the media
MEDIA_CREATION_TIME	Creation time of the media (for example, Mon Dec 13 19:29:04 EST 1999)
MEDIA_PRODUCER	Producer of the media
MEDIA_DURATION	Duration of the media, in the form hour:minute:second:mantissa, where mantissa is the fraction of a second in the units defined in MEDIA_TIMESCALE
MEDIA_TRACK_ID	Track identifier for the annotation
MEDIA_TIMESCALE	Number of units in a second
MEDIA_CONTENT_DATE	Creation date of the media content
MEDIA_MODIFICATION_TIME	Modification time of type java.util.Date
MEDIA_CREDITS	Credits for content providers
MEDIA_SIZE	Size of the media source, in bytes

Table C–1 MediaAnn Codes(Cont.)

Attribute	Description
MEDIA_FORMAT_ENCODING_CODE	Short form (4 characters) of the media encoding
MEDIA_FORMAT_ENCODING	Format of the media encoding
MEDIA_USER_DATA	String containing miscellaneous user data
MEDIA_LANGUAGE	Language of the media
MEDIA_BITRATE	Bit rate of the media (in bits per second)
MEDIA_CATEGORY	Media category or genre
MEDIA_AUTHORING_TOOL	Authoring tool used to create the media
MEDIA_SOURCE_URL	Location or URL of the parsed media source
MEDIA_SOURCE_PROTOCOL	URL protocol of the media source
MEDIA_SOURCE_MIME_TYPE	MIME type of the media and its samples
MEDIA_SOURCE_DIRECTORY	Directory where the source is stored
MEDIA_SOURCE_FILENAME	File name of the source
MEDIA_SOURCE_FILE_FORMAT	Media file format
MEDIA_SOURCE_FILE_FORMAT_CODE	Short form (4 characters) of media file format
MEDIA_SOURCE_STREAMABLE	The streaming server for which the media is optimized, if any

Table C–2 AudioAnn Codes (extends MediaAnn)

Attribute	Description
AUDIO_BITS_PER_SAMPLE	Number of bits per sound sample
AUDIO_SAMPLE_RATE	Audio sampling rate (in samples per second)
AUDIO_NUM_CHANNELS	Number of audio channels
AUDIO_ARTIST	Main artist for the audio clip

Table C–3 VideoAnn Codes (extends MediaAnn)

Attribute	Description
VIDEO_FRAME_RATE	Video frame rate (in frames per second)

Table C–3 VideoAnn Codes (extends MediaAnn)(Cont.)

Attribute	Description
VIDEO_FRAME_SIZE	Video frame size (in bytes)
VIDEO_SRC_HEIGHT	Video height (in pixels)
VIDEO_SRC_WIDTH	Video width (in pixels)
VIDEO_HORIZONTAL_RES	Horizontal resolution (in pixels per inch)
VIDEO_VERTICAL_RES	Vertical resolution (in pixels per inch)
VIDEO_IS_GRAYSCALE	Whether or not the video has colors
VIDEO_DEPTH	Number of bits for the color depth

Table C–4 TextAnn Codes (extends MediaAnn)

Attribute	Description
TEXT_FONTSIZE	Point size of the text track
TEXT_FONTFACE	Font styles used (such as italics or boldface)
TEXT_FONTNAME	Name of the font used
TEXT_BG_COLOR	Background color (for example, 0x00RRGGBB)
TEXT_FG_COLOR	Foreground color (for example, 0x00RRGGBB)
TEXT_ALIGN	Left justified, centered, right justified, or full justified
TEXT_DEF_BOX	Default text box size, consisting of four instances of the Java primitive type *short*

Table C–5 MovieAnn Codes (extends MediaAnn)

Attribute	Description
MOVIE_DIRECTOR	Director of the movie
MOVIE_CAST	Names of the performers in the movie
MOVIE_EDIT_INFORMATION	Information about the editing
MOVIE_WARNING	Movie rating and warning information

Table C–6 AudioCDAnn Codes (extends MediaAnn)

Attribute	Description
AUDIO_CD_ID	CD identifier, recognized by CDDB
AUDIO_CD_NUM_OF_TRACKS	Number of tracks on the CD
AUDIO_CD_ARTIST	Main artist of the CD

Table C–7 AudioCDTrackAnn Codes (extends AudioAnn)

Attribute	Description
AUDIO_CD_TRACK_MINUTE	Starting minute of the track
AUDIO_CD_TRACK_SECOND	Starting second of the track
AUDIO_CD_TRACK_FRAME	Starting frame of the track
AUDIO_CD_TRACK_LBA	Logical block address associated with the track
AUDIO_CD_TRACK_CDID	CD identifier, recognized by CDDB
AUDIO_CD_TRACK_ALBUM	Audio CD title
AUDIO_CD_TRACK_DURATION	Duration of the track

Table C–8 ImageAnn Codes (extends MediaAnn)

Attribute	Description
IMAGE_HEIGHT	Height of the image
IMAGE_WIDTH	Width of the image
IMAGE_COUNT	Number of images stored in the file
IMAGE_PIXEL_FORMAT	The color space of the image, including the resolution
IMAGE_BITS_PER_PIXEL	Number of bits per image pixel
IMAGE_HORIZONTAL_RES	Horizontal resolution (in pixels per inch)
IMAGE_VERTICAL_RES	Vertical resolution (in pixels per inch)
MEDIA_TIMESCALE	Number of units in a second

Table C–9 IptclimAnn Codes

Attribute	Description
IIM_RECORD_VERSION	Version of the record
IIM_OBJECT_NAME	Object name
IIM_SPECIAL_INSTRUCTION	Special instructions
IIM_ACTION_ADVISED	Action to be taken: either 01, 02, or 03
IIM_CREATION_DATE	Creation date, consisting of the Date Created and Time Created record sets
IIM_DIGITAL_CREATION_DATE	Date of creation of the digitized version, consisting of the Digital Creation Date and Digital Creation Time record sets
IIM_ORIGINATING_PROG	Originating program
IIM_PROGRAM_VERSION	Version of the originating program
IIM_OBJECT_CYCLE	Either a.m. (a), p.m. (p), or both (b)
IIM_BYLINE	Creator of the image
IIM_BYLINE_TITLE	Title of the creator of the image
IIM_CITY	ID of the city of creation
IIM_SUB_LOCATION	ID of the location within the city of creation
IIM_PROVINCE_STATE	ID of the province or state of creation
IIM_COUNTRY_CODE	ID of the country of creation
IIM_LOCATION_NAME	Name of the location of creation
IIM_TRANSMISSION_REF	Transmission reference
IIM_HEADLINE	Headline associated with image
IIM_CREDIT	Provider of the object
IIM_SOURCE	Owner of the object
IIM_COPYRIGHT	Copyright notice
IIM_CONTACT	Contact for further information
IIM_CAPTION	Caption or abstract
IIM_WRITER	Writer or editor
IIM_IMAGE_TYPE	Image type

Table C–9 IptcIimAnn Codes(Cont.)

Attribute	Description
IIM_LANGUAGE	Language ID
IIM_KEYWORDS	Keywords associated with the image (supported with version 2 of IIM)

Table C–10 SampleAnn Codes (extends MediaAnn)

Attribute	Description
SAMPLE_TIMESTAMP	Time stamp of the specified sample, in the form hour:minute:second:mantissa, where mantissa is the fraction of a second in the units defined in MEDIA_TIMESCALE

Table C–11 TextSampleAnn Codes (extends SampleAnn)

Attribute	Description
TEXTSAMPLE_VALUE	String value of the text sample

Table C–12 VideoFrameSampleAnn Codes (extends SampleAnn)

Attribute	Description
VIDEO_FRAME_SAMPLE_HEIGHT	Height of the frame extracted from the video track
VIDEO_FRAME_SAMPLE_WIDTH	Width of the video frame extracted from the video track

D

Frequently Asked Questions

How do I find out which attributes go with an annotation?

<*ORACLE_HOME*>\ord\Annotator\lib\descriptors\annotations contains the XML files that define attributes for each annotation type.

Why won't my media file load into the database?

- The directory specified in the PL/SQL Upload Template is not accessible to the database server. This means that the directory does not exist or read access is not granted for that directory. See Section 4.1.2 for more details.
- For the file being uploaded, there is no read access granted to the database server.
- The INSERT statement in the PL/SQL Upload Template is incorrect. Refer to the reported SQL error in the **Console** window for an indication of the problem.
- If the media source is imported through an HTTP stream, an error may occur depending on your proxy settings. Make sure that the UTL_HTTP package in your Oracle database is correctly configured.
- Either the specified WHERE clause returns no results or it returns more than one resulting row. See Section 4.1.6 for more details.

How do I build an index on an attribute value?

See Appendix A.

How do I change the mapping between a file extension and its MIME type?

See Section 2.2.5.

When I am parsing a media source using the HTTP protocol, I encounter the following error: Unsupported Annotation for Content Type text/html

Check the path of the URL pointing to the resource. It is possible that the URL is invalid.

In the Insert New Row window of the PL/SQL Template Wizard, why can't I edit the values of the columns with *inter*Media Audio, Image, or Video type?

*inter*Media Annotator replaces the values of those columns with the media data and the annotation. Therefore, users are not expected to edit those columns, as their values will be written over automatically.

How can I change my startup settings?

If you are familiar with DOS batch files or UNIX shell scripts, you can modify the environment variables at the beginning of the startup script.

When I run Annotator.bat on my Windows NT system, why do I see the following error: "Could not load runtime library: d:\JRE\bin\javai.dll"?

The error appears because the NT registry is not consistent with the JDK that is being used. The error can be corrected by reinstalling the JDK. Also, check the directory names in `Annotator.bat`.

Where can I find the latest information on *inter*Media Annotator?

The latest information, known problems, and FAQ are available at:
`http://technet.oracle.com/software/products/intermedia/software_index.htm`

On the Macintosh platform, where is the Oracle JDBC driver located?

A copy of the Oracle JDBC Thin driver release 8.1.5 is included with the Macintosh version of Annotator. It is located at <*ORACLE_HOME*>\ord\Annotator\lib\classes111.zip.

Does QuickTime support include sprite track and flash track?

Sprite track and flash track are not supported.

How do I submit feedback on *inter*Media Annotator?

Please send e-mail feedback to imedsup@us.oracle.com.

Index

A

C

D

E

G

I

J

L

M

O

P

Q

R

S

T

U

W