Oracle® *inter*Media Annotator

User's Guide

Release 9.2

March 2002

Part No. A96120-01

Oracle *inter*Media Annotator is an API 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 9.2

Part No. A96120-01

Primary Authors: Helen Grembowicz, Sue Pelski

Contributors: Fengting Chen, Dongbai Guo, Susan Mavris, Susan Shepard, Rod Ward

Contents

3 Annotator Engine API Example

4 Annotator Engine API Reference Information

7 Annotator Parser API Reference Information

E Frequently Asked Questions

List of Examples

List of Figures

List of Tables

Send Us Your Comments

Oracle *inter*Media Annotator User's Guide, Release 9.2

Part No. A96120-01

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

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

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

- Electronic mail: nedc-doc_us@oracle.com
- FAX: 603.897.3825 Attn: Oracle *inter*Media 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 electronic mail address (optional).

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

Preface

Oracle *inter*Media Annotator is a Java API 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. Users who want to write their own PL/SQL Upload Templates should be familiar with PL/SQL. Users who want to write their own annotation types or parsers should be familiar with Java and XML.

Documentation Accessibility

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

`http://www.oracle.com/accessibility/`

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

Accessibility of Links to External Web Sites in Documentation This documentation may contain links to Web sites of other companies or organizations that Oracle Corporation does not own or control. Oracle Corporation neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Organization

This guide contains 8 chapters and 5 appendixes:

Book Element	Description
Chapter 1	Introduces Oracle *inter*Media Annotator.
Chapter 2	Describes how to set up your environment and the basic steps needed to use Oracle *inter*Media Annotator.
Chapter 3	Contains a full-length example of a Java application using the Annotator engine.
Chapter 4	Contains reference information on the Java API associated with the Annotator engine.
Chapter 5	Contains instructions for writing a PL/SQL Upload Template to upload your annotation to an Oracle database.
Chapter 6	Contains a full-length example of a custom parser.
Chapter 7	Contains reference information on the Java API associated with writing custom parsers.
Chapter 8	Contains information on creating your own annotation types.
Appendix A	Contains information on using Oracle Text to query stored annotations.
Appendix B	Contains reference information on supported formats.
Appendix C	Contains reference information on annotation attributes.
Appendix D	Contains information about deprecated and obsolete features.
Appendix E	Contains answers to frequently asked questions.

Related Documentation

For information about related topics, see the following documentation:

- *Oracle interMedia User's Guide and Reference*
- *Oracle Text Application Developer's Guide*

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

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

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

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

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

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

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

`http://otn.oracle.com/docs/index.htm`

To access the database documentation search engine directly, go to

`http://tahiti.oracle.com`

Conventions

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

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

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.

Convention	Meaning
. . .	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.
italic font	Italic font in examples indicates a user-supplied name.
[ ]	Brackets enclose optional clauses from which you can choose one or none.

1

Introduction to Oracle *inter*Media Annotator

This chapter provides an overview of Oracle *inter*Media Annotator, which extracts information (**metadata**) from media sources of certain formats and inserts the metadata, along with the media source, into an Oracle database. Oracle *inter*Media Annotator uses Oracle *inter*Media. This chapter discusses the following topics:

- Purpose
- Operations Overview
- Prerequisites

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 *inter*Media 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 annotation will usually contain one or more **subannotations**, 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 subannotations, you can

define your own subannotations by adding an empty annotation and then populating it with your own values. 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.

Oracle *inter*Media Annotator 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 are subannotations of the movie annotation.

You can use *inter*Media Annotator to insert the annotation along with the media source into an Oracle database.

You can use the *inter*Media Annotator engine API to integrate the *inter*Media Annotator functions into your application. You can also use this API as a tool for bulk loading many multimedia files into the database.

For example, if you have a large number of movie trailers to store, 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 I, "Oracle interMedia Annotator Engine" for more information on the Oracle *inter*Media Annotator Java engine.

Oracle *inter*Media Annotator is extensible; you can use the *inter*Media Annotator parser Java API to write a custom parser for your media source files, or to 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 *inter*Media 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 an Oracle database 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 subannotation describing the picture of the

exterior or a movie subannotation describing the movie of the interior of the house. The developer can also write a new parser to create the property annotation.

See Part I, "Oracle interMedia Annotator Engine", for more information on extending Oracle *inter*Media Annotator.

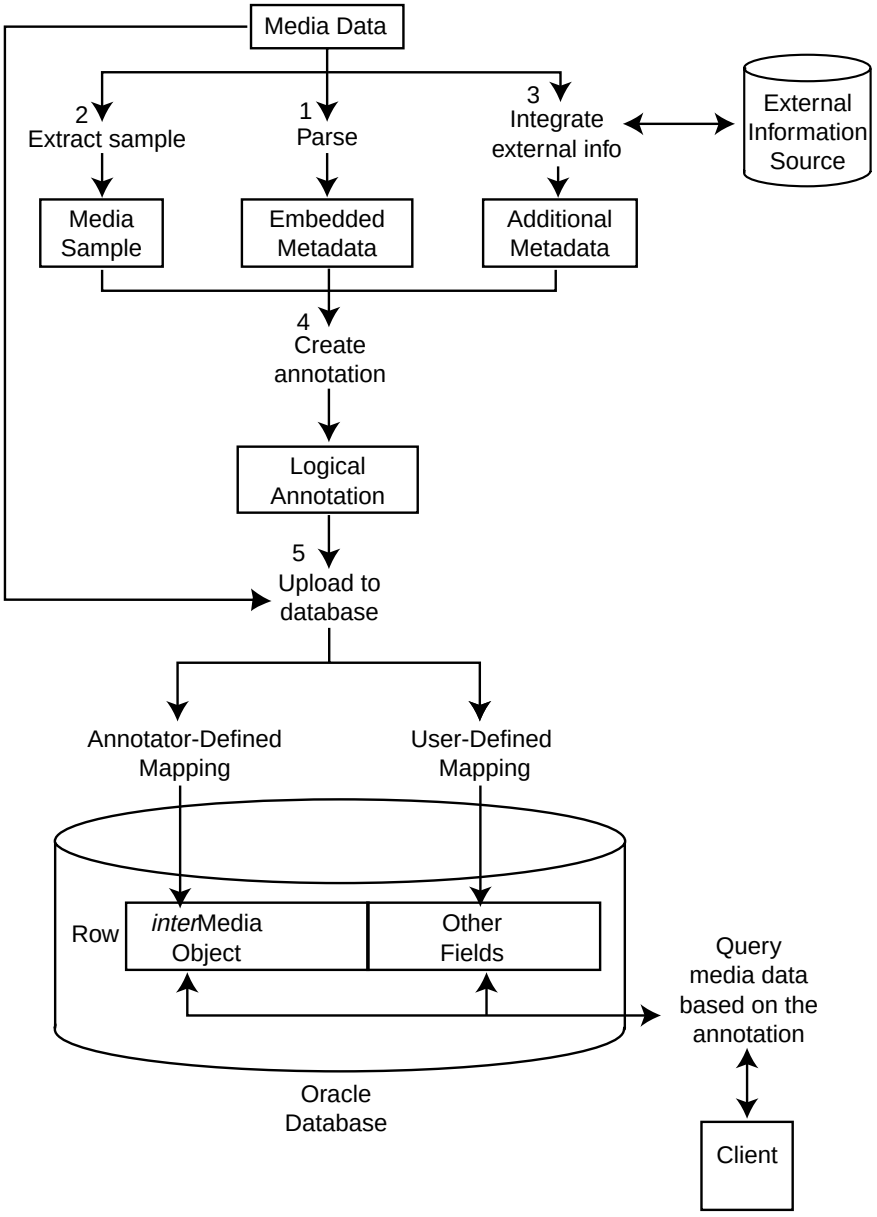
After the annotation is stored in the database, you can use Oracle Text to query the annotation. For more information, see Appendix A.

1.2 Operations Overview

The main functions of Oracle *inter*Media Annotator are to build a logical annotation from a media source and then to 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 illustrates an overview of this process.

Figure 1–1 Overview of Oracle interMedia Annotator Operations



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

1. Parse the media source. Oracle *inter*Media Annotator extracts the metadata from the source file.
2. Extract samples from the media source. For some formats, Oracle *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. Oracle *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.

   Oracle *inter*Media Annotator uploads the media source and the annotation (in XML format) into an *inter*Media object in the database. Oracle *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.

   See Chapter 5 for more information on the upload process.

After you have completed these steps, you can 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 Oracle Text. For more information, see Appendix A.

1.3 Prerequisites

To use Oracle *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), release 1.2 or later, for Oracle 8.1.5 or later.

To use Oracle *inter*Media functions in your Java applications, you should use the Java Development Kit 1.2 or later.

Part I

Oracle *inter*Media Annotator Engine

This part discusses the Oracle *inter*Media Annotator Java-based engine and contains the following chapters:

2

Getting Started with Oracle *inter*Media Annotator

This chapter describes how to get started in using Oracle *inter*Media Annotator. It discusses the following topics:

- Setting Preferences
- Available URL Protocols
- Using Oracle interMedia Annotator

2.1 Setting Preferences

Before you can use Oracle *inter*Media Annotator, you must specify preferences for the environment. You specify the preferences in the `Annotator.prefs` file, located in the configuration directory. By default, Oracle *inter*Media Annotator assumes that the configuration directory is the `\lib\conf` or `/lib/conf` subdirectory of the current directory.

However, at installation, the configuration files are placed in the following directory:

- On UNIX: *`$ORACLE_HOME`*`/ord/Annotator/lib/conf`
- On Windows: *`ORACLE_HOME`*`\ord\Annotator\lib\conf`

You must ensure that the configuration directory contains the preferences file and other configuration files before you use Oracle *inter*Media Annotator. For example, if you run *inter*Media Annotator from the directory `/usr5/myfiles`, *inter*Media Annotator assumes the configuration directory to be `/usr5/myfiles/lib/conf`.

The format of entries in the preferences file is:

preference_parameter=preference_value

Table 2–1 shows the names of the preference parameters and the possible values for each parameter.

Table 2–1 Preferences

Parameter	Description
`MimeMapFile`	The name of the MIME types mapping file, which maps MIME types to a file extension, annotation, parser, and player. The file must be located in the configuration directory as specified by the `configDirectory` parameter. By default, the file name is `Annotator.mime`.
`MimeTypesFile`	The name of the MIME types file, which controls what MIME types are sent to the client for a given extension. The file must be located in the configuration directory as specified by the `configDirectory` parameter. By default, the file name is `mime.types`.
`configDirectory`	Optional parameter representing the specification for the configuration directory. By default, *inter*Media Annotator assumes that the configuration directory is the `lib/conf` or `lib\conf` subdirectory of the current directory. If this parameter is set, the configuration files, such as the preference file and MIME types mapping file, must be located in the specified directory and the value of the `configDirectory` parameter in the preferences file in that directory must be consistent with the actual location of the files. At installation, the configuration files are placed in the following directory: On UNIX: *$ORACLE_HOME*`/ord/Annotator/lib/conf` On Windows: *ORACLE_HOME*`\ord\Annotator\lib\conf`

Table 2–1 (Cont.) Preferences

Parameter	Description
descriptorDirectory	Optional parameter representing the specification for the descriptor directory. By default, *inter*Media Annotator assumes that the descriptor directory is the `lib/descriptors` or `lib\descriptors` subdirectory of the current directory. If this parameter is set, the descriptor files must be located in the specified directory. At installation, the descriptor files are placed in the following directory: On UNIX: *$ORACLE_HOME*`/ord/Annotator/lib/descriptors` On Windows: *ORACLE_HOME*`\ord\Annotator\lib\descriptors`
`connectDriver`	The name of the JDBC driver you are using. For example: `oracle.jdbc.driver.OracleDriver`

`connectJDBCProt`	The prefix for the JDBC driver, which describes whether it is the JDBC OCI driver or the JDBC Thin driver.
	■ For the JDBC OCI driver:
	`jdbc:oracle:oci8:@`
	■ For the JDBC Thin driver:
	`jdbc:oracle:thin`

`connectHost`	The name of the host on which the Oracle database server to which you want to upload media data is installed. For example: `myhost`
`connectPort`	The port number of the host on which the Oracle database server is installed.
connectSID	The SID for the Oracle database.
`connectUserName`	A database user name, which is used to upload media data into the database.
`connectPassword`	The password of the database user.

Table 2–1 (Cont.) Preferences

Parameter	Description
`serviceName`	The service name of the database. Use the following format: *host-name*:*port_name*:*oracle-sid*
`useHttpProxy`	Whether or not to use an HTTP proxy server to annotate media sources that are available remotely over the Internet through the HTTP protocol. Specify `true` if you are running in a secure environment. Valid values are: ■ `true` ■ `false`
`httpProxyServer`	The URL of the HTTP proxy server. The following shows an example of a URL for a proxy server: `www-ourproxy.ourcompany.com`
`httpProxyPort`	The port number of the proxy server.
`mediaDirectory`	The directory that contains the source media.
`uploadOci8BlobBlockSize`	The block size to use to upload BLOBs when you are using the JDBC OCI driver. In most cases, use the default value in the preferences file.
`uploadOci8ClobBlockSize`	The block size to use to upload CLOBs when you are using the JDBC OCI driver. In most cases, use the default value in the preferences file.
`uploadThinBlobBlockSize`	The block size to use to upload BLOBs when you are using the JDBC Thin driver. In most cases, use the default value in the preferences file.
`uploadThinClobBlockSize`	The block size to use to upload CLOBs when you are using the JDBC Thin driver. In most cases, use the default value in the preferences file.
`uploadRootAnn`	Whether or not to upload, not only a subannotation, but any annotations and subannotations that are predecessors of the subannotation. Valid values are: ■ `true` ■ `false`
`sqlFileName`	The script to run to insert an annotation.

Table 2–1 (Cont.) Preferences

Parameter	Description
`openSaveDirectory`	The default directory to which annotations will be saved and from which saved annotations will be opened. This parameter is specific to the demo utility.
`defaultOfm`	The full path of the default PL/SQL upload template ((file extension .ofm), which is used to upload media data to the database. This parameter is specific to the demo utility.
`ofmDirectory`	The directory to which PL/SQL upload templates (file extension .ofm), will be written. This parameter is specific to the demo utility.

Note: For information about the demo utility, see Section D.1.

You can also set preferences by using the Preferences.setProperty( ) method. See "setProperty( )" on page 4-77 for more information.

The following sections discuss using some of the preferences to configure your environment before you begin to use Oracle *inter*Media Annotator.

2.1.1 Setting the Connection to the Database

You can connect to an Oracle database using the JDBC Thin driver or the JDBC OCI driver.

To use the JDBC Thin driver, specify the following in the `Annotator.prefs` file:

- For the value of the parameter `connectDriver`, enter the following:

  ```
  oracle.jdbc.driver.OracleDriver
  ```

- For value of the parameter `connectJDBCProt`, enter the following:

  ```
  jdbc:oracle:thin
  ```

- For value of the parameter `serviceName`, enter the service name of your database, using the following format:

  host-name:*port_name*:*oracle-sid*

To use the JDBC OCI driver, specify the following:

- For the value of the parameter `connectDriver`, enter the following:

  ```
  oracle.jdbc.driver.OracleDriver
  ```

- For value of the parameter `connectJDBCProt`, enter the following:

  ```
  jdbc:oracle:oci8:@
  ```

- For the value of the parameter `serviceName`, enter the service name of your database, using the syntax used by Oracle Net. See the *Oracle9i Net Services Administrator's Guide* for more information.

2.1.2 Specifying the Proxy Settings

Oracle *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 must configure Oracle *inter*Media Annotator to use your proxy server before you can access the Internet. To configure the proxy server, specify the following in the `Annotator.prefs` file:

- For the parameter `useHttpProxy`, enter the value `true`.
- For the parameter `httpProxyServer`, enter the address of your HTTP proxy server. For example:

  ```
  www-ourproxy.ourcompany.com
  ```

- For the parameter `httpProxyPort`, enter the port number for the HTTP proxy server.

2.2 Available URL Protocols

Oracle *inter*Media Annotator can parse media sources accessible through the URL protocols shown in Table 2–2.

Table 2–2 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.

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

If you are parsing a media source with multiple tracks, such as a video source, a subannotation is created for each track.

2.3 Using Oracle *inter*Media Annotator

To use Oracle *inter*Media Annotator, you take the following general steps:

1. Create and initialize an instance of the annotator client.
2. Optionally, set preferences using the Preferences.setProperty( ) method. You can set preferences such as the type of JDBC driver or information about the HTTP Server proxy. See setProperty( ) on page 4-77 for more information about this method.
3. Parse a media source file from a given URL to create an annotation about the source. Use the Annotation.parseMedia( ) method to parse the source file.
4. Optionally, get the defined attributes of the annotation and set additional attributes for the annotation.

   Oracle *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 the Annotation.getAttribute( ) method to get the existing attributes and the Annotation.setAttribute( ) method to add a value to your annotation for any attribute that does not have a value.
5. Get any subannotations using the Annotation.getSubAnnotations( ) method and, optionally, define additional subannotations, using the Annotation.addSubAnnotation( ) method.

   An annotation will usually contain one or more subannotations, 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 subannotations, you can define your own subannotations by adding an empty annotation and then populating it with your own values.
6. For parsers such as the QuickTime parser, you can extract media samples from the media source file, using the AnnotationHandler.extractMedia( ) method.
7. Upload the annotation to an Oracle database, using the AnnotationHandler.insertMedia( ) method.

Note: Oracle *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.

Chapter 3 describes a sample program that parses a media source file, creates an annotation, and uploads the annotation to an Oracle database.

3

Annotator Engine API Example

This chapter provides a description of a sample program that parses a media source file, creates an annotation, and uploads the annotation to an Oracle database. This program, `SimpleAnnotator.java`, is an example of a user-developed application that was written using the Oracle *inter*Media Annotator engine API.

You can find the source code at the following location:

- On UNIX:

  $ORACLE_HOME`/ord/Annotator/demo/examples/src/SimpleAnnotator.java`

- On Windows:

  ORACLE_HOME`\ord\Annotator\demo\examples\src\SimpleAnnotator.java`

In addition, an asynchronous version of the sample program, named `SimpleAnnotatorAsynch.java`, is available in the same directory.

The code that appears in this chapter will not necessarily match the code shipped as `SimpleAnnotator.java`. If you want to run this sample program on your system, use the file provided with the 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 numbers enclosed in brackets; these indicate that further explanation of that code is in the numbered list immediately following the example.

The sample program contains user-defined methods that use Java and Oracle *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 subannotations
- Extract media samples from the media source file
- Upload the annotation and media source to an Oracle database

> **Note:** Oracle *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.

3.1 Import Statements

Example 3–1 shows the import statements that must be included in the application to properly run an Annotator client.

Example 3–1 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;
```

3.2 Class Definition and Instance Variables

Example 3–2 shows the class definition and instance variables for the sample Annotator client.

Example 3–2 Class Definition and Instance Variables

```
public class SimpleAnnotator implements AnnListener, OutputListener{
     private Status m_st;
     private AnnotationHandler m_ah;
     static String m_output_file;
     static boolean m_output_file_defined;
```

An Annotator client must implement the AnnListener interface to have access to the callback methods used for Annotator engine operations. See "Class oracle.ord.media.annotator.listener.AnnListener" on page 4-61 for more information.

An Annotator client must implement OutputListener to get access to trace information from the engine. See "Class oracle.ord.media.annotator.listener.OutputListener" on page 4-67 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 application. See "Class oracle.ord.media.annotator.utils.Status" on page 4-78 for more information.
- `m_ah` is the AnnotationHandler object that will actually produce the annotation for the given content source. See "Class oracle.ord.media.annotator.handlers.AnnotationHandler" on page 4-27 for more information.

Two additional variables are declared. The variable `m_output_file` will hold the name of the output file. The variable `m_output_file_defined` is a Boolean that will indicate if an output file has been passed as an argument.

3.3 main( ) Method

The main( ) method in the sample program tests the URL passed to the application, creates an empty instance of SimpleAnnotator, and calls the init( ) method. After the instance is initialized, it invokes a method that parses the media. Example 3–3 shows the contents of the main( ) method.

Example 3–3 main( ) Method (SimpleAnnotator)

```
[1] public static void main(String[] argv){
[2] if(argv.length == 0 )
      {
        System.err.println("Usage: java SimpleAnnotator mediaURL [-w xmlfile] [-e enc]");
        System.err.println("mediaURL: URL of the media you want to parse");
        System.err.println("          (for example. file:/myjpeg.jpg");
        System.err.println("xmlfile: The output file for XML annotations");
        System.err.println("encoding: an optional argument of the character");
        System.err.println("            encoding of the media");
       return;
      }

[3] String szURL = argv[0];
    String szEncoding=null;
    boolean encoding_define=false;
    m_output_file_defined=false;

[4] for(int i=1; i<argv.length; i++) {
      if (argv[i].charAT(0) == '-') {

        if (argv[i].charAT(1) == 'w') {
          if (i == argv.length - 1) break;
          int j = i+1;
          if (argv[j].charAT(0) != '-') {
             m_output_file_defined=true;
             m_output_file = argv[j];
             i++
          }
        }
        if (argv[i].charAT(1) == 'e') {
          if (i == argv.length - 1) break;
          int j = i+1;
          if (argv[j].charAT(0) != '-') {
             encoding_defined=true;
             szEncoding =argv[j];
             i++;
          }
        }
      }
    }

[5] SimpleAnnotator sa = new SimpleAnnotator();
[6] sa.init();

[7]   if (encoding_defined)
      {
          sa.parse(szURL, szEncoding]);
      }
```

```
        else
        {
        sa.parse(szURL);
        }
}
```

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

1. Accepts the URL of the media source file to be operated upon as the first argument. Optional arguments are:
   - A String representing the output file for the XML annotations.
   - A String representing the character encoding of the media.
2. 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.
3. Defines the following variables:
   - A String named `szURL`. Its value is set to the value of the first parameter, the URL of the media source file.
   - A String named `szEncoding`, representing the character encoding of the media. Its value is set to `null`.
   - A Boolean named `encoding_defined` that defines whether or not an argument for the character encoding is passed to the method.

   In addition, sets the value of the `m_output_file_defined` variable to `false`.
4. Tests to see how many arguments were passed to the method and assigns values to variables.
5. Creates an empty instance of SimpleAnnotator.
6. Calls the init( ) method to initialize the newly created SimpleAnnotator instance. Section 3.4 describes the init( ) method and lists its code.

   After the SimpleAnnotator instance is initialized, the client can invoke the Annotator engine operations, such as parsing, extraction, and insertion.
7. Calls the parse( ) method to parse the media source file. It passes the string containing the URL and, if supplied, the encoding method.

   Section 3.5 describes the parse( ) method and lists its code.

3.4 init( ) Method

The init( ) method in the sample program initializes the Annotator client and sets preferences. Example 3–4 shows the contents of this method.

Example 3–4 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 3.12 for more information on the report(String) method.
2. Initializes the Status object and implements the OutputListener interface so that the current instance of SimpleAnnotator can receive the status messages. See "initStatus( )" on page 4-81 for more information.
3. Sets the initialized Status object to the `m_st` instance variable.
4. Uses the Status.SetOutputMode( ) method to set the status output mode to VERBOSE. See "SetOutputMode( )" on page 4-85 for more information.
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 3.15 for more information on the reportError( ) method.

   If the Status and AnnotationHandler objects were both created with no errors, you can set any necessary preferences.

7. Creates a Preferences object and initializes it.

8. Uses the Preferences.setProperty( ) method to set a new preference named SZ_CONN_PASSWORD. See "setProperty( )" on page 4-77 for more information.

9. Prints a message that initialization was successful.

3.5 parse( ) Method

The parse(String) method in the sample program calls the method AnnotationHandler.parseMedia( ), passing the URL, to parse the source file and create the annotation. Example 3–5 shows the contents of the parse(String) method.

Example 3–5 parse(String) Method

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

The code in the parse(String) method calls the AnnotationHandler.parseMedia( ) method and passes the URL to it. The client (`this`) implements the AnnListener interface.

The parse(String, String) method in the sample program calls the AnnotationHandler.parseMedia( ) method, passing the URL and character encoding, to parse the source file and create the annotation. Example 3–6 shows the contents of the parse(String, String) method.

Example 3–6 Parse(String, String) Method

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

The code in the parse(String, String) method calls the AnnotationHandler.parseMedia( ) method and passes the URL and the character encoding to it. The client (`this`) implements the AnnListener interface.

See "parseMedia(String, AnnListener)" on page 4-42 and "parseMedia(String, AnnListener, String)" on page 4-43 for more information about the AnnotationHandler.parseMedia( ) method.

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

The AnnotationHandler.parseMedia( ) method parses the source file and builds the annotation. Then, 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 3.6 for more information.

3.6 parsePerformed( ) Method

The parsePerformed( ) method in the sample program manipulates the annotation before it is uploaded to the database. Because your application must implement AnnListener, this method is required. Example 3–7 shows the contents of the parsePerformed( ) method.

Example 3–7 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()){
        try {
 [4]      Object att_val = eAttrs.nextElement();
        }
        Catch (Exeption e) {
          System.err.println ("exception caught " + e.getMessage());
        }
 [5] Enumeration eSubAnns = ann.getSubAnnotations();
     while (eSubAnns.hasMoreElements()){
 [6] Annotation subAnn = (Annotation)eSubAnns.nextElement();
     }
     // Advanced Example
     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, this);
     }
}
```

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 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. Uses the Annotation.getAttribute( ) method to retrieve the value of the MEDIA_SOURCE_MIME_TYPE attribute of the annotation and casts it into a String object. See "getAttribute( )" on page 4-5 for more information.

3. Uses the Annotation.getAttributes( ) method to get a list of all attributes that have a valid value and stores their names in an Enumeration object. See "getAttributes( )" on page 4-6 for more information.

4. Accesses the values stored in the Enumeration object.

5. Uses the Annotation.getSubAnnotations( ) method to get all subannotations of the annotation and stores them in an Enumeration object. See "getSubAnnotations( )" on page 4-12 for more information.

6. Accesses the values stored in the Enumeration object.

7. Creates an empty annotation named inventoryAnn.

8. Uses the Annotation.addSubAnnotation( ) method to add inventoryAnn to ann as a subannotation. See "addSubAnnotation( )" on page 4-4 for more information.

9. Uses the Annotation.setAttribute( ) method to set the SALES_PRICE attribute in inventoryAnn to 19.99. See "setAttribute( )" on page 4-18 for more information.
10. Catches any errors raised in steps 7 through 9 and reports them with the errorOccured( ) method, which is discussed in Section 3.10.
11. Uses the report(Annotation) method, which is discussed in Section 3.13, to print the annotation as an XML file.
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, that step is omitted.
13. Uses the AnnotationHandler.extractMedia( ) method to extract the media samples from the annotation. See "extractMedia( )" on page 4-32 for more information.

    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( ). This method is discussed in Section 3.7.

3.7 extractionPerformed( ) Method

The extractionPerformed( ) method in the sample program maps an annotation to a file and uploads the annotation to the database. Because your application must implement AnnListener, this method is required. Example 3–8 shows the contents of the extractionPerformed( ) method.

Example 3–8 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, which is discussed in Section 3.13, to print the annotation as an XML file.

2. Creates a new OrdFileMapping object based on the mapping file. For this example, the mapping file is located at `e:\\mylogic.ofm`. See "OrdFileMapping Constructor" on page 4-47 for more information.

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 and the AnnotationHandler.insertMedia( ) method to insert the media. See "insertMedia(Annotation, OrdMapping, AnnListener)" on page 4-37 for more information on this method.

   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 "insertMedia(Annotation, OrdMapping, AnnListener, Connection)" on page 4-38 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( ), which is discussed in Section 3.8.

3.8 insertionPerformed( ) Method

The insertionPerformed( ) method in the sample program closes the connection to the database and catches any exceptions. Because your application must implement AnnListener, this method is required. Example 3–9 shows the contents of the insertionPerformed( ) method.

Example 3–9 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 3.10 for more information on the errorOccured( ) method.

3.9 warningOccured( ) Method

The warningOccured( ) method in the sample program captures any warnings. Because your application must implement AnnListener, this method is required. Example 3–10 shows the contents of the warningOccured( ) method.

Example 3–10 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, discussed in Section 3.15, to capture the warning and report it. If a warning occurs and this method is called, the Annotator engine continues to operate. See warningOccured( ) on page 4-66 for more information.

3.10 errorOccured( ) Method

The errorOccured( ) method in the sample program captures any errors. Because your application must implement AnnListener, this method is required. Example 3–11 shows the contents of the errorOccured( ) method.

Example 3–11 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, discussed in Section 3.15, 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 "errorOccured( )" on page 4-62 for more information.

3.11 ConsoleOutput( ) Method

The ConsoleOutput( ) method in the sample program prints messages during execution. Because your application must implement OutputListener, this method is required. Example 3–12 shows contents of the ConsoleOutput( ) method.

Example 3–12 ConsoleOutput( ) Method

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

The code in this example implements the OutputListener.ConsoleOutput( ) method. This method uses the report(String) method, discussed in Section 3.12, to print messages during execution. See "ConsoleOutput( )" on page 4-68 for more information.

3.12 report(String) Method

The report(String) method in the sample program prints strings to the error stream. Example 3–13 shows contents of the method.

Example 3–13 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.

3.13 report(Annotation) Method

The report(Annotation) method in the sample program prints the annotation to an XML file. Example 3–14 shows the contents of the method.

Example 3–14 report(Annotation) Method

```
public void report(Annotation ann){
 [1]  try
      {
        File tmp_file;
        if (m_output_file_defined)
        {
```

```
            tmp_file = new File(m_output_file);
        }
        else
        {  tmp_file = File.createTempFile("xml", ".dat");
        }
        System.out.println ("created file to save annotation:"
                            + tmp_file.getPath());
 [2]    FileOutputStream fo = new FileOutputStream (tmp_file);
        OutputStreamWriter w = new OutputStreamWriter(fo, "UTF-8");
 [3]    m_ah.exportToXML(w, ann);
        w.flush();
        w.close();
        fo.close();
}
 catch (Exception e) { reportError(e);   }
{
```

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

1. Tests to see if a temporary output file exists and creates one if it does not.
2. Writes the output stream to a temporary file.
3. Uses the AnnotationHandler.exportToXML( ) method to create an XML representation of the given annotation. See "exportToXML( )" on page 4-31 for more information.

3.14 reportWarning( ) Method

The reportWarning( ) method in the sample program prints warnings to the error stream. Example 3–15 shows the contents of the method.

Example 3–15 reportWarning( ) Method

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

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

3.15 reportError( ) Method

The reportError( ) method in the sample program prints exceptions to the error stream. Example 3–16 shows the contents of the method.

Example 3–16 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.

4

Annotator Engine API Reference Information

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

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

This section presents reference information on the constructors and 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.

See Appendix C for descriptions of the attributes defined by this class.

This class extends java.lang.Object.

Annotation Constructor

Format

public Annotation( )

Description

Creates an annotation object.

Parameters

None.

Return Value

None.

Exceptions

None.

Example

```
Annotation ann = new Annotation( );
```

addSubAnnotation( )

Format

public void addSubAnnotation(Annotation annChild)

Description

Adds the given annotation as a subannotation of the current annotation.

Parameters

annChild
The annotation to be added as a subannotation.

Return Value

None.

Exceptions

None.

Example

See Section 3.6 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. See Appendix C for descriptions of the attributes defined by *inter*Media Annotator.

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 3.6 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. See Appendix C for descriptions of the attributes defined by *inter*Media Annotator.

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 3.6 for an example of this method.

getDescriptor( )

Format

public AnnotationDesc getDescriptor( )

Description

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

See "Class oracle.ord.media.annotator.descriptors.AnnotationDesc" on page 7-2 for more information about the AnnotationDesc object.

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 subannotations of the current annotation.

Parameters

None.

Return Value

This method returns the number of subannotations, 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 subannotations of the current annotation.

Parameters

None.

Return Value

This method returns an Enumeration object that contains a list of the subannotations 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 subannotations.

Parameters

None.

Return Value

This method returns an Enumeration object of the vector of subannotations.

Exceptions

None.

Example

See Section 3.6 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 subannotation of the given annotation.

Parameters

szAncestor
The annotation of which the current annotation may be a subannotation.

Return Value

This method returns true if the current annotation is a subannotation 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. See Appendix C for descriptions of the attributes defined by *inter*Media Annotator.

Return Value

None.

Exceptions

None.

Example

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

removeSampleAnns( )

Format

public void removeSampleAnns( )

Description

Removes all subannotations 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 subannotation from the current annotation.

Parameters

ann
The subannotation to be removed.

Return Value

This method returns true if the subannotation was removed successfully; false otherwise.

Exceptions

None.

Example

See "isDescendantOf( )" on page 4-14 for an example of this method.

setAttribute( )

Format

public void setAttribute(java.lang.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. See Appendix C for descriptions of the attributes defined by *inter*Media Annotator.

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

Return Value

None.

Exceptions

None.

Example

See Section 3.6 for an example of this method.

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

This section presents reference information on the constructor and 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**.

See "Class oracle.ord.media.annotator.handlers.AnnTaskManager" on page 7-10 for more information on the annotation task manager.

This class extends java.lang.Object.

AnnTaskMonitor Constructor

Format

public AnnTaskMonitor(oracle.ord.media.annotator.handlers.AnnTaskManager annTaskMgr)

Description

Creates an AnnTaskMonitor object.

Parameters

annTaskMgr
The annotation task manager that tracks the current task on the server side.

Return Value

None.

Exceptions

None.

Example

```
AnnTaskManager tskmgr = new AnnTaskManager();
AnnTaskMonitor mon = new AnnTaskMonitor(tskmgr);
```

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. Typically, it returns a number between the start value and the end value of the task. See getTaskStart( ) on page 4-24 and getTaskEnd( ) on page 4-23 for more information. You can set the current value using the AnnTaskManager.setTaskCurrent( ) method. See "setTaskCurrent(int)" on page 7-27 for more information.

Exceptions

None.

Example

See "isDone( )" on page 4-25 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. Typically, the task end value is set to 100. You can set the value using the AnnTaskManager.setTask( ) method. See "setTask( )" on page 7-26 for more information.

Exceptions

None.

Example

See the isInitialized( ) method on page 4-26 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. Typically, the task start value is set to zero. You can set the value using the AnnTaskManager.setTask( ) method. See "setTask( )" on page 7-26 for more information.

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();
```

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

This section presents reference information on the constructors and 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 "Class oracle.ord.media.annotator.listener.AnnListener" on page 4-61 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 Section 3.4 for an example of creating an AnnotationHandler object. In addition, see the online examples `SimpleAnnotator.java` and `SimpleAnnotatorAsynch.java`, available in the following directory:

- On UNIX:

  $ORACLE_HOME`/ord/Annotator/demo/examples/src`

- On Windows:

  ORACLE_HOME`\ord\Annotator\demo\examples\src`

AnnotationHandler Constructor

Format

public AnnotationHandler( )

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 "Class oracle.ord.media.annotator.utils.Status" on page 4-78 for more information about Status.

Parameters

None.

Return Value

None.

Exceptions

oracle.ord.media.annotator.handlers.AnnotationHandlerException

Example

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

AnnotationHandler(int) Constructor

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 "Class oracle.ord.media.annotator.utils.Status" on page 4-78 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 3.4 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 3.6 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 subannotations 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 3.13 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 extraction.

Return Value

This method returns the AnnTaskMonitor object associated with this task.

Exceptions

None.

Example

See Section 3.6 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 annotation descriptor XML files.

Parameters

None.

Return Value

This method returns a list of String objects with the names of the annotation types that are defined in the annotation descriptor XML files.

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 parser descriptor XML files.

Parameters

None.

Return Value

This method returns a list of the parser types defined in the parser descriptor XML files.

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 to be inserted.

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 operation.

Return Value

This method returns the AnnTaskMonitor object associated with this task.

Exceptions

None.

Example

See Section 3.7 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 to be inserted.

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

isExtractable( )

Format

public boolean isExtractable (Annotation ann)

Description

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

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 3.6 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)
}
```

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. Unlike parseMedia(String, AnnListener) and parseMedia(String, AnnListener, String), this method sets only one attribute.
- 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 is = new FileInputStream("test.mpg");
AnnTaskMonitor atm = handler.parseMedia(is, 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 3.5 for an example of this method.

parseMedia(String, AnnListener, String)

Format

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

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( )

This method allows parsing of annotations from a media file, regardless of the character set used in the media file. For example, this method can parse media that contains international character sets, such as Shift-JIS for Japanese characters.

Parameters

sURL
The URL of the media file to be parsed.

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

szCharEncoding
The character encoding.

Return Value

This method returns the AnnTaskMonitor object associated with this task.

Exceptions

None.

Example

See Section 3.5 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 operation.

Return Value

None.

Exceptions

None.

Example

See the isPlayable( ) method on page 4-40 for an example of this method.

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

This section presents reference information on the constructor and 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.

OrdFileMapping Constructor

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

```
OrdFileMapping ofm = new OrdFileMapping("e:\\mylogic.ofm");
```

See Section 3.7 for an example that uses this constructor.

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 *inter*Media Annotator preprocessor to insert *inter*Media 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);
```

Class oracle.ord.media.annotator.handlers.utils.MimeMap

This section presents reference information on the constructor and methods of the MimeMap class. This class holds the mapping between a MIME type or file extension and the annotation, parser, and player names as specified in the MIME map file, `Annotator.mime`. This class has dependencies on properties from the Preferences class in that the file to be read for the mappings is specified by the Preferences class.

MimeMap Constructor

Format

public MimeMap( )

Description

Reads the mapping between a MIME type and the annotation, parser, and player names from the MIME mapping file into a hash table. The mapping file is specified by the MimeMapFile parameter in the `Annotator.prefs` file. By default, the MIME mapping file is named `Annotator.mime` and is located in the following directory:

- On UNIX: *`$ORACLE_HOME`*`/ord/Annotator/lib/conf`
- On Windows: *`ORACLE_HOME`*`\ord\Annotator\lib\conf`

The `Annotator.prefs` file is also located in this directory. For more information about setting the location of the `Annotator.mime` file, see Section 2.1.

Parameters

None.

Return Value

None.

Exceptions

oracle.ord.media.annotator.AnnotatorException

Example

```
MimeMap m_map = new MimeMap();
```

clone( )

Format

public java.lang.Object clone( )

Description

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

Parameters

None.

Return Value

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

Exceptions

java.lang.CloneNotSupportedException

Example

None; this method should be called only by advanced programmers.

getAnnotationName(String)

Format

public java.lang.String getAnnotationName(java.lang.String szMimeType)

Description

Returns the class name of the annotation with the specified MIME type.

Parameters

szMimeType
The name of the MIME type.

Return Value

The fully qualified class name of the Annotation that is mapped to the specified MIME type.

Exceptions

None.

Example

```
String name = m_map.getAnnotationName("image/jpeg");
```

getMimeTypes( )

Format

public java.util.Enumeration getMimeTypes( )

Description

Returns the enumeration of all the MIME types currently registered to be handled by *inter*Media Annotator.

Parameters

None.

Return Value

This method returns the enumeration of all the MIME types.

Exceptions

None.

Example

```
Enumeration eMimeTypes = m_map.getMimeTypes();
```

getMimeTypesCount( )

Format

public int getMimeTypesCount( )

Description

Returns the number of MIME types in the hash table.

Parameters

None.

Return Value

This method returns the number of MIME types in the hash table, as an integer.

Exceptions

None.

Example

```
int mimeTypeCount = m_map.getMimeTypesCount();
```

getParserName( )

Format

public java.lang.String getParserName(java.lang.String szMIMEType)

Description

Returns the class name of the parser that is mapped to the specified MIME type.

Parameters

szMIMEType
The name of the MIME type.

Return Value

This method returns the fully qualified class name of the parser that is mapped to the specified MIME type.

Exceptions

None.

Example

```
String name = m_map.getParserName("image/jpeg");
```

getParsers( )

Format

public java.util.Enumeration getParsers( )

Description

Returns the enumeration of all the parsers currently registered to a MIME type by *inter*Media Annotator.

Parameters

None.

Return Value

This method returns the enumeration of all the parsers.

Exceptions

None.

Example

```
Enumeration eparsers = m_map.getParsers();
```

handlesMime( )

Format

public boolean handlesMime(java.lang.String szMIMEType)

Description

Determines if an annotation, parser, or player has been defined in the hash table for the specified MIME type.

Parameters

szMIMEType
The name of the MIME type.

Return Value

A Boolean value. True is returned if the specified MIME type has been defined for an annotation, parser, or player; false otherwise.

Exceptions

None.

Example

```
if(m_map.handlesMime(szMIMEtype) == false)
   m_map.removeMimeType(szMIMEType);
```

removeMimeType( )

Format

public void removeMimeType(java.lang.String szMIMEType)

Description

Removes the MIME type from the hash table.

Parameters

szMIMEType
The name of the MIME type.

Return Value

None.

Exceptions

None.

Example

```
m_map.removeMimeType(szMIMEType);
```

saveMIMEMappings( )

Format

public void saveMIMEMappings()

Description

Writes the MIME settings to the MIME mapping file, `Annotator.mime`.

Parameters

None.

Return Value

None.

Exceptions

java.io.Exception

Example

None; this method should be called only by advanced programmers.

setMimeMap( )

Format

```
public void setMimeMap(java.lang.String szMimeType, java.lang.String szAnnotation,
                       java.lang.String szParser, java.lang.String szPlayer)
```

Description

Maps a MIME type to an annotation, parser, and player.

Parameters

szMimeType
The name of the MIME type.

szAnnotation
The name of the annotation to be mapped to the MIME type.

szParser
The name of the parser to be mapped to the MIME type.

szPlayer
The name of the player to be mapped to the MIME type.

Return Value

None.

Exceptions

java.io.IOException

Example

None; this method should be called only by advanced programmers.

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 *inter*Media 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 3.10 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. To view the new attributes, the client may need to refresh the annotation.

Parameters

ann
The annotation instance from which the extraction was performed.

Return Value

None.

Exceptions

None.

Example

See Section 3.7 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 3.8 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 3.6 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 nonfatal errors.

Parameters

ann
The annotation instance.

e
An exception that explains why the failure occurred.

Return Value

None.

Exceptions

None.

Example

See Section 3.9 for an example of this method.

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 ConsoleOutput(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 3.11 for an example of this method.

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

This section presents reference information on the constructors and 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.

When the engine is initialized, it loads a static copy of the system Preference object, reading from the `Annotator.prefs` file in the configuration directory. By default, Oracle *inter*Media Annotator assumes that the configuration directory is the `\lib\conf` or `/lib/conf` subdirectory of the current directory.

However, at installation, the configuration files are placed in the following directory:

- On UNIX: *`$ORACLE_HOME`*`/ord/Annotator/lib/conf`
- On Windows: *`ORACLE_HOME`*`\ord\Annotator\lib\conf`

You must ensure that the configuration directory contains the preferences file and other configuration files before you use Oracle *inter*Media Annotator. For example, if you run *inter*Media Annotator from the directory `/usr5/myfiles`, *inter*Media Annotator assumes the configuration directory to be `/usr5/myfiles/lib/conf`.

For more information about the preferences file, see Section 2.1.

You can set the preferences using the setPreferences( ) method and you can retrieve the preferences using the getPrefs( ) method. See "setPreferences( )" on page 4-76 and "getPrefs( )" on page 4-73 for more information.

The implementation of this class is independent of the other *inter*Media Annotator classes.

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

Preferences Constructor

Format

public Preferences( )

Description

Creates a Preferences object, reading preferences from the default preferences file, `Annotator.prefs`. For more information about this file, see Section 2.1.

Parameters

None.

Return Value

None.

Exceptions

None.

Example

```
Preferences prefs = new Preferences();
```

Preferences(Properties) Constructor

Format

public Preferences(java.util.Properties props)

Description

Creates a Preferences object, reading preferences from a property list and populating the object with the preferences. The list can be loaded from a file, which must use the format and parameters described in Section 2.1.

Parameters

props
The property list.

Return Value

None.

Exceptions

None.

Example

```
// Create a new property list.
Properties myprop = new Properties();
// Populate the list with the preferences specified in a file.
// Use a full or relative path for the file name.
myprop.load(new FileInputStream("my_file_name");
Preferences pref = new Preferences(myprop);
Preferences.setPreferences(pref);
```

clone( )

Format

public java.lang.Object clone( )

Description

Creates and returns a copy of this object. For more information, see the Java 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 *inter*Media Annotator preferences.

getPrefs( )

Format

public static Preferences getPrefs( )

Description

Gets the system 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 3.4 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 *inter*Media 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 *inter*Media Annotator preferences.

setPreferences( )

Format

public static void setPreferences(Preferences prefs)

Description

Sets the system 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

```
Preferences pref = new Preferences(myprop);
Preferences.setPreferences(pref);
```

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 3.4 for an example of this method.

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 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 *inter*Media Annotator engine in the Java virtual machine. Note that the Status class is not thread-safe.

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 3.4 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 3.4 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 3.4 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 3.4 for an example of this method.

5

Creating PL/SQL Upload Templates

Oracle *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 keywords specific to *inter*Media Annotator.

You create your own PL/SQL Upload Templates using a text editor.

5.1 Overview of Uploading Media Data

Oracle *inter*Media Annotator can use two different methods to upload the media data and associated 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). The media source will be loaded directly from the file system to the database. The import upload method uses the Oracle *inter*Media import( ) method.

In the **remote** upload method, the media source does not have to be visible to the database. The file is loaded into *inter*Media Annotator, which loads the file into the database through JDBC calls. The remote upload method uses the ${MANN_UPLOAD_SRC} *inter*Media Annotator-specific keyword.

Note the following about the upload methods:

- If you use the import upload method 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.

- If you use the import upload method 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 use the remote upload method and you are using the JDBC Thin driver, your upload performance may be poor, especially if you are uploading large files.

Using a combination of PL/SQL and keywords specific to Oracle *inter*Media Annotator, you upload the media data and the annotation into a table in an Oracle database. The table must have at least one column of the appropriate *inter*Media object type, such as ORDSYS.ORDAudio for annotations for audio files, ORDSYS.ORDImage for annotations for image files, or ORDSYS.ORDVideo for annotations for video files.

You can insert the data into a new row in the table, or you can update an existing row in the table.

Section 5.3 describes the *inter*Media Annotator-specific keywords.

5.2 Creating a PL/SQL Upload Template

Use any text editor to create the PL/SQL Upload Template. Note the following about the structure of a PL/SQL Upload Template:

- 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 keywords that are specific to *inter*Media Annotator. For more information on the keywords, see Section 5.3. 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.

5.3 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 (}). These keywords are interpreted by the *inter*Media Annotator preprocessor, which then generates the appropriate PL/SQL code.

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

You can use attributes names as keywords, as described in Section 5.3.1. In addition, you can use the keywords described in Section 5.3.2 through Section 5.3.6.

5.3.1 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 5–1 shows keywords that will later be replaced with attribute values.

Example 5–1 Attribute Names as Keywords

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

INSERT INTO movieTable VALUES (videoSeq.NEXTVAL -- VideoID
                      '${MEDIA_SOURCE_FILENAME}',
                      '${MEDIA_TITLE}',
                      '${MOVIE_DIRECTOR}',
                      '${MOVIE_CAST}',
                      ORDSYS.ORDVIDEO.init());
```

For a list of attribute names defined by *inter*Media Annotator, see Appendix C.

5.3.2 ${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 subannotation of the given type. The name of the annotation type follows the ${MANN_BEGIN_ITERATE} keyword.

Example 5–2 shows a block of code that will be run for each TextSampleAnn annotation that exists as a subannotation of the current annotation.

Example 5–2 ${MANN_BEGIN_ITERATE} and ${MANN_END_ITERATE}

```
${MANN_BEGIN_ITERATE} TextSampleAnn
     INSERT INTO txtSampleTable VALUES (currClipId,  -- VideoID
                                        trackId,
                                        ROUND(${(seconds)SAMPLE_TIMESTAMP}, 4),
                                               '${TEXTSAMPLE_VALUE}' );
```

5.3.3 ${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 executed 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 5–3 shows a block of code that executes only if the MEDIA_SOURCE_MIME_TYPE attribute is defined in the current annotation.

Example 5–3 ${MANN_BEGIN_IFDEF} and ${MANN_END_IFDEF}

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

5.3.4 ${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 executed 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 5–4 shows a block of code that is executed only if the MEDIA_SOURCE_MIME_TYPE attribute is defined as `audio/basic` in the current annotation.

Example 5–4 ${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}
```

5.3.5 ${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 *inter*Media Annotator, which 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 5.1 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 5–5 shows a block of code that uploads the current media source data to the source.localData attribute of the server-side *inter*Media object videoObj.

Example 5–5 ${MANN_UPLOAD_SRC}

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

5.3.6 ${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 5–6 shows a block of code that uploads the current annotation to the comments attribute of the server-side *inter*Media object videoObj.

Example 5–6 ${MANN_UPLOAD_XML}

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

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

5.4 Complete PL/SQL Upload Template Example

Example 5–7 contains a sample PL/SQL Upload Template. It uploads a video object and its associated annotation to an Oracle database table named MediaTable. The sample contains one anonymous PL/SQL block containing a mix of PL/SQL calls and keywords specific to *inter*Media Annotator.

Example 5–7 PL/SQL Upload Template Sample

```
DECLARE
   videoObj       ORDSYS.ORDVIDEO;
   ctx            RAW(64) := NULL;
BEGIN
   INSERT INTO MediaTable VALUES (
      1,
      ORDSYS.ORDVIDEO.init();

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

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

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

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

END;
```

In addition to setting the MEDIA_SOURCE_MIME_TYPE as shown in Example 5–7, you can set other attributes, such as the MEDIA_TITLE, MEDIA_SOURCE_DIRECTORY, and MEDIA_SOURCE_FILE_FORMAT. See *Oracle interMedia User's Guide and Reference* for more information.

5.5 Saving Files

After you have written your PL/SQL Upload Template, save it with the suffix `.ofm`. Oracle *inter*Media Annotator uses the following default directory for PL/SQL Upload Templates:

- On UNIX: *$ORACLE_HOME*/ord/Annotator/ofm
- On Windows: *ORACLE_HOME*\ord\Annotator\ofm

To change the default directory, modify the value of the ofmDirectory parameter in the Annotator.prefs file.

Part II

Oracle *inter*Media Annotator Extensibility

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

- Chapter 6, "Custom Parser Example"
- Chapter 7, "Annotator Parser API Reference Information"
- Chapter 8, "Creating New Annotation Types"

6

Custom Parser Example

This chapter provides an example of creating a new parser for a custom content format. It describes AuParser, which is an example of a user-developed parser for a custom content format that was written using the Oracle *inter*Media Annotator parser API. This example 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 source code is contained in the file `AuParser.java`, which is included in the `parsers.zip` file at the following location:

- On UNIX: *`$ORACLE_HOME`*`/ord/Annotator/src`
- On Windows: *`ORACLE_HOME`*`\ord\Annotator\src`

The example shown in this chapter will not necessarily match the code shipped as `AuParser.java` with the Oracle *inter*Media Annotator installation. If you want to run this example on your system, use the file provided with the 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 numbers enclosed in brackets; these indicate that further explanation of that code is in the numbered list immediately following the example.

6.1 Parser Creation Overview

To define a new parser, perform the following operations:

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

   The new 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 the parser to perform.

   Section 6.3 through Section 6.10 describe the contents of the `AuParser.java` example.

2. Write a parser descriptor XML file and add it to the following directory:

   - On UNIX:

     $ORACLE_HOME`/ord/Annotator/lib/descriptors/parsers`

   - On Windows:

     ORACLE_HOME`\ord\Annotator\lib\descriptors\parsers`

   This directory also contains an example of a parser descriptor XML file, `AuParser.xml`.

3. Optionally, modify the `Annotator.mime` file to set your parser to be used for a specific MIME type. This file is located in the following directory:

   - On UNIX: *$ORACLE_HOME*`/ord/Annotator/lib/conf`
   - On Windows: *ORACLE_HOME*`\ord\Annotator\lib\conf`

6.2 AU File Structure

Example 6–1 shows the basic structure of an AU formatted file.

Example 6–1 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 parameter `magic` must be equal to SND_MAGIC ((int)0x2e736e64), which is a representation of the ASCII characters ".snd". The parameter `info` will be associated with the user data attribute of the annotation.

6.3 Package and Import Statements

Example 6–2 shows the import statements that must be included in the Java program to properly run an *inter*Media Annotator parser, and the package statements that set the package of this class.

Example 6–2 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.*;
```

6.4 Class Definition and Instance Variables

Example 6–3 shows the class definition and instance variables for the AuParser class.

Example 6–3 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 "Class oracle.ord.media.annotator.parsers.Parser" on page 7-32 for more information on the fields and methods of the Parser class.

The AuParser class 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.

The AuParser class contains a FormatInfo object named m_fiFormatInfo, which is used to encapsulate information related to a specific data format. See Section 6.5 for more information. These are specific to the `AuParser.java` example and may not be necessary for your parser.

The AuParser class also contains the following instance variables:

- m_iBitsPerSample: The number of audio bits per sample, as an integer.
- m_iSampleRate: The sampling rate of the AU file, as an integer.
- m_iChannelCount: The number of channels in the AU file, as an integer.
- m_szUserData: Optional text information related to the AU file.

6.5 FormatInfo Class

Example 6–4 shows the contents of the FormatInfo class, which is used to encapsulate information related to a specific data format.

Example 6–4 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.

6.6 AuParser( ) Method

Example 6–5 shows the contents of the AuParser( ) method, which is the constructor of this class.

Example 6–5 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 object.
2. Populates the m_htFormatInfo object with data by calling the FillFormatHashTable( ) method. See Example 6–9 for more information.

6.7 parse( ) Method

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

Example 6–6 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. Using the readInt( ) method, reads the first integer (the magic number) in the AU file and sets it to iMagicNumber.

   The m_madisResource field represents the MADataInputStream object. The MADataInputStream object holds the input stream that contains the AU file to be processed. See "Class oracle.ord.media.annotator.utils.MADataInputStream" on page 7-38 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. Using the readInt( ) method, reads the next integer in the m_madisResource field, 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 using the AnnTaskManager.setTask( ) method. 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. See "setTask( ) on page 7-26 for more information.

6. Sets the current task in the AnnTaskMonitor object using the AnnTaskManager.setTaskCurrent( ) method. The current value is set to 8 (the number of bytes read), and the message is set to "Parsing AU Header..." See "setTaskCurrent(int)" on page 7-27 for more information.

7. Reads the next integer in the m_madisResource field, 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 the m_madisResource field, 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 the m_madisResource field, 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 the m_madisResource field, 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. Using the readString( ) method, 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 the m_fiFormatInfo variable by getting the appropriate value from the Hashtable object named m_htFormatInfo.
15. Sets the value of the m_iBitsPerSample variable to zero as a default.
16. Checks the value of the iDataFormat variable against a series of values in the m_htFormatInfo Hashtable object and sets the m_iBitsPerSample variable 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 6.8 for more information. This method should be called at the end of any implementation of the parse( ) method.

6.8 saveToAnnotation( ) Method

Example 6–7 shows the saveToAnnotation( ) method. This method should be called after the parse( ) method has successfully finished.

Example 6–7 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 the m_fiFormatInfo variable 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 the m_szUserData variable has a value, sets it to the MEDIA_USER_DATA attribute in the annotation.
4. Using the setAttribute( ) method, sets the values of the m_iBitsPerSample, m_iSampleRate, and m_iChannelCount variables to the AUDIO_BITS_PER_SAMPLE, AUDIO_SAMPLE_RATE, and AUDIO_NUM_CHANNELS attributes in the annotation, respectively.

To create a subannotation, a parser uses the AnnotationFactory class to create a subannotation and attach it to the m_annInst variable. However, the `AuParser.java` example does not create subannotations. See the

`QtParser.java` example, which is the QuickTime parser included with *inter*Media Annotator for an example of a parser that creates subannotations.

6.9 extractSamples( ) Method

Example 6–8 shows the extractSamples( ) method. This method is invoked by the AnnotationHandler.extractMedia( ) method.

Example 6–8 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();
}
```

Oracle *inter*Media 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 and the Report( ) method to print a message stating that this parser does not support sample extraction. See "Class oracle.ord.media.annotator.utils.Status" on page 4-78 for more information.
2. Ends the current task with the AnnTaskManager.done( ) method. See "done( )" on page 7-14 for more information.

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

6.10 FillFormatHashTable( ) Method

Example 6–9 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 documentation for more information. This method is specific to the `AuParser.java` example and may not be needed for your parser.

Example 6–9 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"));
}
```

7

Annotator Parser API Reference Information

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

To create a custom parser, in addition to using this API to create a Java class, you must write a parser descriptor XML file and add it to the following directory:

- On UNIX:

  $ORACLE_HOME/ord/Annotator/lib/descriptors/parsers

- On Windows:

  ORACLE_HOME\ord\Annotator\lib\descriptors\parsers

For an example of a parser descriptor XML file, see Chapter 6 and the `AuParser.xml` file in the `parsers` directory.

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 of methods is intended for advanced users (those who write or add parsers to *inter*Media Annotator).

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.

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 of methods is intended for advanced users (those who write or add parsers to *inter*Media Annotator).

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.

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

This section presents reference information on the constructor and 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 "Class oracle.ord.media.annotator.handlers.AnnTaskMonitor" on page 4-19.

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

AnnTaskManager Constructor

Format

public AnnTaskManager( )

Description

Creates an AnnTaskManager object. This constructor is used by the annotation handler to create an annotation task manager when a new task begins.

Parameters

None.

Return Value

None.

Exceptions

None.

Example

```
AnnTaskManager tskmgr = new AnnTaskManager();
```

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 6.9 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 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 on page 7-23 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 on page 7-22 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 6.7 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 6.7 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 6.7 for an example of this method.

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

This section presents reference information on the constructor and methods of the AnnotationFactory class. This class is the factory class for annotations; it contains two subfactories (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.

AnnotationFactory Constructor

Format

public AnnotationFactory(oracle.ord.media.annotator.handlers.utils.MimeMap mm)

Description

This constructor deals with the mapping from a MIME type or file extension to an annotation, parser, and a player instance.

Parameters

mm
The MimeMap instance. See "Class oracle.ord.media.annotator.handlers.utils.MimeMap" on page 4-49 for more information about the MimeMap object.

Return Value

None.

Exceptions

None.

Example

```
MimeMap() m_map = new MimeMap();
AnnotationFactory m_annfact = new AnnotationFactory(m_map);
```

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 3.6 for an example of this method.

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

This section presents reference information on the constructor and 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 progress information related to the parsing of the media data. The Parser object is associated with an AnnotationFactory object to create a subannotation 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 subannotations.

- 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 application.

Parser Constructor

Format

public Parser( )

Description

Creates a new Parser object. Any subclass of the Parser class must have an empty constructor.

Parameters

None.

Return Value

None.

Exceptions

None.

Example

None; the user should not call this constructor directly.

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 6.9 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 6.7 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 6.7 for an example of this method.

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

This section presents reference information on the constructors and 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.

MADataInputStream(InputStream, boolean, String, String) Constructor

Format

```
public MADataInputStream(java.io.InputStream is, boolean bLittleEndian,
                         java.lang.String szCharEncoding,
                         java.lang.String mimetype)
```

Description

Creates an MADataInputStream object and saves the input stream argument for later use. This constructor sets whether the data is in little endian format or big endian format. Integers (short, int, long) of the input stream have the specified endian format.

Parameters

is
The underlying input stream.

bLittleEndian
Whether the data is in little endian format or big endian format. True indicates that the data is in little endian format.

szCharEncoding
The character encoding to be used in reading the input string.

mimetype
The MIME type of the input stream.

Return Value

None.

Exceptions

oracle.ord.media.annotator.AnnotatorException

Example

None; the user should not call this constructor directly.

MADataInputStream(MADataInputStream, boolean, String, String) Constructor

Format

```
public MADataInputStream(MADataInputStream is, boolean bLittleEndian,
                         java.lang.String szCharEncoding,
                         java.lang.String mimetype)
```

Description

Creates an MADataInputStream object and saves the input stream argument for later use. This constructor sets whether the data is in little endian format or big endian format. Integers (short, int, long) of the input stream have the specified endian.

Parameters

is
The underlying input stream.

bLittleEndian
Whether the data is in little endian format or big endian format. True indicates that the data is in little endian format.

szCharEncoding
The character encoding to be used in reading the input string.

mimetype
The MIME type of the input stream.

Return Value

None.

Exceptions

None.

Example

None; the user should not call this constructor directly.

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

```
m_madisResource.close();
```

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 on page 7-72 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 the next caller 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 the next caller 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 Annotator 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 Annotator 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 Annotator 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 Annotator 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 Annotator 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 on page 7-44 for an example of this method.

readLong( )

Format

public long readLong( )

Description

Reads the next 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 Annotator 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 primitive type short in the underlying input stream.

Parameters

None.

Return Value

This method returns the value of the primitive type 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 on page 7-44 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 on page 7-44 for an example of this method.

skipBytes(long)

Format

public long skipBytes(long 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 skipped.

Exceptions

java.io.IOException

Example

```
long number = 256;
if(number == m_madisResource.skipBytes(number)
     int data = m_madisResource.readInt();
```

8

Creating New Annotation Types

In addition to the supplied annotation types, you can use Oracle *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 the following file, which is included when you install Oracle *inter*Media Annotator:

- On UNIX:

  $ORACLE_HOME/ord/Annotator/demo/examples/SampleInventoryAnn.xml

- On Windows:

  ORACLE_HOME\ord\Annotator\demo\examples\SampleInventoryAnn.xml

8.1 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), which is defined in the `AnnotationDescriptor.dtd` file. This file is located in the `annotations` subdirectory. The following shows the full specification for this directory:

- On UNIX:

  $ORACLE_HOME/ord/Annotator/lib/descriptors/annotations

- On Windows:

  ORACLE_HOME\ord\Annotator\lib\descriptors\annotations

The `AnnotationDescriptor.dtd` file describes the AnnotationDescriptor DTD, which contains two elements: AnnotationProperties and AttributeDescriptors.

8.1.1 AnnotationProperties Element

The AnnotationProperties element contains elements that provide information about the annotation as a whole. These elements are the following:

- Name: Required element. It contains the name of the new annotation type.
- Version: Required element. It contains the version number.
- Description: Optional element. It contains a brief description of the annotation as a whole.
- Extends: 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: Reserved element. Do not assign a value to this element.
- ClassName: Reserved element. Do not assign a value to this element.
- IconFileName: Reserved element. Do not assign a value to this element.

8.1.2 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: Required element. It contains the name of your new attribute.
- AttributeType: Required element. It contains the Java object type of the attribute value. AttributeType must be a Java object type; Java primitive types 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 return 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.

- AttributeTypePattern: Optional element which should be used only if the AttributeType is java.lang.Date. This element specifies the String pattern that should be used when displaying the date. The pattern follows the syntax in java.text.SimpleDateFormat.
- AttributeAlias: Optional element. It defines a shorter attribute name for display purposes.
- AttributeDescription: Optional element. It defines a short description of the attribute.
- AttributeDefaultValue: Optional element. It defines the default value of the attribute to be inserted in the annotation.

8.1.3 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>
```

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.

8.2 Using a New Annotation Type

When you finish writing the XML file, you should save it to the `annotations` directory.

Now you can use your new annotation type in the same way that you use the predefined annotation types. See Chapter 2 and Chapter 3 for more information on creating new annotations.

Part III

Appendixes

This part contains the following appendixes:

- Appendix A, "Querying Stored Annotations"
- Appendix B, "Supported File Formats"
- Appendix C, "Defined Annotation Attributes"
- Appendix D, "Deprecated Features"
- Appendix E, "Frequently Asked Questions"

A

Querying Stored Annotations

After the media data and the annotation are uploaded into an Oracle database, you can use Oracle 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 Oracle Text index on a database table, VideoStorage. This code excerpt generates an Oracle 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');
```

The following example shows the SQL statement that creates the VideoStorage table:

```
CREATE TABLE VideoStorage OF VideoType (ID PRIMARY KEY)
             LOB(vsrc.source.localdata) STORE AS (NOCACHE NOLOGGING);
```

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 Oracle Text) of the video clips that contain John Doe in the MOVIE_CAST attribute of the associated annotation.

The preceding PL/SQL statements are available in the following file:

- On UNIX:

  $ORACLE_HOME/ord/Annotator/demo/examples/SampleCode.sql

- On Windows:

  ORACLE_HOME\ord\Annotator\demo\examples\SampleCode.sql

For more information, see the Oracle Text documentation, especially *Oracle Text Application Developer's Guide.*

B

Supported File Formats

Oracle *inter*Media Annotator supports the following file formats:

- AIFF / AIFC
- Apple QuickTime 5.0
- AU
- BMP
- GIF87a / GIF89a
- JPEG / JFIF
- MPEG I / II Audio, all layers
- MPEG I Video
- Microsoft AVI
- RealMedia
- TIFF
- WAV

Table B–1 shows the media source formats that have built-in support from Oracle *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
AIFF	audio / x-aiff audio / x-aif	None

Table B–1 (Cont.) Built-in Parsers

File Format	MIME type	Extraction
Apple QuickTime 5.0	video/quicktime	Text track
		Video frame (with Apple QuickTime Library only)
BMP	image/bmp	None
GIF	image/gif	None
JPEG/JFIF	image/jpeg	None
	image/jpg	
MPEG I/II Audio	audio/x-mpeg	None
MPEG I video	video-mpeg	None
RealMedia	video/x-realvideo	None
	video/x-realaudio	
RIFF	video/x-msvideo	None
	application/x-troff-msvideo	
	audio/x-wav	
Sun Audio	audio/basic	None
TIFF	image/tiff	None

C

Defined Annotation Attributes

The tables in this appendix show the defined annotation attributes for each Oracle *inter*Media Annotator.

Table C–1 lists the attributes for the MediaAnn annotation type, which is used for all media, and the data type and description of each attribute.

Table C–1 MediaAnn Annotation Attributes

Attribute	Data Type	Description
MEDIA_AUTHORING_TOOL	java.lang.String	Authoring tool used to create the media
MEDIA_BITRATE	java.lang.Long	Bit rate of the media (in bits per second)
MEDIA_CATEGORY	java.lang.String	Media category or genre
MEDIA_CONTENT_DATE	java.lang.String	Creation date of the media content
MEDIA_COPYRIGHT	java.lang.String	Copyright information for the media
MEDIA_CREATION_TIME	java.util.Date	Creation time of the media (for example, Mon Dec 13 19:29:04 EST 2001)
MEDIA_CREDITS	java.lang.String	Credits for content providers
MEDIA_DESCRIPTION	java.lang.String	Description of the media
MEDIA_DURATION	oracle.ord.media.annotator.types.TimeCodeString	Duration of the media, in the form hour:minute:second:mantissa, (for example, HH:MM:SS:FF) where mantissa is the fraction of a second in the units defined by MEDIA_TIMESCALE

Table C–1 (Cont.) MediaAnn Annotation Attributes

Attribute	Data Type	Description
MEDIA_FORMAT_ENCODING	java.lang.String	Format of the media encoding
MEDIA_FORMAT_ENCODING_CODE	java.lang.String	Short form (4 characters) of the media encoding
MEDIA_INFORMATION	java.lang.String	Information on the media
MEDIA_LANGUAGE	java.lang.String	Language of the media
MEDIA_MODIFICATION_TIME	java.util.Date	The time of the last modification of the media (for example, Mon Dec 13 19:29:04 EST 2001)
MEDIA_PRODUCER	java.lang.String	Producer of the media
MEDIA_SIZE	java.lang.Long	Size of the media source, in bytes
MEDIA_SOURCE_DIRECTORY	java.lang.String	Directory where the source is stored
MEDIA_SOURCE_FILE_FORMAT	java.lang.String	Media file format
MEDIA_SOURCE_FILE_FORMAT_CODE	java.lang.String	Short form (4 characters) of the media file format
MEDIA_SOURCE_FILENAME	java.lang.String	File name of the source
MEDIA_SOURCE_MIME_TYPE	java.lang.String	MIME type of the media and its samples
MEDIA_SOURCE_PROTOCOL	java.lang.String	URL protocol of the parsed media source
MEDIA_SOURCE_STREAMABLE	java.lang.String	The streaming server for which the media is optimized, if any
MEDIA_SOURCE_URL	java.lang.String	Location or URL of the parsed media source
MEDIA_TIMESCALE	java.lang.Float	Number of units in a second
MEDIA_TITLE	java.lang.String	Title of the media
MEDIA_TRACK_ID	java.lang.Integer	Track identifier for the annotation
MEDIA_USER_DATA	java.lang.String	String containing miscellaneous user data

Table C–2 lists the attributes for the AudioAnn annotation type, which is used for for audio media, and the data type and description of each attribute. The AudioAnn annotation type extends MediaAnn.

Table C–2 AudioAnn Annotation Attributes

Attribute	Data Type	Description
AUDIO_ARTIST	java.lang.String	Main artist for the audio clip
AUDIO_BITS_PER_SAMPLE	java.lang.Integer	Number of bits per sound sample
AUDIO_NUM_CHANNELS	java.lang.Integer	Number of audio channels
AUDIO_SAMPLE_RATE	java.lang.Integer	Audio sampling rate (in samples per second)

Table C–3 lists the attributes for the VideoAnn annotation type, which is used for for a video track annotation, and the data type and description of each attribute. The VideoAnn annotation type extends MediaAnn.

Table C–3 VideoAnn Annotation Attributes

Attribute	Data Type	Description
VIDEO_DEPTH	java.lang.Integer	Number of bits for the color depth
VIDEO_FRAME_RATE	java.lang.Long	Video frame rate (in frames per second)
VIDEO_FRAME_SIZE	java.lang.Long	Video frame size (in bytes)
VIDEO_HORIZONTAL_RES	java.lang.Integer	Horizontal resolution (in pixels per inch)
VIDEO_IS_GRAYSCALE	java.lang.Boolean	Whether or not the video has colors
VIDEO_SRC_HEIGHT	java.lang.Long	Video height (in pixels)
VIDEO_SRC_WIDTH	java.lang.Long	Video width (in pixels)
VIDEO_VERTICAL_RES	java.lang.Integer	Vertical resolution (in pixels per inch)

Table C–4 lists the attributes for the TextAnn annotation type, which is used for a movie text track or any time-based text, and the data type and description of each attribute. The TextAnn annotation type extends MediaAnn.

Table C–4 TextAnn Annotation Attributes

Attribute	Data Type	Description
TEXT_ALIGN	java.lang.String	Alignment of the text track: left-justified, right-justified, fully justified, or centered
TEXT_BG_COLOR	java.lang.String	Background color (for example, 0x00RRGGBB) of the text track

Table C–4 (Cont.) TextAnn Annotation Attributes

Attribute	Data Type	Description
TEXT_DEF_BOX	oracle.ord.media.annotator.types.Rectangle	Default text box size, consisting of four instances of the Java primitive type short
TEXT_FG_COLOR	java.lang.String	Foreground color (for example, 0x00RRGGBB) of the text track
TEXT_FONTFACE	java.lang.String	Font styles used (such as italics or boldface)
TEXT_FONTNAME	java.lang.String	Name of the font used
TEXT_FONTSIZE	java.lang.Short	Point size of the text track

Table C–5 lists the attributes for the MovieAnn annotation type, which is used for movie media, and the data type and description of each attribute. The MovieAnn annotation type extends MediaAnn.

Table C–5 MovieAnn Annotation Attributes

Attribute	Data Type	Description
MOVIE_CAST	java.lang.String	Names of the performers in the movie
MOVIE_DIRECTOR	java.lang.String	Director of the movie
MOVIE_EDIT_INFORMATION	java.lang.String	Information about the editing
MOVIE_WARNING	java.lang.String	Movie rating and warning information

Table C–6 lists the attributes for the ImageAnn annotation type, which is used for image tracks, and a description of each attribute. The ImageAnn annotation type extends MediaAnn.

Table C–6 ImageAnn Annotation Attributes

Attribute	Data Type	Description
IMAGE_BITS_PER_PIXEL	java.lang.Integer	Number of bits per image pixel
IMAGE_COUNT	java.lang.Integer	Number of images stored in the file
IMAGE_HEIGHT	java.lang.Long	Height of the image
IMAGE_HORIZONTAL_RES	java.lang.Double	Horizontal resolution (in pixels per inch)
IMAGE_PIXEL_FORMAT	java.lang.String	The color space of the image, including the resolution

Table C–6 (Cont.) ImageAnn Annotation Attributes

Attribute	Data Type	Description
MEDIA_TIMESCALE	java.lang.Float	Number of units in a second
IMAGE_VERTICAL_RES	java.lang.Double	Vertical resolution (in pixels per inch)
IMAGE_WIDTH	java.lang.Long	Width of the image

Table C–7 lists the attributes for the IptcIimAnn annotation type, which is used for the Information Interchange Model (IIM) format, and the data type and description of each attribute. The IIM format is a container file format for news information. The format was developed by the International Press Telecommunications Council (IPTC).

Table C–7 IptcIimAnn Annotation Attributes

Attribute	Data Type	Description
IIM_ACTION_ADVISED	java.lang.Long	Action to be taken after viewing the object. Valid values and their meanings are: 01: Destroy 02: Replace 03: Append
IIM_BYLINE	java.lang.String	Creator of the object.
IIM_BYLINE_TITLE	java.lang.String	Title of the creator of the object.
IIM_CAPTION	java.lang.String	Caption or abstract of the object, used especially when the object is not text.
IIM_CITY	java.lang.String	City of origin of the object.
IIM_CONTACT	java.lang.String	Contact for further information.
IIM_COPYRIGHT	java.lang.String	Copyright information.
IIM_COUNTRY_CODE	java.lang.String	ID of the country or geographic location of the origin of the object.
IIM_CREATION_DATE	java.util.Date	Creation date of the physical object. The date consists of the Date Created and Time Created record sets.
IIM_CREDIT	java.lang.String	Name of the service transmitting the object.

Table C–7 (Cont.) IptcImAnn Annotation Attributes

Attribute	Data Type	Description
IIM_DIGITAL_CREATION_DATE	java.util.Date	Date of creation of the digitized version of the object. The date consists of the Digital Creation Date and Digital Creation Time record sets.
IIM_HEADLINE	java.lang.String	Synopsis of the content of the object listing keywords to add in a more detailed search for an object.
IIM_IMAGE_TYPE	java.lang.String	A code for the image type. The code consists of two bytes. Possible values for the first byte and their meanings are: 0: No object 1: Monochrome 2, 3, or 4: Multiple components for a color project 9: Object data contains supplementary data Possible values for the second byte and their meanings are: W: Monochrome Y: Yellow component M: Magenta component C: Cyan component K: Black component R: Red component G: Green component B: Blue component T: Text only F: Full color, frame sequential L: Full color, line sequential P: Full color, pixel sequential S: Full color, special interleaving
IIM_KEYWORDS	java.lang.String	Keywords associated with the object (supported with version 2 of IIM).
IIM_LANGUAGE	java.lang.String	ID for the major language.
IIM_LOCATION_NAME	java.lang.String	Full name of the country of origin of the object.

Table C–7 (Cont.) IptcIimAnn Annotation Attributes

Attribute	Data Type	Description
IIM_OBJECT_CYCLE	java.lang.Character	The part of the day. Valid values are: a: a.m. (morning) p: p.m. (afternoon and evening b: both)
IIM_OBJECT_NAME	java.lang.String	Title of the object.
IIM_ORIGINATING_PROG	java.lang.String	The software used to create the object.
IIM_PROGRAM_VERSION	java.lang.String	Version of the software used to create the object.
IIM_PROVINCE_STATE	java.lang.String	ID of the province or state of origin of the object.
IIM_RECORD_VERSION	java.lang.Long	Version of the IPTC IIM specification.
IIM_SOURCE	java.lang.String	Owner of the object.
IIM_SPECIAL_INSTRUCTION	java.lang.String	Special instructions concerning the use of the objects, such as embargoes and warnings.
IIM_SUB_LOCATION	java.lang.String	ID of the location within the city of origin of the object.
IIM_TRANSMISSION_REF	java.lang.String	Code representing the location of the original transmission. For example, BER-5, PAR-12-11-01.
IIM_WRITER	java.lang.String	Writer or editor.

Table C–8 lists the attributes for the SampleAnn annotation type, which is used for media samples, and the data type and description of the attribute. The SampleAnn annotation type extends MediaAnn.

Table C–8 SampleAnn Annotation Attributes

Attribute	Data Type	Description
SAMPLE_TIMESTAMP	oracle.ord.media.annotator.types.TimeCodeString	Time stamp of the specified sample, in the form hour:minute:second:mantissa (for example, HH:MM:SS:FF), where mantissa is the fraction of a second in the units defined in MEDIA_TIMESCALE

Table C–9 lists the attributes for the TextSampleAnn annotation type, which is used for a movie text track sample or any time-based text. It also lists the data type and description of each attribute. TextSampleAnn extends SampleAnn.

Table C–9 TextSampleAnn Annotation Attributes

Attribute	Data Type	Description
TEXTSAMPLE_VALUE	java.lang.String	String value of the text sample

Table C–10 lists the attributes for the VideoFrameSampleAnn annotation type, which is used for a video frame extracted from a video track as a sample, and the data type and description of each attribute. The VideoFrameSampleAnn annotation type extends SampleAnn.

Table C–10 VideoFrameSampleAnn Annotation Attributes

Attribute	Data Type	Description
VIDEO_FRAME_SAMPLE_HEIGHT	java.lang.Integer	Height of the frame extracted from the video track
VIDEO_FRAME_SAMPLE_WIDTH	java.lang.Integer	Width of the video frame extracted from the video track

D

Deprecated Features

This appendix lists the features that have been removed or deprecated from Oracle *inter*Media Annotator.

D.1 Features Removed

The following features have been removed from the product:

- The *inter*Media Annotator demonstration (demo) utility, including documentation

  The demo utility is a graphical user interface. You can use the Java API to perform the functions previously available in the demo utility or you can download the demo utility and its documentation from:

  `http://otn.oracle.com`

- The PL/SQL Upload Template Wizard

  Use a text editor to create PL/SQL upload templates.

- Support for audio CD track annotation. This includes the AudioCDAnn and AudioCDTrackAnn annotation types, and the `cd` URL protocol. If you require this support, contact your local Oracle Support Services.

D.2 General Deprecated Features

The following features have been deprecated:

- Connections to a CDDB: This feature has been disabled.

D.3 Methods Deprecated

The following sections list methods that have been deprecated.

D.3.1 Methods Deprecated from MADataInputStream Class

The following constructors and methods have been deprecated from the MADataInputStream class of the Parser API:

- MADataInputStream(InputStream) constructor
- MADataInputStream(InputStream, boolean) constructor
- endBlock(FourCC fccChunk) method
- getBytesRead( ) method
- getLeft( ) method
- startBlock(lSizeLeft) method

E

Frequently Asked Questions

How do I find out which attributes go with an annotation?

The following directory contains the XML files that define attributes for each annotation type.

- On UNIX:

  $ORACLE_HOME/ord/Annotator/lib/descriptors/annotations

- On Windows NT:

  ORACLE_HOME\ord\Annotator\lib\descriptors\annotations

Why won't my media file load into the database?

The file may not load because of one of the following reasons:

- 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 5.1 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.

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?

Modify the `mime.types` file, located in the following directory:

- On Unix: *`$ORACLE_HOME`*`/ord/Annotator/lib/conf`
- On Windows: *`ORACLE_HOME`*`\ord\Annotator\lib\conf`

When I am parsing a media source using the HTTP protocol, I encounter the following error: Unsupported Annotation for Content Type text/htm

Check the path of the URL pointing to the resource. It is possible that the URL is invalid.

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 Oracle *inter*Media Annotator?

The latest information, known problems, and FAQ are available at:

`http://otn.oracle.com`

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\classes12.zip`.

Does QuickTime support include sprite track and flash track?

Sprite track and flash track are not supported.

How do I submit feedback on Oracle *inter*Media Annotator?

Please contact your local Oracle Support Services.

Index

A

B

C

D

E

F

G

H

I

J

L

M

O

P

Q

R

S

T

U

V

W