Oracle *inter*Media Annotator

ユーザーズ・ガイド

リリース 9.2

2002 年 7 月

部品番号 : J06313-01

ORACLE®

Oracle *inter*Media Annotator ユーザーズ・ガイド , リリース 9.2

部品番号：J06313-01

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

原本部品番号：A96120-01

原本著者：Helen Grembowicz、Sue Pelski

原本協力者：Fengting Chen、Dongbai Guo、Susan Mavris、Susan Shepard、Rod Ward

目次

例目次

図目次

表目次

はじめに

Oracle *inter*Media Annotator は、特定のフォーマットのオーディオ・ソース、イメージ・ソースおよびビデオ・ソースから、メタデータをメディア・ソース・ファイルとともに Oracle データベースに抽出する Java API です。

対象読者

このマニュアルは、マルチメディア・ファイルからメタデータを抽出し、Oracle データベースにメタデータとマルチメディア・ファイルの両方を格納するユーザーを対象としています。*inter*Media Annotator エンジンをそれらのアプリケーションと統合するユーザーは、Java および JDBC を十分に理解している必要があります。独自の PL/SQL Upload Template を作成するユーザーは、PL/SQL を十分に理解している必要があります。独自のアノテーション型またはパーサーを作成するユーザーは、Java および XML を十分に理解している必要があります。

このマニュアルの構成

このマニュアルは、次の章および付録で構成されています。

章番号	説明
第 1 章	この章では、Oracle *inter*Media Annotator の概要を説明します。
第 2 章	この章では、Oracle *inter*Media Annotator を使用するために行う必要のある環境の設定方法および基本的な手順を説明します。
第 3 章	この章では、*inter*Media Annotator エンジンを使用した Java アプリケーションの完全な例を示します。
第 4 章	この章では、*inter*Media Annotator エンジンに対応付けられた Java API のリファレンス情報を示します。
第 5 章	この章では、PL/SQL Upload Template を作成して Oracle データベースにアノテーションをアップロードする方法を説明します。
第 6 章	この章では、カスタム・パーサーの完全な例を示します。
第 7 章	この章では、カスタム・パーサーの作成に関連する Java API のリファレンス情報を示します。
第 8 章	この章では、独自のアノテーション型を作成する方法を説明します。
付録 A	この付録では、Oracle Text を使用して格納されているアノテーションを問い合せる方法を説明します。
付録 B	この付録では、サポートされているフォーマットのリファレンス情報を示します。
付録 C	この付録では、アノテーション特性のリファレンス情報を示します。
付録 D	この付録では、使用されなくなった機能および廃止された機能について説明します。
付録 E	この付録では、FAQ を示します。

関連文書

関連する内容については、次のドキュメントを参照してください。

- 『Oracle *inter*Media ユーザーズ・ガイドおよびリファレンス』
- 『Oracle Text アプリケーション開発者ガイド』

リリース・ノート、インストール・マニュアル、ホワイト・ペーパーまたはその他の関連文書は、OTN-J（Oracle Technology Network Japan）に接続すれば、無償でダウンロードできます。OTN-J を使用するには、オンラインでの登録が必要です。次の URL で登録できます。

```
http://otn.oracle.co.jp/membership/
```

すでに OTN-J のユーザー名およびパスワードを取得済であれば、次の OTN-J Web サイトの文書セクションに直接接続できます。

```
http://otn.oracle.co.jp/document/
```

表記規則

例では、特に示していないかぎり、各行の終わりで暗黙的に改行されています。入力の各行の終わりで [Enter] キーを押す必要があります。

java.lang.String オブジェクトは、String と省略される場合があります。

このマニュアルで使用する表記規則は次のとおりです。

規則	意味
. . .	例中の垂直の省略記号は、例に直接関連しない複数の行が省略されていることを示します。
...	文中またはコマンド中の水平の省略記号は、例に直接関連しない文またはコマンドの一部が省略されていることを示します。
太字	太字は、本文中で定義されている用語、ウィンドウ名、ボタン名、メニュー名またはメニュー項目を示します。
固定幅フォント	文中の固定幅フォントは、コード例、URL または絶対パス名を示します。
イタリック体	例に使用されているイタリック体は、ユーザーが提供する名前を示します。
[ ]	大カッコは、任意に 1 つ選択するか、または選択しない句を示します。

1

Oracle *inter*Media Annotator の概要

この章では、Oracle *inter*Media Annotator について説明します。Oracle *inter*Media Annotator は、特定のフォーマットのメディア・ソースから情報（**メタデータ**）を抽出し、そのメタデータをメディア・ソースとともに Oracle データベースに挿入します。Oracle *inter*Media Annotator は、Oracle *inter*Media を使用します。この章の内容は次のとおりです。

1.1 用途

オブジェクト・リレーショナル・データベース・システムでマルチメディア・データを管理する場合、メディア・ソースに対応付けられたメタデータの抽出、処理および管理方法の問題に直面する可能性が高くなります。通常、メタデータはメディア・ソースを記述するテキストベースの情報で構成され、独自のフォーマットを使用してメディア・ソース内に埋め込まれているため、簡単にアクセスできない場合があります。メタデータの効率的な管理および使用を可能にするには、様々なタイプのメディア・ソースからメタデータを抽出できる必要があります。抽出後、元のメディア・ソースに関係なく、メタデータが一貫して正確に表現される必要があります。

Oracle *inter*Media Annotator は、Java ベースのエンジンです。これを使用すると、一連のマルチメディア・コンテンツおよびメタデータを編成し、それを Oracle データベースにアップロードできます。

*inter*Annotator を使用すると、メディア・ソースを解析したり、そのメタデータを抽出したり、論理アノテーション（またはアノテーション）という編成構造にメタデータをグループ化することができます。すべてのアノテーションは、テキスト属性とオプションのサンプルの集合として編成されます。通常、アノテーションには 1 つ以上の副アノテーションが含まれます。副アノテーションには、メディア・ソースの一部（テキスト・トラックやオーディオ・トラックなど）に対応付けられたメタデータが含まれています。これらの移入済の副アノテーションに加えて、空のアノテーションを追加し、それに独自の値を移入することによって、独自の副アノテーションを定義することができます。**属性**は、データ・フォーマット（MIME タイプやフォーマットなど）またはデータ・コンテンツ（曲名や映画監督など）のいずれかのメディア・ソース情報を提供します。

サンプルは、メディア・ソースから抽出されたマルチメディア・データ（オーディオ・クリップや字幕など）です。

*inter*Media Annotator を使用すると、オーディオ・ファイル、イメージ・ファイルまたはビデオ・ファイル（サポートされているファイル・フォーマットのリストは、付録 B「サポートされているフォーマット」を参照）を解析し、属性を抽出してアノテーションを作成することができます。

また、Oracle *inter*Media Annotator は、メディア・ソースのトラックごとに個別のアノテーションを作成します。たとえば、ムービーを含むメディア・ソースの場合、*inter*Media Annotator はビデオ・データとオーディオ・データ用に個別のアノテーションを作成することができます。これらのアノテーションは、ムービーのアノテーションの副アノテーションとなります。

また、*inter*Media Annotator を使用して、アノテーションをメディア・ソースとともに Oracle データベースに挿入することもできます。

*inter*Media Annotator エンジン API を使用して、*inter*Media Annotator 機能をアプリケーションに統合できます。この API は、多くのマルチメディア・ファイルをデータベースにバルク・ロードするためのツールとしても使用できます。

たとえば、多くの予告ムービーを格納する必要がある場合は、Web ベースのカスタム・アプリケーションを作成して、予告ムービーの解析、各予告ムービーのアノテーションの生成、および Oracle データベースへの予告ムービーのアップロードを自動的に行うことができます。

Java の Oracle *inter*Media Annotator エンジンの詳細は、第 I 部「Oracle interMedia Annotator エンジン」を参照してください。

Oracle *inter*Media Annotator は拡張可能です。*inter*Media Annotator パーサーの Java API を使用して、メディア・ソース・ファイル用のカスタム・パーサーを作成したり、独自のアノテーション型を作成することができます。

たとえば、不動産会社では、不動産物件のエントリごとの外観写真、短い室内ムービー、技術文書、販売価格およびその他の情報を含む、販売不動産物件のリストを管理する場合があります。

開発者は、*inter*Media Annotator の拡張機能を使用して、販売代理店がマルチメディア・ファイルを含むエントリを提供し、それを不動産物件のコンテンツ・ユニットとして Oracle データベースにアップロードできるようにする Web ベースのアプリケーションを簡単に作成できます。開発者は、名前、販売価格、締切日およびその他のテキスト情報を含む、`property` という名前の新しいアノテーション型を定義できます。新しいアノテーション型には、住宅の外観写真を記述するイメージ副アノテーションまたは室内ムービーを記述するムービー副アノテーションを含めることができます。また、開発者は、新しいパーサーを作成して、不動産物件のアノテーションを作成することもできます。

Oracle *inter*Media Annotator の拡張の詳細は、第 I 部「Oracle interMedia Annotator エンジン」を参照してください。

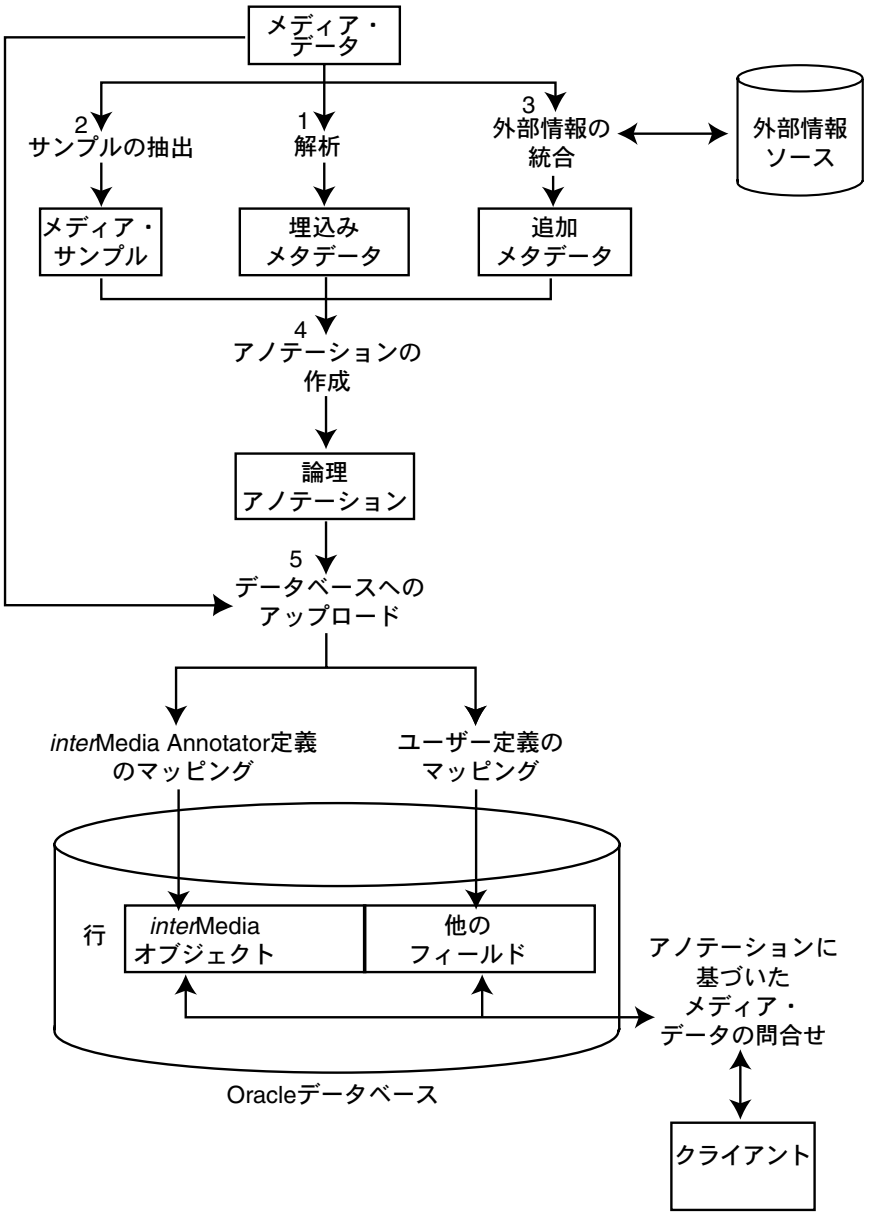
アノテーションがデータベースに格納された後は、Oracle Text を使用してそのアノテーションへの問合せができます。詳細は、付録 A を参照してください。

1.2 操作の概要

Oracle *inter*Media Annotator の主な機能は、メディア・ソースから論理アノテーションを作成し、そのアノテーションとソース・ファイルの両方を Oracle データベースにアップロードすることです。これによって、ユーザーは、アノテーションの情報に基づいて、データベース内のメディア・データへの問合せができます。

図 1-1 に、このプロセスの概要を示します。

図 1-1 Oracle *inter*Media Annotator の操作の概要



Oracle *inter*Media Annotator を使用して、次の操作を次の順序で実行できます。

1. メディア・ソースの解析。Oracle *inter*Media Annotator は、ソース・ファイルからメタデータを抽出します。
2. メディア・ソースからのサンプルの抽出。フォーマットによっては、Oracle *inter*Media Annotator は、メディア・データからサンプル（ムービー・ファイルのテキスト・トラックなど）を抽出します。
3. 追加のソースからの情報の統合。アノテーションに有効ないくつかの情報がメタデータに含まれていない場合があります。たとえば、生成済のアノテーションからデータをインポートできます。
4. 論理アノテーションの作成。Oracle *inter*Media Annotator は、抽出したサンプルとメタデータを組み合せ、論理アノテーションを作成します。

   この時点で、アプリケーションはアノテーションをさらにカスタマイズすることができます。
5. Oracle データベースへのアノテーションおよびメディア・ソースのアップロード。

   Oracle *inter*Media Annotator は、メディア・ソースおよびアノテーション（XML 形式）をデータベース内の *inter*Media オブジェクトにアップロードします。また、Oracle *inter*Media Annotator は、アノテーションの個々の属性をデータベース内の他の列にアップロードすることもできます。PL/SQL Upload Template に、アップロード先の *inter*Media オブジェクトおよび残りの情報を指定します。テンプレートは、テキスト・エディタを使用して作成できます。

   アップロード・プロセスの詳細は、第 5 章を参照してください。

一度これらの手順を完了すると、アノテーション内の情報を問い合せ、直接抽出できないメディア・ソースに関する情報を使用できます。また、Oracle Text を使用して、アノテーションの情報に索引を作成することもできます。詳細は、付録 A を参照してください。

1.3 前提条件

Oracle *inter*Media Annotator を使用するには、Oracle *inter*Media がインストールされている Oracle データベース、および Oracle8*i* リリース 8.1.5 以上に対応する、リリース 1.2 以上の Oracle JDBC ドライバ（Thin または OCI）へのアクセス権限（ローカルまたはリモートのいずれか）が必要です。

Java アプリケーションで Oracle *inter*Media の機能を使用するには、Java Development Kit（JDK）の 1.2 以上を使用する必要があります。

第 I 部

Oracle *inter*Media Annotator エンジン

第 I 部では、Java ベースの Oracle *inter*Media Annotator エンジンについて説明します。第 I 部に含まれる章は、次のとおりです。

- 第 2 章「Oracle interMedia Annotator の使用方法」
- 第 3 章「interMedia Annotator エンジン API の例」
- 第 4 章「interMedia Annotator エンジン API のリファレンス情報」
- 第 5 章「PL/SQL Upload Template の作成」

2

Oracle *inter*Media Annotator の使用方法

この章では、Oracle *inter*Media Annotator の使用方法について説明します。この章の内容は次のとおりです。

- プリファレンスの設定
- 使用可能な URL プロトコル
- Oracle interMedia Annotator 使用の手順

2.1 プリファレンスの設定

Oracle *inter*Media Annotator を使用する前に、作業環境のプリファレンスを指定する必要があります。プリファレンスは、構成ディレクトリ内の Annotator.prefs ファイルに指定します。デフォルトでは、現行ディレクトリの ¥lib¥conf サブディレクトリまたは /lib/conf サブディレクトリが、構成ディレクトリとみなされます。

ただし、インストール時、構成ファイルは次のディレクトリに含まれています。

- UNIX の場合 : *$ORACLE_HOME*/ord/Annotator/lib/conf
- Windows の場合 : *ORACLE_HOME*¥ord¥Annotator¥lib¥conf

Oracle *inter*Media Annotator を使用する前に、プリファレンス・ファイルおよび他の構成ファイルが構成ディレクトリに含まれていることを確認する必要があります。たとえば、ディレクトリ /usr5/myfiles から *inter*Media Annotator が実行された場合は、/usr5/myfiles/lib/conf が構成ディレクトリであるとみなされます。

プリファレンス・ファイルのエントリのフォーマットは、次のとおりです。

preference_parameter=preference_value

表 2-1 に、プリファレンス・パラメータおよび各パラメータの使用可能な値を示します。

表 2-1 プリファレンス

パラメータ	説明
MimeMapFile	MIME タイプをファイル拡張子、アノテーション、パーサーおよびプレーヤにマップする MIME タイプのマッピング・ファイルの名前。このファイルは、configDirectory パラメータで指定されているとおり、構成ディレクトリに格納する必要があります。 デフォルトのファイル名は、Annotator.mime です。
MimeTypesFile	任意の拡張子のクライアントに送信する MIME タイプの種類を制御する MIME タイプのファイルの名前。このファイルは、configDirectory パラメータで指定されているとおり、構成ディレクトリに格納する必要があります。 デフォルトのファイル名は、mime.types です。

表 2-1 プリファレンス（続き）

パラメータ	説明
configDirectory	構成ディレクトリの仕様を表すオプションのパラメータ。デフォルトでは、現行ディレクトリの lib/conf サブディレクトリまたは lib¥conf サブディレクトリが、構成ディレクトリとみなされます。 このパラメータを設定する場合は、構成ファイル（プリファレンス・ファイル、MIME タイプのマッピング・ファイルなど）を、指定されたディレクトリに格納する必要があります。また、プリファレンス・ファイル内の configDirectory パラメータの値を、これらのファイルの実際の位置と一致させる必要があります。 インストール時、構成ファイルは次のディレクトリに含まれています。 UNIX の場合 : *$ORACLE_HOME*/ord/Annotator/lib/conf Windows の場合 : *ORACLE_HOME*¥ord¥Annotator¥lib¥conf
descriptorDirectory	記述子ディレクトリの仕様を表すオプションのパラメータ。デフォルトでは、現行ディレクトリの lib/descriptors サブディレクトリまたは lib¥descriptors サブディレクトリが、記述子ディレクトリとみなされます。 このパラメータを設定する場合は、記述子ファイルを、指定されたディレクトリに格納する必要があります。 インストール時、記述子ファイルは次のディレクトリに含まれています。 UNIX の場合 : *$ORACLE_HOME*/ord/Annotator/lib/descriptors Windows の場合 : *ORACLE_HOME*¥ord¥Annotator¥lib¥descriptors
connectDriver	使用している JDBC ドライバの名前。次に例を示します。 oracle.jdbc.driver.OracleDriver

表 2-1 プリファレンス（続き）

パラメータ	説明
connectJDBCProt	JDBC ドライバが、JDBC OCI ドライバまたは JDBC Thin ドライバのどちらであるかを示す JDBC ドライバの接頭辞。 ■ JDBC OCI ドライバの場合 jdbc:oracle:oci8:@ ■ JDBC Thin ドライバの場合 jdbc:oracle:thin
connectHost	メディア・データのアップロード先となる Oracle データベース・サーバーがインストールされているホストの名前。次に例を示します。 myhost
connectPort	Oracle データベース・サーバーがインストールされているホストのポート番号。
connectSID	Oracle データベースの Oracle システム識別子（SID）。
connectUserName	メディア・データをデータベースにアップロードするために使用するデータベースのユーザー名。
connectPassword	データベース・ユーザーのパスワード。
serviceName	データベースのサービス名。次のフォーマットを使用します。 *host-name*:*port_name*:*oracle-sid*
useHttpProxy	インターネット上で Hypertext Transfer Protocol（HTTP）プロトコルを介して、リモートで使用可能なメディア・ソースにアノテーションを付けるために HTTP プロキシ・サーバーを使用するかどうかを示します。保護環境で実行中の場合は、true を指定します。有効な値は、次のとおりです。 ■ true ■ false
httpProxyServer	HTTP プロキシ・サーバーの URL。プロキシ・サーバーの URL の例を、次に示します。 www-ourproxy.ourcompany.com
httpProxyPort	プロキシ・サーバーのポート番号。
mediaDirectory	ソース・メディアが含まれているディレクトリ。

表 2-1 プリファレンス（続き）

パラメータ	説明
uploadOci8BlobBlockSize	JDBC OCI ドライバの使用中に、BLOB のアップロードのために使用するブロック・サイズ。ほとんどの場合、プリファレンス・ファイルのデフォルト値を使用します。
uploadOci8ClobBlockSize	JDBC OCI ドライバの使用中に、CLOB のアップロードのために使用するブロック・サイズ。ほとんどの場合、プリファレンス・ファイルのデフォルト値を使用します。
uploadThinBlobBlockSize	JDBC Thin ドライバの使用中に、BLOB のアップロードのために使用するブロック・サイズ。ほとんどの場合、プリファレンス・ファイルのデフォルト値を使用します。
uploadThinClobBlockSize	JDBC Thin ドライバの使用中に、CLOB のアップロードのために使用するブロック・サイズ。ほとんどの場合、プリファレンス・ファイルのデフォルト値を使用します。
uploadRootAnn	副アノテーションのみではなく、副アノテーションの継承元であるすべてのアノテーションおよび副アノテーションもアップロードするかどうかを示します。有効な値は、次のとおりです。 ■ true ■ false
sqlFileName	アノテーションを挿入するために実行するスクリプト。
openSaveDirectory	アノテーションの保存先となり、保存したアノテーションをオープンする元になるデフォルトのディレクトリ。これは、demo ユーティリティ固有のパラメータです。
defaultOfm	メディア・データをデータベースにアップロードするために使用するデフォルトの PL/SQL Upload Template（ファイル拡張子 .ofm）のフルパス。これは、demo ユーティリティ固有のパラメータです。
ofmDirectory	PL/SQL Upload Template（ファイル拡張子 .ofm）の書込み先のディレクトリ。これは、demo ユーティリティ固有のパラメータです。

注意： demo ユーティリティの詳細は、D.1 項を参照してください。

プリファレンスは、Preferences.setProperty( ) メソッドを使用しても設定できます。詳細は、4-77 ページの「setProperty( )」を参照してください。

次の項では、Oracle *inter*Media Annotator の使用前に、プリファレンスの一部を使用して環境を構成する手順を説明します。

2.1.1 データベースへの接続の設定

Oracle データベースへは、JDBC Thin ドライバまたは JDBC OCI ドライバを使用して、接続できます。

JDBC Thin ドライバを使用する場合は、`Annotator.prefs` ファイルに次の情報を指定してください。

- `connectDriver` パラメータの値には、次のように入力します。

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

- `connectJDBCProt` パラメータの値には、次のように入力します。

  ```
  jdbc:oracle:thin
  ```

- `serviceName` パラメータの値には、次のフォーマットを使用して、データベースのサービス名を入力します。

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

JDBC OCI ドライバを使用する場合は、次の情報を指定してください。

- `connectDriver` パラメータの値には、次のように入力します。

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

- `connectJDBCProt` パラメータの値には、次のように入力します。

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

- `serviceName` パラメータの値には、Oracle Net で使用されている構文を使用して、データベースのサービス名を入力します。詳細は、『Oracle9*i* Net Services 管理者ガイド』を参照してください。

2.1.2 プロキシ設定の指定

Oracle *inter*Media Annotator は、インターネット上で Hypertext Transfer Protocol（HTTP）プロトコルを介してリモートで使用可能なメディア・ソースにアノテーションを付けることができます。

保護環境で実行中の場合は、インターネットにアクセスする前に、Oracle *inter*Media Annotator を構成して、プロキシ・サーバーを使用する必要があります。プロキシ・サーバーを構成するには、`Annotator.prefs` ファイルに次の情報を指定してください。

- useHttpProxy パラメータに、値 true を入力します。
- httpProxyServer パラメータに、HTTP プロキシ・サーバーのアドレスを入力します。次に例を示します。

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

- httpProxyPort パラメータに、HTTP プロキシ・サーバーのポート番号を入力します。

2.2 使用可能な URL プロトコル

Oracle *inter*Media Annotator は、表 2-2 に示す URL プロトコルを介してアクセス可能なメディア・ソースを解析できます。

表 2-2 使用可能な URL プロトコル

URL プロトコル	説明
file	コンピュータ内のローカル・ディスクまたはリモートでマウントされたディスク上のすべてのファイルにアクセスします。
http	インターネット Web サーバーを介して使用可能なメディアにアクセスします。

ローカル・ファイル、または HTTP プロトコルを介してインターネット上で使用可能なファイルを解析している場合、Oracle *inter*Media Annotator はメディア・ファイルから時間に無関係な属性を抽出し、それを論理アノテーションに挿入します。

ビデオ・ソースなどの複数のトラックを含むメディア・ソースを解析している場合は、トラックごとに副アノテーションが作成されます。

2.3 Oracle *inter*Media Annotator 使用の手順

Oracle *inter*Media Annotator を使用するには、通常、次の手順を実行します。

1. *inter*Media Annotator クライアントのインスタンスを作成し、初期化します。
2. Preferences.setProperty( ) メソッドを使用してプリファレンスを設定します（オプション）。JDBC ドライバのタイプ、HTTP プロキシ・サーバーの情報などのプリファレンスを設定できます。このメソッドの詳細は、4-77 ページの「setProperty( )」を参照してください。
3. 指定された URL からメディア・ソース・ファイルを解析して、ソースのアノテーションを作成します。Annotation.parseMedia( ) メソッドを使用して、ソース・ファイルを解析します。
4. アノテーションの定義済属性を取得し、アノテーションの追加属性を設定します（オプション）。

Oracle *inter*Media Annotator は、指定された数の属性を定義します（属性の完全なリストは、付録 C を参照）。ただし、すべてのメディア・ソースがすべての属性の値を提供するわけではありません。既存の属性を取得する場合は、Annotation.getAttribute( ) メソッドを、値を持たない任意の属性に対する値をアノテーションに追加する場合は、Annotation.setAttribute( ) メソッドを使用できます。

5. Annotation.getSubAnnotations( ) メソッドを使用して任意の副アノテーションを取得します。また、Annotation.addSubAnnotation( ) メソッドを使用して追加副アノテーションを定義します（オプション）。

   通常、アノテーションには 1 つ以上の副アノテーションが含まれます。副アノテーションには、メディア・ソースの一部（テキスト・トラックやオーディオ・トラックなど）に対応付けられたメタデータが含まれています。これらの移入済の副アノテーションに加えて、空のアノテーションを追加し、それに独自の値を移入することによって、独自の副アノテーションを定義することができます。

6. パーサー（QuickTime パーサーなど）の場合は、AnnotationHandler.extractMedia( ) メソッドを使用して、メディア・ソース・ファイルからメディア・サンプルを抽出できます。

7. AnnotationHandler.insertMedia( ) メソッドを使用して、アノテーションを Oracle データベースにアップロードします。

注意： Oracle *inter*Media Annotator は、メディア自体の属性値は変更できません。変更できるのは、抽出されたアノテーションの属性値のみです。再度メディア・ファイルを解析すると、アノテーションが上書きされ、編集したすべての属性が元の値に戻ります。

第 3 章では、メディア・ソース・ファイルを解析し、アノテーションを作成し、アノテーションを Oracle データベースへアップロードするサンプル・プログラムについて説明します。

3

*inter*Media Annotator エンジン API の例

この章では、メディア・ソース・ファイルを解析し、アノテーションを作成し、アノテーションを Oracle データベースへアップロードするサンプル・プログラムについて説明します。SimpleAnnotator.java というこのプログラムは、Oracle *inter*Media Annotator エンジン API を使用して作成されたユーザー開発アプリケーションの一例です。

ソース・コードは次の場所にあります。

- UNIX の場合 :

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

- Windows の場合 :

  ORACLE_HOME¥ord¥Annotator¥demo¥examples¥src¥SimpleAnnotator.java

SimpleAnnotatorAsynch.java という名前のサンプル・プログラムの非同期バージョンも同じディレクトリ内にあります。

この章に記載するコードは、SimpleAnnotator.java のコードとは異なる場合があります。システム上でこのサンプル・プログラムを実行する場合は、インストール時に提供されるファイルを使用してください。この章に記載するコードをコンパイルして実行しないでください。

> **注意 :** この章には、Java コードの例が含まれます。コード例に、カッコで囲まれた数字が含まれている場合があります。これは、例のすぐ後にある手順にそのコードの詳細が記載されていることを示します。

このサンプル・プログラムには、Java および Oracle *inter*Media Annotator API を使用して次の操作を行うユーザー定義のメソッドが含まれます。

- *inter*Media Annotator クライアントのインスタンスの初期化
- プリファレンスの設定

- メディア・ソース・ファイルの解析
- アノテーションの属性の取得および設定
- 副アノテーションの定義
- メディア・ソース・ファイルからのメディア・サンプルの抽出
- Oracle データベースへのアノテーションおよびメディア・ソースのアップロード

注意： Oracle *inter*Media Annotator は、メディア自体の属性値は変更できません。変更できるのは、抽出されたアノテーションの属性値のみです。再度メディア・ファイルを解析すると、アノテーションが上書きされ、編集したすべての属性が元の値に戻ります。

3.1 文のインポート

例 3-1 に、*inter*Media Annotator クライアントを適切に実行するためにアプリケーションに必要なインポート文を示します。

例 3-1 インポート文

```
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 クラス定義およびインスタンス変数

例 3-2 に、*inter*Media Annotator クライアントのサンプルに関するクラス定義およびインスタンス変数を示します。

例 3-2 クラス定義およびインスタンス変数

```
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;
```

*inter*Media Annotator クライアントは、*inter*Media Annotator エンジンの操作に使用するコールバック・メソッドにアクセスするために、AnnListener インタフェースを実装する必要があります。詳細は、4-61 ページの「oracle.ord.media.annotator.listener.AnnListener クラス」を参照してください。

*inter*Media Annotator クライアントは、OutputListener を実装して、エンジンからのトレース情報にアクセスする必要があります。詳細は、4-67 ページの「oracle.ord.media.annotator.listener.OutputListener クラス」を参照してください。

クラスには、次の 2 つのインスタンス変数が含まれます。

- m_st は、アプリケーションのステータスを更新するために使用する Status オブジェクトです。詳細は、4-78 ページの「oracle.ord.media.annotator.utils.Status クラス」を参照してください。
- m_ah は、指定されたコンテンツのソースに対して実際にアノテーションを生成する AnnotationHandler オブジェクトです。詳細は、4-27 ページの「oracle.ord.media.annotator.handlers.AnnotationHandler クラス」を参照してください。

2 つの追加変数が宣言されています。m_output_file 変数は、出力ファイルの名前を保持します。m_output_file_defined 変数は、出力ファイルが引数として渡されたかどうかを示すブーリアンです。

3.3 main( ) メソッド

サンプル・プログラム内の main( ) メソッドは、アプリケーションに渡された URL を確認し、SimpleAnnotator の空のインスタンスを作成し、init( ) メソッドをコールします。インスタンスの初期化後、メディアを解析するメソッドをコールします。例 3-3 に、main( ) メソッドの内容を示します。

例 3-3 main( ) メソッド（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);
         }
}
```

main( ) メソッドのコードは、次の手順を実行します。

1. 操作の対象となるメディア・ソース・ファイルの URL を、1 つ目の引数として指定します。オプションの引数は、次のとおりです。
   - XML アノテーション用の出力ファイルを表す文字列
   - メディアのキャラクタ・エンコーディングを表す文字列
2. 解析されるメディア・ソース・ファイルの URL が引数として渡されたことを確認するテストを行います。引数がない場合、エラー・メッセージが出力されます。
3. 次の変数を定義します。
   - `szURL` という名前の文字列。この値は、1 つ目のパラメータの値（メディア・ソース・ファイルの URL）に設定されます。
   - メディアのキャラクタ・エンコーディングを表す `szEncoding` という文字列。この値は、`null` に設定されます。
   - キャラクタ・エンコーディング用の引数がメソッドに渡されたかどうかを定義する `encoding_defined` という名前のブーリアンです。

   また、`m_output_file_defined` 変数の値を `false` に設定します。

4. メソッドに渡された引数の数を確認し、値を変数に割り当てます。
5. SimpleAnnotator の空のインスタンスを作成します。
6. init( ) メソッドをコールして、新しく作成した SimpleAnnotator インスタンスを初期化します。init( ) メソッドおよびそのコードのリストは、3.4 項を参照してください。

   SimpleAnnotator インスタンスの初期化後、クライアントは、解析、抽出、挿入などの *inter*Media Annotator エンジンの操作を開始できます。
7. parse( ) メソッドをコールして、メディア・ソース・ファイルを解析します。このメソッドは、URL が含まれている文字列およびエンコーディング・メソッド（提供されている場合）を渡します。

   parse( ) メソッドおよびそのコードのリストは、3.5 項を参照してください。

3.4 init( ) メソッド

サンプル・プログラム内の init( ) メソッドは、*inter*Media Annotator クライアントを初期化し、プリファレンスを設定します。例 3-4 に、このメソッドの内容を示します。

例 3-4 init( ) メソッド

```
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");
}
```

init( ) メソッドのコードは、次の手順を実行します。

1. 初期化が開始したことを示すメッセージを出力します。report(String) メソッドの詳細は、3.12 項を参照してください。

2. SimpleAnnotator の現行のインスタンスがステータス・メッセージを受信できるように、Status オブジェクトを初期化し、OutputListener インタフェースを実装します。詳細は、4-81 ページの「initStatus( )」を参照してください。
3. 初期化した Status オブジェクトを m_st インスタンス変数に設定します。
4. Status.SetOutputMode( ) メソッドを使用して、ステータス出力モードを VERBOSE に設定します。詳細は、4-85 ページの「SetOutputMode( )」を参照してください。
5. 同期モードで新しい AnnotationHandler インスタンスを作成し、m_ah インスタンス変数に設定します。
6. 前の手順で発生したすべての例外をキャッチし、report( ) メソッドおよび reportError( ) メソッドを使用して画面にエラーを出力します。reportError( ) メソッドの詳細は、3.15 項を参照してください。

   Status オブジェクトおよび AnnotationHandler オブジェクトの両方が正常に作成された場合、必要なプリファレンスを設定できます。
7. Preferences オブジェクトを作成し、初期化します。
8. Preferences.setProperty( ) メソッドを使用して、SZ_CONN_PASSWORD という名前の新しいプリファレンスを設定します。詳細は、4-77 ページの「setProperty( )」を参照してください。
9. 初期化が正常終了したことを示すメッセージを出力します。

3.5 parse( ) メソッド

サンプル・プログラム内の parse(String) メソッドは、AnnotationHandler.parseMedia( ) メソッドをコールし、URL を渡して、ソース・ファイルの解析およびアノテーションの作成を行います。例 3-5 に、parse(String) メソッドの内容を示します。

例 3-5 parse(String) メソッド

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

parse(String) メソッドのコードは、AnnotationHandler.parseMedia( ) メソッドをコールし、URL を渡します。クライアント（this）は、AnnListener インタフェースを実装します。

サンプル・プログラム内の parse(String, String) メソッドは、AnnotationHandler.parseMedia( ) メソッドをコールし、URL およびキャラクタ・エンコーディングを渡し、ソース・ファイルの解析およびアノテーションの作成を行います。例 3-6 に、parse(String, String) メソッドの内容を示します。

例 3-6 Parse(String, String) メソッド

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

parse(String, String) メソッドのコードは、AnnotationHandler.parseMedia( ) メソッドをコールし、URL およびキャラクタ・エンコーディングを渡します。クライアント（this）は、AnnListener インタフェースを実装します。

AnnotationHandler.parseMedia( ) メソッドの詳細は、4-42 ページの「parseMedia(String, AnnListener)」および 4-43 ページ の「parseMedia(String, AnnListener, String)」を参照してください。

init( ) メソッドでアノテーションハンドラが適切に初期化された場合にのみ、AnnotationHandler.parseMedia( ) がコールされます。

AnnotationHandler.parseMedia( ) メソッドは、ソース・ファイルを解析し、アノテーションを作成します。その後、コールバック関数 AnnListener.parsePerformed( ) をコールします。このメソッドは SimpleAnnotator クラスで上書きされるため、実際にコールされるメソッドは SimpleAnnotator.parsePerformed( ) になります。詳細は、3.6 項を参照してください。

3.6 parsePerformed( ) メソッド

このサンプル・プログラム内の parsePerformed( ) メソッドは、アノテーションを処理してからデータベースにアップロードします。アプリケーションは AnnListener を実装する必要があるため、このメソッドは必須です。例 3-7 に、parsePerformed( ) メソッドの内容を示します。

例 3-7 parsePerformed( ) メソッド

```
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);
    }
}
```

parsePerformed( ) メソッドのコードは、次の手順を実行します。

1. コール元から有効なアノテーションが渡された場合にのみ、次のブロックのコードを実行します。

   コール元から有効なアノテーションが渡された場合、通常、アノテーションを処理してからデータベースにアップロードします。手順 2 ～ 11 に、実行できる操作の例を示します。手順 7 ～ 11 のタスクは、上級プログラマ向けです。

2. Annotation.getAttribute( ) メソッドを使用して、アノテーションの MEDIA_SOURCE_MIME_TYPE 属性の値を取得し、String オブジェクトにキャストします。詳細は、4-5 ページの「getAttribute( )」を参照してください。

3. Annotation.getAttributes( ) メソッドを使用して、有効な値を持つすべての属性のリストを取得し、その属性の名前を Enumeration オブジェクトに格納します。詳細は、4-6 ページの「getAttributes( )」を参照してください。

4. Enumeration オブジェクトに格納された値にアクセスします。

5. Annotation.getSubAnnotations( ) メソッドを使用して、アノテーションのすべての副アノテーションを取得し、Enumeration オブジェクトに格納します。詳細は、4-12 ページの「getSubAnnotations( )」を参照してください。

6. Enumeration オブジェクトに格納された値にアクセスします。

7. inventoryAnn という名前の空のアノテーションを作成します。

8. Annotation.addSubAnnotation( ) メソッドを使用して、アノテーションに inventoryAnn を副アノテーションとして追加します。詳細は、4-4 ページの「addSubAnnotation( )」を参照してください。

9. Annotation.setAttribute( ) を使用して、inventoryAnn の SALES_PRICE 属性を 19.99 に設定します。詳細は、4-18 ページの「setAttribute( )」を参照してください。

10. 手順 7 ～ 9 で発生したすべてのエラーをキャッチし、errorOccured( ) メソッド（3.10 項を参照）を使用してレポートします。
11. report(Annotation) メソッド（3.13 項を参照）を使用して、アノテーションを XML ファイルとして出力します。
12. アノテーションまたはその副アノテーションからサンプルを抽出できるかどうかを確認します。サンプルを抽出できる場合、手順 13 のコードを実行します。サンプルを抽出できない場合は、この手順は省略されます。
13. AnnotationHandler.extractMedia( ) メソッドを使用して、アノテーションからメディア・サンプルを抽出します。詳細は、4-32 ページの「extractMedia( )」を参照してください。

    AnnotationHandler.extractMedia( ) は、終了時にコールバック関数 AnnListener.extractionPerformed( ) をコールします。このメソッドは SimpleAnnotator クラスで上書きされるため、実際にコールされるメソッドは SimpleAnnotator.extractionPerformed( ) になります。このメソッドの詳細は、3.7 項を参照してください。

3.7 extractionPerformed( ) メソッド

サンプル・プログラム内の extractionPerformed( ) メソッドは、アノテーションをファイルにマップし、そのアノテーションをデータベースにアップロードします。アプリケーションは AnnListener を実装する必要があるため、このメソッドは必須です。例 3-8 に、extractionPerformed( ) メソッドの内容を示します。

例 3-8 extractionPerformed( ) メソッド

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

extractionPerformed( ) メソッドのコードは、次の手順を実行します。

1. report(Annotation) メソッド（3.13 項を参照）を使用して、アノテーションを XML ファイルとして出力します。
2. マッピング・ファイルに基づいて、新しい OrdFileMapping オブジェクトを作成します。この例では、マッピング・ファイルは `e:¥¥mylogic.ofm` に格納されています。詳細は、4-47 ページの「OrdFileMapping コンストラクタ」を参照してください。

3. データベースにアノテーションをアップロードし、OrdFileMapping オブジェクトを使用して、データベースの適切な位置にアノテーションの内容をマップし、AnnotationHandler.insertMedia( ) メソッドを使用してメディアを挿入します。このメソッドの詳細は、4-37 ページの「insertMedia(Annotation, OrdMapping, AnnListener)」を参照してください。

   また、アップロード・プロセスで使用される JDBC 接続を表す Connection オブジェクトを指定することもできます。複数のアップロード操作に同一の JDBC 接続を使用できます。詳細は、4-38 ページの「insertMedia(Annotation, OrdMapping, AnnListener, Connection)」を参照してください。

   AnnotationHandler.insertMedia( ) は、終了時にコールバック関数 AnnListener.insertionPerformed( ) をコールします。このメソッドは SimpleAnnotator クラスで上書きされるため、実際にコールされるメソッドは SimpleAnnotator.insertionPerformed( )（3.8 項を参照）になります。

3.8 insertionPerformed( ) メソッド

サンプル・プログラム内の insertionPerformed( ) メソッドは、データベースへの接続をクローズし、すべての例外をキャッチします。アプリケーションは AnnListener を実装する必要があるため、このメソッドは必須です。例 3-9 に、insertionPerformed( ) メソッドの内容を示します。

例 3-9 insertionPerformed( ) メソッド

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

insertionPerformed( ) メソッドのコードは、次の手順を実行します。

1. データベースに対するすべての変更をコミットします。
2. データベースへの接続をクローズします。

   クライアントは、接続をクローズすることなく他の AnnotationHandler コールに渡すことによって、その接続を再利用することもできます。

3. 手順 1 および 2 で発生したすべてのエラーをキャッチし、errorOccured( ) メソッドを使用してレポートします。errorOccured( ) メソッドの詳細は、3.10 項を参照してください。

3.9 warningOccured( ) メソッド

サンプル・プログラム内の warningOccured( ) メソッドは、すべての警告を取得します。アプリケーションは AnnListener を実装する必要があるため、このメソッドは必須です。例 3-10 に、warningOccured( ) メソッドの内容を示します。

例 3-10 warningOccured( ) メソッド

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

warningOccured( ) メソッドのコードは、AnnListener.warningOccured( ) メソッドを実装します。このメソッドは、reportError( ) メソッド（3.15 項を参照）を使用して警告を取得し、レポートします。警告が発生してこのメソッドがコールされた場合、*inter*Media Annotator エンジンは操作を継続します。詳細は、4-66 ページの「warningOccured( )」を参照してください。

3.10 errorOccured( ) メソッド

サンプル・プログラム内の errorOccured( ) メソッドは、すべてのエラーを取得します。アプリケーションは AnnListener を実装する必要があるため、このメソッドは必須です。例 3-11 に、errorOccured( ) メソッドの内容を示します。

例 3-11 errorOccured( ) メソッド

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

errorOccured( ) メソッドのコードは、AnnListener.errorOccured( ) メソッドを実装します。このメソッドは、reportError( ) メソッド（3.15 項を参照）を使用してエラーを取得し、レポートします。エラーが発生してこのメソッドがコールされた場合、*inter*Media Annotator エンジンは操作を継続しません。詳細は、4-62 ページの「errorOccured( )」を参照してください。

3.11 ConsoleOutput( ) メソッド

サンプル・プログラム内の ConsoleOutput( ) メソッドは、実行中にメッセージを出力します。アプリケーションは OutputListener を実装する必要があるため、このメソッドは必須です。例 3-12 に、ConsoleOutput( ) メソッドの内容を示します。

例 3-12 ConsoleOutput( ) メソッド

```
public void ConsoleOutput(String sz){
```

```
    report(sz);
}
```

この例のコードは、OutputListener.ConsoleOutput( ) メソッドを実装します。このメソッドは、report(String) メソッド（3.12 項を参照）を使用して実行中にメッセージを出力します。詳細は、4-68 ページの「ConsoleOutput( )」を参照してください。

3.12 report(String) メソッド

サンプル・プログラム内の report(String) メソッドは、文字列をエラー・ストリームに出力します。例 3-13 に、このメソッドの内容を示します。

例 3-13 report(String) メソッド

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

report(String) メソッドのコードは、指定されたストリームをエラー・ストリームに出力します。

3.13 report(Annotation) メソッド

サンプル・プログラム内の report(Annotation) メソッドは、アノテーションを XML ファイルに出力します。例 3-14 に、このメソッドの内容を示します。

例 3-14 report(Annotation) メソッド

```
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);   }
{
```

report(Annotation) メソッドのコードは、次の手順を実行します。

1. 一時出力ファイルが存在するかどうかを確認し、存在しない場合はそのファイルを作成します。
2. 出力ストリームを一時ファイルに書き込みます。
3. AnnotationHandler.exportToXML( ) メソッドを使用して、指定されたアノテーションの XML 表現を作成します。詳細は、4-31 ページの「exportToXML( )」を参照してください。

3.14 reportWarning( ) メソッド

サンプル・プログラム内の reportWarning( ) メソッドは、警告をエラー・ストリームに出力します。例 3-15 に、このメソッドの内容を示します。

例 3-15 reportWarning( ) メソッド

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

このメソッドは、reportError( ) メソッドを使用して指定されたエラーをレポートします。

3.15 reportError( ) メソッド

サンプル・プログラム内の reportError( ) メソッドは、例外をエラー・ストリームに出力します。例 3-16 に、このメソッドの内容を示します。

例 3-16 reportError( ) メソッド

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

reportError( ) メソッドのコードは、例外の内容を取得し、String オブジェクトにキャストし、report(String) メソッドを使用して例外をエラー・ストリームに出力します。

4

*inter*Media Annotator エンジン API のリファレンス情報

この章には、初心者のユーザーが Oracle *inter*Media Annotator エンジンを使用する Java アプリケーションを作成するために必要な、クラス、コンストラクタおよびメソッドに関する参照データが含まれます。詳細は、*inter*Media Annotator のインストール時に提供される Javadoc を参照してください。

oracle.ord.media.annotator.annotations.Annotation クラス

この項では、Annotation クラスのコンストラクタおよびメソッドに関するリファレンス情報を示します。このクラスはすべてのアノテーションのスーパークラスです。論理アノテーション、修飾子およびアクセッサ・メソッドを保持するために必要なデータ構造を提供します。

このクラスで定義される属性の詳細は、付録 C を参照してください。

このクラスは java.lang.Object を拡張します。

Annotation コンストラクタ

構文

```
public Annotation( )
```

説明

Annotation オブジェクトを作成します。

パラメータ

なし

戻り値

なし

例外

なし

例

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

addSubAnnotation( )

構文

```
public void addSubAnnotation(Annotation annChild)
```

説明

指定されたアノテーションを現行のアノテーションの副アノテーションとして追加します。

パラメータ

annChild
副アノテーションとして追加するアノテーション

戻り値

なし

例外

なし

例

このメソッドの例は、3.6 項を参照してください。

getAttribute( )

構文

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

説明

指定された属性の値をオブジェクトとして取得します。クライアントは、戻り値にアクセスするために、オブジェクトを適切にキャストする必要があります。

パラメータ

szAttrCode
取得される属性の属性コード（文字列）。*inter*Media Annotator で定義される属性の詳細は、付録 C を参照してください。

戻り値

指定された属性の値をオブジェクトとして戻します。指定された属性に値がない場合、NULL を戻します。

例外

なし

例

このメソッドの例は、3.6 項を参照してください。

getAttributes( )

構文

```
public java.util.Enumeration getAttributes( )
```

説明

値が設定された属性コードのリストを戻します。このリストには、値が NULL である属性は含まれません。*inter*Media Annotator で定義される属性の詳細は、付録 C を参照してください。

パラメータ

なし

戻り値

値が設定された属性コードのリストを含む Enumeration オブジェクトです。各コードは整数で戻されます。

例外

なし

例

このメソッドの例は、3.6 項を参照してください。

getDescriptor( )

構文

```
public AnnotationDesc getDescriptor( )
```

説明

XML エクスポータに必要な AnnotationDesc オブジェクトを戻します。

AnnotationDesc オブジェクトの詳細は、7-2 ページの「oracle.ord.media.annotator.descriptors.AnnotationDesc クラス」を参照してください。

パラメータ

なし

戻り値

XML エクスポータに必要な AnnotationDesc オブジェクトを戻します。

例外

なし

例

なし。上級ユーザーのみが、このメソッドを直接コールしてください。

getName( )

構文

```
public java.lang.String getName( )
```

説明

現行のアノテーションの名前を戻します。

パラメータ

なし

戻り値

現行のアノテーションの名前を戻します。

例外

なし

例

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

getNumSubAnnotations( )

構文

```
public int getNumSubAnnotations( )
```

説明

現行のアノテーションの副アノテーションの数を戻します。

パラメータ

なし

戻り値

副アノテーションの数を整数で戻します。

例外

なし

例

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

getParent( )

構文

```
public Annotation getParent( )
```

説明

アノテーションの親オブジェクトを戻します。

パラメータ

なし

戻り値

現行のアノテーションの親オブジェクトを戻します。

例外

なし

例

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

getSampleAnns( )

構文

```
public java.util.Enumeration getSampleAnns( )
```

説明

現行のアノテーションの副アノテーションのリストを取得します。

パラメータ

なし

戻り値

現行のアノテーションの副アノテーションのリストを含む Enumeration オブジェクトを戻します。

例外

なし

例

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

getSubAnnotations( )

構文

```
public java.util.Enumeration getSubAnnotations( )
```

説明

副アノテーションのベクトルの Enumeration オブジェクトを取得します。

パラメータ

なし

戻り値

副アノテーションのベクトルの Enumeration オブジェクトを戻します。

例外

なし

例

このメソッドの例は、3.6 項を参照してください。

getURL( )

構文

```
public java.net.URL getURL( )
```

説明

アノテーションの URL を戻します。

パラメータ

なし

戻り値

アノテーションの URL を戻します。

例外

なし

例

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

isDescendantOf( )

構文

```
public boolean isDescendantOf(java.lang.String szAncestor)
```

説明

現行のアノテーションが指定されたアノテーションの副アノテーションかどうかを確認します。

パラメータ

szAncestor
現行のアノテーションが副アノテーションである可能性があるアノテーション

戻り値

現行のアノテーションが指定されたアノテーションの副アノテーションである場合は true、そうでない場合は false を戻します。

例外

なし

例

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

removeAttribute( )

構文

```
public void removeAttribute(java.lang.Object key)
```

説明

現行のアノテーションから属性およびその値を削除します。

パラメータ

key
削除する属性。*inter*Media Annotator で定義される属性の詳細は、付録 C を参照してください。

戻り値

なし

例外

なし

例

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

removeSampleAnns( )

構文

```
public void removeSampleAnns( )
```

説明

現行のアノテーションのすべての副アノテーションを削除します。

パラメータ

なし

戻り値

なし

例外

なし

例

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

removeSubAnnotation( )

構文

```
public boolean removeSubAnnotation(Annotation ann)
```

説明

現行のアノテーションから指定された副アノテーションを削除します。

パラメータ

ann
削除する副アノテーション

戻り値

副アノテーションが正常に削除された場合は true、そうでない場合は false を戻します。

例外

なし

例

このメソッドの例は、4-14 ページの「isDescendantOf( )」を参照してください。

setAttribute( )

構文

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

説明

現行のアノテーションに新しい属性を挿入します。

パラメータ

szAttrCode
値を変更する属性の属性コード（文字列）。*inter*Media Annotator で定義される属性の詳細は、付録 C を参照してください。

oValue
属性の新しい値（オブジェクト）

戻り値

なし

例外

なし

例

このメソッドの例は、3.6 項を参照してください。

oracle.ord.media.annotator.handlers.AnnTaskMonitor クラス

この項では、アノテーション・タスク・モニターを作成する、AnnTaskMonitor クラスのコンストラクタおよびメソッドに関するリファレンス情報を示します。アノテーション・タスク・モニター・オブジェクトは、AnnotationHandler オブジェクト（アノテーション・ハンドラ）で実行中のタスクを監視するために使用されるコンポーネントの 1 つです。アノテーション・ハンドラでタスクが起動されるたびに、アノテーション・タスク・マネージャおよびアノテーション・タスク・モニターが作成されます。アノテーション・タスク・マネージャはサーバー側で実行し、データベース・サーバー上でのタスクの処理過程を追跡します。アノテーション・タスク・モニターはクライアント側で実行し、タスク処理過程モニターを介して、戻されたアノテーション・タスク・モニターのインスタンスから処理過程の値およびメッセージを追跡します。

アノテーション・タスク・マネージャの詳細は、7-10 ページの「oracle.ord.media.annotator.handlers.AnnTaskManager クラス」を参照してください。

このクラスは java.lang.Object を拡張します。

AnnTaskMonitor コンストラクタ

構文

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

説明

AnnTaskMonitor オブジェクトを作成します。

パラメータ

annTaskMgr
サーバー側の現行のタスクを追跡するアノテーション・タスク・マネージャ

戻り値

なし

例外

なし

例

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

getMessage( )

構文

```
public java.lang.String getMessage( )
```

説明

タスク処理過程モニターから現在のメッセージを取得します。

パラメータ

なし

戻り値

タスク処理過程モニターの現在のメッセージを文字列として戻します。

例外

なし

例

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

getTaskCurrent( )

構文

```
public int getTaskCurrent( )
```

説明

タスク処理過程モニターの現在の値を取得します。

パラメータ

なし

戻り値

タスク処理過程モニターの現在の値を戻します。通常、タスクの開始値から終了値の範囲内の数値を戻します。詳細は、4-24 ページの「getTaskStart( )」および4-23 ページの「getTaskEnd( )」を参照してください。現在の値は、AnnTaskManager.setTaskCurrent( ) メソッドを使用して設定できます。詳細は、7-27 ページの「setTaskCurrent(int)」を参照してください。

例外

なし

例

このメソッドの例は、4-25 ページの「isDone( )」を参照してください。

getTaskEnd( )

構文

```
public int getTaskEnd( )
```

説明

タスク処理過程モニターの終了値を取得します。

パラメータ

なし

戻り値

タスク処理過程モニターの終了値を戻します。通常、タスクの終了値は 100 に設定されます。この値は、AnnTaskManager.setTask( ) メソッドを使用して設定できます。詳細は、7-26 ページの「setTask( )」を参照してください。

例外

なし

例

このメソッドの例は、4-26 ページの「isInitialized( )」メソッドを参照してください。

getTaskStart( )

構文

```
public int getTaskStart( )
```

説明

タスク処理過程モニターの開始値を取得します。

パラメータ

なし

戻り値

タスク処理過程モニターの開始値を戻します。通常、タスクの開始値は 0（ゼロ）に設定されます。この値は、AnnTaskManager.setTask( ) メソッドを使用して設定できます。詳細は、7-26 ページの「setTask( )」を参照してください。

例外

なし

例

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

isDone( )

構文

```
public boolean isDone( )
```

説明

タスクが完了しているかどうかを判断します。

パラメータ

なし

戻り値

タスクが完了している場合は true、そうでない場合は false を戻します。

例外

なし

例

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

isInitialized( )

構文

```
public boolean isInitialized( )
```

説明

アノテーション・タスク・モニターが初期化されているかどうかを確認します。初期化されている場合、getStartTask( ) メソッドおよび getEndTask( ) メソッドをコールして、タスクの開始時と終了時を検出できます。

パラメータ

なし

戻り値

アノテーション・タスク・モニターが初期化されている場合は true、そうでない場合は false を戻します。

例外

なし

例

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

oracle.ord.media.annotator.handlers.AnnotationHandler クラス

この項では、アノテーション・ハンドラを作成する、AnnotationHandler クラスのコンストラクタおよびメソッドに関するリファレンス情報を示します。このクラスは、指定されたコンテンツ・ソースにアノテーションを生成するメソッドを提供します。AnnotationHandler をコールするアプリケーションは、ハンドラへの様々な応答をリスニングするために、AnnListener インタフェースを実装する必要があります。詳細は、4-61 ページの「oracle.ord.media.annotator.listener.AnnListener クラス」を参照してください。

アプリケーションで作成および使用する必要がある AnnotationHandler インスタンスは 1 つのみです。AnnotationHandler はステートレスかつスレッド・セーフであるため、複数のスレッドで同一の AnnotationHandler インスタンスをコールすることができます。

このクラスは java.lang.Object を拡張します。

次のフィールドを含みます。

- public static final int OP_MODE_ASYNCH

  非同期モードを示します。

- public static final int OP_MODE_SYNCH

  同期モードを示します。

この項に示す例は、AnnotationHandler オブジェクトの名前でハンドラが作成されていることを前提としています。AnnotationHandler オブジェクトの作成例は、3.4 項を参照してください。また、次のディレクトリに格納されているオンライン例 `SimpleAnnotator.java` および `SimpleAnnotatorAsynch.java` も参照してください。

- UNIX の場合 :

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

- Windows の場合 :

  ORACLE_HOME`¥ord¥Annotator¥demo¥examples¥src`

AnnotationHandler コンストラクタ

構文

```
public AnnotationHandler( )
```

説明

AnnotationHandler オブジェクトを作成します。デフォルトでは、コンストラクタは操作に非同期モードを使用します。

すべてのエンジン・トレースが処理されるように、コール元はアノテーション・ハンドラを作成する前に Status インスタンスを作成する必要があります。Status の詳細は、4-78 ページの「oracle.ord.media.annotator.utils.Status クラス」を参照してください。

パラメータ

なし

戻り値

なし

例外

oracle.ord.media.annotator.handlers.AnnotationHandlerException

例

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

AnnotationHandler(int) コンストラクタ

構文

```
public AnnotationHandler(int iOperationMode)
```

説明

AnnotationHandler オブジェクトを作成します。デフォルトでは、コンストラクタは操作に非同期モードを使用します。

AnnotationHandler クラスは、OP_MODE_ASYNCH および OP_MODE_SYNCH という名前の 2 つの静的整数を含みます。非同期モードで実行するアノテーション・ハンドラを作成するには、iOperationMode を OP_MODE_ASYNCH に設定します。同期モードで実行するアノテーション・ハンドラを作成するには、iOperationMode を OP_MODE_SYNCH に設定します。

すべてのエンジン・ステータス・メッセージが処理されるように、コール元はアノテーション・ハンドラを作成する前に Status インスタンスを作成する必要があります。Status の詳細は、4-78 ページの「oracle.ord.media.annotator.utils.Status クラス」を参照してください。

パラメータ

iOperationMode
アノテーション・ハンドラが使用するモード（同期または非同期）

戻り値

なし

例外

oracle.ord.media.annotator.handlers.AnnotationHandlerException

例

このメソッドの例は、3.4 項を参照してください。

createAnnotationByName( )

構文

```
public Annotation createAnnotationByName(java.lang.String szAnnName)
```

説明

アノテーション型を指定して、アノテーションの新しいインスタンスを作成します。

パラメータ

szAnnName
作成するアノテーションのアノテーション型

戻り値

新しく作成されたアノテーションを戻します。

例外

AnnotatorException

例

このメソッドの例は、3.6 項を参照してください。

exportToXML( )

構文

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

説明

アノテーションおよびその副アノテーションの XML 表現を作成し、XML ファイルにエクスポートします。

パラメータ

w
内容の XML 表現を作成する Writer オブジェクト

ann
エクスポートするアノテーション

戻り値

なし

例外

なし

例

このメソッドの例は、3.13 項を参照してください。

extractMedia( )

構文

```
public AnnTaskMonitor extractMedia(Annotation ann,
                                   AnnListener annListener)
```

説明

アノテーションからメディア・サンプルを抽出します。抽出が完了すると、コールバック関数 AnnListener.extractionPerformed( ) がコールされます。

パラメータ

ann
サンプルを抽出するアノテーション

annListener
抽出の完了時に通知されるリスナー

戻り値

このタスクに対応付けられた AnnTaskMonitor オブジェクトを戻します。

例外

なし

例

このメソッドの例は、3.6 項を参照してください。

getAnnotationNames( )

構文

```
public java.util.Enumeration getAnnotationNames( )
```

説明

アノテーション記述子の XML ファイルに定義されたアノテーション型の名前を持つ String オブジェクトのリストを戻します。

パラメータ

なし

戻り値

アノテーション記述子の XML ファイルに定義されたアノテーション型の名前を持つ String オブジェクトのリストを戻します。

例外

AnnotatorException

例

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

getParserNames( )

構文

```
public java.util.Enumeration getParserNames( )
```

説明

パーサー記述子の XML ファイルに定義されたパーサー型のリストを戻します。

パラメータ

なし

戻り値

パーサー記述子の XML ファイルに定義されたパーサー型のリストを戻します。

例外

AnnotatorException

例

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

getRelVersion( )

構文

```
public final java.lang.String getRelVersion( )
```

説明

*inter*Media Annotator リリースのバージョンを戻します。

パラメータ

なし

戻り値

*inter*Media Annotator リリースのバージョンを戻します。

例外

なし

例

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

importFromXML( )

構文

```
public Annotation importFromXML(java.io.Reader r)
```

説明

XML ファイルから読み込まれたコンテンツを持つ新しい Annotation オブジェクトを作成します。

パラメータ

r
XML ファイルからコンテンツを読み込む Reader オブジェクト

戻り値

新しい Annotation オブジェクトを戻します。

例外

oracle.ord.media.annotator.annotations.AnnotationException

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

例

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

insertMedia(Annotation, OrdMapping, AnnListener)

構文

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

説明

データベースへの新しい接続を作成し、データベース・サーバー上の Oracle *inter*Media オブジェクトにアノテーションを挿入します。

パラメータ

ann
挿入されるアノテーション

om
アノテーションとデータベース・サーバー上の Oracle *inter*Media オブジェクト間のマッピング

annListener
操作の完了時に通知されるリスナー

戻り値

このタスクに対応付けられた AnnTaskMonitor オブジェクトを戻します。

例外

なし

例

このメソッドの例は、3.7 項を参照してください。

insertMedia(Annotation, OrdMapping, AnnListener, Connection)

構文

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

説明

データベースへの新しい接続を作成し、Oracle *inter*Media オブジェクトにアノテーションを挿入します。解析が完了すると、コールバック・メソッド AnnListener.insertionPerformed( ) をコールします。

パラメータ

ann
挿入されるアノテーション

om
アノテーションとデータベース・サーバー上の Oracle *inter*Media オブジェクト間のマッピング。OrdMapping オブジェクトの詳細は、*inter*Media Annotator の Javadoc を参照してください。

annListener
操作の完了時に通知されるリスナー

conn
データベースへの接続。このパラメータを NULL に設定すると、新しい接続が作成されます。

戻り値

このタスクに対応付けられた AnnTaskMonitor オブジェクトを戻します。

例外

なし

例

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

isExtractable( )

構文

```
public boolean isExtractable (Annotation ann)
```

説明

指定されたアノテーションまたはその副アノテーションからサンプルを抽出できるかどうかを判断します。

パラメータ

ann
サンプルを抽出するアノテーション

戻り値

サンプルを抽出できる場合は true、そうでない場合は false を戻します。

例外

なし

例

このメソッドの例は、3.6 項を参照してください。

isPlayable( )

構文

```
public boolean isPlayable(Annotation ann)
```

説明

指定されたアノテーションで表現されるメディア・コンテンツを再生できるかどうかを判断します。

パラメータ

ann
コンテンツを再生するアノテーション

戻り値

メディア・コンテンツを再生できる場合は true、そうでない場合は false を戻します。

例外

なし

例

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

parseMedia(InputStream, String, AnnListener)

構文

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

説明

指定された InputStream に対応付けられたソースを解析し、指定された URL のアノテーションを作成します。解析が完了すると、次の操作を実行します。

- アノテーションへの MEDIA_SOURCE_MIME_TYPE 属性の設定を試みます。parseMedia(String, AnnListener) および parseMedia(String, AnnListener, String) とは異なり、このメソッドは 1 つの属性のみを設定します。
- コールバック関数 AnnListener.parsePerformed( ) をコールします。

パラメータ

is
解析されるメディア・ファイルの InputStream

sURL
解析されるメディア・ファイルの URL

annListener
解析の完了時に通知されるリスナー

戻り値

このタスクに対応付けられた AnnTaskMonitor オブジェクトを戻します。

例外

なし

例

```
//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)

構文

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

説明

ソースを解析し、指定された URL のアノテーションを作成します。解析が完了すると、次の操作を実行します。

- アノテーションへの次の属性の設定を試みます。
  - MEDIA_SIZE
  - MEDIA_SOURCE_DIRECTORY
  - MEDIA_SOURCE_FILENAME
  - MEDIA_SOURCE_PROTOCOL
  - MEDIA_SOURCE_URL
  - MEDIA_SOURCE_MIME_TYPE
- コールバック関数 AnnListener.parsePerformed( ) をコールします。

パラメータ

sURL
解析されるメディア・ファイルの URL

annListener
解析の完了時に通知されるリスナー

戻り値

このタスクに対応付けられた AnnTaskMonitor オブジェクトを戻します。

例外

なし

例

このメソッドの例は、3.5 項を参照してください。

parseMedia(String, AnnListener, String)

構文

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

説明

ソースを解析し、指定された URL のアノテーションを作成します。解析が完了すると、次の操作を実行します。

- アノテーションへの次の属性の設定を試みます。
  - MEDIA_SIZE
  - MEDIA_SOURCE_DIRECTORY
  - MEDIA_SOURCE_FILENAME
  - MEDIA_SOURCE_PROTOCOL
  - MEDIA_SOURCE_URL
  - MEDIA_SOURCE_MIME_TYPE
- コールバック関数 AnnListener.parsePerformed( ) をコールします。

このメソッドでは、メディア・ファイルで使用されているキャラクタ・セットに関係なく、メディア・ファイルからアノテーションを解析できます。たとえば、このメソッドは各国語のキャラクタ・セット（日本語の Shift-JIS など）が含まれているメディアを解析できます。

パラメータ

sURL
解析されるメディア・ファイルの URL

annListener
解析の完了時に通知されるリスナー

szCharEncoding
キャラクタ・エンコーディング

戻り値

このタスクに対応付けられた AnnTaskMonitor オブジェクトを戻します。

例外

なし

例

このメソッドの例は、3.5 項を参照してください。

playMedia( )

構文

```
public void playMedia(Annotation ann, AnnListener annListener)
```

説明

指定されたアノテーションが示すコンテンツを再生します。このメソッドは同期であるため、AnnTaskMonitor オブジェクトを戻しません。

パラメータ

ann
コンテンツを再生するアノテーション

annListener
操作の完了時に通知されるリスナー

戻り値

なし

例外

なし

例

このメソッドの例は、4-40 ページの「isPlayable( )」メソッドを参照してください。

oracle.ord.media.annotator.handlers.db.OrdFileMapping クラス

この項では、アノテーション・インスタンスの内容をデータベースの特定の表および特定の行にマップする、OrdFileMapping オブジェクトに対応付けられたコンストラクタおよびメソッドに関するリファレンス情報を示します。

このクラスは、oracle.ord.media.annotator.handlers.db.OrdMapping を拡張します。

OrdFileMapping コンストラクタ

構文

```
public OrdFileMapping(java.lang.String szFileName)
```

説明

データベースへのアノテーションの内容のマッピングを含む、OrdFileMapping オブジェクトを作成します。

パラメータ

szFileName
マッピングを含むファイルの名前

戻り値

なし

例外

なし

例

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

このコンストラクタの使用例は、3.7 項を参照してください。

generateStatement( )

構文

```
public java.lang.String generateStatement(Annotation ann)
```

説明

データベースへのアノテーションの挿入に使用される PL/SQL 文を戻します。この文は、*inter*Media Annotator 固有のディレクティブを挿入するため、*inter*Media Annotator プリプロセッサによって処理されます。

このメソッドは、OrdMapping.generateStatement( ) を上書きします。

パラメータ

ann
挿入されるアノテーション

戻り値

データベースへのアノテーションの挿入に使用される PL/SQL 文を戻します。

例外

java.io.IOException

例

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

oracle.ord.media.annotator.handlers.utils.MimeMap クラス

この項では、MimeMap クラスのコンストラクタおよびメソッドに関するリファレンス情報を示します。このクラスは、MIME マップ・ファイル Annotator.mime で指定された MIME タイプ（またはファイル拡張子）とアノテーション名、パーサー名およびプレーヤ名との間のマッピングを保持します。このクラスは、マッピング用に読み込むファイルが、Preferences クラスによって指定されるという点では、Preferences クラスからのプロパティに依存しています。

MimeMap コンストラクタ

構文

```
public MimeMap( )
```

説明

MIME タイプとアノテーション名、パーサー名およびプレーヤ名との間のマッピングを、MIME マッピング・ファイルからハッシュテーブルに読み込みます。マッピング・ファイルは、Annotator.prefs ファイルの MimeMapFile パラメータで指定します。デフォルトでは、MIME マッピング・ファイルは Annotator.mime という名前で、次のディレクトリに格納されています。

- UNIX の場合 : *$ORACLE_HOME*/ord/Annotator/lib/conf
- Windows の場合 : *ORACLE_HOME*¥ord¥Annotator¥lib¥conf

Annotator.prefs ファイルも、このディレクトリに格納されています。
Annotator.mime ファイルの場所の設定の詳細は、2.1 項を参照してください。

パラメータ

なし

戻り値

なし

例外

oracle.ord.media.annotator.AnnotatorException

例

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

clone( )

構文

```
public java.lang.Object clone( )
```

説明

このオブジェクトのコピーを作成して戻します。詳細は、Java 1.2 のマニュアルに記載されている java.lang.Object.clone( ) メソッドの説明を参照してください。

パラメータ

なし

戻り値

現行の MimeMapping オブジェクトのコピーをオブジェクトとして戻します。

例外

java.lang.CloneNotSupportedException

例

なし。上級プログラマのみが、このメソッドをコールしてください。

getAnnotationName(String)

構文

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

説明

指定された MIME タイプを持つアノテーションのクラス名を戻します。

パラメータ

szMimeType
MIME タイプの名前

戻り値

指定された MIME タイプにマップされたアノテーションの完全修飾クラス名を戻します。

例外

なし

例

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

getMimeTypes( )

構文

```
public java.util.Enumeration getMimeTypes( )
```

説明

*inter*Media Annotator での処理のために、現在登録されているすべての MIME タイプのリストを戻します。

パラメータ

なし

戻り値

すべての MIME タイプのリストを戻します。

例外

なし

例

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

getMimeTypesCount( )

構文

```
public int getMimeTypesCount( )
```

説明

ハッシュテーブル内の MIME タイプの数を戻します。

パラメータ

なし

戻り値

ハッシュテーブル内の MIME タイプの数を整数で戻します。

例外

なし

例

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

getParserName( )

構文

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

説明

指定された MIME タイプにマップされたパーサーのクラス名を戻します。

パラメータ

szMIMEType
MIME タイプの名前

戻り値

指定された MIME タイプにマップされたパーサーの完全修飾クラス名を戻します。

例外

なし

例

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

getParsers( )

構文

```
public java.util.Enumeration getParsers( )
```

説明

*inter*Media Annotator によって MIME タイプに現在登録されているすべてのパーサーのリストを戻します。

パラメータ

なし

戻り値

すべてのパーサーのリストを戻します。

例外

なし

例

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

handlesMime( )

構文

```
public boolean handlesMime(java.lang.String szMIMEType)
```

説明

指定された MIME タイプに対して、アノテーション、パーサーまたはプレーヤが、ハッシュテーブルにすでに定義されているかどうかを判断します。

パラメータ

szMIMEType
MIME タイプの名前

戻り値

ブーリアン値。指定された MIME タイプが、アノテーション、パーサーまたはプレーヤにすでに定義されている場合は true を戻し、それ以外の場合は false を戻します。

例外

なし

例

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

removeMimeType( )

構文

```
public void removeMimeType(java.lang.String szMIMEType)
```

説明

MIME タイプをハッシュテーブルから削除します。

パラメータ

szMIMEType
MIME タイプの名前

戻り値

なし

例外

なし

例

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

saveMIMEMappings( )

構文

```
public void saveMIMEMappings()
```

説明

MIME 設定を MIME マッピング・ファイル Annotator.mime に書き込みます。

パラメータ

なし

戻り値

なし

例外

java.io.Exception

例

なし。上級プログラマのみが、このメソッドをコールしてください。

setMimeMap( )

構文

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

説明

MIME タイプをアノテーション、パーサーおよびプレーヤにマップします。

パラメータ

szMimeType
MIME タイプの名前

szAnnotation
MIME タイプにマップするアノテーションの名前

szParser
MIME タイプにマップするパーサーの名前

szPlayer
MIME タイプにマップするプレーヤの名前

戻り値

なし

例外

java.io.IOException

例

なし。上級プログラマのみが、このメソッドをコールしてください。

oracle.ord.media.annotator.listener.AnnListener クラス

この項では、AnnListener インタフェースのメソッドに関するリファレンス情報を示します。クライアントは、*inter*Media Annotator エンジンを起動するため、このインタフェースを実装する必要があります。

このクラスは java.util.EventListener を拡張します。

errorOccured( )

構文

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

説明

致命的エラーが発生した場合に例外を戻します。

AnnotationHandler.insertMedia( ) によってエラーが生成された場合、JDBC 接続が自動的にロールバックされ、クローズされます。

パラメータ

ann
アノテーション・インスタンス

e
障害が発生した理由を説明する例外

戻り値

なし

例外

なし

例

このメソッドの例は、3.10 項を参照してください。

extractionPerformed( )

構文

```
public void extractionPerformed(Annotation ann)
```

説明

メディア・サンプルの抽出の完了後、すべての必要な操作を実行します。このメソッドは、AnnotationHandler.extractMedia( ) のコールバック関数です。

抽出が完了すると、アノテーションに新しい属性が定義されます。新しく定義された属性は、抽出されたサンプルに関連しています。新しい属性を表示するには、クライアントでアノテーションをリフレッシュする必要があります。

パラメータ

ann
抽出を実行したアノテーション・インスタンス

戻り値

なし

例外

なし

例

このメソッドの例は、3.7 項を参照してください。

insertionPerformed( )

構文

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

説明

データベースへのアノテーションの挿入の完了後、すべての必要な操作を実行します。これらの操作には、データベースへの変更の明示的なコミットまたはロールバック、およびデータベースへの接続のクローズが含まれます。

データベースへの接続をオープンな状態に保持し、AnnotationHandler.insertMedia( ) の他のコールに渡すことができます。ただし、接続がスレッド・セーフであることを確認する必要があります。

このメソッドは、AnnotationHandler.extractMedia( ) のコールバック関数です。

パラメータ

ann
データベースに挿入されたアノテーション・インスタンス

conn
挿入の実行に使用される JDBC 接続

戻り値

なし

例外

なし

例

このメソッドの例は、3.8 項を参照してください。

parsePerformed( )

構文

```
public void parsePerformed(Annotation ann)
```

説明

アノテーションの作成後、データベースにアップロードする前に、すべての必要な操作を実行します。このメソッドは、AnnotationHandler.parseMedia( ) のコールバック関数です。

パラメータ

ann
新しく作成されたメディア・アノテーション

戻り値

なし

例外

なし

例

このメソッドの例は、3.6 項を参照してください。

warningOccured( )

構文

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

説明

致命的でないエラーが発生した場合に例外を戻します。

パラメータ

ann
アノテーション・インスタンス

e
障害が発生した理由を説明する例外

戻り値

なし

例外

なし

例

このメソッドの例は、3.9 項を参照してください。

oracle.ord.media.annotator.listener.OutputListener クラス

この項では、OutputListener インタフェースのメソッドに関するリファレンス情報を示します。クライアントは、このメソッドをコールしてエンジンからのステータスの出力を処理します。

このクラスは java.util.EventListener を拡張します。

ConsoleOutput( )

構文

```
public void ConsoleOutput(java.lang.String szOutput)
```

説明

エンジンの実行中にステータス・メッセージを出力します。

パラメータ

szOutput
出力するステータス・メッセージ

戻り値

なし

例外

なし

例

このメソッドの例は、3.11 項を参照してください。

oracle.ord.media.annotator.utils.Preferences クラス

この項では、Preferences クラスに対応付けられたコンストラクタおよびメソッドに関するリファレンス情報を示します。このクラスは、主にエンジンによって使用されます。プリファレンスのロード、プリファレンスの動的変更およびファイルへのプリファレンスの保存をサポートします。

エンジンは、初期化されるとシステム Preferences オブジェクトの静的コピーをロードし、構成ディレクトリ内の `Annotator.prefs` ファイルから読み込みます。デフォルトでは、現行ディレクトリの `¥lib¥conf` サブディレクトリまたは `/lib/conf` サブディレクトリが構成ディレクトリとみなされます。

ただし、インストール時、構成ファイルは次のディレクトリに格納されています。

- UNIX の場合 : *$ORACLE_HOME*`/ord/Annotator/lib/conf`
- Windows の場合 : *ORACLE_HOME*`¥ord¥Annotator¥lib¥conf`

Oracle *inter*Media Annotator を使用する前に、プリファレンス・ファイルおよび他の構成ファイルが構成ディレクトリに含まれていることを確認する必要があります。たとえば、ディレクトリ `/usr5/myfiles` から *inter*Media Annotator が実行された場合は、`/usr5/myfiles/lib/conf` が構成ディレクトリであるとみなされます。

プリファレンス・ファイルの詳細は、2.1 項を参照してください。

プリファレンスは、setPreferences( ) メソッドを使用して設定できます。また、プリファレンスの取得には、getPrefs( ) メソッドが使用できます。詳細は、4-76 ページの「setPreferences( )」および 4-73 ページの「getPrefs( )」を参照してください。

このクラスの実装は、他の *inter*Media Annotator クラスに依存しません。

このクラスは、java.lang.Object を拡張し、oracle.ord.media.annotator.utils.PreferenceConstants および java.lang.Cloneable を実装します。

Preferences コンストラクタ

構文

```
public Preferences( )
```

説明

Preferences オブジェクトを作成して、デフォルトのプリファレンス・ファイル Annotator.prefs からプリファレンスを読み込みます。プリファレンス・ファイルの詳細は、2.1 項を参照してください。

パラメータ

なし

戻り値

なし

例外

なし

例

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

Preferences(Properties) コンストラクタ

構文

```
public Preferences(java.util.Properties props)
```

説明

Preferences オブジェクトを作成して、プロパティ・リストからプリファレンスを読み込み、そのプリファレンスをオブジェクトに移入します。このリストは、2.1 項で説明したフォーマットおよびパラメータを使用するファイルからロードできます。

パラメータ

props
プロパティ・リスト

戻り値

なし

例外

なし

例

```
// 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( )

構文

```
public java.lang.Object clone( )
```

説明

このオブジェクトのコピーを作成して戻します。詳細は、Java 1.2 のマニュアルに記載されている java.lang.Object.clone( ) メソッドの説明を参照してください。

パラメータ

なし

戻り値

現行の Preferences オブジェクトのコピーをオブジェクトとして戻します。

例外

java.lang.CloneNotSupportedException

例

なし。*inter*Media Annotator プリファレンスに手動でアクセスする上級プログラマのみが、このメソッドを直接コールしてください。

getPrefs( )

構文

```
public static Preferences getPrefs( )
```

説明

現行のアノテーションのシステム Preferences オブジェクトを取得します。

パラメータ

なし

戻り値

現行のアノテーションの Preferences オブジェクトを戻します。

例外

なし

例

このメソッドの例は、3.4 項を参照してください。

getProperty( )

構文

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

説明

現行のアノテーションのプリファレンスから指定されたプロパティの値を取得します。

パラメータ

s
値を取得するプロパティの名前

戻り値

プロパティの値を文字列として戻します。

例外

なし

例

なし。*inter*Media Annotator プリファレンスに手動でアクセスする上級プログラマのみが、このメソッドを直接コールしてください。

saveToFile( )

構文

```
public void saveToFile( )
```

説明

ファイルにプリファレンスを保存します。

パラメータ

なし

戻り値

なし

例外

なし

例

なし。*inter*Media Annotator プリファレンスに手動でアクセスする上級プログラマのみが、このメソッドを直接コールしてください。

setPreferences( )

構文

```
public static void setPreferences(Preferences prefs)
```

説明

指定された Preferences オブジェクトに一致するように、現行のアノテーションのシステム・プリファレンスを設定します。

パラメータ

prefs
アノテーションに設定するプリファレンス

戻り値

なし

例外

なし

例

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

setProperty( )

構文

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

説明

指定されたプロパティを指定された値に設定します。

パラメータ

s
設定するプロパティの名前

o
設定する値

戻り値

なし

例外

なし

例

このメソッドの例は、3.4 項を参照してください。

oracle.ord.media.annotator.utils.Status クラス

この項では、Status クラスに対応付けられたメソッドに関するリファレンス情報を示します。このクラスは、アプリケーションの現行のステータスを更新します。ユーザーは、サポートされている 3 つのステータス・モードから選択できます。モードには、出力される可能性が低いものから順番に、STATUS（または TERSE）、VERBOSE および TRACE があります。

Status クラスは単一のパターンに従うため、Java Virtual Machine（JVM）の *inter*Media Annotator エンジンのすべてのインスタンスに対して、1 つのインスタンスのみが必要です。Status クラスがスレッド・セーフでないことに注意してください。

このクラスは java.lang.Object を拡張します。

このクラスは、エラー・レベルの設定に使用される次のフィールドを含みます。

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

このクラスは、出力モードの設定に使用される次のフィールドを含みます。

- 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( )

構文

```
public short GetOutputMode( )
```

説明

Status オブジェクトの現行の出力モードを戻します。

パラメータ

なし

戻り値

Status オブジェクトの現行の出力モードを戻します。可能な値は、OUTPUT_MODE_STATUS、OUTPUT_MODE_TERSE、OUTPUT_MODE_TRACE または OUTPUT_MODE_VERBOSE です。

例外

なし

例

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

getStatus( )

構文

```
public static Status getStatus( )
```

説明

現行のアノテーションの Status オブジェクトを取得します。

パラメータ

なし

戻り値

Status オブジェクトを戻します。

例外

なし

例

このメソッドの例は、3.4 項を参照してください。

initStatus( )

構文

```
public static void initStatus(OutputListener ol)
```

説明

Status オブジェクトを初期化します。このメソッドは、AnnotationHandler オブジェクトを初期化する前にコールする必要があります。

パラメータ

ol
AnnotationHandler オブジェクトからステータス・メッセージを受信する OutputListener クラスのインスタンス

戻り値

なし

例外

なし

例

このメソッドの例は、3.4 項を参照してください。

Report( )

構文

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

説明

指定されたメッセージを適切な出力ソースに出力します。出力ソースは、Status オブジェクトがインスタンス化されるときに内部的に設定されます。

このメソッドは、パーサー開発者のみが使用してください。

パラメータ

omDesignated
出力モード。ここで指定する出力モードの優先順位がエンジンに設定されている出力モードより低い場合、メッセージがレポートされます。

szStatus
レポートするメッセージ

戻り値

なし

例外

なし

例

このメソッドの例は、3.4 項を参照してください。

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

構文

```
public void setProperty(java.lang.String s,
                        java.lang.Object oInstance,
                        java.lang.String szMethodName,
                        int iLineNumber, java.lang.String szDesc)
```

説明

System.err ストリームを介してエラーをレポートします。複数のエラー・レベルを設定して結果を指定することができます。

パラメータ

sErrLevel
16 ビットのエラー・レベル（ERR_LEVEL_WARNING、ERR_LEVEL_ERROR または ERR_LEVEL_FATALERROR）

oInstance
エラー元のオブジェクト・ポインタ

szMethodName
エラーが発生したメソッドの名前（文字列）

iLineNumber
エラーが発生した行番号

szDesc
エラーの詳細な記述

戻り値

なし

例外

なし

例

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

ReportError(short, Throwable)

構文

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

説明

System.err ストリームを介してエラーをレポートします。複数のエラー・レベルを設定して結果を指定することができます。

パラメータ

sErrLevel
16 ビットのエラー・レベル（ERR_LEVEL_WARNING、ERR_LEVEL_ERROR または ERR_LEVEL_FATALERROR）

sException
発生した例外（Throwable オブジェクト）

戻り値

なし

例外

なし

例

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

SetOutputMode( )

構文

```
public void SetOutputMode(short omNew)
```

説明

ステータス出力モードを、STATUS、TERSE、TRACE または VERBOSE に設定します。

パラメータ

omNew
設定する出力モード。値は、OUTPUT_MODE_STATUS、OUTPUT_MODE_TERSE、OUTPUT_MODE_TRACE または OUTPUT_MODE_VERBOSE である必要があります。

戻り値

なし

例外

なし

例

このメソッドの例は、3.4 項を参照してください。

5

PL/SQL Upload Template の作成

Oracle *inter*Media Annotator は、メディア・データおよび対応付けられたアノテーションを、Oracle *inter*Media がインストールされている Oracle データベースにアップロードできます。*inter*Media Annotator は、Oracle PL/SQL Upload Template を介してこれを行います。Oracle PL/SQL Upload Template には、PL/SQL コールと *inter*Media Annotator 固有のキーワードの両方が含まれています。

テキスト・エディタを使用して、独自の PL/SQL Upload Template を作成します。

5.1 メディア・データのアップロードの概要

Oracle *inter*Media Annotator は、インポートおよびリモートという 2 つの異なる方法で、メディア・ソースおよび対応付けられたアノテーションをデータベースにアップロードできます。

インポート・アップロード方法では、データベース・サーバーが（ファイル・システム内で、または HTTP ストリームを介して）メディア・ソースを参照できる必要があります。この方法では、メディア・ソースはファイル・システムから直接データベースにアップロードされます。インポート・アップロード方法では、Oracle *inter*Media import( ) メソッドを使用します。

リモート・アップロード方法では、データベースがメディア・ソースを参照できる必要はありません。ファイルは *inter*Media Annotator にロードされ、*inter*Media Annotator が JDBC コールを介してファイルをデータベースにロードします。リモート・アップロード方法では、${MANN_UPLOAD_SRC} という *inter*Media Annotator 固有のキーワードを使用します。

アップロード方法では、次のことに注意してください。

- インポート・アップロード方法を使用し、メディア・ソースがファイル・システム内に存在する場合は、そのメディア・ファイルが存在するディレクトリへのパスを指定する必要があります。この場合、アップロード先の Oracle データベース・サーバーから見たディレクトリ・パスを指定する必要があります。たとえば、Windows NT 上で *inter*Media Annotator を実行中に、UNIX プラットフォーム上で実行している Oracle データベースにデータをアップロードする場合、両方のマシンがアクセスできるディレクトリ内にメディア・データがある必要があります。これを行うには、サーバー上の UNIX ディレクトリを Windows NT ネットワーク・ドライブにマウントします。
- インポート・アップロード方法を使用し、メディア・ソースが HTTP ストリームの場合は、メディア・ソースをデータベースにインポートするか、または URL をデータベースに格納するかのいずれかを選択できます。
- リモート・アップロード方法を使用し、JDBC Thin ドライバを使用している場合は、大規模なファイルをアップロードする場合特に、アップロードのパフォーマンスが低下することがあります。

PL/SQL と Oracle *inter*Media Annotator 固有のキーワードの組合せを使用して、メディア・データおよびアノテーションを Oracle データベース内の表にアップロードします。この表には、適切な *inter*Media オブジェクト型（オーディオ・ファイルのアノテーションには ORDSYS.ORDAudio、イメージ・ファイルのアノテーションには ORDSYS.ORDImage、ビデオ・ファイルのアノテーションには ORDSYS.ORDVideo など）の列が 1 つ以上含まれている必要があります。

表内の新しい行にデータを挿入するか、または表内の既存の行を更新できます。

*inter*Media Annotator 固有のキーワードについては、5.3 項を参照してください。

5.2 PL/SQL Upload Template の作成

任意のテキスト・エディタを使用して PL/SQL Upload Template を作成してください。PL/SQL Upload Template の構造では、次のことに注意してください。

- PL/SQL Upload Template は、DML 文および DDL 文のリストで始まります。このリストは必要に応じて使用します。
- リストの次には、1 つの無名 PL/SQL ブロックが続きます。指定できる無名 PL/SQL ブロックは 1 つのみであり、このブロックの後には何も記載できません。

  無名 PL/SQL ブロックには、標準の PL/SQL コードおよび *inter*Media Annotator 固有のキーワードの両方が含まれます。キーワードの詳細は、5.3 項を参照してください。PL/SQL コードの記述の詳細は、『PL/SQL ユーザーズ・ガイドおよびリファレンス』を参照してください。

データベース・サーバーのプラットフォームによっては、無名 PL/SQL ブロックの最大サイズが制限される場合があります。この場合、いくつかの文を PL/SQL プロシージャにパッケージ化し、PL/SQL ブロックのサイズを小さくすることによって対処できます。

5.3 *inter*Media Annotator 固有のキーワード

PL/SQL Upload Template には、標準の PL/SQL コールに加えて *inter*Media Annotator 固有のキーワードが含まれます。キーワードは、始まりは ${ 、終わりは } で区切られています。これらのキーワードは、適切な PL/SQL コードを生成する、*inter*Media Annotator プリプロセッサで解析されます。

注意： *inter*Media Annotator 固有のキーワードは、PL/SQL Upload Template の特定の行に出現する必要があります。同一の行に複数のキーワードを指定することはできません。

属性名はキーワードとして使用できます。詳細は、5.3.1 項を参照してください。また、5.3.2 項～ 5.3.6 項で説明するキーワードも使用できます。

5.3.1 属性値

PL/SQL Upload Template では、特定の属性値をハード・コードするかわりに、属性名を「${」と「}」という文字で囲んで指定します。これによって、プリプロセッサが現行のアノテーションから属性の実際の値を取得し、その値を PL/SQL Upload Template のキーワードと置換します。この単純な置換方法では、同一の PL/SQL Upload Template を複数のアノテーションに使用できます。

例 5-1 に、属性値で置換されるキーワードを示します。

例 5-1 キーワードとしての属性値

```
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());
```

*inter*Media Annotator で定義される属性名のリストは、付録 C を参照してください。

5.3.2 ${MANN_BEGIN_ITERATE} および ${MANN_END_ITERATE}

${MANN_BEGIN_ITERATE} キーワードおよび ${MANN_END_ITERATE} キーワードは、指定された型の副アノテーションごとに、これらのキーワードで囲まれたコードを繰り返し使用する必要があることを示します。${MANN_BEGIN_ITERATE} キーワードの後には、アノテーション型の名前を指定します。

例 5-2 に、現行のアノテーションの副アノテーションとして存在する TextSampleAnn アノテーションごとに実行されるコードのブロックを示します。

例 5-2 ${MANN_BEGIN_ITERATE} および ${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} および ${MANN_END_IFDEF}

${MANN_BEGIN_IFDEF} キーワードおよび ${MANN_END_IFDEF} キーワードは、現行のアノテーションで任意の属性に値が定義されている場合にのみ、これらのキーワードで囲まれたコードを実行する必要があることを示します。${MANN_BEGIN_IFDEF} キーワードの後には、属性の名前を指定します。

例 5-3 に、現行のアノテーションで MEDIA_SOURCE_MIME_TYPE 属性が定義されている場合にのみ実行されるコードのブロックを示します。

例 5-3 ${MANN_BEGIN_IFDEF} および ${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} および ${MANN_END_IFEQUALS}

${MANN_BEGIN_IFEQUALS} キーワードおよび ${MANN_END_IFEQUALS} キーワードは、現行のアノテーションで任意の属性に任意の値が含まれる場合にのみ、これらのキーワードで囲まれたコードを実行する必要があることを示します。
${MANN_BEGIN_IFEQUALS} キーワードの後には、属性の名前および値を指定します。文字列の比較では、大 / 小文字が区別されます。

例 5-4 に、現行のアノテーションで MEDIA_SOURCE_MIME_TYPE 属性が `audio/basic` と定義されている場合にのみ実行されるコードのブロックを示します。

例 5-4 ${MANN_BEGIN_IFEQUALS} および ${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}

${MANN_UPLOAD_SRC} キーワードは、現行のアノテーションに対応付けられたメディア・ソース・データを、JDBC を使用して現行の Oracle データベース表にアップロードする必要があることを示します。ファイルは *inter*Media Annotator にロードされ、*inter*Media Annotator によってデータベースにロードされます。${MANN_UPLOAD_SRC} キーワードの後には、サーバー側のオブジェクトの名前および属性（BLOB 型）を指定します。

大きいメディア・ソースのアップロードに JDBC Thin ドライバを使用する場合、またはネットワーク接続が低速の場合、${MANN_UPLOAD_SRC} キーワードによるアップロードの速度が遅くなる場合があります。また、*inter*Media import( ) メソッドを使用する方が、速度が早くなる場合があります。2 つのアップロード・オプションの詳細は、5.1 項を参照してください。import( ) メソッドの詳細は、『Oracle *inter*Media ユーザーズ・ガイドおよびリファレンス』を参照してください。

例 5-5 に、サーバー側の *inter*Media オブジェクトである videoObj の source.localData 属性に現行のメディア・ソース・データをアップロードするコードのブロックを示します。

例 5-5 ${MANN_UPLOAD_SRC}

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

5.3.6 ${MANN_UPLOAD_XML}

${MANN_UPLOAD_XML} キーワードは、現行の Oracle データベース表に現行のアノテーションをアップロードする必要があることを示します。アノテーションは、Oracle *inter*Media オブジェクトの CLOB にアップロードする必要があります。
${MANN_UPLOAD_XML} キーワードの後には、サーバー側のオブジェクトの名前および CLOB 属性を指定します。

例 5-6 に、サーバー側の *inter*Media オブジェクトである videoObj のコメント属性に現行のアノテーションをアップロードするコードのブロックを示します。

例 5-6 ${MANN_UPLOAD_XML}

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

Oracle *inter*Media API の詳細は、『Oracle *inter*Media ユーザーズ・ガイドおよびリファレンス』を参照してください。

5.4 完全な PL/SQL Upload Template の例

例 5-7 に、PL/SQL Upload Template のサンプルを示します。このサンプルは、ビデオ・オブジェクトおよび対応付けられたアノテーションを MediaTable という名前の Oracle データベース表にアップロードします。このサンプルには、PL/SQL コールと *inter*Media Annotator 固有のキーワードが混在する 1 つの無名 PL/SQL ブロックが含まれます。

例 5-7 PL/SQL Upload Template のサンプル

```
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;
```

例 5-7 に示されている MEDIA_SOURCE_MIME_TYPE の設定に加え、MEDIA_TITLE, MEDIA_SOURCE_DIRECTORY、MEDIA_SOURCE_FILE_FORMAT などの他の属性も設定できます。詳細は、『Oracle *inter*Media ユーザーズ・ガイドおよびリファレンス』を参照してください。

5.5 ファイルの保存

PL/SQL Upload Template の作成後、接尾辞 `.ofm` を付けて保存します。Oracle *inter*Media Annotator は、PL/SQL Upload Template に対して次のデフォルトのディレクトリを使用します。

- UNIX の場合 : *`$ORACLE_HOME`*`/ord/Annotator/ofm`
- Windows の場合 : *`ORACLE_HOME`*`¥ord¥Annotator¥ofm`

デフォルトのディレクトリを変更するには、`Annotator.prefs` ファイル内の ofmDirectory パラメータの値を変更してください。

第 II 部

Oracle *inter*Media Annotator の拡張性

第 II 部では、Oracle *inter*Media Annotator の拡張性について説明します。第 II 部に含まれる章は、次のとおりです。

- 第 6 章「カスタム・パーサーの例」
- 第 7 章「interMedia Annotator パーサー API のリファレンス情報」
- 第 8 章「新しいアノテーション型の作成」

6

カスタム・パーサーの例

この章では、ユーザー定義のコンテンツ・フォーマットに使用する新しいパーサーの作成例を示します。また、この章では、AuParser について説明します。AuParser は、ユーザー定義のコンテンツ・フォーマットに使用するユーザー開発パーサーの一例で、*inter*Media Annotator パーサー API を使用して作成されます。この AuParser には、Java および *inter*Media Annotator API を使用して NeXT/Sun AU ファイル・フォーマット用のパーサーを定義する、ユーザー定義のメソッドが含まれています。このパーサーを使用すると、フォーマットのエンコーディングに関する情報および関連付けられたユーザー・データをファイルから抽出できます。

ソース・コードは、`AuParser.java` ファイルに含まれています。このファイルは、次の場所に格納されている `parsers.zip` ファイルに含まれています。

- UNIX の場合 : *$ORACLE_HOME*`/ord/Annotator/src`
- Windows の場合 : *ORACLE_HOME*`¥ord¥Annotator¥src`

この章で示す例は、Oracle *inter*Media Annotator のインストール時に `AuParser.java` として提供されるコードとは異なる場合があります。システム上でこの例を実行する場合は、インストール時に提供されるファイルを使用してください。この章で示すコードは、コンパイルして実行しないでください。

注意 : この章には、Java コードの例が含まれます。コード例には、カッコで囲まれた数字が含まれている場合があります。これは、例のすぐ後にある手順にそのコードの詳細が説明されていることを示します。

6.1 パーサーの作成の概要

新しいパーサーを定義するには、次の手順を実行します。

1. oracle.ord.media.annotator.parsers.Parser クラスからプロパティおよびメソッドを継承する新しい Java クラスを作成します。この例では、Java クラスは、AuParser という名前で、`AuParser.java` ファイルに定義されています。

   新しい Java クラスは次のメソッドを実装する必要があります。

   - parse( ): InputStream オブジェクトの内容を解析します。
   - saveToAnnotation( ): m_annInst という名前のアノテーションとして解析の結果を保存します。
   - extractSamples( ): メディア・ソース・ファイルからサンプルを抽出します。このメソッドは、サンプルの抽出をサポートするかどうかに関係なく、実装する必要があります。

   パーサーで実行する必要がある操作に応じて、他のメソッドを追加することもできます。

   `AuParser.java` の例の内容については、6.3 項～ 6.10 項を参照してください。

2. パーサー記述子 XML ファイルを作成し、次のディレクトリに追加します。

   - UNIX の場合 :

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

   - Windows の場合 :

     ORACLE_HOME`¥ord¥Annotator¥lib¥descriptors¥parsers`

   このディレクトリには、パーサー記述子 XML ファイル `AuParser.xml` の例も含まれています。

3. `Annotator.mime` ファイルを変更して、特定の MIME タイプに使用するパーサーを設定します（オプション）。このファイルは、次のディレクトリに格納されています。

   - UNIX の場合 :

     $ORACLE_HOME`/ord/Annotator/lib/conf`

   - Windows の場合 :

     ORACLE_HOME`¥ord¥Annotator¥lib¥conf`

6.2 AU ファイルの構造

例 6-1 に、AU フォーマットのファイルの基本構造を示します。

例 6-1 AU ファイルの基本構造

```
<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>
```

`magic` パラメータは、ASCII 文字「.snd」を表す SND_MAGIC ((int)0x2e736e64) と同等である必要があります。`info` パラメータは、アノテーションのユーザー・データ属性に対応付けられます。

6.3 パッケージ文およびインポート文

例 6-2 に、*inter*Media Annotator パーサーの適切な実行のために Java プログラムに含める必要があるインポート文、およびこのクラスのパッケージを設定するパッケージ文を示します。

例 6-2 パッケージ文およびインポート文

```
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 クラス定義およびインスタンス変数

例 6-3 に、AuParser クラスのクラス定義およびインスタンス変数を示します。

例 6-3 クラス定義およびインスタンス変数

```
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;
```

パーサーは Parser クラスを拡張する必要があります。Parser クラスのフィールドおよびメソッドの詳細は、7-32 ページの「oracle.ord.media.annotator.parsers.Parser クラス」を参照してください。

AuParser クラスは、値にキーをマップする、m_htFormatInfo という名前の Hashtable オブジェクトを含みます。キーはプライベート静的整数オブジェクトとしてインスタンス化され、シーケンシャル値を与えられます。これらは `AuParser.java` に固有です。パーサーによっては不要な場合があります。

AuParser クラスは、特定のデータ・フォーマットに関連する情報をカプセル化するために使用される、m_fiFormatInfo という名前の FormatInfo オブジェクトを含みます。詳細は、6.5 項を参照してください。これらは `AuParser.java` の例に固有です。パーサーによっては不要な場合があります。

AuParser クラスは、次のインスタンス変数も含みます。

- m_iBitsPerSample: サンプルごとのオーディオ・ビット数（整数）
- m_iSampleRate: AU ファイルのサンプル・レート（整数）
- m_iChannelCount: AU ファイルのチャネルの数（整数）
- m_szUserData: AU ファイルに関連するオプションのテキスト情報

6.5 FormatInfo クラス

例 6-4 に、特定のデータ・フォーマットに関連する情報をカプセル化するために使用される、FormatInfo クラスの内容を示します。

例 6-4 FormatInfo クラス

```
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;
     }
}
```

FormatInfo クラスは、次のコードを含みます。

1. フォーマット文字列および書式コードの値を格納する 2 つのインスタンス変数
2. フォーマット文字列および書式コードを設定する 2 つのパラメータを持つコンストラクタ
3. コール元にフォーマット文字列を戻すメソッド
4. コール元に書式コードを戻すメソッド

6.6 AuParser( ) メソッド

例 6-5 に、このクラスのコンストラクタである、AuParser( ) メソッドの内容を示します。

例 6-5 AuParser( ) メソッド

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

パーサーには、パラメータがないコンストラクタを含める必要があります。

AuParser( ) メソッドのコードは、次の手順を実行します。

1. m_htFormatInfo オブジェクトをインスタンス化します。
2. FillFormatHashTable( ) メソッドをコールして m_htFormatInfo オブジェクトにデータを移入します。詳細は、例 6-9 を参照してください。

6.7 parse( ) メソッド

例 6-6 に、AU ファイルを解析して一部のメタデータを抽出する、parse( ) メソッドを示します。

例 6-6 parse( ) メソッド

```
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();
}
```

parse( ) メソッドのコードは、次の手順を実行します。

1. readInt( ) メソッドを使用して、AU ファイルの最初の整数（マジック・ナンバー）を読み込み、iMagicNumber に設定します。

   m_madisResource フィールドは、MADataInputStream オブジェクトを表します。MADataInputStream オブジェクトには、処理する AU ファイルが含まれている入力ストリームが保持されます。詳細は、7-37 ページの「oracle.ord.media.annotator.utils.MADataInputStream クラス」を参照してください。

2. 読み込んだマジック・ナンバーが AU ファイルのマジック・ナンバーに一致するかどうかをテストします。一致しない場合、ファイルは AU ファイルではなく、例外が発生します。

3. readInt( ) メソッドを使用して、m_madisResource フィールドの次の整数（メディア・データが開始するストリームの始まりからのオフセット）を読み込み、iDataLocation に設定します。

4. カウンタを設定します。すでに 2 つの整数（合計 8 バイト）が読み込まれているため、カウンタは 2 に設定されます。

5. AnnTaskManager.setTask( ) メソッドを使用して、AnnTaskMonitor オブジェクトの開始値および終了値を設定します。終了値は、メディア・データの始まりを示す iDataLocation の値です。この値は、ヘッダー情報の長さも表します。詳細は、7-26 ページの「setTask( )」を参照してください。

6. AnnTaskManager.setTaskCurrent( ) メソッドを使用して、AnnTaskMonitor オブジェクトに現行のタスクを設定します。現在の値は 8（読込み済のバイト数）に、メッセージは「Parsing AU Header...」に設定されています。詳細は、7-27 ページの「setTaskCurrent(int)」を参照してください。

7. m_madisResource フィールドの次の整数（ファイルのメディア・データのバイト数）を読み込みます。値を iDataSize に設定します。読み込んだ 4 バイトを反映するように、タスク処理過程モニターを設定します。

8. m_madisResource フィールド次の整数（データ・フォーマット・コード）を読み込みます。値を iDataFormat に設定します。読み込んだ 4 バイトを反映するように、タスク処理過程モニターを設定します。
9. m_madisResource フィールドの次の整数（ファイルのメディア・データのサンプル・レート）を読み込みます。値を iSampleRate に設定します。読み込んだ 4 バイトを反映するように、タスク処理過程モニターを設定します。
10. m_madisResource フィールドの次の整数（ファイルのメディア・データのチャネル数）を読み込みます。値を iChannelCount に設定します。読み込んだ 4 バイトを反映するように、タスク処理過程モニターを設定します。
11. readString( ) メソッドを使用して、残りのヘッダー情報を読み込み、値を m_szUserData に設定します。

    残りのヘッダー情報の長さは、ヘッダー情報の合計の長さから読込み済のバイト数（24）を引いて決定されます。
12. 読込み済のすべてのヘッダー情報を表示するように、タスク処理過程モニターの値を設定します。
13. AnnTaskMonitor オブジェクトの現行のタスクを終了します。
14. m_htFormatInfo という名前の Hashtable オブジェクトから適切な値を取得して、m_fiFormatInfo 変数の値を設定します。
15. m_iBitsPerSample 変数のデフォルトの値を 0（ゼロ）に設定します。
16. m_htFormatInfo Hashtable オブジェクトの一連の値に対する iDataFormat 変数の値を確認し、m_iBitsPerSample に適切な値を設定します。
17. 前の手順で発生したエラーまたは例外があれば、それらをキャッチします。
18. saveToAnnotation( ) メソッドをコールして、アノテーションを保存します。詳細は、6.8 項を参照してください。このメソッドは、parse( ) メソッドの実装が終了するたびにコールする必要があります。

6.8 saveToAnnotation( ) メソッド

例 6-7 に、saveToAnnotation( ) メソッドを示します。このメソッドは、parse( ) メソッドが正常に終了した後にコールする必要があります。

例 6-7 saveToAnnotation( ) メソッド

```
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));
}
```

saveToAnnotation( ) メソッドのコードは、次の手順を実行します。

1. アノテーションの MEDIA_SOURCE_FILE_FORMAT_CODE 属性および MEDIA_SOURCE_FILE_FORMAT 属性を AU ファイルの値に設定します。
2. m_fiFormatInfo 変数に値が指定されている場合は、その値のフォーマット文字列をアノテーションの MEDIA_FORMAT_ENCODING 属性に、書式コードを MEDIA_FORMAT_ENCODING_CODE 属性に設定します。
3. m_szUserData 変数に値が指定されている場合は、その値をアノテーションの MEDIA_USER_DATA 属性に設定します。
4. setAttribute( ) メソッドを使用して、m_iBitsPerSample 変数の値をアノテーションの AUDIO_BITS_PER_SAMPLE 属性に、m_iSampleRate 変数の値を AUDIO_SAMPLE_RATE 属性に、m_iChannelCount 変数の値を AUDIO_NUM_CHANNELS 属性に設定します。

パーサーは、AnnotationFactory クラスを使用して副アノテーションを作成し、m_annInst 変数に連結します。ただし、`AuParser.java` の例では、副アノテーションは作成されません。副アノテーションを作成するパーサーの例については、*inter*Media Annotator に含まれている QuickTime パーサー `QtParser.java` を参照してください。

6.9 extractSamples( ) メソッド

例 6-8 に、extractSamples( ) メソッドを示します。このメソッドは、AnnotationHandler.extractMedia( ) メソッドによってコールされます。

例 6-8 extractSamples( ) メソッド

```
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 は、AU ファイルからのサンプルの抽出をサポートしていません。このメソッドは、エラーまたは例外を発生させるかわりに、次の手順を実行します。

1. Status オブジェクトおよび Report( ) メソッドを使用して、このパーサーがサンプルの抽出をサポートしていないことを示すメッセージを出力します。詳細は、4-78 ページの「oracle.ord.media.annotator.utils.Status クラス」を参照してください。
2. AnnTaskManager.done( ) メソッドを使用して現行のタスクを終了します。詳細は、7-14 ページの「done( )」を参照してください。

サンプルの抽出をサポートするパーサーの例は、QuickTime パーサーを参照してください。

6.10 FillFormatHashTable( ) メソッド

例 6-9 に、Hashtable.put( ) メソッドを使用して m_htFormatString Hashtable オブジェクトの各キーに値を割り当てる FillFormatHashTable( ) メソッドを示します。詳細は、Java 1.2 のマニュアルを参照してください。このメソッドは、`AuParser.java` に固有です。パーサーによっては不要な場合があります。

例 6-9 FillFormatHashTable( ) メソッド

```
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

*inter*Media Annotator パーサー API のリファレンス情報

この章には、初心者のユーザーがユーザー定義の *inter*Media Annotator パーサーを作成するために必要なクラス、コンストラクタおよびメソッドに関する参照データが含まれます。詳細は、*inter*Media Annotator のインストール時に提供される Javadoc を参照してください。

ユーザー定義のパーサーを作成するには、この API を使用して Java クラスを作成する他に、パーサー記述子の XML ファイルを作成し、次のディレクトリに追加する必要があります。

- UNIX の場合 :

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

- Windows の場合 :

  ORACLE_HOME¥ord¥Annotator¥lib¥descriptors¥parsers

パーサー記述子の XML ファイルの例については、第 6 章および `parsers` ディレクトリの `AuParser.xml` ファイルを参照してください。

oracle.ord.media.annotator.descriptors.AnnotationDesc クラス

この項では、アノテーションの記述子オブジェクトを作成する、AnnotationDesc クラスのメソッドに関するリファレンス情報を示します。このクラスは、アノテーションの属性の定義を提供します。このクラスのメソッドは、*inter*Media Annotator へのパーサーの書込みまたは追加を実行する上級ユーザーを対象としています。

このクラスは、oracle.ord.media.annotator.descriptors.Descriptor を拡張します。

getAncestors( )

構文

```
public java.util.Vector getAncestors( )
```

説明

現行のアノテーションの親アノテーションを取得します。

パラメータ

なし

戻り値

現行のアノテーションの親アノテーションを含む Vector オブジェクトを戻します。

例外

なし

例

なし。上級ユーザーのみが、このメソッドを直接コールしてください。

getAttributeDesc( )

構文

```
public AttributeDesc getAttributeDesc
  (java.lang.String szAttributeName)
```

説明

指定された属性の属性記述子を取得します。

パラメータ

szAttributeName
取得する属性記述子を持つ属性の名前

戻り値

指定された属性の属性記述子を AttributeDesc オブジェクトとして戻します。

例外

なし

例

なし。上級ユーザーのみが、このメソッドを直接コールしてください。

getSuppAttributes( )

構文

```
public java.util.Enumeration getSuppAttributes( )
```

説明

アノテーション型に定義された、サポートされている属性記述子を取得します。

パラメータ

なし

戻り値

アノテーション型に定義された、サポートされている属性記述子を含む Enumeration オブジェクトを、AttributeDesc オブジェクトとして戻します。

例外

なし

例

なし。上級ユーザーのみが、このメソッドを直接コールしてください。

oracle.ord.media.annotator.descriptors.ParserDesc クラス

この項では、パーサー記述子オブジェクトを作成する、ParserDesc クラスのメソッドに関するリファレンス情報を示します。このクラスは、パーサー、パーサーのパラメータ、パーサーのオプションおよびオプション・パラメータで定義される操作の定義を提供します。このクラスのメソッドは、*inter*Media Annotator へのパーサーの書込みまたは追加を実行する上級ユーザーを対象としています。

このクラスは、oracle.ord.media.annotator.descriptors.Descriptor を拡張します。

getOperationDesc( )

構文

```
public OperationDesc getOperationDesc(java.lang.String szOpName)
```

説明

指定された操作の操作記述子を取得します。

パラメータ

szOpName
記述子を戻す操作の名前

戻り値

指定された操作の操作記述子を OperationDesc オブジェクトとして戻します。

例外

なし

例

なし。上級ユーザーのみが、このメソッドを直接コールしてください。

getOperations( )

構文

```
public java.util.Enumeration getOperations( )
```

説明

パーサー記述子に定義された、パーサーによってサポートされている操作の記述子を取得します。

パラメータ

なし

戻り値

パーサーによってサポートされている操作の記述子を含む Enumeration オブジェクトを、OperationDesc オブジェクトとして戻します。

例外

なし

例

なし。上級ユーザーのみが、このメソッドを直接コールしてください。

isEnabledAndExecutable( )

構文

```
public boolean isEnabledAndExecutable(java.lang.szOpName)
```

説明

指定された操作が有効で実行可能かどうかを確認します。

パラメータ

szOpName
確認する操作の名前

戻り値

メソッドが有効な場合は true、そうでない場合は false を戻します。

例外

oracle.ord.media.annotator.descriptors.DescriptorException

例

なし。上級ユーザーのみが、このメソッドを直接コールしてください。

oracle.ord.media.annotator.handlers.AnnTaskManager クラス

この項では、アノテーション・タスク・マネージャを作成する、AnnTaskManager クラスのコンストラクタおよびメソッドに関するリファレンス情報を示します。アノテーション・タスク・マネージャ・オブジェクトは、AnnotationHandler オブジェクト（アノテーション・ハンドラ）で実行中のタスクを監視するために使用されるコンポーネントの 1 つです。アノテーション・ハンドラでタスクが起動されるたびに、アノテーション・タスク・マネージャおよびアノテーション・タスク・モニターが作成されます。アノテーション・タスク・マネージャはサーバー側で実行し、データベース・サーバー上でのタスクの処理過程を追跡します。アノテーション・タスク・モニターはクライアント側で実行し、タスク処理過程モニターを介して、戻されたアノテーション・タスク・モニターのインスタンスから処理過程の値およびメッセージを追跡します。

アノテーション・タスク・モニターの詳細は、4-19 ページの「oracle.ord.media.annotator.handlers.AnnTaskMonitor クラス」を参照してください。

このクラスは java.lang.Object を拡張します。

次のフィールドを含みます。

- 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 コンストラクタ

構文

```
public AnnTaskManager( )
```

説明

AnnTaskManager オブジェクトを作成します。新しいタスクの開始時に、アノテーション・ハンドラでこのコンストラクタを使用して、アノテーション・タスク・マネージャを作成します。

パラメータ

なし

戻り値

なし

例外

なし

例

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

addIterCounter( )

構文

```
public void addIterCounter(int iIterCounter)
```

説明

指定された数をカウンタに追加します。

パラメータ

iIterCounter
カウンタに追加する値

戻り値

なし

例外

なし

例

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

decrIterCounter( )

構文

```
public void decrIterCounter( )
```

説明

カウンタの値を 1 つずつ減らします。

パラメータ

なし

戻り値

なし

例外

なし

例

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

done( )

構文

```
public void done( )
```

説明

現行のタスクが完了したことを示します。

パラメータ

なし

戻り値

なし

例外

なし

例

このメソッドの例は、6.9 項を参照してください。

getIterCounter( )

構文

```
public int getIterCounter( )
```

説明

カウンタの現在の値を取得します。

パラメータ

なし

戻り値

カウンタの現在の値を戻します。

例外

なし

例

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

getMessage( )

構文

```
public java.lang.String getMessage( )
```

説明

タスク処理過程モニターから現在のメッセージを取得します。

パラメータ

なし

戻り値

タスク処理過程モニターの現在のメッセージを戻します。

例外

なし

例

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

getTaskCurrent( )

構文

```
public int getTaskCurrent( )
```

説明

タスク処理過程モニターの現在の値を取得します。

パラメータ

なし

戻り値

タスク処理過程モニターの現在の値を戻します。

例外

なし

例

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

getTaskEnd( )

構文

```
public int getTaskEnd( )
```

説明

タスク処理過程モニターの終了値を取得します。

パラメータ

なし

戻り値

タスク処理過程モニターの終了値を戻します。

例外

なし

例

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

getTaskStart( )

構文

```
public int getTaskStart( )
```

説明

タスク処理過程モニターの開始値を取得します。

パラメータ

なし

戻り値

タスク処理過程モニターの開始値を戻します。

例外

なし

例

このメソッドの例は、7-23 ページの「isInitialized( )」メソッドを参照してください。

incrIterCounter( )

構文

```
public void incrIterCounter( )
```

説明

カウンタの値を 1 つずつ増やします。

パラメータ

なし

戻り値

なし

例外

なし

例

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

incrTaskCurrent( )

構文

```
public void incrTaskCurrent(int iTaskToAdd)
```

説明

タスク処理過程モニターの現在の値に、指定された値を追加します。

パラメータ

iTaskToAdd
タスク処理過程モニターの現在の値に追加する量

戻り値

なし

例外

なし

例

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

isDone( )

構文

```
public boolean isDone( )
```

説明

現行のタスクが完了しているかどうかを判断します。

パラメータ

なし

戻り値

現行のタスクが完了している場合は true、そうでない場合は false を戻します。

例外

なし

例

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

isInitialized( )

構文

```
public boolean isInitialized( )
```

説明

アノテーション・タスク・モニターが初期化されているかどうかを確認します。初期化されている場合、getTaskStart( ) および getTaskEnd( ) メソッドを使用できます。

パラメータ

なし

戻り値

アノテーション・タスク・モニターが初期化されている場合は true、そうでない場合は false を戻します。

例外

なし

例

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

setIterCounter( )

構文

```
public void setIterCounter(int iIterCounter)
```

説明

反復処理を追跡するようにカウンタを設定します。done( ) メソッドがコールされると、カウンタが 1 つ減ります。カウンタが 0（ゼロ）になると、isDone( ) メソッドは true を戻します。

パラメータ

iIterCounter
カウンタの初期値。デフォルト値は 1 です。

戻り値

なし

例外

なし

例

このメソッドの例は、7-22 ページの「isDone( )」メソッドを参照してください。

setMessage( )

構文

```
public void setMessage(java.lang.String szMessage)
```

説明

タスク処理過程モニターのメッセージを設定します。

パラメータ

szMessage
設定するメッセージ

戻り値

なし

例外

なし

例

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

setTask( )

構文

```
public void setTask(int iTaskStart, int iTaskEnd)
```

説明

タスク処理過程モニターの開始値および終了値を設定します。

パラメータ

iTaskStart
タスク処理過程モニターの開始値

iTaskEnd
タスク処理過程モニターの終了値

戻り値

なし

例外

なし

例

このメソッドの例は、6.7 項を参照してください。

setTaskCurrent(int)

構文

```
public void setTaskCurrent(int iTaskCurrent)
```

説明

タスク処理過程モニターの現在の値を設定します。

パラメータ

iTaskCurrent
タスク処理過程モニターに設定する値

戻り値

なし

例外

なし

例

このメソッドの例は、6.7 項を参照してください。

setTaskCurrent(int, String)

構文

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

説明

タスク処理過程モニターの現在の値およびメッセージを設定します。

パラメータ

iTaskCurrent
タスク処理過程モニターに設定する値

szMessage
タスク処理過程モニターに設定するメッセージ

戻り値

なし

例外

なし

例

このメソッドの例は、6.7 項を参照してください。

oracle.ord.media.annotator.handlers.annotation.AnnotationFactory クラス

この項では、AnnotationFactory クラスのコンストラクタおよびメソッドに関するリファレンス情報を示します。このクラスはアノテーションのファクトリ・クラスです。パーサーおよびアノテーションを作成するために使用される、（パーサー記述子用とアノテーション記述子用の）2 つのサブファクトリを含みます。AnnotationFactory クラスは、名前でアノテーションを作成することもできます。

このクラスは java.lang.Object を拡張します。

AnnotationFactory コンストラクタ

構文

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

説明

このコンストラクタは、MIME タイプまたはファイル拡張子からアノテーション、パーサーおよびプレーヤ・インスタンスへのマッピングを処理します。

パラメータ

mm
MimeMap インスタンス。MimeMap オブジェクトの詳細は、4-49 ページの「oracle.ord.media.annotator.handlers.utils.MimeMap クラス」を参照してください。

戻り値

なし

例外

なし

例

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

createAnnotationByName( )

構文

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

説明

アノテーション記述子ファクトリからアノテーション記述子を取得することによって、アノテーションをインスタンス化します。

パラメータ

szAnnName
新しいアノテーションの名前

戻り値

新しく作成されたアノテーションを戻します。

例外

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

例

このメソッドの例は、3.6 項を参照してください。

oracle.ord.media.annotator.parsers.Parser クラス

この項では、Parser クラスのコンストラクタおよびメソッドに関するリファレンス情報を示します。このクラスはすべてのパーサーのベース・クラスです。独自のパーサーを作成するには、このクラスを拡張する必要があります。Parser クラスは、メタデータの解析、サンプルの抽出、アノテーションへのメタデータの保存など、パーサーに求められる機能を定義する抽象クラスです。Parser クラスのすべてのサブクラスに、これらの機能を提供するメソッドを実装する必要があります。

Parser クラスは、解析処理中オブジェクトの読込みに使用される DataInputStream オブジェクトの基礎となるラッパーに基づいて動作します。Parser オブジェクトは、パーサーによってストリームで検出されるメタデータを移入するアノテーション・インスタンスに対応付けられます。また、メディア・データの解析に関連する処理過程情報を提供する、AnnTaskManager クラスのインスタンスに対応付けられます。さらに、対応付けられたアノテーション・インスタンスの副アノテーションを作成するため、AnnotationFactory オブジェクトに対応付けられます。

このクラスは java.lang.Object を拡張します。次のフィールドを含みます。

- protected AnnotationFactory m_annFactory

  副アノテーションを作成するために使用される AnnotationFactory オブジェクトです。

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

  パーサーで処理されるアノテーションです。

- protected AnnTaskManager m_annTaskMan

  処理過程情報を生成するために使用される AnnTaskManager オブジェクトです。

- protected boolean m_bExtractable

  パーサーがアノテーションからサンプルを抽出できるかどうかを判断します。デフォルトは false です。

- protected MADataInputStream m_madisResource

  処理されるメディア・ソースです。

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

  パーサー記述子の XML ファイルのメモリー内表現です。

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

  アプリケーションの追跡情報の生成に使用される Status オブジェクトです。

Parser コンストラクタ

構文

```
public Parser( )
```

説明

新しい Parser オブジェクトを作成します。Parser クラスのすべてのサブクラスに、空のコンストラクタが存在する必要があります。

パラメータ

なし

戻り値

なし

例外

なし

例

なし。ユーザーがこのコンストラクタを直接コールすることはできません。

extractSamples( )

構文

```
public abstract void extractSamples( )
```

説明

現行のメディア・ソースからサンプルを抽出し、そのサンプルを現行のアノテーション・インスタンスに設定します。AnnotatorEngine オブジェクトは、m_madisResource フィールドに setSource( ) メソッドを、m_annInst フィールドに setAnnotation( ) メソッドを自動的に設定します。

パラメータ

なし

戻り値

なし

例外

ParserException

詳細は、*inter*Media Annotator の Javadoc を参照してください。

例

このメソッドの例は、6.9 項を参照してください。

parse( )

構文

```
public abstract void parse( )
```

説明

ソースを解析し、メタデータを抽出します。AnnotatorEngine オブジェクトは、m_madisResource フィールドに setSource( ) メソッドを、m_annInst フィールドに setAnnotation( ) メソッドを自動的に設定します。

このメソッドを実行した後、saveToAnnotation( ) メソッドをコールしてアノテーションにメタデータを設定する必要があります。

パラメータ

なし

戻り値

なし

例外

ParserException

詳細は、*inter*Media Annotator の Javadoc を参照してください。

例

このメソッドの例は、6.7 項を参照してください。

saveToAnnotation( )

構文

```
public abstract void saveToAnnotation( )
```

説明

抽出されたメタデータを m_annInst フィールドのアノテーションに設定します。アノテーションは、解析の前に setAnnotation( ) メソッドで自動的に設定されます。

このメソッドは、メディア・ソースの解析の直後にコールする必要があります。これによって、解析が正常に終了した場合にのみアノテーションが変更されることを保証できます。

パラメータ

なし

戻り値

なし

例外

なし

例

このメソッドの例については、6.7 項を参照してください。

oracle.ord.media.annotator.utils.MADataInputStream クラス

この項では、MADataInputStream クラスのコンストラクタおよびメソッドに関するリファレンス情報を示します。このクラスは、入力ストリームから次の型を読み込むメソッドを提供します。

- 16 ビットの固定小数点数
- 32 ビットの固定小数点数
- 80 ビットの拡張浮動小数点数
- Audio Video Interleaved（AVI）言語コード
- 日付（整数および書式化された文字列の両方）
- 4 文字コード（FourCC）
- 整数（ビッグエンディアンおよびリトルエンディアンの両方）
- LONG（ビッグエンディアンおよびリトルエンディアンの両方）
- Pascal 文字列（可変サイズおよび固定サイズの両方）
- QuickTime 言語コード
- SHORT（ビッグエンディアンおよびリトルエンディアンの両方）
- 符号なし整数（ビッグエンディアンおよびリトルエンディアンの両方）
- 符号なし LONG（ビッグエンディアンおよびリトルエンディアンの両方）
- 符号なし SHORT（ビッグエンディアンおよびリトルエンディアンの両方）

このクラスは java.lang.Object を拡張します。

MADataInputStream(InputStream, boolean, String, String) コンストラクタ

構文

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

説明

MADataInputStream オブジェクトを作成し、後で使用できるように入力ストリーム引数を保存します。このコンストラクタは、データがリトルエンディアン形式かビッグエンディアン形式のいずれであるかを設定します。入力ストリームの整数（SHORT、整数、LONG）には、指定されたエンディアン形式があります。

パラメータ

is
基礎となる入力ストリーム

bLittleEndian
データがリトルエンディアン・フォーマットかビッグエンディアン・フォーマットのいずれであるかを示します。true は、データがリトルエンディアン形式であることを示します。

szCharEncoding
入力文字列の読込みに使用するキャラクタ・エンコーディング

mimetype
入力ストリームの MIME タイプ

戻り値

なし

例外

oracle.ord.media.annotator.AnnotatorException

例

なし。ユーザーがこのコンストラクタを直接コールすることはできません。

MADataInputStream(MADataInputStream, boolean, String, String) コンストラクタ

構文

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

説明

MADataInputStream オブジェクトを作成し、後で使用できるように入力ストリーム引数を保存します。このコンストラクタは、データがリトルエンディアン形式かビッグエンディアン形式のいずれであるかを設定します。入力ストリームの整数（SHORT、整数、LONG）には、指定されたエンディアン形式があります。

パラメータ

is
基礎となる入力ストリーム

bLittleEndian
データがリトルエンディアン形式かビッグエンディアン形式のいずれであるかを示します。true は、データがリトルエンディアン形式であることを示します。

szCharEncoding
入力文字列の読込みに使用するキャラクタ・エンコーディング

mimetype
入力ストリームの MIME タイプ

戻り値

なし

例外

なし

例

なし。ユーザーがこのコンストラクタを直接コールすることはできません。

available( )

構文

```
public int available( )
```

説明

この入力ストリームに対するメソッドの次のコール元によってブロックされることなく読込みまたはスキップできるバイト数を戻します。

パラメータ

なし

戻り値

メソッドの次のコール元によってブロックされることなく読込みまたはスキップできるバイト数を戻します。

例外

java.io.IOException

例

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

close( )

構文

```
public void close( )
```

説明

入力ストリームをクローズし、ストリームに対応付けられているすべてのシステム・リソースを解放します。

パラメータ

なし

戻り値

なし

例外

なし

例

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

isLittleEndian( )

構文

```
public boolean isLittleEndian( )
```

説明

入力ストリームがリトルエンディアン形式のデータを読み込むことができるかどうかを確認します。

パラメータ

なし

戻り値

リトルエンディアン・データを読み込むことができる場合は true、そうでない場合は false を戻します。

例外

なし

例

このメソッドの例は、7-71 ページの「setLittleEndian( )」メソッドを参照してください。

mark( )

構文

```
public void mark(int readLimit)
```

説明

入力ストリームでの現在の位置をマークします。次に reset( ) メソッドがコールされると、最後にマークされた位置にストリームが戻ります。

パラメータ

readLimit
マーク位置が無効になる前に読込みが可能なバイトの最大数

戻り値

なし

例外

なし

例

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

read(byte[ ])

構文

```
public final int read(byte[ ] b)
```

説明

入力ストリームから数バイトを読み込み、バッファ配列 b に格納します。実際に読み込んだバイト数は整数で戻します。データの一部が常にバッファに読み込まれるように、入力データが有効になるか、ファイルの終了が検出されるか、または例外が発生するまで、このメソッドはメソッドの次のコール元をブロックします。

b が NULL の場合は、NullPointerException が発生します。b の長さが 0 の場合は、読み込むバイトがないため 0 を戻します。そうでない場合は、1 バイト以上の読込みを試みます。ストリームでファイルが終了し、読み込むバイトがない場合は、-1 という値を戻します。そうでない場合は、1 バイト以上を読み込み、b に格納します。

パラメータ

b
読み込んだデータを格納するバッファ

戻り値

読み込んだバイト数を戻します。

例外

java.io.IOException

例

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

read(byte[ ], int, int)

構文

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

説明

入力ストリームからバイト配列にデータのバイト（len の値まで）を読み込みます。可能なかぎり多くのバイト数の読込みを試みますが、読み込んだバイト数が小さい場合もあります（0（ゼロ）の場合もあります）。実際に読み込んだバイト数は、整数で戻します。

len が 0（ゼロ）の場合は、読み込むバイトがないため 0（ゼロ）を戻します。そうでない場合は、1 バイト以上の読込みを試みます。ストリームでファイルが終了し、読み込むバイトがない場合は、-1 という値を戻します。そうでない場合は、1 バイト以上を読み込み、配列に格納します。

データの一部が常にバッファに読み込まれるように、入力データが有効になるか、ファイルの終了が検出されるか、または例外が発生するまで、このメソッドは次のメソッドのコール側をブロックします。

b が NULL の場合は、NullPointerException が発生します。off の値が負の場合、len が負の場合または off+len が配列 b の長さより大きい場合は、IndexOutOfBoundsException が発生します。ファイルの終了以外の理由で最初のバイトの読込みができない場合は、IOException が発生します。IOException は、特に、入力ストリームがクローズした場合に発生します。

パラメータ

b
読み込んだデータを格納するバッファ

off
データが読み込まれるバッファの始まりからのオフセット

len
読み込むバイトの最大数

戻り値

読み込んだバイト数を戻します。

例外

java.io.IOException

例

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

readAVILanguage( )

構文

```
public AVILanguage readAVILanguage( )
```

説明

基礎となる入力ストリームの次の AVI 言語コードを読み込みます。

AVILanguage クラスの詳細は、*inter*Media Annotator の Javadoc を参照してください。

パラメータ

なし

戻り値

入力ストリームから読み込んだ AVI 言語コードを戻します。

例外

java.io.IOException

例

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

readByte( )

構文

```
public byte readByte( )
```

説明

入力ストリームから次のバイトを読み込みます。

パラメータ

なし

戻り値

入力ストリームから読み込んだバイトを戻します。

例外

java.io.IOException

例

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

readByteArray(byte[ ],int)

構文

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

説明

指定されたバイト数を、基礎となるデータ入力ストリームから指定されたバイト配列に読み込みます。

パラメータ

bBuffer
読み込んだデータを格納するバイト配列

iNumBytesToRead
読み込むバイト数

戻り値

読み込んだバイト数を戻します。

例外

java.io.IOException

例

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

readByteArray(int)

構文

```
public byte[ ] readByteArray(int iNumBytesToRead)
```

説明

指定されたバイト数を、基礎となるデータ入力ストリームからバイト配列に読み込みます。

パラメータ

iNumBytesToRead
読み込むバイト数

戻り値

読み込んだバイト配列を戻します。

例外

java.io.IOException

例

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

readColor48( )

構文

```
public long readColor48( )
```

説明

データ・ストリームから 6 バイトを読み込み、RGB 形式でカラーを表現する LONG 型の値を戻します。

これは QuickTime ファイル・フォーマットに使用されます。

パラメータ

なし

戻り値

RGB 形式のカラーの値を LONG 型で戻します。

例外

java.io.IOException

例

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

readDate( )

構文

```
public java.util.Date readDate( )
```

説明

次の java.util.Date オブジェクトを基礎となる入力ストリームから読み込みます。

パラメータ

なし

戻り値

入力ストリームから読み込んだ java.util.Date オブジェクトを戻します。

例外

java.io.IOException

例

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

readDate(int, String)

構文

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

説明

指定された名前のパターンを使用して、データ・ストリームから java.util.Date オブジェクトとして読み込んだバイトを戻します。

パラメータ

iLen
読み込むバイト数

szPattern
java.text.SimpleDateFormat の仕様に従ったバイトの日付パターン

戻り値

読み込んだ java.util.Date オブジェクトを戻します。

例外

java.io.IOException

例

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

readExtended( )

構文

```
public Extended readExtended( )
```

説明

基礎となる入力ストリームで次の 80 ビットの拡張浮動小数点数を読み込みます。

Extended クラスの詳細は、*inter*Media Annotator の Javadoc を参照してください。

パラメータ

なし

戻り値

入力ストリームから読み込んだ 80 ビットの拡張浮動小数点数を戻します。

例外

java.io.IOException

例

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

readFixedPoint16( )

構文

```
public FixedPoint16 readFixedPoint16( )
```

説明

基礎となる入力ストリームで次の 16 ビットの固定小数点数を読み込みます。

FixedPoint16 クラスの詳細は、*inter*Media Annotator の Javadoc を参照してください。

パラメータ

なし

戻り値

入力ストリームから読み込んだ 16 ビットの固定小数点数を戻します。

例外

java.io.IOException

例

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

readFixedPoint32( )

構文

```
public FixedPoint32 readFixedPoint32( )
```

説明

基礎となる入力ストリームで次の 32 ビットの固定小数点数を読み込みます。

FixedPoint32 クラスの詳細は、*inter*Media Annotator の Javadoc を参照してください。

パラメータ

なし

戻り値

入力ストリームから読み込んだ 32 ビットの固定小数点数を戻します。

例外

java.io.IOException

例

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

readFourCC( )

構文

```
public FourCC readFourCC( )
```

説明

基礎となる入力ストリームで次の 4 文字コードを読み込みます。

FourCC クラスの詳細は、*inter*Media Annotator の Javadoc を参照してください。

パラメータ

なし

戻り値

入力ストリームから読み込んだ 4 文字コードを戻します。

例外

java.io.IOException

例

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

readInt( )

構文

```
public int readInt( )
```

説明

次の整数を入力ストリームから読み込みます。

パラメータ

なし

戻り値

入力ストリームから読み込んだ整数を戻します。

例外

なし

例

このメソッドの例は、7-43 ページの「mark( )」メソッドを参照してください。

readLong( )

構文

```
public long readLong( )
```

説明

基礎となる入力ストリームから次の LONG 型を読み込みます。

パラメータ

なし

戻り値

入力ストリームから読み込んだ LONG 型の値を戻します。

例外

java.io.IOException

例

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

readPascalString( )

構文

```
public java.lang.String readPascalString( )
```

説明

次の Pascal 文字列を基礎となる入力ストリームから読み込み、Pascal 文字列の内容を Java String オブジェクトとして戻します。

パラメータ

なし

戻り値

Pascal 文字列の内容を Java String オブジェクトとして戻します。

例外

java.io.IOException

例

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

readPascalString(int)

構文

```
public java.lang.String readPascalString(int iNumBytesToRead)
```

説明

次の Pascal 文字列を基礎となる入力ストリームから読み込み、Pascal 文字列の内容を Java String オブジェクトとして戻します。

パラメータ

iNumBytesToRead
入力ストリームから読み込むバイト数

戻り値

Pascal 文字列の内容を Java String オブジェクトとして戻します。

例外

java.io.IOException

例

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

readPascalString(Short)

構文

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

説明

次の拡張 Pascal 文字列を基礎となる入力ストリームから読み込み、Pascal 文字列の内容を Java String オブジェクトとして戻します。

拡張 Pascal 文字列は、最初に文字列の長さ（8 ビット、16 ビットまたは 32 ビット）が設定され、次に文字列の内容が続く Pascal 文字列です。長さは 8、16 または 32 ビットである必要があります。

パラメータ

sLengthSize
文字列のビット数。8、16 または 32 である必要があります。

戻り値

Pascal 文字列の内容を Java String オブジェクトとして戻します。

例外

java.io.IOException

例

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

readQTLanguage( )

構文

```
public QTLanguage readQTLanguage( )
```

説明

次の QuickTime 言語コードを基礎となる入力ストリームから読み込みます。

QTLanguage オブジェクトの詳細は、*inter*Media Annotator の Javadoc を参照してください。

パラメータ

なし

戻り値

入力ストリームから読み込んだ QuickTime 言語コードを戻します。

例外

java.io.IOException

例

```
QTLanguage code = m_madisResource.readQTLanguage();
```

readRectangle( )

構文

```
public Rectangle readRectangle( )
```

説明

入力ストリームから、四角形の座標として解析される 8 バイトのデータを読み込みます。これは QuickTime および RIFF データ・フォーマットに使用されます。

パラメータ

なし

戻り値

入力ストリームから読み込んだ 8 バイトを Rectangle オブジェクトとして戻します。

例外

java.io.IOException

例

```
Rectangle size = m_madisResource.readRectangle();
```

readShort( )

構文

```
public short readShort( )
```

説明

基礎となる入力ストリームから次の SHORT 型を読み込みます。

パラメータ

なし

戻り値

入力ストリームから読み込んだ SHORT 型の値を戻します。

例外

なし

例

```
short number = m_madisResource.readShort();
```

readString( )

構文

```
public java.lang.String readString(int iNumBytesToRead)
```

説明

指定されたバイト数を基礎となる入力ストリームから読み込み、文字列としてフォーマットします。

パラメータ

iNumBytesToRead
読み込むバイト数

戻り値

入力ストリームから読み込んだ文字列を戻します。

例外

java.io.IOException

例

```
String info = m_madisResource.readString(24);
```

readUnsignedByte( )

構文

```
public int readUnsignedByte( )
```

説明

基礎となる入力ストリームから次の符号なしバイトを読み込みます。

パラメータ

なし

戻り値

入力ストリームから読み込んだ符号なしバイトを戻します。

例外

java.io.IOException

例

```
int number = m_madisResource.readUnsignedByte();
```

readUnsignedInt( )

構文

```
public long readUnsignedInt( )
```

説明

基礎となる入力ストリームから次の符号なし整数を読み込みます。

パラメータ

なし

戻り値

入力ストリームから読み込んだ符号なし整数を戻します。

例外

java.io.IOException

例

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

readUnsignedShort( )

構文

```
public int readUnsignedShort( )
```

説明

基礎となる入力ストリームから次の符号なし SHORT を読み込みます。

パラメータ

なし

戻り値

入力ストリームから読み込んだ SHORT を戻します。

例外

java.io.IOException

例

```
int number = m_madisResource.readUnsignedShort();
```

reset( )

構文

```
public void reset( )
```

説明

最後にマークされた位置に基礎となる入力ストリームを戻します。

パラメータ

なし

戻り値

なし

例外

java.io.IOException

例

このメソッドの例は、7-43 ページの「mark( )」メソッドを参照してください。

setLittleEndian( )

構文

```
public void setLittleEndian(boolean bLittleEndian)
```

説明

リトルエンディアン形式で格納されたデータを入力ストリームが読み込むことができるかどうかを設定します。

パラメータ

bLittleEndian
リトルエンディアン形式で格納されたデータを入力ストリームが読み込むことができるかどうかを判断します。

戻り値

なし

例外

なし

例

```
if(m_madisResource.isLittleEndian())
     m_madisResource.setLittleEndian(false);
```

skipBytes(int)

構文

```
public int skipBytes(int iNum)
```

説明

指定されたバイト数を基礎となる入力ストリームでスキップします。

パラメータ

iNum
スキップするバイト数

戻り値

スキップしたバイト数を戻します。

例外

java.io.IOException

例

このメソッドの例は、7-43 ページの「mark( )」メソッドを参照してください。

skipBytes(long)

構文

```
public long skipBytes(long iNum)
```

説明

指定されたバイト数を基礎となる入力ストリームでスキップします。

パラメータ

iNum
スキップするバイト数

戻り値

スキップしたバイト数を戻します。

例外

java.io.IOException

例

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

8

新しいアノテーション型の作成

アプリケーションの要件をより適切に満たすために、提供されているアノテーション型に加えて、Oracle *inter*Media Annotator を使用して独自のアノテーション型を作成することができます。たとえば、オンライン販売データベースの所有者は、メディア・データおよび抽出されたメタデータの他に、在庫および価格の情報を含むアノテーションを定義できます。

ユーザー定義のアノテーション型の例は、Oracle *inter*Media Annotator のインストール時に提供される次のファイルを参照してください。

- UNIX の場合 :

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

- Windows の場合 :

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

8.1 新しいアノテーション型の作成

新しいアノテーション型は、XML ファイルに定義します。XML ファイルは、AnnotationDescriptor.dtd ファイルに定義されている、AnnotationDescriptor の Document Type Definition（DTD）に準拠する必要があります。このファイルは、annotations サブディレクトリに格納されています。このディレクトリの詳細は、次のとおりです。

- UNIX の場合 :

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

- Windows の場合 :

  ORACLE_HOME¥ord¥Annotator¥lib¥descriptors¥annotations

AnnotationDescriptor.dtd ファイルには、AnnotationProperties および AttributeDescriptors の 2 つの要素を含む、AnnotationDescriptor DTD が記述されています。

8.1.1 AnnotationProperties 要素

AnnotationProperties 要素は、アノテーション全体の情報を提供する要素を含みます。含まれる要素は次のとおりです。

- Name: 必須要素。新しいアノテーション型の名前を含みます。
- Version: 必須要素。バージョン番号を含みます。
- Description: オプション要素。アノテーション全体の概要を含みます。
- Extends: オプション要素。作成した新しいアノテーション型が拡張する、別のアノテーション型の名前を含みます。作成した新しいアノテーション型には、追加で定義するすべての属性の他に、指定されたアノテーション型から継承されるすべての属性の定義が含まれます。

  ただし、既存のアノテーション型から継承された属性は上書きできません。新しい属性の作成のみ実行できます。別のアノテーション型に関連しないアノテーション型を作成する場合は、Extends 要素を含めないでください。
- Contains: 予約要素。この要素には値を割り当てないでください。
- ClassName: 予約要素。この要素には値を割り当てないでください。
- IconFileName: 予約要素。この要素には値を割り当てないでください。

8.1.2 AttributeDescriptors 要素

AttributeDescriptors 要素には、1 つ以上の AttributeDescriptor 要素が含まれます。AttributeDescriptor 要素には、1 つの AttributeProperties 要素が含まれます。

AttributeProperties 要素には、作成した新しいアノテーション型の特定の属性に関する情報を提供する要素が含まれます。含まれる要素は次のとおりです。

- AttributeName: 必須要素。新しい属性の名前を含みます。
- AttributeType: 必須要素。属性値の Java オブジェクト型を含みます。AttributeType は Java オブジェクト型である必要があります。XML 文書では Java 基本型は使用できません。たとえば、整数を使用する場合は、int ではなく、java.lang.Integer を使用します。

  Java オブジェクト型が、public String toString( ) および public static *Object* valueOf(String)（ここで *Object* は Java オブジェクト型）の 2 つの有効なメソッドを定義するかぎり、ほぼすべての Java オブジェクト型を AttributeType として使用できます。public String toString( ) メソッドはオブジェクトの内容を有効な文字列として戻し、public static Object valueOf(String) メソッドは指定された文字列の内容を *Object* 型の有効なオブジェクトとして戻します。

  java.util.Date クラスは特別なケースで、オブジェクトの内容の文字列表現を提供するために、前述のメソッドを使用しません。かわりに、AttributeTypePattern 要素を使用します。
- AttributeTypePattern: オプション要素。AttributeType が java.lang.Date の場合にのみ使用します。日付の表示に使用する必要がある文字列パターンを指定します。このパターンは、java.text.SimpleDateFormat の構文に従います。
- AttributeAlias: オプション要素。表示に使用する属性名の短縮形を定義します。
- AttributeDescription: オプション要素。属性の概要を定義します。
- AttributeDefaultValue: オプション要素。アノテーションに挿入する属性のデフォルト値を定義します

8.1.3 要素の階層

通常、XML 文書の構造は次のようにする必要があります。

```
<?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>
```

注意： XML ファイルは空白の有無を区別します。「java.lang.Double」は有効ですが、「java.lang.Double 」は無効です。エラーの原因となるため、XML ファイルに不要な空白を含めないようにしてください。

8.2 新しいアノテーション型の使用

XML ファイルの作成完了後、そのファイルを `annotations` ディレクトリに保存する必要があります。

作成した新しいアノテーション型は、事前定義されたアノテーション型と同じ方法で使用できます。新しいアノテーションの作成の詳細は、第 2 章および第 3 章を参照してください。

第 III 部

付録

第 III 部に含まれる付録は、次のとおりです。

A

格納されたアノテーションの問合せ

メディア・データおよびアノテーションが Oracle データベースにアップロードされると、Oracle Text を使用して、アノテーション（データベースに XML 形式で保存されている）への問合せができます。

次の PL/SQL コードの抜粋は、データベース表 VideoStorage に Oracle Text の索引を作成する方法の例です。このコードの抜粋は、セクション・グループに定義されたリスト・プリファレンスおよび XML タグとともに、VideoStorage 表の `vsrc` 列の comments フィールドに Oracle Text の索引を生成します。

```
-- 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');
```

次の例は、VideoStorage 表を作成する SQL 文を示します。

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

次の PL/SQL コードの抜粋は、VideoStorage 表を問い合せる方法の例です。

```
-- Perform a query
SELECT id, score(99)
FROM VideoStorage V
WHERE
     CONTAINS(V.vsrc.comments, '(John Doe) WITHIN MOVIECASTTAG',
     99) > 0;
```

前述の問合せは、対応付けられたアノテーションの MOVIE_CAST 属性に John Doe を含む、ビデオ・クリップのクリップ ID および関連スコア（Oracle Text が生成）を戻します。

前述の PL/SQL 文は、次のファイルに含まれています。

- UNIX の場合 :

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

- Windows の場合 :

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

詳細は、Oracle Text マニュアル（特に『Oracle Text アプリケーション開発者ガイド』）を参照してください。

B

サポートされているフォーマット

Oracle *inter*Media Annotator は、次のフォーマットをサポートします。

- AIFF/AIFC
- Apple QuickTime 5.0
- AU
- BMP
- GIF87a/GIF89a
- JPEG/JFIF
- MPEG I/II Audio（すべてのレイヤー）
- MPEG I Video
- Microsoft AVI
- RealMedia
- TIFF
- WAV

表 B-1 に、Oracle *inter*Media Annotator で組込みサポートされているメディア・ソース・フォーマット、およびそれに対応する組込みパーサーが提供する抽出機能（ある場合）を示します。

表 B-1 組込みパーサー

ファイル・フォーマット	MIME タイプ	抽出
AIFF	audio/x-aiff audio/x-aif	なし
Apple QuickTime 5.0	video/quicktime	テキスト・トラック ビデオ・フレーム（Apple QuickTime Library でのみ）
BMP	image/bmp	なし
GIF	image/gif	なし
JPEG/JFIF	image/jpeg image/jpg	なし
MPEG Audio Layer I/II	audio/x-mpeg	なし
MPEG I Video	video-mpeg	なし
RealMedia	video/x-realvideo video/x-realaudio	なし
RIFF	video/x-msvideo application/x-troff-msvideo audio/x-wav	なし
Sun Audio	audio/basic	なし
TIFF	image/tiff	なし

C

定義済のアノテーション特性

この付録の表では、各 Oracle *inter*Media Annotator について定義されているアノテーション特性を示します。

表 C-1 に、すべてのメディアに使用される MediaAnn アノテーション型の属性、および各属性のデータ型と説明を示します。

表 C-1 MediaAnn のアノテーション特性

属性	データ型	説明
MEDIA_AUTHORING_TOOL	java.lang.String	メディアの作成に使用されたオーサリング・ツール
MEDIA_BITRATE	java.lang.Long	メディアのビット・レート（ビット / 秒）
MEDIA_CATEGORY	java.lang.String	メディアのカテゴリまたはジャンル
MEDIA_CONTENT_DATE	java.lang.String	メディア・コンテンツの作成日
MEDIA_COPYRIGHT	java.lang.String	メディアの著作権情報
MEDIA_CREATION_TIME	java.util.Date	メディアの作成日時（たとえば、Mon Dec 13 19:29:04 JST 2001）
MEDIA_CREDITS	java.lang.String	コンテンツ・プロバイダのクレジット
MEDIA_DESCRIPTION	java.lang.String	メディアの説明
MEDIA_DURATION	oracle.ord.media.annotator.types.TimeCodeString	時 : 分 : 秒 : 仮数（ここで仮数とは、MEDIA_TIMESCALE で定義されるユニットでの秒の小数部分。例 : HH:MM:SS:FF）の書式で表されるメディアの継続時間
MEDIA_FORMAT_ENCODING	java.lang.String	メディア・エンコーディングのフォーマット

表 C-1 MediaAnn のアノテーション特性（続き）

属性	データ型	説明
MEDIA_FORMAT_ENCODING_CODE	java.lang.String	メディア・エンコーディングの短縮形（4 文字）
MEDIA_INFORMATION	java.lang.String	メディアの情報
MEDIA_LANGUAGE	java.lang.String	メディアの言語
MEDIA_MODIFICATION_TIME	java.util.Date	メディアの最終変更日時（たとえば、Mon Dec 13 19:29:04 JST 2001）
MEDIA_PRODUCER	java.lang.String	メディアの作成者
MEDIA_SIZE	java.lang.Long	メディア・ソースのサイズ（バイト）
MEDIA_SOURCE_DIRECTORY	java.lang.String	ソースの格納ディレクトリ
MEDIA_SOURCE_FILE_FORMAT	java.lang.String	メディアのファイル・フォーマット
MEDIA_SOURCE_FILE_FORMAT_CODE	java.lang.String	メディアのファイル・フォーマットの短縮形（4 文字）
MEDIA_SOURCE_FILENAME	java.lang.String	ソースのファイル名
MEDIA_SOURCE_MIME_TYPE	java.lang.String	メディアおよびそのサンプルの MIME タイプ
MEDIA_SOURCE_PROTOCOL	java.lang.String	解析済メディア・ソースの URL プロトコル
MEDIA_SOURCE_STREAMABLE	java.lang.String	メディアを最適化する対象のストリーム・サーバー（存在する場合）
MEDIA_SOURCE_URL	java.lang.String	解析済メディア・ソースの場所または URL
MEDIA_TIMESCALE	java.lang.Float	1 秒間のユニット数
MEDIA_TITLE	java.lang.String	メディアのタイトル
MEDIA_TRACK_ID	java.lang.Integer	アノテーションのトラック識別子
MEDIA_USER_DATA	java.lang.String	その他のユーザー・データを含む文字列

表 C-2 に、オーディオ・メディアに使用される AudioAnn アノテーション型の属性、および各属性のデータ型と説明を示します。AudioAnn アノテーション型は、MediaAnn を拡張します。

表 C-2 AudioAnn のアノテーション特性

属性	データ型	説明
AUDIO_ARTIST	java.lang.String	オーディオ・クリップの主なアーティスト
AUDIO_BITS_PER_SAMPLE	java.lang.Integer	1 音声サンプル当たりのビット数
AUDIO_NUM_CHANNELS	java.lang.Integer	オーディオ・チャネル数
AUDIO_SAMPLE_RATE	java.lang.Integer	オーディオのサンプル・レート（サンプル数 / 秒）

表 C-3 に、ビデオ・トラックのアノテーションに使用される VideoAnn アノテーション型の属性、および各属性のデータ型と説明を示します。VideoAnn アノテーション型は、MediaAnn を拡張します。

表 C-3 VideoAnn のアノテーション特性

属性	データ型	説明
VIDEO_DEPTH	java.lang.Integer	色の深度のビット数
VIDEO_FRAME_RATE	java.lang.Long	ビデオのフレーム・レート（フレーム数 / 秒）
VIDEO_FRAME_SIZE	java.lang.Long	ビデオ・フレームのサイズ（バイト）
VIDEO_HORIZONTAL_RES	java.lang.Integer	水平解像度（ピクセル / インチ）
VIDEO_IS_GRAYSCALE	java.lang.Boolean	ビデオがカラーかどうか
VIDEO_SRC_HEIGHT	java.lang.Long	ビデオの高さ（ピクセル）
VIDEO_SRC_WIDTH	java.lang.Long	ビデオの幅（ピクセル）
VIDEO_VERTICAL_RES	java.lang.Integer	垂直解像度（ピクセル / インチ）

表 C-4 に、ムービー・テキスト・トラックまたは時間ベースのテキストに使用される TextAnn アノテーション型の属性、および各属性のデータ型と説明を示します。TextAnn アノテーション型は、MediaAnn を拡張します。

表 C-4 TextAnn のアノテーション特性

属性	データ型	説明
TEXT_ALIGN	java.lang.String	テキスト・トラックの位置合せ（左揃え、中央揃え、右揃えまたは両端揃え）
TEXT_BG_COLOR	java.lang.String	テキスト・トラックのバックグラウンド・カラー（たとえば、0x00RRGGBB）
TEXT_DEF_BOX	oracle.ord.media.annotator.types.Rectangle	Java プリミティブ型 short の 4 つのインスタンスで構成されるデフォルトのテキスト・ボックスのサイズ
TEXT_FG_COLOR	java.lang.String	テキスト・トラックのフォアグラウンド・カラー（たとえば、0x00RRGGBB）
TEXT_FONTFACE	java.lang.String	使用されるフォント・スタイル（イタリック体やボールド体など）
TEXT_FONTNAME	java.lang.String	使用されるフォントの名前
TEXT_FONTSIZE	java.lang.Short	テキスト・トラックのポイント・サイズ

表 C-5 に、ムービー・メディアに使用される MovieAnn アノテーション型の属性、および各属性のデータ型と説明を示します。MovieAnn アノテーション型は、MediaAnn を拡張します。

表 C-5 MovieAnn のアノテーション特性

属性	データ型	説明
MOVIE_CAST	java.lang.String	映画の出演者の名前
MOVIE_DIRECTOR	java.lang.String	映画の監督
MOVIE_EDIT_INFORMATION	java.lang.String	編集情報
MOVIE_WARNING	java.lang.String	映画のレーティング情報および警告情報

表 C-6 に、イメージ・トラックに使用される ImageAnn アノテーション型の属性および各属性の説明を示します。ImageAnn アノテーション型は、MediaAnn を拡張します。

表 C-6 ImageAnn のアノテーション特性

属性	データ型	説明
IMAGE_BITS_PER_PIXEL	java.lang.Integer	イメージの 1 ピクセル当たりのビット数
IMAGE_COUNT	java.lang.Integer	ファイルに格納されたイメージの数
IMAGE_HEIGHT	java.lang.Long	イメージの高さ
IMAGE_HORIZONTAL_RES	java.lang.Double	水平解像度（ピクセル / インチ）
IMAGE_PIXEL_FORMAT	java.lang.String	イメージのカラー領域（解像度を含む）
MEDIA_TIMESCALE	java.lang.Float	1 秒間のユニット数
IMAGE_VERTICAL_RES	java.lang.Double	垂直解像度（ピクセル / インチ）
IMAGE_WIDTH	java.lang.Long	イメージの幅

表 C-7 に、Information Interchange Model（IIM）形式に使用される IptcIimAnn アノテーション型の属性、および各属性のデータ型と説明を示します。IIM 形式は、新しい情報用のコンテナ・ファイル・フォーマットで、国際新聞通信委員会（International Press Telecommunications Council: IPTC）によって開発されました。

表 C-7 IptcIimAnn のアノテーション特性

属性	データ型	説明
IIM_ACTION_ADVISED	java.lang.Long	オブジェクトの表示後必要な処理。有効な値とその意味は、次のとおりです。 01: 破棄 02: 置換 03: 追加
IIM_BYLINE	java.lang.String	オブジェクトの作成者
IIM_BYLINE_TITLE	java.lang.String	オブジェクト作成者の肩書き
IIM_CAPTION	java.lang.String	特にオブジェクトがテキストではない場合に使用されるオブジェクトの注釈または要約
IIM_CITY	java.lang.String	オブジェクトが作成された都市
IIM_CONTACT	java.lang.String	詳細情報の問合せ先
IIM_COPYRIGHT	java.lang.String	著作権情報

表 C-7 IptclimAnn のアノテーション特性（続き）

属性	データ型	説明
IIM_COUNTRY_CODE	java.lang.String	オブジェクトが作成された国または地理的場所の ID
IIM_CREATION_DATE	java.util.Date	物理オブジェクトの作成日。Date Created レコードおよび Time Created レコードのセットで構成されます。
IIM_CREDIT	java.lang.String	オブジェクトを送信するサービスの名前
IIM_DIGITAL_CREATION_DATE	java.util.Date	Digital Creation Date レコードと Digital Creation Time レコードのセットで構成される、デジタル・バージョンの作成日
IIM_HEADLINE	java.lang.String	オブジェクトの詳細検索を行うためのキーワードを示すオブジェクトの内容の概要
IIM_IMAGE_TYPE	java.lang.String	イメージ・タイプを表すコード（2 バイトで構成） 最初のバイトの可能な値およびその意味は、次のとおりです。 0: オブジェクトなし 1: モノクロ 2、3 または 4: カラー・プロジェクトの複数色素 9: 補足データが含まれているオブジェクト・データ 2 つ目のバイトの可能な値およびその意味は、次のとおりです。 W: モノクロ Y: 黄色の色素 M: マゼンタの色素 C: シアンの色素 K: 黒の色素 R: 赤の色素 G: 緑の色素 B: 青の色素 T: テキストのみ F: フル・カラー（フレーム順序） L: フル・カラー（ライン順序） P: フル・カラー（ピクセル順序） S: フル・カラー（特別な Interleaving）
IIM_KEYWORDS	java.lang.String	オブジェクトに対応付けられたキーワード（IIM のバージョン 2 でサポート）

表 C-7 IptclimAnn のアノテーション特性（続き）

属性	データ型	説明
IIM_LANGUAGE	java.lang.String	主要言語の ID
IIM_LOCATION_NAME	java.lang.String	オブジェクトが作成された国のフルネーム
IIM_OBJECT_CYCLE	java.lang.Character	1 日の一部。有効な値は、次のとおりです。 a: A.M.（午前） p: P.M.（午後および夜間） b: 午前および午後
IIM_OBJECT_NAME	java.lang.String	オブジェクトのタイトル
IIM_ORIGINATING_PROG	java.lang.String	オブジェクトの作成に使用するソフトウェア
IIM_PROGRAM_VERSION	java.lang.String	オブジェクトの作成に使用するソフトウェアのバージョン
IIM_PROVINCE_STATE	java.lang.String	オブジェクトが作成された州の ID
IIM_RECORD_VERSION	java.lang.Long	IPTC IIM 仕様のバージョン
IIM_SOURCE	java.lang.String	オブジェクトの所有者
IIM_SPECIAL_INSTRUCTION	java.lang.String	オブジェクトの使用に関する特別な指示（禁止、警告など）
IIM_SUB_LOCATION	java.lang.String	オブジェクトが作成された都市内の場所の ID
IIM_TRANSMISSION_REF	java.lang.String	オリジナルの転送場所を表すコード（BER-5、PAR-12-11-01 など）
IIM_WRITER	java.lang.String	著者または編集者

表 C-8 に、メディア・サンプルに使用される SampleAnn アノテーション型の属性、および各属性のデータ型と説明を示します。SampleAnn アノテーション型は、MediaAnn を拡張します。

表 C-8 SampleAnn のアノテーション特性

属性	データ型	説明
SAMPLE_TIMESTAMP	oracle.ord.media.annotator.types.TimeCodeString	時 : 分 : 秒 : 仮数（ここで仮数とは、MEDIA_TIMESCALE で定義されるユニットでの秒の小数部分。例 : HH:MM:SS:FF）の書式で表される、指定されたサンプルのタイム・スタンプ

表 C-9 に、ムービー・テキスト・トラックのサンプルまたは時間ベースのテキストに使用される TextSampleAnn アノテーション型の属性、および各属性のデータ型と説明を示します。TextSampleAnn アノテーション型は、SampleAnn を拡張します。

表 C-9 TextSampleAnn のアノテーション特性

属性	データ型	説明
TEXTSAMPLE_VALUE	java.lang.String	テキスト・サンプルの文字列値

表 C-10 に、サンプルとしてビデオ・トラックから抽出されたビデオ・フレームに使用される VideoFrameSampleAnn アノテーション型の属性、および各属性のデータ型と説明を示します。VideoFrameSampleAnn アノテーション型は、SampleAnn を拡張します。

表 C-10 VideoFrameSampleAnn のアノテーション特性

属性	データ型	説明
VIDEO_FRAME_SAMPLE_HEIGHT	java.lang.Integer	ビデオ・トラックから抽出されたフレームの高さ
VIDEO_FRAME_SAMPLE_WIDTH	java.lang.Integer	ビデオ・トラックから抽出されたビデオ・フレームの幅

D

使用されなくなった機能

この付録では、Oracle *inter*Media Annotator から削除されたか、または使用されなくなった機能を示します。

D.1 削除された機能

Oracle *inter*Media Annotator から削除された機能は、次のとおりです。

- *inter*Media Annotator デモ・ユーティリティ（マニュアルを含む）

  デモ・ユーティリティは、Graphical User Interface（GUI）です。デモ・ユーティリティで使用可能だった機能は、Java API を使用して実行することができます。

- PL/SQL Upload Template Wizard

  任意のテキスト・エディタを使用して PL/SQL Upload Template を作成してください。

- オーディオ CD トラックのアノテーションのサポート。

  このサポートには、AudioCDAnn アノテーション型、AudioCDTrackAnn アノテーション型、`cd` URL プロトコルが含まれます。

D.2 使用されなくなった一般的機能

使用されなくなった機能は、次のとおりです。

- CDDB への接続。この機能は使用禁止になりました。

D.3 使用されなくなったメソッド

次の項に、使用されなくなったメソッドを示します。

D.3.1 MADataInputStream クラスで使用されなくなったメソッド

パーサー API の MADataInputStream クラスでは、次のコンストラクタおよびメソッドが使用されなくなりました。

- MADataInputStream(InputStream) コンストラクタ
- MADataInputStream(InputStream, boolean) コンストラクタ
- endBlock(FourCC fccChunk) メソッド
- getBytesRead( ) メソッド
- getLeft( ) メソッド
- startBlock(lSizeLeft) メソッド

E

FAQ

アノテーションに対応付けられた属性を確認する方法を教えてください。
次のディレクトリに、各アノテーション型の属性を定義している XML ファイルがあります。

- UNIX の場合 :

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

- Windows NT の場合 :

  ORACLE_HOME¥ord¥Annotator¥lib¥descriptors¥annotations

メディア・ファイルがデータベースにロードされない原因を教えてください。
次のいずれかの理由で、ファイルがロードされない可能性があります。

- データベース・サーバーが、PL/SQL Upload Template で指定されたディレクトリにアクセスできません。これは、そのディレクトリが存在しないか、またはそのディレクトリに対する読込み権限が付与されていないことを意味します。詳細は、5.1 項を参照してください。
- アップロードしているファイルに対する読込み権限が、データベース・サーバーに付与されていません。
- PL/SQL Upload Template 内の INSERT 文が不適切です。問題の詳細は、「Console」タブに表示される SQL エラーを参照してください。
- HTTP ストリームを介してメディア・ソースをインポートする場合、プロキシ設定によっては、エラーが発生する場合があります。Oracle データベースの UTL_HTTP パッケージが正しく構成されていることを確認してください。
- 指定された WHERE 句が結果を戻さないか、または複数の結果行を戻します。

属性値に索引を作成する方法を教えてください。
付録 A を参照してください。

ファイル拡張子とその MIME タイプ間のマッピングを変更する方法を教えてください。
次のディレクトリに格納されている mime.types ファイルを変更してください。

- UNIX の場合 : *$ORACLE_HOME*/ord/Annotator/lib/conf
- Windows の場合 : *ORACLE_HOME*¥ord¥Annotator¥lib¥conf

HTTP プロトコルを使用してメディア・ソースを解析中に、「Unsupported Annotation for Content Type text/htm」というエラーが発生する原因を教えてください。
リソースを指す URL のパスを確認してください。URL が無効である可能性があります。

起動設定を変更する方法を教えてください。
DOS バッチ・ファイルまたは UNIX シェル・スクリプトを十分に理解している場合は、起動スクリプトの先頭にある環境変数を変更できます。

Windows NT システム上で Annotator.bat を実行すると、「Could not load runtime library: d:´JRE´bin´javai.dll」というエラーが発生する原因を教えてください。
このエラーは、使用中の JDK と NT レジストリに一貫性がないために発生します。このエラーを回避するには、JDK を再インストールします。また、Annotator.bat 内のディレクトリ名も確認してください。

Oracle *inter*Media Annotator の最新情報を参照できる場所を教えてください。
最新情報、既知の問題および FAQ については、ONN-J（Oracle Technology Network Japan）を参照してください。

http://otn.oracle.co.jp

QuickTime では、スプライト・トラックおよびフラッシュ・トラックがサポートされていますか？
スプライト・トラックおよびフラッシュ・トラックはサポートされていません。

索引

A

C

D

E

G

H

I

J

M

O

P

Q

R

S

T

U

V

W

X

あ

い

え

た

つ

て

に

は

ひ

ふ

ほ

む

め

ゆ

よ

り

ろ