Oracle® *inter*Media

User's Guide and Reference

Release 9.0.1

June 2001

Part No. A88786-01

Oracle *inter*Media audio, document, image, and video is designed to manage Internet media content. *inter*Media is a standard feature, enabling Oracle9*i* to manage rich content, including text, documents, images, audio, video, and location information, in an integrated fashion with traditional business data.

ORACLE®

Oracle *inter*Media User's Guide and Reference, Release 9.0.1

Part No. A88786-01

Copyright © 1999, 2001, Oracle Corporation. All rights reserved.

Primary Author: Rod Ward

Contributors: Susan Mavris, Melli Annamalai, Todd Rowell, Raja Chatterjee, Robert Abbott, Albert Landeck, Vishal Rao, Dongbai Guo, Fengting Chen, Joseph Mauro, Rabah Mediouni, Sanjay Agarwal, Bill Voss, Susan Kotsovolos, Rosanne Toohey, Bill Beauregard, Susan Shepard, Helen Grembowicz

The Programs (which include both the software and documentation) contain proprietary information of Oracle Corporation; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. Oracle Corporation does not warrant that this document is error free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Oracle Corporation.

If the Programs are delivered to the U.S. Government or anyone licensing or using the programs on behalf of the U.S. Government, the following notice is applicable:

Restricted Rights Notice Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication, and disclosure of the Programs, including documentation, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, Programs delivered subject to the Federal Acquisition Regulations are "restricted computer software" and use, duplication, and disclosure of the Programs shall be subject to the restrictions in FAR 52.227-19, Commercial Computer Software - Restricted Rights (June, 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and Oracle Corporation disclaims liability for any damages caused by such use of the Programs.

Oracle and SQL*Plus are registered trademarks, and Oracle9*i* and PL/SQL are trademarks of Oracle Corporation. Other names may be trademarks of their respective owners.

Contents

Send Us Your Comments xxiii

Preface xxv

Audience xxv
Organization xxvi
Related Documents xxvii
Conventions xxvii
Changes to This Guide xxviii
Documentation Accessibility xxix

1 Introduction

1.1 Object Relational Technology 1-1
1.2 Multimedia Content Management 1-2
1.3 Audio Concepts 1-5
1.3.1 Digitized Audio 1-6
1.3.2 Audio Components 1-6
1.4 ORDDoc or Heterogeneous Media Data Concepts 1-6
1.4.1 Digitized Heterogeneous Media Data 1-7
1.4.2 Heterogeneous Media Data Components 1-7
1.5 Image Concepts 1-7
1.5.1 Digitized Images 1-8
1.5.2 Image Components 1-8
1.6 Video Concepts 1-9

1.6.1 Digitized Video ... 1-9
1.6.2 Video Components ... 1-9
1.7 Multimedia Object Types and Methods ... 1-10
1.8 Multimedia Storage ... 1-10
1.8.1 Storing Multimedia Data ... 1-11
1.8.2 Querying Multimedia Data ... 1-12
1.8.3 Accessing Multimedia Data ... 1-12
1.9 Extending Oracle *inter*Media ... 1-12
1.9.1 Supporting Other External Sources and Other Media Data Formats ... 1-13
1.9.2 Supporting Audio Data Processing ... 1-14
1.9.3 Supporting Video Data Processing ... 1-14
1.10 Relational Interface ... 1-15
1.11 Loading Multimedia Data into Oracle9*i* Using *inter*Media ... 1-15
1.12 Reading Data from a LOB ... 1-16
1.13 *inter*Media Architecture ... 1-17
1.13.1 Oracle *inter*Media Java Classes ... 1-21
1.13.2 Oracle *inter*Media Java Classes for Servlets and JSPs ... 1-22
1.13.3 Annotation Services for Multimedia Data ... 1-23
1.13.4 Streaming Content from an Oracle Database ... 1-24
1.13.5 Support for Web Technologies ... 1-25
1.13.6 Geocoding Services ... 1-25

2 Content-Based Retrieval Concepts

2.1 Overview and Benefits ... 2-1
2.2 How Content-Based Retrieval Works ... 2-2
2.2.1 Color ... 2-5
2.2.2 Texture ... 2-7
2.2.3 Shape ... 2-7
2.3 How Matching Works ... 2-8
2.3.1 Weight ... 2-8
2.3.2 Score ... 2-8
2.3.3 Similarity Calculation ... 2-9
2.3.4 Threshold Value ... 2-11
2.4 Using an Index to Compare Signatures ... 2-12
2.5 Preparing or Selecting Images for Useful Matching ... 2-13

3 *inter*Media Examples

3.1 Audio Data Examples 3-1
3.1.1 Defining a Song Object 3-2
3.1.2 Creating an Object Table SongsTable 3-2
3.1.3 Creating a List Object Containing a List of References to Songs 3-2
3.1.4 Defining the Implementation of the songList Object 3-3
3.1.5 Creating a CD Object and a CD Table 3-3
3.1.6 Inserting a Song into the SongsTable Table 3-4
3.1.7 Inserting a CD into the CdTable Table 3-5
3.1.8 Loading a Song into the SongsTable Table 3-5
3.1.9 Inserting a Reference to a Song Object into the Songs List in the CdTable Table 3-6
3.1.10 Adding a CD Reference to a Song 3-8
3.1.11 Retrieving Audio Data from a Song in a CD 3-8
3.1.12 Extending *inter*Media to Support a New Audio Data Format 3-9
3.1.13 Extending *inter*Media with a New Type 3-9
3.1.14 Using Audio Types with Object Views 3-10
3.1.15 Scripts for Creating and Populating an Audio Table from a BFILE Data Source 3-11
3.2 Media Data Examples 3-19
3.2.1 Defining a Media Object 3-20
3.2.2 Creating an Object Table DocumentsTable 3-21
3.2.3 Creating a List Object Containing a List of References to Media 3-21
3.2.4 Defining the Implementation of the documentList Object 3-21
3.2.5 Creating a Library Object and a Library Table 3-22
3.2.6 Inserting Media into the DocumentsTable Table 3-23
3.2.7 Inserting a Library into the LibraryTable Table 3-24
3.2.8 Loading Media into the DocumentsTable Table 3-24
3.2.9 Inserting a Reference to a Document Object into the Documents List in the LibraryTable Table 3-25
3.2.10 Adding a Library Reference to a Document 3-26
3.2.11 Extending *inter*Media to Support a New Media Data Format 3-27
3.2.12 Extending *inter*Media with a New Type 3-28
3.2.13 Using Document Types with Object Views 3-28
3.2.14 Using the ORDDoc Object Type as a Repository 3-29
3.2.15 Scripts for Creating and Populating a Media Table from a BFILE Data Source. 3-34

3.3 Image Data Examples 3-41
3.3.1 Adding Image Types to an Existing Table 3-42
3.3.2 Adding Image Types to a New Table 3-42
3.3.3 Inserting a Row Using BLOB Images 3-43
3.3.4 Populating a Row Using BLOB Images 3-44
3.3.5 Inserting a Row Using BFILE Images 3-45
3.3.6 Populating a Row Using BFILE Images 3-46
3.3.7 Querying a Row 3-46
3.3.8 Importing an Image from an External File into the Database 3-47
3.3.9 Retrieving an Image 3-48
3.3.10 Retrieving Images Similar to a Comparison Image (Content-Based Retrieval) .. 3-50
3.3.11 Creating a Domain Index 3-52
3.3.12 Retrieving Images Similar to a Comparison Image Using Index Operations (Indexed Content-Based Retrieval) 3-53
3.3.13 Copying an Image 3-53
3.3.14 Converting an Image Format 3-54
3.3.15 Copying and Converting in One Step 3-54
3.3.16 Extending *inter*Media with a New Type 3-55
3.3.17 Using Image Types with Object Views 3-57
3.3.18 Scripts for Creating and Populating an Image Table from a BFILE Data Source 3-59
3.3.19 Scripts for Populating an Image Table from an HTTP Data Source 3-66
3.3.20 Addressing Globalization Support Issues 3-68
3.4 Video Data Examples 3-69
3.4.1 Defining a Clip Object 3-70
3.4.2 Creating an Object Table ClipsTable 3-70
3.4.3 Creating a List Object Containing a List of Clips 3-71
3.4.4 Defining the Implementation of the clipList Object 3-71
3.4.5 Creating a Video Object and a Video Table 3-71
3.4.6 Inserting a Video Clip into the ClipsTable Table 3-72
3.4.7 Inserting a Row into the VideoTable Table 3-73
3.4.8 Loading a Video into the ClipsTable Table 3-73
3.4.9 Inserting a Reference to a Clip Object into the Clips List in the VideoTable Table 3-74
3.4.10 Inserting a Reference to a Video Object into the Clip 3-75
3.4.11 Retrieving a Video Clip from the VideoTable Table 3-76
3.4.12 Extending *inter*Media to Support a New Video Data Format 3-76

3.4.13 Extending *inter*Media with a New Object Type 3-77
3.4.14 Using Video Types with Object Views 3-77
3.4.15 Scripts for Creating and Populating a Video Table from a BFILE Data Source.. 3-79
3.5 Extending *inter*Media to Support a New Data Source 3-86

4 Ensuring Future Compatibility with Evolving *inter*Media Object Types

4.1 When and How to Call the Compatibility Initialization Function 4-1
compatibilityInit( ) 4-3

5 Common Methods for *inter*Media Object Types Reference Information

5.1 Important Notes 5-2
5.2 Methods 5-3
clearLocal( ) 5-5
closeSource( ) 5-6
deleteContent( ) 5-8
export( ) 5-9
getBFILE( ) 5-13
getContent( ) 5-15
getMimeType( ) 5-17
getSource( ) 5-19
getSourceLocation( ) 5-21
getSourceName( ) 5-22
getSourceType( ) 5-23
getUpdateTime( ) 5-25
isLocal( ) 5-26
openSource( ) 5-27
processSourceCommand( ) 5-29
readFromSource( ) 5-32
setLocal( ) 5-34
setMimeType( ) 5-35
setSource( ) 5-37
setUpdateTime( ) 5-39

trimSource( ) 5-40
writeToSource( ) 5-42

6 ORDAudio Reference Information

6.1 Object Types 6-2
ORDAudio Object Type 6-3
6.2 Constructors 6-7
init( ) 6-8
init(srcType,srcLocation,srcName) 6-10
6.3 Methods 6-12
6.3.1 Example Table Definitions 6-16
checkProperties( ) 6-17
getAllAttributes( ) 6-19
getAttribute( ) 6-21
getAudioDuration( ) 6-23
getContentLength( ) 6-24
getCompressionType( ) 6-25
getContentInLob( ) 6-26
getDescription( ) 6-28
getEncoding( ) 6-29
getFormat( ) 6-30
getNumberOfChannels( ) 6-31
getSampleSize( ) 6-32
getSamplingRate( ) 6-33
import( ) 6-34
importFrom( ) 6-36
processAudioCommand( ) 6-39
setAudioDuration( ) 6-42
setCompressionType( ) 6-43
setDescription( ) 6-44
setEncoding( ) 6-46

setFormat( ) 6-47
setKnownAttributes( ) 6-49
setNumberOfChannels( ) 6-51
setProperties( ) 6-52
setSamplingRate( ) 6-54
setSampleSize( ) 6-55
6.4 Packages or PL/SQL Plug-ins 6-56
6.4.1 ORDPLUGINS.ORDX_DEFAULT_AUDIO Package 6-56
6.4.2 Extending *inter*Media to Support a New Audio Data Format 6-59

7 ORDDoc Reference Information

7.1 Object Types 7-2
ORDDoc Object Type 7-3
7.2 Constructors 7-5
init( ) 7-6
init(srcType,srcLocation,srcName) 7-8
7.3 Methods 7-10
7.3.1 Example Table Definitions 7-12
getContentInLob( ) 7-14
getContentLength( ) 7-16
getFormat 7-17
import( ) 7-18
importFrom( ) 7-21
setFormat( ) 7-24
setProperties( ) 7-26
7.4 Packages or PL/SQL Plug-ins 7-29
7.4.1 ORDPLUGINS.ORDX_DEFAULT_DOC Package 7-29
7.4.2 Extending *inter*Media to Support a New Media Data Format 7-30

8 Image Object Types Reference Information

8.1 ORDImage Object Types 8-2
ORDImage Object Type 8-3

8.1.1 Constructors ... 8-6
init( ) for ORDImage ... 8-7
init(srcType,srcLocation,srcName) for ORDImage ... 8-9
8.1.2 Methods ... 8-10
8.1.3 Example Table Definitions ... 8-13
checkProperties ... 8-15
copy( ) ... 8-16
getCompressionFormat ... 8-18
getContentFormat ... 8-19
getContentLength ... 8-20
getFileFormat ... 8-21
getHeight ... 8-22
getWidth ... 8-23
import( ) ... 8-24
importFrom( ) ... 8-26
process( ) ... 8-29
processCopy( ) ... 8-34
setProperties ... 8-36
setProperties( ) for Foreign Images ... 8-38
8.2 ORDImageSignature Object Type ... 8-40
ORDImageSignature Object Type ... 8-42
8.2.1 Constructors ... 8-42
init( ) for ORDImageSignature ... 8-44
8.2.2 Methods ... 8-45
evaluateScore( ) ... 8-46
generateSignature( ) ... 8-48
isSimilar( ) ... 8-49
8.2.3 ORDImageSignature Operators ... 8-51
IMGSimilar Operator ... 8-52
IMGScore Operator ... 8-56

9 ORDVideo Reference Information

9.1 Object Types 9-2
ORDVideo Object Type 9-3
9.2 Constructors 9-8
init( ) 9-9
init(srcType,srcLocation,srcName) 9-11
9.3 Methods 9-13
9.3.1 Example Table Definitions 9-17
checkProperties( ) 9-18
getAllAttributes( ) 9-20
getAttribute( ) 9-22
getBitRate 9-24
getCompressionType 9-25
getContentInLob( ) 9-26
getContentLength( ) 9-28
getDescription 9-29
getFormat 9-30
getFrameRate 9-32
getFrameResolution 9-33
getFrameSize( ) 9-34
getNumberOfColors 9-36
getNumberOfFrames 9-37
getVideoDuration 9-38
import( ) 9-39
importFrom( ) 9-41
processVideoCommand( ) 9-44
setBitRate( ) 9-47
setCompressionType( ) 9-48
setDescription( ) 9-49
setFormat( ) 9-51
setFrameRate( ) 9-53

setFrameResolution( ) 9-54
setFrameSize( ) 9-55
setKnownAttributes( ) 9-57
setNumberOfColors( ) 9-60
setNumberOfFrames( ) 9-61
setProperties( ) 9-62
setVideoDuration( ) 9-64
9.4 Packages or PL/SQL Plug-ins 9-65
9.4.1 ORDPLUGINS.ORDX_DEFAULT_VIDEO Package 9-65
9.4.2 Extending *inter*Media to Support a New Video Data Format 9-68

10 *inter*Media Relational Interface Reference

10.1 Static Methods for the Relational Interface 10-2
10.1.1 Static Methods Common to All Object Types 10-2
10.1.2 Static Methods Uniquely Associated with Each Object Type 10-2
10.2 Static Methods Common to All Object Types 10-4
export( ) 10-5
importFrom( ) 10-8
importFrom( ) (all attributes) 10-11
10.3 Static Methods Unique to the ORDAudio Object Type Relational Interface 10-13
10.3.1 Example Table Definitions 10-14
getProperties( ) for BLOBs 10-16
getProperties( ) (all attributes) for BLOBs 10-18
getProperties( ) for BFILEs 10-22
getProperties( ) (all attributes) for BFILEs 10-24
10.4 Static Methods Unique to the ORDDoc Object Type Relational Interface 10-26
10.4.1 Example Table Definitions 10-28
getProperties( ) for BLOBs 10-29
getProperties( ) (all attributes) for BLOBs 10-31
getProperties( ) for BFILEs 10-34
getProperties( ) (all attributes) for BFILEs 10-36
10.5 Static Methods Unique to the ORDImage Object Type Relational Interface 10-38
10.5.1 Example Table Definitions 10-39

getProperties( ) for BLOBs 10-41
getProperties( ) (all attributes) for BLOBs 10-43
getProperties( ) for BFILEs 10-46
getProperties( ) (all attributes) for BFILEs 10-48
process( ) 10-51
processCopy( ) for BLOBs 10-53
processCopy( ) for BFILEs 10-55
10.6 Static Methods Unique to the ORDVideo Object Type Relational Interface 10-56
10.6.1 Example Table Definitions 10-58
getProperties( ) for BLOBs 10-60
getProperties( ) (all attributes) for BLOBs 10-62
getProperties( ) for BFILEs 10-66
getProperties( ) (all attributes) for BFILEs 10-68

11 Tuning Tips for the DBA

11.1 Setting Database Initialization Parameters 11-2
11.2 Issues to Consider in Creating Tables with *inter*Media Column Objects Containing BLOBs 11-8
11.2.1 Initializing Internal *inter*Media Column Objects Containing BLOBs to NULL or EMPTY 11-8
11.2.2 Specifying Tablespace and Storage Characteristics for *inter*Media Column Objects Containing BLOBs 11-9
11.2.3 Segment Attributes and Physical Attributes 11-15
11.2.4 Accommodating Temporary LOBs in the Buffer Cache 11-16
11.2.5 Using *inter*Media Column Objects Containing BLOBs in Table Partitions 11-17
11.2.6 LOB Buffering for Client Applications 11-17
11.3 Improving Multimedia Data INSERT Performance in *inter*Media Objects Containing LOBs 11-18
11.4 Loading Multimedia Data Using the *inter*Media Clipboard 11-25
11.5 Loading Multimedia Data Using *inter*Media Annotator Utility 11-25
11.6 Reading Data from an ORDVideo Object Using the *inter*Media readFromSource( ) Method in a PL/SQL Script 11-25
11.7 Reading Results of an *inter*Media Benchmark 11-26
11.8 Getting the Best Performance Results 11-28

11.9 Improving Multimedia LOB Data Retrieval and Update Performance 11-29

A Audio File and Compression Formats

A.1 Supported Audio File and Compression Formats A-1

B Image File and Compression Formats

B.1 Supported Image File and Compression Formats B-1
B.1.1 Image File Formats B-1
B.1.2 Image Compression Formats B-7
B.1.3 Summary of Image File Format and Image Compression Format B-11

C Video File and Compression Formats

C.1 Supported Video File and Compression Formats C-1

D Image process( ) and processCopy( ) Operators

D.1 Common Concepts D-1
D.1.1 Source and Destination Images D-1
D.1.2 process( ) and processCopy( ) D-2
D.1.3 Operator and Value D-2
D.1.4 Combining Operators D-2
D.2 Image Formatting Operators D-2
D.2.1 FileFormat D-3
D.2.2 ContentFormat D-3
D.2.3 CompressionFormat D-4
D.2.4 CompressionQuality D-5
D.3 Image Processing Operators D-6
D.3.1 Cut D-6
D.3.2 Scale D-6
D.3.3 XScale D-6
D.3.4 YScale D-7
D.3.5 FixedScale D-7
D.3.6 MaxScale D-7
D.4 Format-Specific Operators D-8
D.4.1 ChannelOrder D-8

D.4.2 Interleaving D-9
D.4.3 PixelOrder D-9
D.4.4 ScanlineOrder D-9
D.4.5 InputChannels D-9
D.4.6 Dither D-10
D.4.7 Page D-10
D.4.8 Tiled D-11

E Image Raw Pixel Format

E.1 Raw Pixel Introduction E-1
E.2 Raw Pixel Image Structure E-2
E.3 Raw Pixel Header Field Descriptions E-3
E.4 Raw Pixel Post-Header Gap E-7
E.5 Raw Pixel Data Section and Pixel Data Format E-8
E.5.1 Scanline Ordering E-8
E.5.2 Pixel Ordering E-8
E.5.3 Band Interleaving E-9
E.5.4 N-Band Data E-10
E.6 Raw Pixel Header “C” Structure E-11
E.7 Raw Pixel Header “C” Constants E-12
E.8 Raw Pixel PL/SQL Constants E-13
E.9 Raw Pixel Images Using CCITT Compression E-13
E.10 Foreign Image Support and the Raw Pixel Format E-14

F Sample Programs

F.1 Sample Audio Scripts F-1
F.2 Sample Document Scripts F-2
F.3 Sample Program for Modifying Images or Testing the Image Installation F-2
F.3.1 Demonstration (Demo) Installation Steps F-3
F.3.2 Running the Demo F-3
F.4 Sample Video Scripts F-4
F.5 Java Demo F-5

G Frequently Asked Questions

H Exceptions and Error Messages

H.1 Exceptions H-1
H.1.1 ORDAudioExceptions Exceptions H-1
H.1.2 ORDDocExceptions Exceptions H-3
H.1.3 ORDImageExceptions Exceptions H-3
H.1.4 ORDVideoExceptions Exceptions H-4
H.1.5 ORDSourceExceptions Exceptions H-5
H.2 ORDAudio Error Messages H-6
H.3 ORDImage Error Messages H-7
H.4 ORDVideo Error Messages H-25

I ORDSource Reference Information

I.1 Object Types I-2
ORDSource Object Type I-3
I.2 Methods I-6
clearLocal I-9
close( ) I-10
deleteLocalContent I-12
export( ) I-13
getBFile I-16
getContentInTempLob( ) I-17
getContentLength( ) I-19
getLocalContent I-21
getSourceAddress( ) I-22
getSourceInformation I-24
getSourceLocation I-25
getSourceName I-26
getSourceType I-27
getUpdateTime I-28
import( ) I-29

importFrom( ) I-31
isLocal I-33
open( ) I-34
processCommand( ) I-36
read( ) I-38
setLocal I-40
setSourceInformation( ) I-41
setUpdateTime( ) I-43
trim( ) I-44
write( ) I-46
I.3 Packages or PL/SQL Plug-ins I-47
I.3.1 ORDPLUGINS.ORDX_FILE_SOURCE Package I-48
I.3.2 ORDPLUGINS.ORDX_HTTP_SOURCE Package I-50
I.3.3 ORDPLUGINS.ORDX_<srcType>_SOURCE Package I-52
I.3.4 Extending *inter*Media to Support a New Data Source I-52

J Deprecated Methods

J.1 Deprecated Audio and Video Methods J-1

Index

List of Examples

3–1 Define a Song Object 3-2
3–2 Create a Table Named SongsTable 3-2
3–3 Create a List Object Containing a List of References to Songs 3-2
3–4 Define the Implementation of the songList Object 3-3
3–5 Create a CD Table Containing CD Information 3-4
3–6 Insert a Song into the SongsTable Table 3-4
3–7 Insert a CD into the CdTable Table 3-5
3–8 Load a Song into the SongsTable Table 3-5
3–9 Insert a Reference to a Song Object into the Songs List in the CdTable Table 3-7
3–10 Add a CD Reference to a Song 3-8
3–11 Retrieve Audio Data from a Song in a CD 3-9
3–12 Define a Relational Table Containing No ORDAudio Object 3-10
3–13 Define an Object View Containing an ORDAudio Object and Relational Columns. 3-11
3–14 Define a Media Object 3-20
3–15 Create a Table Named DocumentsTable 3-21
3–16 Create a List Object Containing a List of References to Media 3-21
3–17 Define the Implementation of the documentList Object 3-21
3–18 Create a Library Table Containing Library Information 3-22
3–19 Insert Media into the DocumentsTable Table 3-23
3–20 Insert a Library into the LibraryTable Table 3-24
3–21 Load Media into the DocumentsTable Table 3-24
3–22 Insert a Reference to a Document Object into the Documents List in the LibraryTable Table 3-25
3–23 Add a Library Reference to a Document 3-26
3–24 Build a Repository of Media 3-29
3–25 Add New Columns of Type ORDImage and ORDImageSignature to the stockphotos Table 3-42
3–26 Create the stockphotos Table and Add ORDImage and ORDImageSignature Types 3-43
3–27 Insert a Row into a Table with Empty Data in the ORDImage Type Column 3-43
3–28 Populate a Row with ORDImage BLOB Data 3-44
3–29 Insert a Row into a Table Pointing to an External Image Data File 3-45
3–30 Populate a Row with ORDImage External File Data 3-46
3–31 Query Rows of ORDImage Data for Widths Greater Than 32 Pixels 3-46
3–32 Query Rows of ORDImage Data for Widths Greater Than 32 Pixels and a Minimum Content Length 3-47
3–33 Import an Image from an External File 3-47
3–34 Table stockphotos Definition Used for Content-Based Retrieval of Images 3-48
3–35 Load the stockphotos Table with Image Data 3-48
3–36 Check the Contents of the stockphotos Table 3-49

3–37 Create the Tablespaces for the Index ... 3-49
3–38 Retrieve an Image (Simple Read) ... 3-50
3–39 Retrieve Images Similar to a Comparison Image ... 3-51
3–40 Find photo_id and Score of Similar Image ... 3-52
3–41 Create an *inter*Media Index ... 3-52
3–42 Copy an Image ... 3-53
3–43 Convert an Image Format ... 3-54
3–44 Copy and Convert an Image Format ... 3-55
3–45 Extend Oracle *inter*Media with a New Object Type ... 3-56
3–46 Define a Relational Table Containing No ORDImage Object ... 3-57
3–47 Define an Object View Containing an ORDImage Object and Relational Columns . 3-58
3–48 Address a Globalization Support Issue ... 3-69
3–49 Define a Clip Object ... 3-70
3–50 Create a Table Named ClipsTable ... 3-71
3–51 Create a List Object Containing a List of Clips ... 3-71
3–52 Define the Implementation of the clipList Object ... 3-71
3–53 Create a Video Table Containing Video Information ... 3-72
3–54 Insert a Video Clip into the ClipsTable Table ... 3-72
3–55 Insert a Row into the VideoTable Table ... 3-73
3–56 Load a Video into the ClipsTable Table ... 3-73
3–57 Insert a Reference to a Clip Object into the Clips List in the VideoTable Table ... 3-74
3–58 Insert a Reference to a Video Object into the Clip ... 3-75
3–59 Retrieve a Video Clip ... 3-76
3–60 Define a Relational Table Containing No ORDVideo Object ... 3-78
3–61 Define an Object View Containing an ORDVideo Object and Relational Columns . 3-78
6–1 Show the Package Body for Extending Support to a New Audio Data Format ... 6-60
7–1 Show the Package Body for Extending Support to a New Media Data Format ... 7-30
9–1 Show the Package Body for Extending Support to a New Video Data Format ... 9-69
11–1 Create a Separate Tablespace to Store an *inter*Media Column Object Containing LOB Data ... 11-9
11–2 Show the Load1.bat File ... 11-18
11–3 Show the T1.SQL File ... 11-19
11–4 Show the Load1.sql File that Executes the load_image Stored Procedure ... 11-22
11–5 Show the Control File for Loading Video Data ... 11-23
11–6 Read Data from an ORDVideo Column Object Using *inter*Media readFromSource( ) Method in a PL/SQL Stored Procedure ... 11-26
F–1 Execute the Demo from the Command Line ... F-4
I–1 Show the Package Body for Extending Support to a New Data Source ... I-53

List of Figures

1–1 *inter*Media Architecture 1-19
2–1 Unsegmented Image 2-3
2–2 Segmented Image 2-4
2–3 Image Comparison: Color and Location 2-5
2–4 Images Very Similar in Color 2-6
2–5 Images Very Similar in Color and Location 2-6
2–6 Fabric Images with Similar Texture 2-7
2–7 Images with Very Similar Shape 2-8
2–8 Score and Distance Relationship 2-9
8–1 Use and Syntax Flow Diagram for the contentFormat Operator Values 8-32

List of Tables

1–1	*inter*Media Services and Features -- Supported Systems and Oracle9*i* Releases	1-20
2–1	Distances for Visual Attributes Between Image1 and Image2	2-10
6–1	PL/SQL Plug-ins Provided in the ORDPLUGINS Schema	6-56
6–2	Methods Supported in the ORDPLUGINS.ORDX_DEFAULT_AUDIO Package	6-58
7–1	PL/SQL Plug-ins Provided in the ORDPLUGINS Schema	7-29
7–2	Method Supported in the ORDPLUGINS.ORDX_DEFAULT_DOC Package	7-29
8–1	Image Processing Operators	8-29
8–2	Additional Image Processing Operators for Raw Pixel and Foreign Images	8-31
8–3	Image Characteristics for Foreign Files	8-39
9–1	PL/SQL Plug-ins Provided in the ORDPLUGINS Schema	9-65
9–2	Methods Supported in the ORDPLUGINS.ORDX_DEFAULT_VIDEO Package	9-67
A–1	AIFF Data Format	A-1
A–2	AIFF-C Data Format	A-2
A–3	AU Data Format	A-2
A–4	WAV Data Format	A-3
A–5	Audio MPEG Data Format	A-5
B–1	Summary of Read/Write Access for Supported Image File Formats -- Content Format Specific Characteristics	B-12
B–2	Summary of Read/Write Access for Supported Image File Formats -- Compression Format and Other Format Specific Characteristics	B-13
C–1	Apple QuickTime 3.0 Data Format	C-2
C–2	Microsoft Video for Windows (AVI) Data Format	C-3
C–3	RealNetworks Real Video Data Format	C-3
I–1	Methods Supported in the ORDPLUGINS.ORDX_FILE_SOURCE Package	I-49
I–2	Methods Supported in the ORDPLUGINS.ORDX_HTTP_SOURCE Package	I-51

Send Us Your Comments

Oracle *inter*Media User's Guide and Reference, Release 9.0.1

Part No. A88786-01

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

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

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

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

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

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

Preface

This guide describes how to use Oracle *inter*Media.

Oracle *inter*Media ships with Oracle9*i*.

For information about Oracle9*i* and the features and options that are available to you, see *Oracle9i Database New Features*.

Audience

This guide is for application developers and database administrators who are interested in storing, retrieving, and manipulating audio, document, image, and video data in an Oracle database, including developers of audio, document, image, and video specialization options.

If you are interested in only one particular object type, see Chapter 1 for general introductory information, then, for a description of the methods that are common for all object types, refer to Chapter 5. If, for example, you are interested in the ORDImage object type, refer to Chapter 8, the ORDImage reference chapter for a description of the image-specific methods, then for a description of content-based retrieval and image matching, refer to Chapter 2.

Also, for examples about using the ORDImage methods, see Chapter 3 and for a description of using the relational interface with images, see Chapter 10. Then, for tuning tips for storing image files, see Chapter 11.

For information on supported image content and compression formats, see Appendix B. For information about using image processing methods, see Appendix D. Finally, for information about the raw pixel image format, see Appendix E.

Organization

This guide contains the following chapters and appendixes:

Chapter 1	Introduces multimedia and Oracle *inter*Media; explains multimedia-related concepts.
Chapter 2	Explains concepts, operations, and techniques related to content-based retrieval.
Chapter 3	Provides basic examples of using Oracle *inter*Media object types and methods.
Chapter 4	Provides compatibility information for ensuring future compatibility with evolving object types.
Chapter 5	Provides reference information about methods that are common to ORDAudio, ORDDoc, ORDImage, and ORDVideo object types.
Chapter 6	Provides reference information on Oracle *inter*Media ORDAudio object type and methods.
Chapter 7	Provides reference information on Oracle *inter*Media ORDDoc object type and methods.
Chapter 8	Provides reference information on Oracle *inter*Media ORDImage object type and methods.
Chapter 9	Provides reference information on Oracle *inter*Media ORDVideo object type and methods.
Chapter 10	Provides reference information on *inter*Media relational interface methods for the ORDAudio, ORDDoc, ORDImage, and ORDVideo object types.
Chapter 11	Provides tuning tips for the DBA for more efficient storage of multimedia data.
Appendix A	Describes the supported audio data formats.
Appendix B	Describes the supported image data formats.
Appendix C	Describes the supported video data formats.
Appendix D	Describes the process and processCopy operators.
Appendix E	Describes the raw pixel format.
Appendix F	Describes how to run the sample program and includes the source program.
Appendix G	Emphasizes several entries from the online FAQ.

Appendix H — Lists exceptions raised and potential errors, their causes, and user actions to correct them.

Appendix I — Provides reference information on Oracle *inter*Media ORDSource object type and methods.

Appendix J — Describes the deprecated audio and video methods.

Related Documents

Note: For information added after the release of this guide, refer to the online README.txt file in your *ORACLE_HOME* directory. Depending on your operating system, this file may be in:

ORACLE_HOME`/ord/im/admin/README.txt`

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

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

`http://otn.oracle.com/`

For more information about using *inter*Media in a development environment, see the following documents in the release 9.0.1 Oracle database server documentation set:

- *Oracle Call Interface Programmer's Guide*
- *Oracle9i Application Developer's Guide - Fundamentals*
- *Oracle9i Application Developer's Guide - Large Objects (LOBs)*
- *Oracle9i Database Concepts*
- *PL/SQL User's Guide and Reference*
- *Oracle interMedia Java Classes User's Guide and Reference*

Conventions

In this guide, Oracle *inter*Media is sometimes referred to as *inter*Media.

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

The following conventions are also used in this guide:

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

Changes to This Guide

The following substantive changes have been made to this guide since its previous version for release 8.1.7 on the Oracle Technology Network (OTN) Web site.

Other minor corrections and clarifications have also been included.

A new document object, ORDDoc, is available. The ORDDoc document type can be used in applications that require you to store different types of documents, such as audio, image, video, and any other type of document in the same column so you can build a common metadata index on all the different types of documents and search across different types of documents using this index. See Chapter 7 for more information.

Content-based retrieval of images with extensible indexing is supported for image matching. See Chapter 2, the ORDImageSignature Object Type, the evaluateScore( ) method, and the image operators described in Section 8.2.3 for more information.

*inter*Media image supports the Java Advanced Imaging engine. See the process( ) method, Appendix B, and Appendix D for more information.

An *inter*Media relational interface is available for application developers, who created multimedia applications without using the *inter*Media object types to store

and manage media data in relational tables, and who do not want to migrate their existing multimedia applications to use *inter*Media objects. See Chapter 10 for more information.

Documentation Accessibility

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

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

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

1 Introduction

Oracle *inter*Media is a feature that enables Oracle9*i* to store, manage, and retrieve geographic location information, images, audio, video, or other heterogeneous media data in an integrated fashion with other enterprise information. Oracle *inter*Media extends Oracle9*i* reliability, availability, and data management to multimedia content in Internet, electronic commerce, and media-rich applications as well as online Internet-based geocoding services for locator applications.

Oracle *inter*Media provides services for managing Web content. These services include:

- Media and application metadata management services (see Section 1.3, Section 1.4, Section 1.5, Section 1.6, and Section 1.13.3)
- Storage and retrieval services (see Section 1.11 and Section 1.12)
- Support for popular formats (see Appendix A, Appendix B, and Appendix C)
- Access through traditional and Web interfaces (see Section 1.13.5) and a search capability using associated relational data or using specialized indexing.

Oracle *inter*Media provides media content services to JDeveloper, Oracle Internet File System, Oracle Portal, and Oracle partners. This guide describes only the management of audio, image, and video, or other heterogeneous media data.

1.1 Object Relational Technology

Oracle9*i* is an *object relational* database management system. This means that in addition to its traditional role in the safe and efficient management of relational data, it provides support for the definition of object types, including the data associated with objects and the operations (methods) that can be performed on them. This powerful mechanism, well established in the object-oriented world,

includes integral support for BLOBs to provide the basis for adding complex objects, such as digitized audio, image, and video to Oracle9*i* databases.

Within Oracle *inter*Media, audio data characteristics have an object relational type known as **ORDAudio**, heterogeneous data characteristics have an object relational type known as **ORDDoc**, image data characteristics have an object relational type known as **ORDImage**, and video data characteristics have an object relational type known as **ORDVideo**. All four store data source information in an object relational type known as **ORDSource**.

See the following references for extensive information on using BLOBs and BFILEs:

- *Oracle9i Application Developer's Guide - Large Objects (LOBs)*
- *Oracle9i Database Concepts* -- see the chapter on Object Views.

See Section 1.7 for more information about the multimedia object types and methods and Section 1.8 for more information about the ORDSource object type and methods.

1.2 Multimedia Content Management

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

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

Multimedia applications have common and unique requirements. Oracle *inter*Media object types support common application requirements and can be extended to address application-specific requirements. With Oracle *inter*Media, multimedia data can be managed as easily as standard attribute data.

Oracle *inter*Media is accessible to applications through both relational and object interfaces. Database applications written in Java, C++, or traditional 3GLs can interact with *inter*Media through modern class library interfaces, or PL/SQL and Oracle Call Interface (OCI).

*inter*Media supports storage of the popular file formats, including desktop publishing image, and streaming audio and video formats in Oracle9*i* databases. *inter*Media provides the means to add audio, image, and video, or other heterogeneous media columns or objects to existing tables, and insert and retrieve multimedia data. This enables database designers to extend existing application databases with multimedia data or to build new end-user multimedia database applications. *inter*Media developers can use the basic functions provided here to build specialized multimedia applications.

Oracle *inter*Media uses object types, similar to Java or C++ classes, to describe multimedia data. These object types are called ORDAudio, ORDDoc, ORDImage, and ORDVideo. An instance of these object types consists of attributes, including metadata and the media data, and methods. **Media data** is the actual audio, image, or video, or other heterogeneous media data. **Metadata** is information about the data, such as object length, compression type, or format. **Methods** are procedures that can be performed on the object like getContent( ) and setProperties( ).

*inter*Media objects have a common media data storage model. The media data component of these objects can be stored in the database, in a binary large object (BLOB) under transaction control. The media data can also be stored outside the database, without transaction control. In this case, a pointer is stored in the database under transaction control, and the media data is stored in:

- An external binary file (BFILE)
- An HTTP server-based URL
- A user-defined source on a specialized media data server or other server

Media data stored outside the database can provide a convenient mechanism for managing large, pre-existing, or new media repositories that reside as flat files on erasable or read-only media. This data can be imported into BLOBs at any time for transaction control. Section 1.11 describes several ways of loading multimedia data into an Oracle9*i* database.

Media metadata is stored in the database under Oracle *inter*Media control. Whether media data is stored within or outside the database, *inter*Media manages metadata for all the media types and may automatically extract it for audio, image, and video. This metadata includes the following attributes:

- Audio, image, and video, or other heterogeneous media data storage information including the source type, location, and source name, and whether the data is stored locally (in the database) or externally
- Audio, image, and video, or other heterogeneous media data update timestamp
- Audio and video data description
- Audio, image, and video, or other heterogeneous media data format
- MIME type of the audio, image, and video, or other heterogeneous media data
- Audio and video metadata, or other heterogeneous media metadata in XML
- Audio characteristics: encoding type, number of channels, sampling rate, sample size, compression type, and play time (duration)
- Image characteristics: height and width, image content length, image content format, and image compression format
- Video characteristics: frame width and height, frame resolution, frame rate, play time (duration), number of frames, compression type, number of colors, and bit rate

In addition to metadata extraction methods, a minimal set of image manipulation methods is provided. For image, this includes performing format conversion and compression, scaling, cropping, and copying images.

*inter*Media is designed to be extensible. It supports a base set of popular audio, image, and video data formats for multimedia processing that also can be extended, for example, to support additional formats, new digital compression and decompression schemes (**codecs**), data sources, and even specialized data processing algorithms for audio and video data.

It is possible to extend Oracle *inter*Media by:

- Creating a new object type or a new composite object type based on the provided multimedia object types. See the examples in Section 3.1.13, Section 3.3.16, and Section 3.4.13 for more information.
- Creating specialized plug-ins to support other external sources of audio, image, and video data, or other heterogeneous media data that are not currently supported. See Section 1.9.1 for more information.
- Creating specialized audio and video data, or other heterogeneous media data format plug-ins to support other audio and video data, or other heterogeneous media data formats that are not currently supported. See Section 1.9.1 for more information.

- Using the setProperties( ) method for foreign images, which allows certain other image formats to be recognized. See Section 1.9.1 and "setProperties( ) for Foreign Images" in Section 8.1.2 for more information.
- Using the audio and video data processing methods to allow a specific audio or video command and its arguments to be passed through to process audio or video data. See Section 1.9.2 and Section 1.9.3 for more information.

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

- Internet music stores that provide music samplings of CD quality
- Digital sound repositories
- Dictation and telephone conversation repositories
- Audio archives and collections (for example, for musicians)
- Digital art galleries
- Real estate marketing
- Document imaging
- Photograph collections (for example, for professional photographers)
- Internet video stores and digital video-clip previews
- Digital video sources for streaming video delivery systems
- Digital video libraries, archives, and repositories
- Libraries of digital video training programs
- Digital video repositories (for example, for motion picture production, television broadcasting, documentaries, advertisements, and so forth)

1.3 Audio Concepts

This section contains information about digitized audio concepts and using the ORDAudio object type to build audio applications or specialized ORDAudio objects.

1.3.1 Digitized Audio

ORDAudio integrates the storage, retrieval, and management of digitized audio data in Oracle databases using Oracle9*i*.

Audio may be produced by an audio recorder, an audio source such as a microphone, digitized audio, other specialized audio recording devices, or even by program algorithms. Audio recording devices take an analog or continuous signal, such as the sound picked up by a microphone or sound recorded on magnetic media, and convert it into digital values with specific audio characteristics such as format, encoding type, number of channels, sampling rate, sample size, compression type, and audio duration.

1.3.2 Audio Components

Digitized audio consists of the audio data (digitized bits) and attributes that describe and characterize the audio data. Audio applications sometimes associate application-specific information, such as the description of the audio clip, date recorded, author or artist, and so forth, with audio data by storing descriptive text in an attribute or column in the database table.

The audio data can have different formats, encoding types, compression types, numbers of channels, sampling rates, sample sizes, and playing times (duration) depending upon how the audio data was digitally recorded. ORDAudio can store and retrieve audio data of any data format. ORDAudio can automatically extract metadata from audio data of a variety of popular audio formats. ORDAudio can also extract application attributes and store them in the comments field of the object in XML form identical to what is provided by *inter*Media Annotator utility. See Appendix A for a list of supported data formats from which ORDAudio can extract and store attributes and other audio features. ORDAudio is extensible and can be made to recognize and support additional audio formats.

The size of digitized audio (number of bytes) tends to be large compared to traditional computer objects, such as numbers and text. Therefore, several encoding schemes are used that squeeze audio data into fewer bytes, thus putting a smaller load on storage devices and networks.

1.4 ORDDoc or Heterogeneous Media Data Concepts

This section contains information about heterogeneous media data concepts and using the ORDDoc object type to build applications or specialized ORDDoc objects.

1.4.1 Digitized Heterogeneous Media Data

ORDDoc integrates the storage, retrieval, and management of heterogeneous media data in Oracle databases using Oracle9*i*.

Text documents may be produced by application software, text conversion utilities, speech to text processing software, and so forth. Heterogeneous media data can be ASCII text files or binary files formatted by a particular application.

*inter*Media ORDDoc can store any heterogeneous media data including audio, image, and video data in a database column. Instead of having separate columns for audio, image, text, and video objects, you can use one column of ORDDoc objects to represent all types of multimedia.

1.4.2 Heterogeneous Media Data Components

Heterogeneous media data consist of the data (digitized bits) and attributes that describe and characterize the heterogeneous media data.

Heterogeneous media data can have different formats depending upon the application generating the media data. *inter*Media can store and retrieve media data of any data format. The ORDDoc heterogeneous media data type can be used in applications that require you to store different types of heterogeneous media data, such as audio, image, video, and any other type of media data in the same column so you can build a common metadata index on all the different types of heterogeneous media data. Using this index, you can search across all the different types of heterogeneous media data. Note that you cannot use this same search technique if the different types of heterogeneous media data are stored in different types of objects in different columns of relational tables.

ORDDoc can automatically extract metadata from data of a variety of popular audio, image, and video data formats. ORDDoc can also extract application attributes and store them in the comments field of the object in XML form. See Appendix A, Appendix B, and Appendix C for a list of supported data formats from which *inter*Media can extract and store attributes. ORDDoc is extensible and can be made to recognize and support other heterogeneous media data formats.

1.5 Image Concepts

This section contains information about digitized image concepts and using the ORDImage object type to build image applications or specialized ORDImage objects.

1.5.1 Digitized Images

ORDImage integrates the storage, retrieval, and management of digitized images in Oracle databases using Oracle9*i*.

ORDImage supports two-dimensional, static, digitized raster images stored as binary representations of real-world objects or scenes. Images may be produced by a document or photograph scanner, a video source such as a camera or VCR connected to a video digitizer or frame grabber, other specialized image capture devices, or even by program algorithms. Capture devices take an analog or continuous signal such as the light that falls onto the film in a camera, and convert it into digital values on a two-dimensional grid of data points known as pixels. Devices involved in the capture and display of images are under application control.

1.5.2 Image Components

Digitized images consist of the image data (digitized bits) and attributes that describe and characterize the image data. Image applications sometimes associate application-specific information, such as including the name of the person pictured in a photograph, description of the image, date photographed, photographer, and so forth, with image data by storing this descriptive text in an attribute or column in the database table.

The image data (pixels) can have varying depths (bits per pixel) depending on how the image was captured, and can be organized in various ways. The organization of the image data is known as the **data format**. ORDImage can store and retrieve image data of any data format. ORDImage can process and automatically extract properties of images of a variety of popular data formats. See Appendix B for a list of supported data formats for which ORDImage can process and extract metadata. In addition, certain foreign images (formats not natively supported by ORDImage) have limited support for image processing. See Appendix E for more information.

The storage space required for digitized images can be large compared to traditional attribute data such as numbers and text. Many compression schemes are available to squeeze an image into fewer bytes, thus reducing storage device and network load. **Lossless compression** schemes squeeze an image so that when it is decompressed, the resulting image is bit-for-bit identical with the original. **Lossy compression** schemes do not result in an identical image when decompressed, but rather, one in which the changes may be imperceptible to the human eye.

Image **interchange format** describes a well-defined organization and use of image attributes, data, and often compression schemes, allowing different applications to create, exchange, and use images. Interchange formats are often stored in or as disk

files. They may also be exchanged in a sequential fashion over a network and be referred to as a **protocol**. There are many application subdomains within the digitized imaging world and many applications that create or utilize digitized images within these. ORDImage supports storage and retrieval of all image data formats and processing and attribute extraction of many image data formats (see Appendix B).

Content-based retrieval of images with extensible indexing is supported for image matching. An overview of the benefits of content-based retrieval is described in Chapter 2 along with how content-based retrieval works, including definitions and explanations of the visual attributes (color, texture, shape, and location) and why you might emphasize specific attributes in certain situations. In addition, the use of indexing to improve search and retrieval performance is described in Section 2.4.

1.6 Video Concepts

This section contains information about digitized video concepts and using ORDVideo to build video applications or specialized ORDVideo objects.

1.6.1 Digitized Video

ORDVideo integrates the storage, retrieval, and management of digitized video data in Oracle databases using Oracle9*i*.

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

1.6.2 Video Components

Digitized video consists of the video data (digitized bits) and the attributes that describe and characterize the video data. Video applications sometimes associate application-specific information, such as the description of the video training tape, date recorded, instructor's name, producer's name, and so forth, with video data by storing descriptive text in an attribute or column in the database table.

The video data can have different formats, compression types, frame rates, frame sizes, frame resolutions, playing times, compression types, number of colors, and

bit rates depending upon how the video data was digitally recorded. ORDVideo can store and retrieve video data of any data format. ORDVideo can automatically extract metadata from video data of a variety of popular video formats. ORDVideo can also extract application attributes and store them in the comments field of the object in XML form identical to what is provided by the *inter*Media Annotator utility. See Appendix C for a list of supported data formats from which *inter*Media can extract and store attributes and other video features. ORDVideo is extensible and can be made to recognize and support additional video formats.

The size of digitized video (number of bytes) tends to be large compared to traditional computer objects, such as numbers and text. Therefore, several encoding schemes are used that squeeze video data into fewer bytes, thus putting a smaller load on storage devices and networks.

1.7 Multimedia Object Types and Methods

Oracle *inter*Media provides the ORDAudio, ORDDoc, ORDImage, and ORDVideo object types and methods for:

- Modifying the time an object was last updated
- Manipulating the location of media data
- Extracting attributes from multimedia data
- Getting and managing multimedia data from Oracle *inter*Media, Web servers, and other servers
- Performing a minimal set of manipulation operations on multimedia data (ORDImage only)
- Performing file operations on the source and metadata extraction in XML format (ORDAudio, ORDDoc, and ORDVideo only)

1.8 Multimedia Storage

Oracle *inter*Media provides the ORDSource object type and methods for multimedia data source manipulation. The ORDAudio, ORDDoc, ORDImage, and ORDVideo object types all contain an attribute of type ORDSource. This section presents a conceptual overview of the ORDSource object type methods.

Note: ORDSource methods should not be called directly. Instead, invoke the wrapper method of the media object corresponding to the ORDSource method. This information is presented for users who want to write their own user-defined sources.

1.8.1 Storing Multimedia Data

*inter*Media can store multimedia data as an internal source within the Oracle9*i* database, under transactional control as a BLOB. It can also externally reference digitized multimedia data stored as an external source in an operating system-specific BFILE in a local file-system, as a URL on an HTTP server, as audio, image, or video stored on media servers, or as a user-defined source on other servers. Although these external storage mechanisms are particularly convenient for integrating pre-existing sets of multimedia data with an Oracle9*i* database, the multimedia data will not be under transactional control.

BLOBs are stored in the database tablespaces in a way that optimizes space and provides efficient access. Large BLOBs may not be stored inline (BLOBs under 4K bytes in size can be stored inline) with other row data. Depending on the size of the BLOB, a locator is stored in the row and the actual BLOB (up to 4 gigabytes) is stored in other tablespaces. The locator can be considered a pointer to the actual location of the BLOB value. When you select a BLOB, you are selecting the locator instead of the value, although this is done transparently. An advantage of this design is that multiple BLOB locators can exist in a single row. For example, you might want to store a short video clip of a training tape, an audio recording containing a brief description of its contents, a syllabus of the course, a picture of the instructor, and a set of maps and directions to each training center.

Because BFILEs are not under the transactional control of the database, users could change the external source without updating the database, thus causing an inconsistency with the BFILE locator. See *Oracle9i Application Developer's Guide - Large Objects (LOBs)* and *Oracle Call Interface Programmer's Guide* for detailed information on using BLOBs and BFILEs.

*inter*Media ORDAudio, ORDDoc, ORDImage, and ORDVideo object types provide wrapper methods to set the source of the data as local or external; modifying the time an object was last updated; setting information about the external source type, location, and file name of the data; transferring data into or out of the database; obtaining information about the local data content such as its length and location, its handle to the BLOB, putting the content into a temporary BLOB, or deleting it; accessing source data by opening it, reading it, writing to it, trimming it, and

closing it; and passing in a series of methods and related arguments to be processed by calling a single method.

1.8.2 Querying Multimedia Data

Once stored within an Oracle9*i* database, multimedia data can be queried and retrieved by using the various alphanumeric columns or object attributes of the table to find a row that contains the desired data. For example, you can select a video clip from the Training table where the course name is 'Oracle9*i* Concepts'.

The collection of multimedia data in the database can be related to some set of attributes or keywords that describe the associated content. The multimedia data content can be described with textual components and numeric attributes such as dates and identification numbers. With Oracle9*i*, data attributes can reside in the same table as the object type with objects also containing the metadata. Alternatively, the application designer could define a composite object type that contains one of the *inter*Media object types along with other attributes.

1.8.3 Accessing Multimedia Data

Applications access and manipulate multimedia data using SQL, PL/SQL, OCI, or Java through the object relational types ORDAudio, ORDDoc, ORDImage, and ORDVideo. See *Oracle interMedia Java Classes User's Guide and Reference* for more information about using Java.

The object syntax for accessing attributes within a complex object is the dot notation:

```
variable.data_attribute
```

The syntax for invoking methods of a complex object is also the dot notation:

```
variable.function(parameter1, parameter2, ...)
```

A complete set of media attribute accessors (get methods) are provided for accessing attributes for each media type.

See *Oracle9i Database Concepts* for information on this and other SQL syntax.

1.9 Extending Oracle *inter*Media

*inter*Media can be extended to support:

- Other external sources of media data not currently supported

- Other media data formats not currently supported
- Audio and video data processing

The following sections describe each of these topics and where to find more information.

1.9.1 Supporting Other External Sources and Other Media Data Formats

For each unique external media data source or each unique ORDAudio, ORDDoc, or ORDVideo data format that you want to support, you must:

1. Design your new data source or new ORDAudio, ORDDoc, or ORDVideo data format.
2. Implement your new data source or new ORDAudio, ORDDoc, or ORDVideo data format.
3. Install your new plug-in in the ORDPLUGINS schema.
4. Grant EXECUTE privileges on your new plug-in to PUBLIC.

Supporting Other External Sources

To implement your new data source, you must implement the required interfaces in the ORDX_<srcType>_SOURCE package in the ORDPLUGINS schema (where <srcType> represents the name of the new external source type). Use the package body example in Section I.3.4 as a template to create the package body. Then set the source type parameter in the setSourceInformation( ) call to the appropriate source value to indicate to the ORDAudio, ORDImage, ORDDoc, or ORDVideo object that package ORDPLUGINS.ORDX_<srcType>_SOURCE is available as a plug-in. Use the ORDPLUGINS.ORDX_FILE_SOURCE and ORDPLUGINS.ORDX_HTTP_SOURCE packages as guides when you extend support to other external audio, image, video, or other heterogeneous media data sources.

See Section 3.5, Section I.3.1, Section I.3.2, and Section I.3.4 for examples and for more information on extending the supported external sources of audio, image, video, or other heterogeneous media data.

Supporting Other ORDAudio, ORDDoc, and ORDVideo Data Formats

To implement your new ORDAudio, ORDDoc, or ORDVideo data format, you must implement the required interfaces in the ORDPLUGINS.ORDX_<format>_<media> package in the ORDPLUGINS schema (where <format> represents the name of the new audio or video, or other heterogeneous media data format and <media> represents the type of media ("AUDIO" or "VIDEO", or "DOC"). Use the

ORDPLUGINS.ORDX_DEFAULT_<media> package as a guide when you extend support to other audio or video data formats or other heterogeneous media data formats. Use the package body examples in Section 6.4.2, Section 7.4.2, and Section 9.4.2 as templates to create the audio or video, or other heterogeneous media data package body, respectively. Then set the new format parameter in the setFormat( ) call to the appropriate format value to indicate to the ORDAudio, ORDDoc, or ORDVideo object that package ORDPLUGINS.ORDX_<format>_<media> is available as a plug-in.

See Section F.1 and Section F.4 for more information on installing your own format plug-in and running the sample scripts provided.

See Section 3.1.12, Section 3.2.11, Section 3.4.12, Section 6.4.2, Section 7.4.2, and Section 9.4.2 for examples and for more information on extending the supported audio and video, or other heterogeneous media data formats.

Supporting Other Image Data Formats

Oracle *inter*Media supports certain other image formats through the setProperties( ) method for foreign images. This method allows other image formats to be recognized by writing the values supplied to the setProperties( ) method for foreign images to the existing ORDImage data attributes. See "setProperties( ) for Foreign Images" in Section 8.1.2 for more information.

1.9.2 Supporting Audio Data Processing

To support audio data processing, that is, the passing of an audio processing command and set of arguments to a format plug-in for processing, use the processAudioCommand( ) method. This method is available only for user-defined formats.

See "processAudioCommand( )" in Section 6.3 and Section 3.1.12 for a description.

1.9.3 Supporting Video Data Processing

To support video data processing, that is, the passing of a command and set of arguments to a format plug-in for processing, use the processVideoCommand( ) method. This method is only available for user-defined formats.

See "processVideoCommand( )" in Section 9.3 and Section 3.4.12 for a description.

1.10 Relational Interface

Oracle *inter*Media relational interface gives developers the power of *inter*Media to annotate and manipulate media data stored in BLOBs and BFILEs without requiring changes to the existing application schema or instantiation of *inter*Media object types, ORDAudio, ORDDoc, ORDVideo, and ORDImage.

Developers can now use static methods of *inter*Media objects with existing and new media stored in BLOBS and BFILEs to move media data between the local file system and the database, to parse and extract the properties of the media data, and to store these properties in an XML formatted CLOB and optionally individual relational columns. *inter*Media static methods can also be used to perform image processing operations such as cut, scale, compress, and convert format.

See Chapter 10 for a description of the relational interface for each media type, including reference information, and information about using the relational interface. See Table 1–1 for a description on the availability of the relational interface and form of distribution.

1.11 Loading Multimedia Data into Oracle9*i* Using *inter*Media

Multimedia data can be managed best by the Oracle9*i* database. Your multimedia data should be loaded into Oracle9*i* to take advantage of its reliability, scalability, availability, and data management capabilities. To bulk load multimedia data into Oracle9*i*, you can use:

- SQL*Loader

  SQL*Loader is an Oracle utility that lets you load data, and in this case, multimedia data (LOB data), from external multimedia files into a table of an Oracle9*i* database containing *inter*Media column objects.

- PL/SQL

  A procedural extension to SQL, PL/SQL is an advanced fourth-generation programming language (4GL) of Oracle Corporation.

An advantage of using SQL*Loader is that it is easy to create and test the control file that controls your data loading operation. See Section 11.3 for a description of a sample control file. See also *Oracle9i Database Utilities* for more information.

An advantage of using PL/SQL scripts to load your data is that you can call methods as you load data to generate image thumbnails, extract properties, or import data. See Section 11.3 for a description of a sample PL/SQL multimedia data load script. See also *PL/SQL User's Guide and Reference* for more information.

Loading Multimedia Data Using Oracle *inter*Media Clipboard, Version 2

You can also use the Oracle *inter*Media feature, the Clipboard Version 2, to individually store and retrieve multimedia objects, such as audio, video, and image data, in an Oracle9*i* database server.

See *Setting Up the Oracle interMedia Clipboard, Version 2* for more information.

The Clipboard can be downloaded from the Oracle *inter*Media Utilities and Plugins section of the Oracle Technology Network Web site:

`http://otn.oracle.com/`

Loading Multimedia Data Using Oracle *inter*Media Annotator Utility

You can use the Oracle *inter*Media Annotator utility to upload media data and an associated annotation into an Oracle9*i* database. Annotator does this using an Oracle PL/SQL Upload Template, which contains both PL/SQL calls and Annotator-specific keywords.

Advanced users with PL/SQL experience can create their own PL/SQL Upload Templates in a text editor. Novice users can use the PL/SQL Template Wizard, which is a graphical user interface that progresses through each step of PL/SQL Upload Template creation.

With a PL/SQL Upload Template created, you use the Annotator utility to invoke the Upload Annotation window and perform a series of operations, entering user name, password, service name, and the path to the PL/SQL Template Folder and file specification for your PL/SQL Upload Template.

See Chapter 5 "Uploading Structured Annotations into a Database" in *Oracle interMedia Annotator User's Guide* for more information.

1.12 Reading Data from a LOB

LOB read tests were conducted with:

- PL/SQL scripts used to read LOBs from the database
- OCI calls to perform LOB read operations from C++

A benchmark measured the performance of an Oracle-based system in a setting modeling a real-life video server application. See Section 11.6 for a description of the PL/SQL script used to read LOBs from the database. See Section 11.7 for a description of the LOB-read benchmark tests and the results of these tests for measuring the performance of an Oracle-based system in a setting modeling a real-life audio server application.

1.13 *inter*Media Architecture

Oracle *inter*Media is a single, integrated feature that extends Oracle9*i* by offering services to store, manage, and retrieve image, audio, and video data, location services, support for Web technologies, and annotation services for multimedia data.

The *inter*Media architecture defines the framework (see Figure 1–1) through which media-rich content as well as traditional data are supported in the database. This content and data can then be securely shared across multiple applications written with popular languages and tools, easily managed and administered by relational database management and administration technologies, and offered on a scalable server that supports thousands of users.

Figure 1–1 shows the *inter*Media architecture from a 3-tier perspective: database server tier -- Oracle9*i*; application server tier -- Internet Application Server (*i*AS); and client tier -- thin and thick Java clients.

Through the use of *inter*Media, Oracle9*i* holds rich content in tables along with traditional data. Through the Oracle9*i* Java Virtual Machine (JVM), a server-side media parser is supported as well as an image processor. The media parser has object-oriented and relational interfaces, supports format and application metadata parsing, and includes a registry for new formats and extensions. The image processor includes Java Advanced Imaging (JAI) and provides image processing for producing thumbnail-sized images, for converting images, and image indexing and matching.

Beginning with Oracle9*i*, *inter*Media supports a heterogeneous media column, known as the ORDDoc object type. This allows a column to hold a mixture of image, audio, and video data, or other heterogeneous media data. Using *inter*Media import and export methods for each object type and for the relational interface, import and export between media objects and operating system files (external file storage) is possible. *inter*Media also supports special delivery types of servers, such as streaming content from an Oracle database. Using the Oracle *inter*Media plug-in for RealServer G2 6.0, 7.0, or 8.0, the RealServer G2 can stream multimedia data to a client directly out of the Oracle9*i* database using rtsp and iip protocols. In addition, media content indexing generators run external to the database.

In the middle tier, the Internet Application Server (*i*AS) or other Web server, provides access to *inter*Media through Oracle *inter*Media Java Classes, which enables Java applications on any tier (client, application server, or database server) to access, manipulate, and modify audio, image, and video data stored in Oracle9*i*. *inter*Media Java Classes makes it possible for JDBC result sets to include both traditional relational data and *inter*Media media objects (OrdAudio, OrdDoc,

OrdImage, and OrdVideo). This support enables applications to easily select and operate on a result set that contains *inter*Media columns plus other relational data. These classes also enable access to *inter*Media object attributes and invocation of *inter*Media object methods.

In addition, Oracle *inter*Media Java classes for servlets and JSPs facilitates the upload and retrieval of multimedia data stored in an Oracle9*i* database using the *inter*Media ORDAudio, ORDDoc, ORDImage, and ORDVideo object types. Oracle *inter*Media Java classes for servlets and JSPs can access data stored in the *inter*Media objects or BLOBs directly.

On the client tier, the browser-based *inter*Media Clipboard is provided and uses the webdav-enabled HTTP protocol for communication with the application server (*i*AS) tier. For thick clients and tools, client-side media processing and media parsing is supported through JAI, and the Java Media Framework (JMF). With Business Components for Java (BC4J), Oracle JDeveloper's programming framework can build scalable, multitier database applications from reusable business components.

Figure 1–1 interMedia Architecture

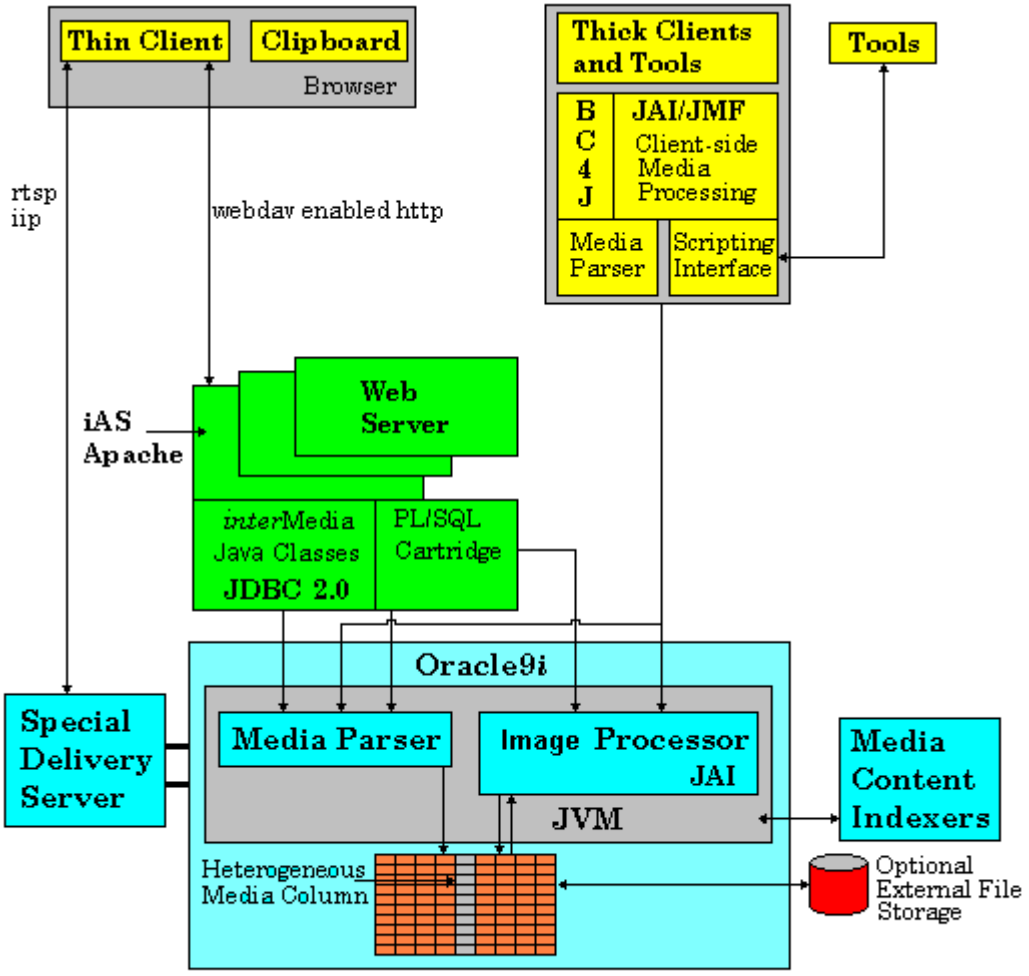


Table 1–1 describes the *inter*Media services and features for specified operating systems and releases of Oracle9*i*.

Table 1–1 interMedia Services and Features -- Supported Systems and Oracle9i Releases

Form of Distribution[1]	*inter*Media Services and Features	Operating Systems and Platforms[2]				Release		
		Solaris	Linux	Windows NT	Macintosh	8.1.5	8.1.6 or 8.1.7	9.0.1
CD-ROM	*inter*Media server-side	Yes	Yes	Yes	No	Yes	Yes	Yes
CD-ROM	Java classes	Yes	Yes	Yes	No	No	Yes	Yes
OTN	Java Classes	Yes	Yes	Yes	No	Yes	Yes	No
OTN and/or CD-ROM	Java Classes for Servlets and JSPs	Yes	Yes	Yes	No	No	Yes[3]	Yes[4]
OTN	Clipboard (Release 2)	Yes	Yes-	Yes	No	No	No	Yes
OTN CD-ROM	Annotator utility	No	No	Yes	Yes (MacOS 8.6)	Yes	Yes	Yes[5]
OTN CD-ROM	MediaFinder	Yes	No	Yes	No	Yes	Yes	Yes[6]
OTN	Plug-in for RealServer G2 6.0, 7.0 or 8.0	Yes	Yes	Yes	No	Yes	Yes	Yes
OTN	Plug-in for Macromedia UltraDev	No	No	Yes	Yes	No	Yes[7]	Yes
CD-ROM	Locator	Yes	Yes	Yes	No	Yes	Yes	Yes
CD-ROM	Generic Geocoding interface[8]	Yes	Yes	Yes	No	No	Yes	Yes
OTN	Custom DataSource and DataSink for JMF 2.0[9]	YesYes	No	Yes	No	No	Yes	Yes

Table 1–1 interMedia Services and Features -- Supported Systems and Oracle9i Releases (Cont.)

Form of Distribution[1]	*inter*Media Services and Features	Operating Systems and Platforms[2]				Release		
		Solaris	Linux	Windows NT	Macintosh	8.1.5	8.1.6 or 8.1.7	9.0.1
CD-ROM	BFILE and BLOB Stream Adaptors for JAI	Yes	Yes	Yes	No	No	No	Yes
OTN / CD-ROM	*inter*Media Relational interface	Yes	Yes	Yes	No	No	Yes[10]	Yes[11]

1 Oracle software is distributed from CD-ROM or OTN -- Oracle Technology Network Web site: `http://otn.oracle.com/`

2 *inter*Media server and client software are available on many other platforms; the platforms shown in this table describe only the ones on which the respective *inter*Media services and features listed are known to run.

3 Available on OTN for release 8.1.7.

4 Available for release 9.0.1 CD-ROM only.

5 Available for release 9.0.1 CD-ROM only.

6 Available for release 9.0.1 CD-ROM only.

7 Available on OTN for release 8.1.7.

8 Generic geocoding client written in Java, is embedded in Oracle9*i* database as a JSP, and published using PL/SQL interface.

9 Requires JMF V2.0, Oracle JDBC 8.1.5 or later. JDK version 1.1.*n*.

10 Available on OTN for release 8.1.7.

11 Available for release 9.0.1 CD-ROM only.

Section 1.13.1 through Section 1.13.6 describe the additional *inter*Media services and features that comprise *inter*Media.

1.13.1 Oracle *inter*Media Java Classes

Oracle *inter*Media Java Classes enables Java applications on any tier (client, application server, or database server) to manipulate and modify audio, image, and video data, or heterogeneous media data stored in Oracle9*i*. *inter*Media Java Classes makes it possible for JDBC result sets to include both traditional relational data and *inter*Media media objects. This support enables applications to easily select and operate on a result set that contains sets of *inter*Media columns plus other relational data. These classes also enable access to object attributes and invocation of object methods. See *Oracle interMedia Java Classes User's Guide and Reference* for more information.

1.13.2 Oracle *inter*Media Java Classes for Servlets and JSPs

Oracle *inter*Media Java Classes for servlets and JSPs facilitates the upload and retrieval of multimedia data stored in an Oracle9*i* database using the *inter*Media ORDAudio, ORDDoc, ORDImage, and ORDVideo object types. Oracle *inter*Media Java Classes for servlets and JSPs accesses data stored in the *inter*Media object types using Oracle *inter*Media Java Classes. However, Oracle *inter*Media Java Classes for servlets and JSPs can also be used to handle upload and retrieval of data using BLOBs directly.

The OrdHttpResponseHandler class facilitates the retrieval of multimedia data from an Oracle9*i* database and its delivery to a browser or other HTTP client from a Java servlet. The OrdHttpJspResponseHandler class provides the same features for Java Server Pages (JSPs).

Note: JSP engines are not required to support access to the servlet binary output stream. Therefore, not all JSP engines support the delivery of multimedia data using the OrdHttpJspResponseHandler class. See *Oracle interMedia Java Classes User's Guide and Reference* for more information.

Form-based file uploading using HTML forms encodes form data and uploaded files in POST requests using the multipart/form-data format. The OrdHttpUploadFormData class facilitates the processing of such requests by parsing the POST data and making the contents of regular form fields and the contents of uploaded files readily accessible to a Java servlet or Java Server Page. The handling of uploaded files is facilitated by the OrdHttpUploadFile class, which provides an easy-to-use API that applications call to load audio, image, and video data, or heterogeneous media data into a database.

To read the Javadoc documentation that describes how to use *inter*Media Java Classes for Servlets and JSPs, expand the zip file:

```
<ORACLE_HOME>/ord/http/doc/ordhttpdoc.zip (on Unix)
<ORACLE_HOME>\ord\http\doc\ordhttpdoc.zip (on Windows NT)
```

Also, see *Oracle interMedia Java Classes User's Guide and Reference* for more information.

1.13.3 Annotation Services for Multimedia Data

One application for which annotation services can be used is for constructing and operating a media archive. In the sections that follow, Oracle *inter*Media Annotator and a MediaFinder sample application are described. An annotation utility shows how content and format properties can be extracted from media data, collected as an annotation, stored in the database, and queried to locate media data based on the annotation's content. MediaFinder is a sample application that demonstrates how to build a media library.

*inter*Media Annotator Utility

Oracle *inter*Media Annotator is a utility that makes it easy to store and search for rich media content in Oracle9*i*. Oracle *inter*Media Annotator utility extracts content and format attributes from media sources (image, audio, and video files, audio CD, and URLs), and organizes the attributes into an XML formatted annotation. It lets you customize annotations to further describe the data, loads the annotation and the media data into Oracle9*i*, and allows you to index the annotation for powerful full text and thematic media searches using Oracle9*i* Text. Thus, the database can be queried to locate the media data based on the annotation's content. See *Oracle interMedia Annotator User's Guide* for more information.

MediaFinder - a Sample Application That Uses Oracle *inter*Media Annotator

MediaFinder is a sample application that demonstrates how to build a media library by using Oracle *inter*Media components. The open source code is provided to assist developers in building their own applications.

MediaFinder is an application that uses Oracle *inter*Media to let you search a video library built using Oracle *inter*Media Annotator. MediaFinder allows searching by movie title or by keyword, retrieving movie annotation information along with the video clip, and launching QuickTime to play the video. During a keyword search that will result in text sample matches, MediaFinder will locate the point where the match occurred, and allow you to start the playback from that point.

With the Apple QuickTime-For-Java library, *inter*Media Annotator can extract video frames as well as the text-track samples from the specified QuickTime movie. Consequently, MediaFinder can enrich the result set of your keyword search by retrieving the video frame that is closest to the matching text sample by means of timestamp comparisons.

MediaFinder uses Oracle9*i* Text to perform a text search against an XML document as well as a plain text string. For more information, refer to the Oracle9*i* Text information provided on the Oracle Technology Network Web site:

`http://otn.oracle.com/`

MediaFinder also uses Oracle *inter*Media ORDImage and ORDVideo objects for the storage of images and video in the Oracle9*i* database.

MediaFinder has a graphical user interface that allows you to use a Web interface to search a video library for a specific text sample, and retrieve the video frames associated with each text sample.

See *"MediaFinder" - a Sample Application That Uses Oracle interMedia Annotator Utility Readme for Installation, Configuration, and Use* for more information, which can be found on the Oracle *inter*Media Utilities and Plugins section of the Oracle Technology Network Web site:

`http://otn.oracle.com/`

1.13.4 Streaming Content from an Oracle Database

You can stream content stored in an Oracle database using an Oracle *inter*Media plug-in that supports the streaming server, and deliver this content for play on a client that uses the browser-supported streaming player.

Oracle *inter*Media Plug-in for RealServer G2 6.0, 7.0, or 8.0

Oracle *inter*Media Plug-in for RealServer G2 6.0, 7.0, or 8.0 allows RealServer G2 to stream multimedia data to a client directly out of the Oracle9*i* database. This plug-in is installed in RealServer G2 and defined in the RealServer G2 configuration file. The data is requested with a URL, which contains information necessary to select the multimedia data from the database.

For information on RealNetwork RealServer G2 Streaming Server, see the following URL:

`http://www.real.com/`

See *Oracle interMedia Plug-in for RealNetworks G2 Streaming Server Readme for Installation and Configuration* for more information. The Oracle *inter*Media Plug-in for RealServer G2 can be downloaded from the Oracle *inter*Media Utilities and Plugins section of the Oracle Technology Network Web site:

`http://otn.oracle.com/`

1.13.5 Support for Web Technologies

Using *inter*Media support for Web technologies, you can easily integrate multimedia data into Web and Java applications. You can also store, retrieve, and manage rich media content in an Oracle9*i* database.

Oracle *inter*Media Clipboard Features

The *inter*Media Clipboard (Version 2) enables users to access Oracle *inter*Media data from the Web. You can configure the Clipboard to enable access in the following ways:

- Through the Clipboard Web browser interface
- Through the OraDav programming interface

See *Setting Up the Oracle interMedia Clipboard, Version 2* for more information.

The Clipboard can be downloaded from the Oracle *inter*Media Utilities and Plugins section of the Oracle Technology Network Web site:

`http://otn.oracle.com/`

1.13.6 Geocoding Services

Geocoding represents addresses and locations of interest (postal codes, demographic regions, and so forth) as geometric factors (points). These enable distances to be calculated and sites to be represented graphically in Web, data warehousing, customer information system, and enterprise resource planning applications. Geocoding services can be used to add the exact location (latitude and longitude) of points of interest to existing data files stored in Oracle9*i*.

A geocoding service is used for converting tables of address data into standardized address, location, and possibly other data.

Oracle9*i* Locator

Oracle9*i* Locator is an Internet-ready tool developed exclusively to support standalone and online geocoding and Internet mapping requirements. Geocoded business information provides a necessary step in cleansing, enhancing, and visualizing customer records. Such information is proving vital in data warehousing, customer information systems, electronic commerce, and enterprise resource planning. In addition to geocoding support, Oracle9*i* Locator provides the technology that enables the deployment of simple, easy-to-use Internet-based mapping applications.

Oracle9*i* Locator enables Oracle9*i* to support online Internet-based geocoding facilities for locator applications and proximity queries.

Oracle9*i* Locator supports the leading online and batch geocoding services including MapXtreme from MapInfo Corporation, Centrus from Qualitative Marketing Software, MapQuest destination information solutions from MapQuest.com ("MapQuest"), and GeoZip from whereonearth.com Ltd.

MapInfo Corporation, Qualitative Marketing Software, MapQuest.com, and whereonearth.com currently provide the online and batch geocoding services for the Locator features. Each service offers a number of free geocoding calls at its Web site for trial purposes for online geocoding, and geocoding service software for batch geocoding. Locator users need to consent to the vendor policies and possibly register with them:

MapInfo Corporation: `http://www.MapMarker.com/`

Qualitative Marketing Software: `http://www.centrus-software.com/oracle/`

MapQuest.com: `http://www.mapquest.com/`

whereonearth.com: `http://www.whereonearth.com/`

During registration for online geocoding services, you are asked to create your own user ID and password. Please make a note of them for embedding into your sample geocoding service because the user ID / password combination is required for each geocoding call. Your free account is limited to a small number of address records per day.

Should you require the ability to geocode larger data sets, or for further information, contact:

- MapInfo technologies to complement your Oracle solution; call 1.800.FASTMAP (1.800.327.8627); or send e-mail to custserv@mapinfo.com (see their Web site for more specific geographic contact information)
- QMSoft technologies to complement your Oracle solution; call QMSoft at 1.800.782.7988; or send e-mail to oracle@qmsoft.com
- MapQuest.com technologies to complement your Oracle solution; call 1.888.MAPQUEST (1.888.627.7837) or in Europe, (31) 70.426.2660; or send e-mail to info@mapquest.com
- whereonearth.com technologies to complement your Oracle solution; call +44 (0) 207 246 1400; or send e-mail to enquiries@whereonearth.com

These companies' Web sites will also have detailed documentation about the vendor-specific parameter information of the Locator features, such as match code

or error code. Because Oracle provides an interface to facilitate the geocoding functions, you should contact the vendors with your questions.

See Oracle9*i* Locator Release Notes (depending on your operating system, <*ORACLE_HOME*>/md/doc/README.txt) for additional information about the geocoding services provided by these Oracle partners.

Oracle9*i* Locator also supports server-based geocoding and data scrubbing operations for data warehouse applications.

Using simple location queries, Oracle9*i* Locator allows Web and other applications to retrieve information based on distance. For example, using a set of geocoded address data and simple query-by-text or query-by-map operations, users can use a Web browser-based application, enter a distance, and identify the nearest location from a specific address or reference point on a map. For example, Oracle9*i* Locator applications can help you locate stores, offices, distribution points, and other points of interest based on their distance from a given postal (zip) code, address, or other reference point.

See the <*ORACLE_HOME*>/md/doc/LOCATOR_README.txt file or <*ORACLE_HOME*>/md/doc/LOCATOR_README.htm file for more information.

These features enable database designers to extend existing application databases with geocoded, spatial-point data, or to build new geocoded spatial-point applications. Web application developers can build specialized Web-enabled Oracle9*i* Locator applications.

Oracle9*i* Locator is Web-based and requests are formatted in HTTP. Thus, each request in SQL must contain the URL of the Web site, proxy for the firewall (if any), and user account information on the service provider's Web site. An HTTP approach potentially limits the utility or practicality of the service when dealing with large tables or undertaking frequent updates to the base address information. In such situations, it is preferable to use a batch geocoding service made available within an Intranet or local area network. The next section describes the interface for a facility that potentially contains this existing Oracle9*i* Locator HTTP-based solution.

Generic Geocoding Interface

A generic geocoding interface is available with Oracle Spatial for release 8.1.6 and later. This is a generic interface to third-party geocoding software that lets users geocode their address information stored in database tables, standardized addresses, and corresponding location information as instances of predefined object types. This interface is part of the geocoding framework in Oracle Spatial for release 8.1.6 and later and Oracle9*i* Locator.

This generic geocoding interface describes a set of interfaces and metadata schema that enables geocoding of an entire address table, or a single row. It also describes the procedures for inserting new or updated standardized address and spatial data into another table (or the same table). The third-party geocoding service is assumed to have been installed on a local network and to be accessible through standard communication protocols such as sockets or HTTP.

The generic geocoding client is written in Java and embedded in the Oracle9*i* database as a Java stored procedure. A fast, scalable, highly available, and secure Java Virtual Machine (Java VM or JVM) is integrated in the Oracle9*i* database server. The Java VM provides an ideal platform for enterprise applications written in Java as Java stored procedures, Enterprise JavaBeans (EJBs), or Java Methods of Oracle9*i* object types.

Java stored procedures are published using the PL/SQL interface; thus, the generic geocoding interface can be compatible with existing Locator APIs.

The stored procedures have an interface, oracle.spatial.geocoder, that must be implemented by each vendor whose geocoder is integrated with Oracle Spatial and Oracle9*i* Locator. The procedures also require certain object types to be defined and metadata tables to be populated. The object types, metadata schema, and geocoder interface are described in <*ORACLE_HOME*>/md/doc/LOCATOR_README.txt file or <*ORACLE_HOME*>/md/doc/LOCATOR_README.htm file and *Oracle Spatial User's Guide and Reference*.

2

Content-Based Retrieval Concepts

This chapter explains, at a high level, why and how to use content-based retrieval. It covers the following topics:

- Overview and benefits of content-based retrieval
- How content-based retrieval works, including definitions and explanations of the visual attributes (color, texture, shape, location) and why you might emphasize specific attributes in certain situations
- Image matching using a specified comparison image, including comparing how the weights of visual attributes determine the degree of similarity between images
- Use of indexing to improve search and retrieval performance
- Image preparation or selection to maximize the usefulness of comparisons

2.1 Overview and Benefits

Inexpensive image-capture and storage technologies have allowed massive collections of digital images to be created. However, as an image database grows, the difficulty of finding relevant images increases. Two general approaches to this problem have been developed, both of which use metadata for image retrieval:

- Using information manually entered or included in the table design, such as titles, descriptive keywords from a limited vocabulary, and predetermined classification schemes
- Using automated image feature extraction and object recognition to classify image content -- that is, using capabilities unique to content-based retrieval

With *inter*Media, you can combine both approaches in designing a table to accommodate images: use traditional text columns to describe the semantic

significance of the image (for example, that the pictured automobile won a particular award, or that its engine has six or eight cylinders), and use the ORDImage type for the image, to permit content-based queries based on intrinsic attributes of the image (for example, how closely its color and shape match a picture of a specific automobile).

As an alternative to defining image-related attributes in columns separate from the image, a database designer could create a specialized composite data type that combines *inter*Media and the appropriate text, numeric, and date attributes.

The primary benefit of using content-based retrieval is reduced time and effort required to obtain image-based information. With frequent adding and updating of images in massive databases, it is often not practical to require manual entry of all attributes that might be needed for queries, and content-based retrieval provides increased flexibility and practical value. It is also useful in providing the ability to query on attributes such as texture or shape that are difficult to represent using keywords.

Examples of database applications where content-based retrieval is useful -- where the query is semantically of the form, “find objects that look like this one” -- include:

- Trademarks, copyrights, and logos
- Art galleries and museums
- Retailing
- Fashion and fabric design
- Interior design or decorating

For example, a Web-based interface to a retail clothing catalog might allow users to search by traditional categories (such as style or price range) and also by image properties (such as color or texture). Thus, a user might ask for formal shirts in a particular price range that are off-white with pin stripes. Similarly, fashion designers could use a database with images of fabric swatches, designs, concept sketches, and finished garments to facilitate their creative processes.

2.2 How Content-Based Retrieval Works

A content-based retrieval system processes the information contained in image data and creates an abstraction of its content in terms of visual attributes. Any query operations deal solely with this abstraction rather than with the image itself. Thus,

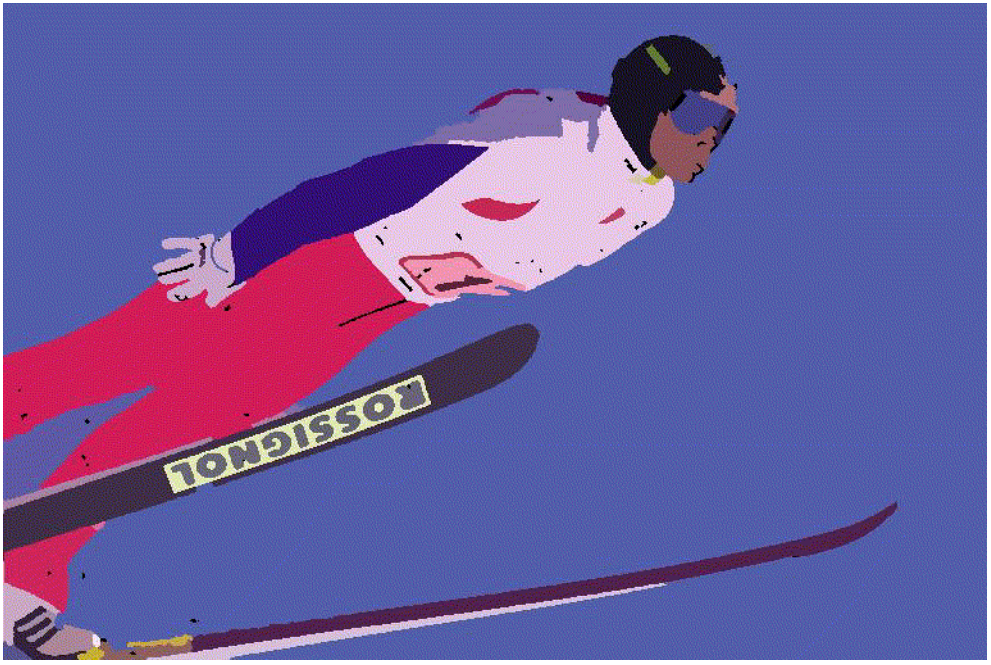
every image inserted into the database is analyzed, and a compact representation of its content is stored in a feature vector, or **signature**.

The signature for the image in Figure 2–1 is extracted by segmenting the image into regions based on color as shown in Figure 2–2. Each region has associated with it color, texture, and shape information. The signature contains this region-based information along with global color, texture, and shape information to represent these attributes for the entire image. In Figure 2–2, there are a total of 55 shapes (patches of connected pixels with similar color) in this segmented image. In addition, there is also a "background" shape, which consists of small disjoint dark patches. These tiny patches (usually having distinct colors) do not belong to any of their adjacent shapes and are all classified into a single "background" shape. This background shape is also taken into consideration for image retrieval.

Figure 2–1 Unsegmented Image



Figure 2–2 Segmented Image



Images are matched based on the color, texture, and shape attributes. The positions of these visual attributes in the image are represented by location. Location by itself is not a meaningful search parameter, but in conjunction with one of the three visual attributes represents a search where the visual attribute and the location of that visual attribute are both important.

The signature contains information about the following visual attributes:

- **Color** represents the distribution of colors within the entire image. This distribution includes the amounts of each color, but not the locations of colors.
- **Texture** represents the low-level patterns and textures within the image, such as graininess or smoothness. Unlike shape, texture is very sensitive to features that appear with great frequency in the image.
- **Shape** represents the shapes that appear in the image, as determined by color-based segmentation techniques. A shape is characterized by a region of uniform color.

- **Location** represents the positions of the shapes, color, and texture components. For example, the color blue could be located in the top half of the image. A certain texture could be located in the bottom right corner of the image.

Feature data for all these visual attributes is stored in the signature, whose size typically ranges from 3000 to 4000 bytes. For better performance with large image databases, you can create an index based on the signatures of your images. See Section 2.4 for more information on indexing.

Images in the database can be retrieved by matching them with a comparison image. The comparison image can be any image inside or outside the current database, a sketch, an algorithmically generated image, and so forth.

The matching process requires that signatures be generated for the comparison image and each image to be compared with it. Images are seldom identical, and therefore matching is based on a similarity-measuring function for the visual attributes and a set of weights for each attribute. The **score** is the relative distance between two images being compared. The score for each attribute is used to determine the degree of similarity when images are compared, with a smaller distance reflecting a closer match, as explained in Section 2.3.3.

2.2.1 Color

Color reflects the distribution of colors within the entire image.

Color and location specified together reflect the color distributions *and* where they occur in the image. To illustrate the relationship between color and location, consider Figure 2–3.

Figure 2–3 Image Comparison: Color and Location

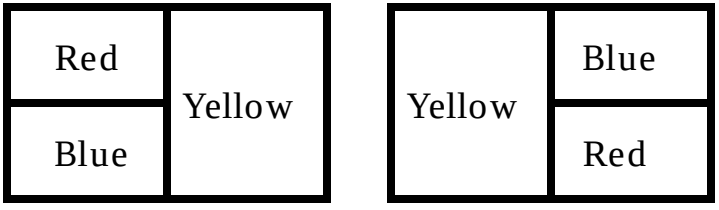


Image 1 and Image 2 are the same size and are filled with solid colors. In Image 1, the top left quarter (25%) is red, the bottom left quarter (25%) is blue, and the right half (50%) is yellow. In Image 2, the top right quarter (25%) is blue, the bottom right quarter (25%) is red, and the left half (50%) is yellow.

If the two images are compared first solely on color and then color and location, the following are the similarity results:

- Color: complete similarity (score = 0.0), because each color (red, blue, yellow) occupies the same percentage of the total image in each one
- Color and location: no similarity (score = 100), because there is no overlap in the placement of any of the colors between the two images

Thus, if you need to select images based on the dominant color or colors (for example, to find apartments with blue interiors), give greater relative weight to color. If you need to find images with common colors in common locations (for example, red dominant in the upper portion to find sunsets), give greater relative weight to location.

Figure 2–4 shows two images very close in color. Figure 2–5 shows two images very close in both color and location.

Figure 2–4 Images Very Similar in Color

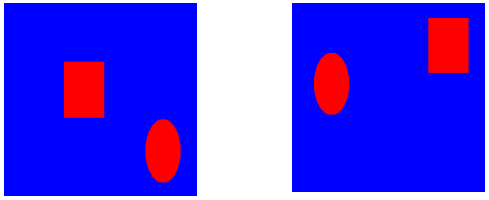


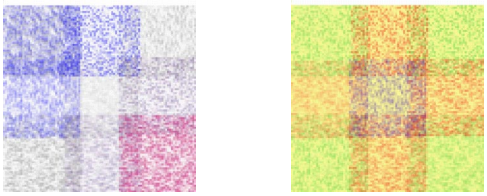
Figure 2–5 Images Very Similar in Color and Location



2.2.2 Texture

Texture reflects the texture of the entire image. Texture is most useful for full images of textures, such as catalogs of wood grains, marble, sand, or stones. These images are generally hard to categorize using keywords alone because our vocabulary for textures is limited. Texture can be used effectively alone (without color) for pure textures, but also with a little bit of color for some kinds of textures, like wood or fabrics. Figure 2–6 shows two similar fabric samples.

Figure 2–6 Fabric Images with Similar Texture



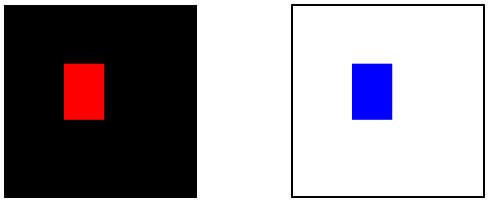
Texture and location specified together compare texture and location of the textured regions in the image.

2.2.3 Shape

Shape represents the shapes that appear in the image. Shapes are determined by identifying regions of uniform color.

Shape is useful to capture objects such as horizon lines in landscapes, rectangular shapes in buildings, and organic shapes such as trees. Shape is very useful for querying on simple shapes (like circles, polygons, or diagonal lines) especially when the query image is drawn by hand and color is not considered important when the drawing is made. Figure 2–7 shows two images very close in shape.

Figure 2–7 Images with Very Similar Shape



Shape and location specified together compare shapes and location of the shapes in the images.

2.3 How Matching Works

When you match images, you assign an importance measure, or weight, to each of the visual attributes, and *inter*Media calculates a similarity measure for each visual attribute.

2.3.1 Weight

Each **weight** value reflects how sensitive the matching process for a given attribute should be to the degree of similarity or dissimilarity between two images. For example, if you want color to be completely ignored in matching, assign a weight of 0.0 to color; in this case, any similarity or difference between the color of the two images is totally irrelevant in matching. On the other hand, if color is extremely important, assign it a weight greater than any of the other attributes; this will cause any similarity or dissimilarity between the two images with respect to color to contribute greatly to whether or not the two images match.

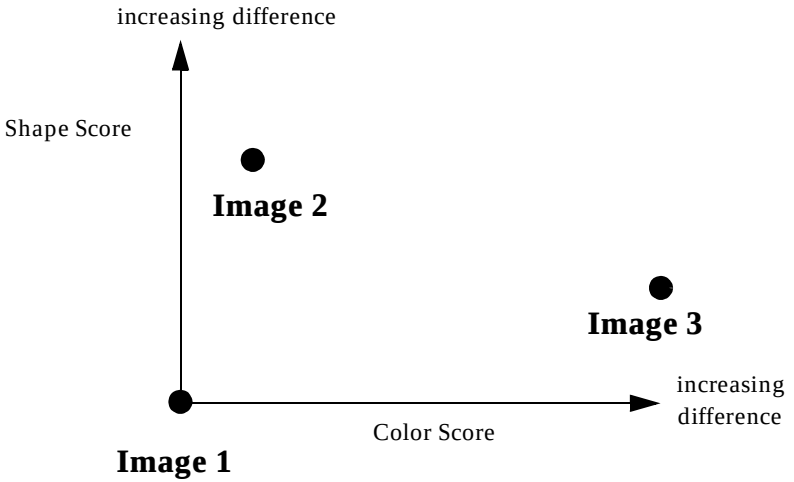
Weight values can be between 0.0 and 1.0. During processing, the values are normalized such that they total 1.0. The weight of at least one of the color, texture, or shape attributes must be set to greater than zero. See Section 2.3.3 for details of the calculation.

2.3.2 Score

The similarity measure for each visual attribute is calculated as the **score** or distance between the two images with respect to that attribute. The score can range from 0.00 (no difference) to 100.0 (maximum possible difference). Thus, the more similar two images are with respect to a visual attribute, the *smaller* the score will be for that attribute.

As an example of how distance is determined, assume that the dots in Figure 2–8 represent scores for three images with respect to two visual attributes, such as color and shape, plotted along the x-axis and y-axis of a graph.

Figure 2–8 Score and Distance Relationship



For matching, assume Image 1 is the comparison image, and Image 2 and Image 3 are each being compared with Image 1. With respect to the color attribute plotted on the x-axis, the distance between Image 1 and Image 2 is relatively small (for example, 15), whereas the distance between Image 1 and Image 3 is much greater (for example, 75). If the color attribute is given more weight, then the fact that the two distance values differ by a great deal will probably be very important in determining whether or not Image 2 and Image 3 match Image 1. However, if color is minimized and the shape attribute is emphasized instead, then Image 3 will match Image 1 better than Image 2 matches Image 1.

2.3.3 Similarity Calculation

In Section 2.3.2, Figure 2–8 showed a graph of only two of the attributes that *inter*Media can consider. In reality, when images are matched, the degree of similarity depends on a weighted sum reflecting the weight and distance of all three of the visual attributes in conjunction with location of the comparison image and the test image.

For example, assume that for the comparison image (Image 1) and one of the images being tested for matching (Image 2), Table 2–1 lists the relative distances between the two images for each attribute. Note that you would never see these individual numbers unless you computed three separate scores, each time highlighting one attribute and setting the others to zero. For simplicity, the three attributes are not considered in conjunction with location in this example.

Table 2–1 Distances for Visual Attributes Between Image1 and Image2

Visual Attribute	Distance
Color	15
Texture	5
Shape	50

In this example, the two images are most similar with respect to texture (distance = 5) and most different with respect to shape (distance = 50).

Assume that for the matching process, the following weights have been assigned to each visual attribute:

- Color = 0.7
- Texture = 0.2
- Shape = 0.1

The weights are supplied in the range of 0.0 to 1.0. Within this range, a weight of 1 indicates the strongest emphasis, and a weight of 0 means the attribute should be ignored. The values you supply are automatically normalized such that the weights total 1.0, still maintaining the ratios you have supplied. In this example, the weights were specified such that normalization was not necessary.

The following formula is used to calculate the weighted sum of the distances, which is used to determine the degree of similarity between two images:

```
weighted_sum = color_weight    * color_distance +
               texture_weight  * texture_distance +
               shape_weight    * shape_distance+
```

The degree of similarity between two images in this case is computed as:

0.7*c_distance + 0.2*tex_distance + 0.1*shape_distance

Using the supplied values, this becomes:

(0.7*15 + 0.2*5 + 0.1*50) = (10.5 + 1.0 + 5.0) = 16.5

To illustrate the effect of different weights in this case, assume that the weights for color and shape were reversed. In this case, the degree of similarity between two images is computed as:

0.1*c_distance +0.2*tex_distance + 0.7*shape_distance

That is:

(0.1*15 + 0.2*5 + 0.7*50) = (1.5 + 1.0 + 35.0) = 37.5

In this second case, the images are considered to be less similar than in the first case, because the overall score (37.5) is greater than in the first case (16.5). Whether or not the two images are considered matching depends on the threshold value (explained in Section 2.3.4). If the weighted sum is less than or equal to the threshold, the images match; if the weighted sum is greater than the threshold, the images do not match.

In these two cases, the *correct* weight assignments depend on what you are looking for in the images. If color is extremely important, then the first set of weights is a better choice than the second set of weights, because the first set of weights grants greater significance to the disparity between these two specific images with respect to color. The two images differ greatly in shape (50) but that difference contributes less to the final score because the weight assigned to the attribute shape is low. With the second set of weights, the images have a higher score when shape is assigned a higher weight and the images are less similar with respect to shape than with respect to color.

2.3.4 Threshold Value

When you match images, you assign a **threshold** value. If the weighted sum of the distances for the visual attributes is less than or equal to the threshold, the images match; if the weighted sum is greater than the threshold, the images do not match.

Using the examples in Section 2.3.3, if you assign a threshold of 20, the images do not match when the weighted sum is 37.5, but they do match when the weighted sum is 16.5. If the threshold is 10, the images do not match in either case; and if the threshold is 37.5 or greater, the images match in both cases.

The following example shows a cursor (getphotos) that selects the photo_id, annotation, and photo from the Pictures table where the threshold value is 20 for comparing photographs with a comparison image:

```
CURSOR getphotos IS
 SELECT photo_id, annotation, photo FROM Pictures WHERE
```

```
ORDSYS.IMGSimilar(photo_sig, comparison_sig, 'color="0.4",
texture="0.10", shape="0.3", location="0.2"', 20)=1;
```

Before the cursor executes, the generateSignature( ) method must be called to compute the signature of the comparison image (comparison_sig), and to compute signatures for each image in the table. Chapter 8 describes all the operators, including IMGSimilar and IMGScore.

The number of matches returned generally increases as the threshold increases. Setting the threshold to 100 would return all images as matches. Such a result, of course, defeats the purpose of content-based retrieval. If your images are all very similar, you may find that even a threshold of 50 returns too many (or all) images as matches. Through trial and error, adjust the threshold to an appropriate value for your application.

You will probably want to experiment with different weights for the visual attributes and different threshold values, to see which combinations retrieve the kinds and approximate number of matches you want.

2.4 Using an Index to Compare Signatures

A domain index, or extensible index, is an approach for supporting complex data objects. The Oracle database and *inter*Media cooperate to define, build, and maintain an index for image data. This index is of type ORDImageIndex. Once it is created, the index automatically updates itself every time an image is inserted or removed from the database table. The index is created, managed, and accessed by routines supplied by the index type.

For better performance with large image databases, you should always create and use an index for searching through the image signatures. The default search model compares the signature of the query image to the signatures of all images stored in the database. This works well for simple queries against a few images such as, "Does this picture of an automobile match the image stored with the client's insurance records?" However, if you want to compare that image with thousands or millions of images to determine what kind of vehicle it is, then a linear search though the database would be impractical. In this case, an index based on the image signatures would greatly improve performance.

Assume you have a table T containing fabric ID numbers and pattern photographs and signatures:

```
CREATE TABLE T (fabric_id NUMBER, pattern_photo ORDSYS.ORDIMAGE, pattern_
signature ORDSYS.ORDImageSignature);
```

Load the table with images, and process each image using the generateSignature( ) method to generate the signatures.

> **Note:** Performance is greatly improved by loading the data tables prior to creating the index.

Once the signatures are created, the following command creates an index on this table, based on the data in the `pattern_photo` column.

```
CREATE INDEX idx1 ON T(pattern_signature) INDEXTYPE IS ORDSYS.ORDIMAGEINDEX
      PARAMETERS ('ORDImage_Filter_Tablespace = <name>,ORDImage_Index_Tablespace =
<name>');
```

The index name is limited to 24 or fewer characters.[1] As with any Oracle table, do not use pound signs (#) or dollar signs ($) in the name. Also as usual, the tablespace must be created before creating the table.

The index data resides in two tablespaces. The first contains the actual index data, and the second is an internal index created on that data. See Section 3.3.11 for suggestions concerning the sizes of these tablespaces.

Finally, as with other Oracle indexes, you should analyze the new index as follows:

```
ANALYZE INDEX idx1 COMPUTE STATISTICS;
```

Two operators, IMGSimilar and IMGScore support queries using the index. The operators automatically use the index if it is present. See Section 8.2.3 for syntax information and examples.

2.5 Preparing or Selecting Images for Useful Matching

The human mind is infinitely smarter than a computer in matching images. If we are near a street and want to identify all red automobiles, we can easily do so because our minds rapidly adjust for the following factors:

- Whether the automobile is stopped or moving
- The distinction between red automobiles, red motorcycles, and red trailers
- The absolute size of the automobile, as well as its relative size in our field of vision (because of its distance from us)

[1] The standard Oracle restriction is 30 characters for table or index names. However, *inter*Media requires an extra 6 characters for internal processing of the domain index.

- The location of the automobile in our field of vision (center, left, right, top, bottom)
- The direction in which the automobile is pointing or traveling (left or right, toward us, or away from us)

However, for a computer to find red automobiles (retrieving all red automobiles and no or very few images that are not red or not automobiles), it is helpful if all the automobile images have the automobile occupy almost the entire image, have no extraneous elements (people, plants, decorations, and so on), and have the automobiles pointing in the same direction. In this case, a match emphasizing color and shape would produce useful results. However, if the pictures show automobiles in different locations, with different relative sizes in the image, pointing in different directions, and with different backgrounds, it will be difficult to perform content-based retrieval with these images.

The following are some suggestions for selecting images or preparing images for comparison. The list is not exhaustive, but the basic principle to keep in mind is this: Know what you are looking for, and use common sense. If possible, crop and edit images in accordance with the following suggestions before performing content-based retrieval:

- Have what you expect to be looking for occupy almost all the image space, or at least occupy the same size and position on each image. For example, if you want to find all the red automobiles, each automobile image should show only the automobile and should have the automobile in approximately the same position within the overall image.
- Minimize any extraneous elements that might prevent desired matches or cause unwanted matches. For example, if you want to match red automobiles and if each automobile has a person standing in front of it, the color, shape, and position of the person (skin and clothing) will cause color and shape similarities to be detected, and might reduce the importance of color and shape similarities between automobiles (because part of the automobile is behind the person and thus not visible). If you know that your images vary in this way, experiment with different thresholds and different weights for the various visual attributes until you find a combination that provides the best result set for your needs.
- During analysis, images are temporarily scaled to a common size such that the resulting signatures are based on a common frame of reference. If you crop a section of an image, and then compare that piece back to the original,

*inter*Media will likely find that the images are less similar than you would expect.

> **Note:** *inter*Media has a fuzzy search engine, and is not designed to do correlations. For example, *inter*Media cannot find a specific automobile in a parking lot. However, if you crop an individual automobile from a picture of a parking lot, you can then compare the automobile to known automobile images.

- When there are several objects in the image, *inter*Media matches them best when:
  - The colors in the image are distinct from each other. For example, an image of green and red as opposed to an image of dark green and light green.
  - The color in adjacent objects in the image contrast with each other.
  - The image consists of a few, simple shapes.

3

*inter*Media Examples

This chapter provides examples that show common operations with Oracle *inter*Media. Examples are presented by audio, media, image, and video data groups followed by a section that describes how to extend *inter*Media to support a new data source.

3.1 Audio Data Examples

Audio data examples using *inter*Media include the following common operations:

- Defining a song object named songObject
- Creating an object table named SongsTable
- Creating a list object named songList that contains a list of songs
- Defining the implementation of the songList object
- Creating a CD object and CdTable table
- Inserting a song into the SongsTable table
- Inserting a CD into the CdTable table
- Loading a song into the SongsTable table
- Inserting a reference to a song object into the songs list in the CdTable table
- Adding a CD reference to a song
- Retrieving audio data from a song in a CD
- Extending *inter*Media to support a new audio data format
- Extending *inter*Media with new object types
- Using *inter*Media with object views

- Using a set of scripts for creating and populating an audio table from a BFILE data source

Reference information on the methods used in these examples is presented in Chapter 6.

3.1.1 Defining a Song Object

Example 3–1 describes how to define a Song object.

Example 3–1 Define a Song Object

```
CREATE TYPE songObject as OBJECT (
  cdRef       REF CdObject,   -- REF into the cd table
  songId      VARCHAR2(20),
  title       VARCHAR2(4000),
  artist      VARCHAR2(4000),
  awards      VARCHAR2(4000),
  timePeriod  VARCHAR2(20),
  duration    INTEGER,
  clipRef     REF clipObject, -- REF into the clips table (music video)
  txtcontent  CLOB,
  audioSource ORDSYS.ORDAUDIO
);
```

3.1.2 Creating an Object Table SongsTable

Example 3–2 describes how to create an object table named SongsTable.

Example 3–2 Create a Table Named SongsTable

```
CREATE TABLE SongsTable of songObject (UNIQUE (songId), songId NOT NULL);
```

3.1.3 Creating a List Object Containing a List of References to Songs

Example 3–3 describes how to create a list object containing a list of references to songs.

Example 3–3 Create a List Object Containing a List of References to Songs

```
CREATE TYPE songNstType AS TABLE of REF songObject;

CREATE TYPE songList AS OBJECT (songs songNstType,
     MEMBER PROCEDURE addSong(s IN REF songObject));
```

3.1.4 Defining the Implementation of the songList Object

Example 3–4 describes how to define the implementation of the songList object.

Example 3–4 Define the Implementation of the songList Object

```
CREATE TYPE BODY songList AS
  MEMBER PROCEDURE addSong(s IN REF songObject)
  IS
    pos INTEGER := 0;
  BEGIN
    IF songs IS NULL THEN
      songs := songNstType(NULL);
      pos := 0;
    ELSE
      pos := songs.count;
    END IF;
    songs.EXTEND;
    songs(pos+1) := s;
  END;
END;
```

3.1.5 Creating a CD Object and a CD Table

This section describes how to create a CD object and a CD table of audio clips that includes, for each audio clip, the following information:

- Item ID
- CD DB ID
- CD title
- CD artist
- CD category
- Copyright
- Name of producer
- Awards
- Time period

- Rating
- Duration
- Text content
- Cover image
- Songs

Example 3–5 creates a CD object named CdObject, and a CD table named CdTable that contains the CD information.

Example 3–5 Create a CD Table Containing CD Information

```
CREATE TYPE CdObject as OBJECT (
  itemId        INTEGER,
  cddbID        INTEGER,
  title         VARCHAR2(4000),
  artist        VARCHAR2(4000),
  category      VARCHAR2(20),
  copyright     VARCHAR2(4000),
  producer      VARCHAR2(4000),
  awards        VARCHAR2(4000),
  timePeriod    VARCHAR2(20),
  rating        VARCHAR2(256),
  duration      INTEGER,
  txtcontent    CLOB,
  coverImg      REF ORDSYS.ORDImage,
  songs         songList);

CREATE TABLE CdTable OF CdObject (UNIQUE(itemId), itemId NOT NULL)
       NESTED TABLE songs.songs STORE AS song_store_table;
```

3.1.6 Inserting a Song into the SongsTable Table

Example 3–6 describes how to insert a song into the SongsTable table.

Example 3–6 Insert a Song into the SongsTable Table

```
-- Insert a song into the songs table
INSERT INTO SongsTable VALUES (NULL,
                               '00',
                               'Under Pressure',
                               'Queen',
                               'no awards',
                               '80-90',
```

```
                                    243,
                                    NULL,
                                    EMPTY_CLOB(),
                                    ORDSYS.ORDAudio.init());

-- Check songs insertion
SELECT s.title
FROM   SongsTable s
WHERE songId = '00';
```

3.1.7 Inserting a CD into the CdTable Table

Example 3–7 describes how to insert a CD into the CdTable table.

Example 3–7 Insert a CD into the CdTable Table

```
-- Insert a cd into the cd table
INSERT INTO CdTable VALUES (1, 23232323,
                            'Queen Classics',
                            'Queen',
                            'rock',
                            'BMV Company',
                            'BMV',
                            'Grammy',
                            '80-90',
                            'no rating',
                            4000,             -- in seconds
                            EMPTY_CLOB(),
                            NULL,
                            songList(NULL));

-- Check cd insertion
SELECT cd.title
FROM   Cdtable cd;
```

3.1.8 Loading a Song into the SongsTable Table

Example 3–8 describes how to load a song into the SongsTable table. This example requires an AUDDIR directory to be defined; see the comments in the example.

Example 3–8 Load a Song into the SongsTable Table

```
-- Load a Song into the SongsTable
```

```
-- Create your directory specification below
-- CREATE OR REPLACE DIRECTORY AUDDIR AS '/audio/';
-- GRANT READ ON DIRECTORY AUDDIR TO PUBLIC WITH GRANT OPTION;
DECLARE
        audioObj ORDSYS.ORDAUDIO;
        ctx RAW(4000) := NULL;
BEGIN
        SELECT S.audioSource INTO audioObj
        FROM   SongsTable S
        WHERE  S.songId = '00'
        FOR UPDATE;

        audioObj.setSource('file', 'AUDDIR', 'UnderPressure.au');
        audioObj.import(ctx);
        audioObj.setProperties(ctx);

        UPDATE SongsTable S
        SET    S.audioSource = audioObj
        WHERE  S.songId = '00';
        COMMIT;
END;

-- Check song insertion
DECLARE
     audioObj ORDSYS.ORDAUDIO;
     ctx RAW(4000) := NULL;
BEGIN
     SELECT S.audioSource INTO audioObj
     FROM   SongsTable S
     WHERE  S.songId = '00';

     dbms_output.put_line('Content Length: ' ||
                    audioObj.getContentLength(ctx));
     dbms_output.put_line('Content MimeType: ' ||
                    audioObj.getMimeType());
END;
```

3.1.9 Inserting a Reference to a Song Object into the Songs List in the CdTable Table

Example 3–9 describes how to insert a reference to a song object into the songs list in the CdTable table.

Example 3–9 Insert a Reference to a Song Object into the Songs List in the CdTable Table

```
-- Insert a reference to a SongObject into the Songs List in the CdTable Table
DECLARE
        songRef REF SongObject;
        songListInstance songList;
BEGIN
        SELECT REF(S) into songRef
        FROM   SongsTable S
        where  S.songId = '00';

        SELECT C.songs INTO songListInstance
        FROM   CdTable C
        WHERE  C.itemId = 1
        FOR UPDATE;

        songListInstance.addSong(songRef);

        UPDATE CdTable C
        SET    C.songs = songListInstance
        WHERE  C.itemId = 1;

        COMMIT;
END;

-- Check insertion of ref
-- This example works for the first entry inserted in the songList
DECLARE
        song             SongObject;
        songRef          REF SongObject;
        songListInstance songList;
        songType         songNstType;
BEGIN
        SELECT C.songs INTO songListInstance
        FROM   CdTable C
        WHERE  C.itemId = 1;

        SELECT songListInstance.songs INTO songType FROM DUAL;
        songRef := songType(1);
        SELECT DEREF(songRef) INTO song FROM DUAL;

        dbms_output.put_line('Song Title: ' ||
                             song.title);
END;
```

3.1.10 Adding a CD Reference to a Song

Example 3–10 describes how to add a CD reference to a song.

Example 3–10 Add a CD Reference to a Song

```
-- Adding a cd reference to a song
DECLARE
     songCdRef  REF CdObject;
BEGIN
     SELECT S.cdRef INTO songCdRef
     FROM   SongsTable S
     WHERE  S.songId = '00'
     FOR UPDATE;

     SELECT REF(C) INTO songCdRef
     FROM  CdTable C
     WHERE C.itemId = 1;

     UPDATE SongsTable S
     SET    S.cdRef = songCdRef
     WHERE  S.songId = '00';

     COMMIT;
END;

-- Check cd Ref
DECLARE
     cdRef REF CdObject;
     cd    CdObject;
BEGIN
     SELECT S.cdRef INTO cdRef
     FROM   SongsTable S
     WHERE  S.songId = '00';

     SELECT DEREF(cdRef) INTO cd FROM DUAL;
     dbms_output.put_line('Cd Title: ' ||
                          cd.title);
END;
```

3.1.11 Retrieving Audio Data from a Song in a CD

Example 3–11 describes how to retrieve audio data from a song in a CD.

Example 3–11 Retrieve Audio Data from a Song in a CD

```
FUNCTION retrieveAudio(itemID IN INTEGER,
                       songId IN INTEGER)
         RETURN BLOB IS
 obj ORDSYS.ORDAudio;
BEGIN
 select S.audioSource into obj from SongsTable S
   where S.songId = songId;
 return obj.getContent();
END;
```

3.1.12 Extending *inter*Media to Support a New Audio Data Format

To support a new audio data format, implement the required interfaces in the ORDX_<format>_AUDIO package in the ORDPLUGINS schema (where <format> represents the name of the new audio data format). See Section 6.4.1 for a complete description of the interfaces for the ORDX_DEFAULT_AUDIO package. Use the package body example in Section 6.4.2 as a template to create the audio package body. Then set the new format parameter in the setFormat call to the appropriate format value to indicate to the audio object that package ORDPLUG-INS.ORDX_<format>_AUDIO is available as a plug-in.

See Section F.1 for more information on installing your own format plug-in and running the sample scripts provided. See the fplugins.sql and fpluginb.sql files that are installed in the *$ORACLE_HOME*/ord/aud/demo/ directory. These are demonstration (demo) plug-ins that you can use as a guideline to write any format plug-in that you want to support. See the auddemo.sql file in this same directory to learn how to install your own format plug-in.

3.1.13 Extending *inter*Media with a New Type

This section describes how to extend Oracle *inter*Media with a new object type.

You can use any of the *inter*Media objects types as the basis for a new type of your own creation.

See Example 3–45 for a more complete example and description.

Note: When a type is altered any dependent type definitions are invalidated. You will encounter this problem if you define a new type that includes an ORDAudio attribute and the *inter*Media ORDAudio type is altered, which always occurs during an *inter*Media installation upgrade.

A workaround to this problem is to revalidate all invalid type definitions with the following SQL statement:

```
SQL> ALTER TYPE <type-name> COMPILE;
```

3.1.14 Using Audio Types with Object Views

This section describes how to use audio types with object views. Just as a view is a virtual table, an object view is a virtual object table.

Oracle provides object views as an extension of the basic relational view mechanism. By using object views, you can create virtual object tables from data -- of either built-in or user-defined types -- stored in the columns of relational or object tables in the database.

Object views can offer specialized or restricted access to the data and objects in a database. For example, you might use an object view to provide a version of an employee object table that does not have attributes containing sensitive data or a deletion method. Object views also let you try object-oriented programming without permanently converting your tables. Using object views, you can convert data gradually and transparently from relational tables to object-relational tables.

In Example 3–12, consider the following relational table (containing no ORDAudio objects).

Example 3–12 Define a Relational Table Containing No ORDAudio Object

```
create table flat (
   id                NUMBER,
   description       VARCHAR2(4000),
   localData         BLOB,
   srcType           VARCHAR2(4000),
   srcLocation       VARCHAR2(4000),
   srcName           VARCHAR2(4000),
   upDateTime        DATE,
   local             NUMBER,
   format            VARCHAR2(31),
   mimeType          VARCHAR2(4000),
```

```
   comments          CLOB,
   encoding          VARCHAR2(256),
   numberOfChannels  NUMBER,
   samplingRate      NUMBER,
   sampleSize        NUMBER,
   compressionType   VARCHAR2(4000),
   audioDuration     NUMBER,
);
```

You can create an object view on the relational table shown in Example 3–12 as follows in Example 3–13.

Example 3–13 Define an Object View Containing an ORDAudio Object and Relational Columns

```
create or replace view object_audio_v as
  select
      id,
      ORDSYS.ORDAudio(
      ORDSYS.ORDSource(
         T.srctype, T.srcLocation, T.srcName,, T.updateTime, T.Local),
         T.description,
         T.localData,
         T.format,
         T.mimeType,
         T.comments,
         T.encoding,
         T.numberOfChannels,
         T.samplingRate,
         T.sampleSize,
         T.compressionType,
         T.audioDuration)
   from flat T;
```

Object views provide the flexibility of looking at the same relational or object data in more than one way. Therefore, you can use different in-memory object representations for different applications without changing the way you store the data in the database. See the *Oracle9i Database Concepts* manual for more information on defining, using, and updating object views.

3.1.15 Scripts for Creating and Populating an Audio Table from a BFILE Data Source

The following scripts can be found on the Oracle Technology Network (OTN) Web site: `http://otn.oracle.com/` as an end-to-end script that creates and

populates an audio table from a BFILE data source. You can get to this site by selecting the Oracle *inter*Media Plugins and Utilities page and from the *inter*Media page, select Sample Code.

The following set of scripts:

1. Creates a tablespace for the audio data, creates a user and grants certain privileges to this new user, creates an audio data load directory (create_auduser.sql).
2. Creates a table with two columns, inserts two rows into the table and initializes the object column to empty with a locator (create_audtable.sql).
3. Loads the audio data with a SELECT FOR UPDATE operation using an import method to import the data from a BFILE (importaud.sql).
4. Performs a check of the properties for the loaded data to ensure that it is really there (chkprop.sql).

The fifth script (setup_audschema.sql) automates this entire process by running each script in the required order. The last script (readaudio.sql) creates a stored procedure that performs a SELECT operation to read a specified amount of audio data from the BLOB, beginning at a particular offset, until all the audio data is read. To successfully load the audio data, you must have an *auddir* directory created on your system. This directory contains the aud1.wav and aud2.mp3 files, which are installed in `<ORACLE_HOME>/ord/aud/demo` directory; this directory path and disk drive must be specified in the CREATE DIRECTORY statement in the create_auduser.sql file.

Script 1: Create a Tablespace, Create an Audio User, Grant Privileges to the Audio User, and Create an Audio Data Load Directory (create_auduser.sql)

This script creates the auddemo tablespace. It contains a data file named auddemo.dbf of 200MB in size, an initial extent of 64K, and a next extent of 128K, and turns on table logging. Next, the auddemo user is created and given connect, resource, create library, and create directory privileges followed by creating the

audio data load directory. Before running this script, you must change the create directory line to point to your data load directory location.

> **Note:** You must edit the create_auduser.sql file and either enter the system password in the connect statement or comment out the connect statement and run this file in the system account. You must specify the disk drive in the CREATE DIRECTORY statement. Also, create the temp temporary tablespace if you have not already created it, otherwise this file will not run.

```
-- create_auduser.sql
-- Connect as admin
connect system/<system password>;

-- Edit this script and either enter your system password here
-- to replace <system password> or comment out this connect
-- statement and connect as system before running this script.

set serveroutput on
set echo on

-- Need system manager privileges to delete a user.
-- Note: There is no need to delete auddemo user if you do not delete
-- the auddemo tablespace, therefore comment out the next line.

-- drop user auddemo cascade;

-- Need system manager privileges to delete a directory. If there is no need to
-- delete it, then comment out the next line.

-- drop directory auddir;

-- Delete then create tablespace.

-- Note: It is better to not delete and create tablespaces,
-- so comment this next line out. The create tablespace statement
-- will fail if it already exists.

-- drop tablespace auddemo including contents;

-- If you uncomment the preceding line and really want to delete the
-- auddemo tablespace, remember to manually delete the auddemo.dbf
-- file to complete this operation. Otherwise, you cannot create
```

```
-- the auddemo tablespace again because the auddemo.dbf file
-- already exists. Therefore, it might be best to create this tablespace
-- once and not delete it.

create tablespace auddemo
       datafile 'auddemo.dbf' size 200M
       minimum extent 64K
       default storage (initial 64K next 128K)
       logging;

-- Create auddemo user.
create user auddemo identified by auddemo
default tablespace auddemo
temporary tablespace temp;

-- Note: If you do not have a temp tablespace already defined, you will have to
-- create it first for this script to work.

grant connect, resource, create library to auddemo;
grant create any directory to auddemo;

-- Note: If this user already exists, you get an error message
-- when you try and create this user again.

-- Connect as auddemo.
connect auddemo/auddemo

-- Create the auddir load directory; this is the directory where the audio
-- files are residing.

create or replace directory auddir
       as 'e:\oracle\ord\aud\demo';
grant read on directory auddir to public with grant option;

-- Note: If this directory already exists, an error message
-- is returned stating the operation will fail; ignore the message.
```

Script 2: Create the Audio Table and Initialize the Column Object (create_audtable.sql)

This script creates the audio table and then performs an insert operation to initialize the column object to empty for two rows. Initializing the column object creates the BLOB locator that is required for populating each row with BLOB data in a subsequent data load operation.

```
--create_audtable.sql

connect auddemo/auddemo;
set serveroutput on
set echo on

drop table audtable;
create table audtable (id number,
       Audio ordsys.ordAudio);

-- Insert a row with empty BLOB.
insert into audtable values(1,ORDSYS.ORDAudio.init());

-- Insert a row with empty BLOB.
insert into audtable values(2,ORDSYS.ORDAudio.init());
commit;
```

Script 3: Load the Audio Data (importaud.sql)

This script performs a SELECT FOR UPDATE operation to load the audio data by first setting the source for loading the audio data from a file, importing the data, setting the properties for the BLOB data, updating the row, and committing the transaction. To successfully run this script, you must copy two audio clips to your AUDDIR directory using the names specified in this script, or modify this script to match the file names of your audio clips.

```
-- importaud.sql

set serveroutput on
set echo on
-- Import two files into the database.

DECLARE
  obj ORDSYS.ORDAUDIO;
  ctx RAW(4000) := NULL;

BEGIN
-- This imports the audio file aud1.wav from the auddir directory
-- on a local file system (srcType=file) and sets the properties.

  select Audio into obj from audtable where id = 1 for update;
  obj.setSource('file','AUDDIR','aud1.wav');
  obj.import(ctx);
  obj.setProperties(ctx);
```

```
  update audtable set audio = obj where id = 1;
  commit;

-- This imports the audio file aud2.mp3 from the auddir directory
-- on a local file system (srcType=file) and sets the properties.

  select Audio into obj from audtable where id = 2 for update;
  obj.setSource('file','AUDDIR','aud2.mp3');
  obj.import(ctx);
  obj.setProperties(ctx);

  update audtable set audio = obj where id = 2;
  commit;
END;
/
```

Script 4: Check the Properties of the Loaded Data (chkprop.sql)

This script performs a SELECT operation of the rows of the audio table, then gets the audio characteristics of the BLOB data to check that the BLOB data is in fact loaded.

```
--chkprop.sql
set serveroutput on;
--Connect auddemo/auddemo
--Query audtable for ORDSYS.ORDAudio.
DECLARE
  audio ORDSYS.ORDAudio;
  idnum integer;
  properties_match BOOLEAN;
  ctx RAW(4000) := NULL;

BEGIN
 FOR I IN 1..2 LOOP
  SELECT id, audio into idnum, audio from audtable where id=I;
   dbms_output.put_line('audio id:          '|| idnum);

  properties_match := audio.checkProperties(ctx);
  IF properties_match THEN DBMS_OUTPUT.PUT_LINE('Check Properties Succeeded');
  END IF;

dbms_output.put_line('audio encoding:           '|| audio.getEncoding()); dbms_
output.put_line('audio number of channels:'|| audio.getNumberOfChannels());
dbms_output.put_line('audio MIME type:          '|| audio.getMimeType());
dbms_output.put_line('audio file format:        '|| audio.getFormat());
 dbms_output.put_line('BLOB Length:             '|| TO_
```

```
CHAR(audio.getContentLength(ctx)));
dbms_output.put_line('--------------------------------------------');

 END loop;
END;
```

Results from running the script chkprop.sql are the following:

```
SQL> @chkprop.sql
audio id:          1
Check Properties Succeeded
audio encoding:        MS-PCM
audio number of channels:  1
audio MIME type:       audio/x-wav
audio file format:     WAVE
BLOB Length:           93594
--------------------------------------------
audio id:          2
Check Properties Succeeded
audio encoding:        LAYER3
audio number of channels: 1
audio MIME type:  audio/mpeg
audio file format:     MPGA
BLOB Length:           51537
--------------------------------------------
PL/SQL procedure successfully completed.
```

Automated Script (setup_audschema.sql)

This script runs each of the previous four scripts in the correct order to automate this entire process.

```
--setup_audschema.sql
-- Create auddemo user, tablespace, and load directory to
-- hold the audio files:
@create_auduser.sql

-- Create Audio table:
@create_audtable.sql

--Import 2 audio clips and set properties:
@importaud.sql

--Check the properties of the audio clips:
@chkprop.sql
```

```
--exit;
```

Read Data from the BLOB (readaudio.sql)

This script creates a stored procedure that performs a SELECT operation to read a specified amount of audio data from the BLOB, beginning at a particular offset, until all the audio data is read.

```
--readaudio.sql

set serveroutput on
set echo on

create or replace procedure readaudio as

   obj ORDSYS.ORDAudio;
   buffer RAW (32767);
   numBytes BINARY_INTEGER := 32767;
   startpos integer := 1;
   read_cnt integer := 1;
   ctx RAW(4000) := NULL;

BEGIN

   Select audio into obj from audtable where id = 1;

   LOOP
           obj.readFromSource(ctx,startPos,numBytes,buffer);
           DBMS_OUTPUT.PUT_LINE('BLOB Length: '   || TO_CHAR(obj.getContentLength(ctx)));
             DBMS_OUTPUT.PUT_LINE('start position: '|| startPos);
             DBMS_OUTPUT.PUT_LINE('doing read: '    || read_cnt);
           startpos := startpos + numBytes;
           read_cnt := read_cnt + 1;
   END LOOP;
-- Note: Add your own code here to process the audio data being read;
--       this routine just reads the data into the buffer 32767 bytes
--       at a time, then reads the next chunk, overwriting the first
--       buffer full of data.

EXCEPTION

   WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('End of data ');
   WHEN ORDSYS.ORDSourceExceptions.METHOD_NOT_SUPPORTED THEN
    DBMS_OUTPUT.PUT_LINE('ORDSourceExceptions.METHOD_NOT_SUPPORTED caught');
   WHEN OTHERS THEN
```

```
    DBMS_OUTPUT.PUT_LINE('EXCEPTION caught');

END;

/
show errors
```

To execute the stored procedure, enter the following SQL statements:

```
SQL> set serveroutput on;
SQL> execute readaudio
Content Length: 93594
start position: 1
doing read: 1
start position: 32768
doing read: 2
start position: 65535
doing read: 3
---------------
End of data

PL/SQL procedure successfully completed.
```

3.2 Media Data Examples

Media data examples using *inter*Media include the following common operations:

- Defining a media object named documentObject
- Creating an object table named DocumentsTable
- Creating a list object named docList that contains a list of media
- Defining the implementation of the docList object
- Creating a library object and LibraryTable table
- Inserting media into the DocumentsTable table
- Inserting a library into the LibraryTable table
- Loading media into the DocumentsTable table
- Inserting a reference to a document object into the media list in the LibraryTable table
- Adding a library reference to document

- Extending *inter*Media to support a new media data format
- Extending *inter*Media with new object types
- Using *inter*Media with object views
- Using the ORDDoc object type as a repository
- Using a set of scripts for creating and populating a media table from a BFILE data source

Reference information on the methods used in these examples is presented in Chapter 7.

3.2.1 Defining a Media Object

Example 3–14 describes how to define a media object. You must create an empty LibraryObject object type first for the REF to work in this example. The actual LibraryObject is created in Example 3–18.

Example 3–14 Define a Media Object

```
-- Forward Declarations --
-- -------------------- --
CREATE OR REPLACE TYPE LibraryObject;
/

CREATE TYPE DocumentObject as OBJECT (
  LibraryRef      REF LibraryObject,   -- REF into the library table
  documentId      VARCHAR2(40),
  title           VARCHAR2(4000),
  author          VARCHAR2(4000),
  category        VARCHAR2(20),
  copyright       VARCHAR2(4000),
  publisher       VARCHAR2(4000),
  awards          VARCHAR2(4000),
  timePeriod      VARCHAR2(20),
  length          INTEGER,
  txtcontent      CLOB,
  coverImage      REF ORDSYS.ORDImage,
  documentSource  ORDSYS.ORDDOC
);
/
show errors
```

3.2.2 Creating an Object Table DocumentsTable

Example 3–15 describes how to create an object table named DocumentsTable.

Example 3–15 Create a Table Named DocumentsTable

```
CREATE TABLE DocumentsTable of DocumentObject (UNIQUE (documentId), documentId
NOT NULL);
```

3.2.3 Creating a List Object Containing a List of References to Media

Example 3–16 describes how to create a list object containing a list of references to media.

Example 3–16 Create a List Object Containing a List of References to Media

```
CREATE TYPE documentNstType AS TABLE of REF DocumentObject;
/
show errors

CREATE TYPE documentList AS OBJECT (documents documentNstType,
     MEMBER PROCEDURE addDocument(d IN REF DocumentObject));
/
show errors
```

3.2.4 Defining the Implementation of the documentList Object

Example 3–17 describes how to define the implementation of the documentList object.

Example 3–17 Define the Implementation of the documentList Object

```
CREATE TYPE BODY documentList AS
  MEMBER PROCEDURE addDocument(d IN REF DocumentObject)
  IS
    pos INTEGER := 0;
  BEGIN
    IF documents IS NULL THEN
      documents := documentNstType(NULL);
      pos := 0;
    ELSE
      pos := document.count;
    END IF;
    documents.EXTEND;
```

```
    documents(pos+1) := d;
  END;
END;
/
show errors
```

3.2.5 Creating a Library Object and a Library Table

This section describes how to create a Library object and a Library table of media abstracts that includes, for each media abstract, the following information:

- Item ID
- Library DB ID
- Library title
- Library author
- Library category
- Copyright
- Name of publisher
- Awards
- Time period
- Rating
- Length of media in bytes
- Text content
- Cover image
- Documents

Example 3–18 creates a Library object named LibraryObject, a Library table named LibraryTable that contains the Library information, and an Image table named ImageTable that contains the media cover images.

Example 3–18 Create a Library Table Containing Library Information

```
CREATE TYPE LibraryObject as OBJECT (
  itemId        INTEGER,
  librarydbID   INTEGER,
  title         VARCHAR2(4000),
  author        VARCHAR2(4000),
```

```
  category      VARCHAR2(20),
  copyright     VARCHAR2(4000),
  publisher     VARCHAR2(4000),
  awards        VARCHAR2(4000),
  timePeriod    VARCHAR2(20),
  rating        VARCHAR2(256),
  length        INTEGER,
  txtcontent    CLOB,
  coverImg      REF ORDSYS.ORDImage,
  documents     documentsList);
/
show errors

CREATE TABLE LibraryTable OF LibraryObject (UNIQUE(itemId), itemId NOT NULL)
       NESTED TABLE documents.documents STORE AS document_store_table;
CREATE TABLE ImageTable OF ORDSYS.ORDImage;
```

3.2.6 Inserting Media into the DocumentsTable Table

Example 3–19 describes how to insert media into the DocumentsTable table.

Example 3–19 Insert Media into the DocumentsTable Table

```
-- Insert media into the documents table
INSERT INTO DocumentsTable VALUES (NULL,
                                   '00',
                                   'The Big Wind Storm',
                                   'author',
                                   'storms',
                                   '1999',
                                   'Windy Rivers Publishers',
                                   'Classic Tales Award',
                                   '1992',
                                   2430000,
                                   EMPTY_CLOB(),
                                   NULL,
                                   ORDSYS.ORDDoc.init());

-- Check media insertion
SELECT d.title
FROM   DocumentsTable d
WHERE documentId = '00';
```

3.2.7 Inserting a Library into the LibraryTable Table

Example 3–20 describes how to insert a Library into the LibraryTable table.

Example 3–20 Insert a Library into the LibraryTable Table

```
-- Insert a library into the library table
INSERT INTO LibraryTable VALUES (1, 23232323,
                            'Sailing Classics',
                            'authors',
                            'sailing',
                            '1998',
                            'BMV Company',
                            'Young Authors Award',
                            '90s',
                            'no rating',
                            4000000,            -- in characters
                            EMPTY_CLOB(),
                            NULL,
                            documentList(NULL));

-- Check library insertion
SELECT library.title
FROM   Librarytable library;
```

3.2.8 Loading Media into the DocumentsTable Table

Example 3–21 describes how to load media into the DocumentsTable table. This example requires a DOCDIR directory to be defined; see the comments in the example.

Example 3–21 Load Media into the DocumentsTable Table

```
-- Load media into the DocumentsTable
-- Create your directory specification below
-- CREATE OR REPLACE DIRECTORY DOCDIR AS '/document/';
DECLARE
        documentObj ORDSYS.ORDDOC;
        ctx RAW(4000) := NULL;
BEGIN
        SELECT D.documentSource INTO documentObj
        FROM   DocumentsTable D
        WHERE  D.documentId = '00'
        FOR UPDATE;
```

```
        documentObj.setSource('file', 'DOCDIR', 'BigWindStorm.pdf');
        documentObj.setMimeType('application/pdf');
        documentObj.import(ctx,FALSE);
        documentObj.setProperties(ctx,FALSE);

        UPDATE DocumentsTable D
        SET    D.documentSource = documentObj
        WHERE  D.documentId = '00';
        COMMIT;
END;
/
-- Check document insertion
DECLARE
     documentObj ORDSYS.ORDDOC;
     ctx RAW(4000) := NULL;
BEGIN
     SELECT D.documentSource INTO documentObj
     FROM   DocumentsTable D
     WHERE  D.documentId = '00';

     dbms_output.put_line('Content Length: ' ||
                    documentObj.getContentLength());
     dbms_output.put_line('Content MimeType: ' ||
                    documentObj.getMimeType());
END;
/
```

3.2.9 Inserting a Reference to a Document Object into the Documents List in the LibraryTable Table

Example 3–22 describes how to insert a reference to a document object into the documents list in the LibraryTable table.

Example 3–22 Insert a Reference to a Document Object into the Documents List in the LibraryTable Table

```
-- Insert a reference to a DocumentObject into the Documents List in the
LibraryTable Table
DECLARE
        documentRef REF DocumentObject;
        documentListInstance documentList;
BEGIN
        SELECT REF(D) into documentRef
```

```
        FROM   DocumentsTable D
        where  D.documentId = '00';

        SELECT L.documents INTO documentListInstance
        FROM   LibraryTable L
        WHERE  L.itemId = 1
        FOR UPDATE;

        documentListInstance.addDocument(documentRef);

        UPDATE LibraryTable L
        SET    L.documents = documentListInstance
        WHERE  L.itemId = 1;

        COMMIT;
END;
-- Check insertion of ref
-- This example works for the first entry inserted in the documentList
DECLARE
        document             DocumentObject;
        documentRef          REF DocumentObject;
        documentListInstance documentList;
        documentType         documentNstType;
BEGIN
        SELECT L.documents INTO documentListInstance
        FROM   LibraryTable L
        WHERE  L.itemId = 1;

        SELECT documentListInstance.documents INTO documentType FROM DUAL;
        documentRef := documentType(1);
        SELECT DEREF(documentRef) INTO document FROM DUAL;

        dbms_output.put_line('Document Title: ' ||
                             document.title);
END;
/
```

3.2.10 Adding a Library Reference to a Document

Example 3–23 describes how to add a library reference to a document.

Example 3–23 Add a Library Reference to a Document

```
-- Adding a library reference to a document
DECLARE
```

```
     documentLibraryRef  REF LibraryObject;
BEGIN
     SELECT D.libraryRef INTO documentLibraryRef
     FROM   DocumentsTable D
     WHERE  D.documentId = '00'
     FOR UPDATE;

     SELECT REF(L) INTO documentLibraryRef
     FROM  LibraryTable L
     WHERE L.itemId = 1;

     UPDATE DocumentsTable D
     SET    D.libraryRef = documentLibraryRef
     WHERE  D.documentId = '00';

     COMMIT;
END;

-- Check library Ref
DECLARE
     libraryRef REF LibraryObject;
     library    LibraryObject;
BEGIN
     SELECT D.libraryRef INTO libraryRef
     FROM   DocumentsTable D
     WHERE  D.documentId = '00';

     SELECT DEREF(libraryRef) INTO library FROM DUAL;
     dbms_output.put_line('Library Title: ' ||
                          library.title);
END;
/
```

3.2.11 Extending *inter*Media to Support a New Media Data Format

To support a new media data format, implement the required interfaces in the ORDX_<format>_DOC package in the ORDPLUGINS schema (where <format> represents the name of the new media data format). See Section 7.4.1 for a complete description of the interfaces for the ORDX_DEFAULT_DOC package. Use the package body example in Section 7.4.2 as a template to create the media package body. Then set the new format parameter in the setFormat call to the appropriate format value to indicate to the media object that package ORDPLUG-INS.ORDX_<format>_DOC is available as a plug-in. See Section 7.4.2 for more information about extending *inter*Media to support a new media data format.

3.2.12 Extending *inter*Media with a New Type

This section describes how to extend Oracle *inter*Media with a new object type.

You can use any of the *inter*Media objects types as the basis for a new type of your own creation.

See Example 3–45 for a more complete example and description.

Note: When a type is altered any dependent type definitions are invalidated. You will encounter this problem if you define a new type that includes an ORDDoc attribute and the *inter*Media ORDDoc type is altered, which always occurs during an *inter*Media installation upgrade.

A workaround to this problem is to revalidate all invalid type definitions with the following SQL statement:

```
SQL> ALTER TYPE <type-name> COMPILE;
```

3.2.13 Using Document Types with Object Views

This section describes how to use document types with object views. Just as a view is a virtual table, an object view is a virtual object table.

The Oracle database provides object views as an extension of the basic relational view mechanism. By using object views, you can create virtual object tables from data -- of either built-in or user-defined types -- stored in the columns of relational or object tables in the database.

Object views can offer specialized or restricted access to the data and objects in a database. For example, you might use an object view to provide a version of an employee object table that does not have attributes containing sensitive data or a deletion method. Object views also let you try object-oriented programming without permanently converting your tables. Using object views, you can convert data gradually and transparently from relational tables to object-relational tables. See Example 3–12 and Example 3–13 for examples of defining a relational table containing no media (ORDAudio) object type and how to define an object view containing a media (ORDAudio) object type and relational columns.

Object views provide the flexibility of looking at the same relational or object data in more than one way. Therefore, you can use different in-memory object representations for different applications without changing the way you store the data in the database. See the *Oracle9i Database Concepts* manual for more information on defining, using, and updating object views.

3.2.14 Using the ORDDoc Object Type as a Repository

The ORDDoc document object type is most useful for applications that require the storage of different types of media, such as audio, image, video, and any other type of document in the same column so you can build a common metadata index on all the different types of media and perform searches across different types of media using this index.

> **Note:** You cannot use this same search technique if the different types of media are stored in different types of objects in different columns of relational tables.

Example 3–24 shows how to create a repository of media using the tdoc table. A requirement for creating the metadata index is to create a primary key constraint on column n. After initializing each row, load each row with a different media, in this case two audio clips, two video clips, and two images. For each media file, call the setProperties( ) method after each row is loaded and specify the setComments = TRUE value for this parameter to populate the comments field of the object with an extensive set of format and application properties in XML form. Because the format of each media type is natively supported by *inter*Media, the setProperties method is used to extract the properties from the media source and the comments field of the object is populated in XML form. If the format of the media type is not known, then the setProperties( ) method raises a DOC_PLUGIN_EXCEPTION exception. *inter*Media does not support any document media type file (html, pdf, doc, and so forth), therefore you must create your own format plug-in in order to extract the media attributes from the media data.

Next, use Oracle Text and create the metadata index on the comments attribute of the doc column. Then, begin to search for interesting formats, mimeTypes, and so forth.

Example 3–24 Build a Repository of Media

```
-- Connect as system manager to create a tablespace and a user.
-- May need to create a temp tablespace for this to work.

CONNECT SYSTEM/MANAGER

--Create tablespace docrepository.

CREATE TABLESPACE docrepository
     DATAFILE 'docrepos.dbf' SIZE 200M
```

```
     MINIMUM EXTENT 64K
     DEFAULT STORAGE (INITIAL 64K NEXT 128K)
     LOGGING;

-- Create a docuser user.
-- Create a temp tablespace if you do not have one.

CREATE USER DOCUSER IDENTIFIED BY DOCUSER
DEFAULT TABLESPACE docrepository;
-- TEMPORARY TABLESPACE temp;

GRANT CONNECT, RESOURCE, CREATE LIBRARY to docuser;
GRANT CREATE ANY DIRECTORY TO docuser;

-- End of system manager tasks.


-- Begin user tasks.

CONNECT docuser/docuser


-- Create the docdir directory.

CREATE OR REPLACE DIRECTORY docdir
              as 'e:\oracle\ord\aud\demo';
GRANT READ ON DIRECTORY docdir TO PUBLIC WITH GRANT OPTION;

-- Create the tdoc table.

CREATE TABLE tdoc (n NUMBER CONSTRAINT n_pk PRIMARY KEY, doc ORDSYS.ORDDoc)
   STORAGE (INITIAL 100K NEXT 100K PCTINCREASE 0);

INSERT INTO tdoc VALUES(1, ORDSYS.ORDDoc.init());
INSERT INTO tdoc VALUES(2, ORDSYS.ORDDoc.init());

DECLARE
  obj ORDSYS.ORDDOC;
  ctx RAW(4000) := NULL;

BEGIN
-- This imports the audio file aud1.wav from the docdir directory
-- on a local file system (srcType=file) and sets the properties.

SELECT doc INTO obj FROM tdoc WHERE n = 1 FOR UPDATE;
```

```
  obj.setSource('file','DOCDIR','aud1.wav');
  obj.import(ctx,FALSE);
  obj.setProperties(ctx,TRUE);
UPDATE tdoc SET doc = obj WHERE n = 1;
COMMIT;

-- This imports the audio file aud2.mp3 from the docdir directory
-- on a local file system (srcType=file) and sets the properties.

SELECT doc INTO obj FROM tdoc WHERE n = 2 FOR UPDATE;
  obj.setSource('file','DOCDIR','aud2.mp3');
  obj.import(ctx,FALSE);
  obj.setProperties(ctx, TRUE);
UPDATE tdoc SET doc = obj WHERE n = 2;
COMMIT;
END;
/

INSERT INTO tdoc VALUES(3, ORDSYS.ORDDoc.init());
INSERT INTO tdoc VALUES(4, ORDSYS.ORDDoc.init());

CREATE OR REPLACE DIRECTORY docdir
       as 'e:\oracle\ord\vid\demo';
GRANT READ ON DIRECTORY docdir TO PUBLIC WITH GRANT OPTION;

DECLARE
  obj ORDSYS.ORDDOC;
  ctx RAW(4000) := NULL;

BEGIN
-- This imports the video file vid1.mov from the docdir directory
-- on a local file system (srcType=file) and sets the properties.

SELECT doc INTO obj FROM tdoc WHERE n = 3 FOR UPDATE;
  obj.setSource('file','DOCDIR','vid1.mov');
  obj.import(ctx,FALSE);
  obj.setProperties(ctx,TRUE);
UPDATE tdoc SET doc = obj WHERE n = 3;
COMMIT;

-- This imports the video file vid2.mov from the docdir directory
-- on a local file system (srcType=file) and sets the properties.

SELECT doc INTO obj FROM tdoc WHERE n = 4 FOR UPDATE;
  obj.setSource('file','DOCDIR','vid2.mov');
```

```
  obj.import(ctx,FALSE);
  obj.setProperties(ctx, TRUE);
UPDATE tdoc SET doc = obj WHERE n = 4;
COMMIT;
END;
/

INSERT INTO tdoc VALUES(5, ORDSYS.ORDDoc.init());
INSERT INTO tdoc VALUES(6, ORDSYS.ORDDoc.init());

CREATE OR REPLACE DIRECTORY docdir
       as 'e:\oracle\ord\img\demo';
GRANT READ ON DIRECTORY docdir TO PUBLIC WITH GRANT OPTION;

DECLARE
  obj ORDSYS.ORDDOC;
  ctx RAW(4000) := NULL;

BEGIN
-- This imports the image file img71.gif from the docdir directory
-- on a local file system (srcType=file) and sets the properties.

SELECT doc INTO obj FROM tdoc WHERE n = 5 FOR UPDATE;
  obj.setSource('file','DOCDIR','img71.gif');
  obj.import(ctx,FALSE);
  obj.setProperties(ctx, TRUE);
UPDATE tdoc SET doc = obj WHERE n = 5;
COMMIT;

-- This imports the image file img50.gif from the docdir directory
-- on a local file system (srcType=file) and sets the properties.

SELECT doc INTO obj FROM tdoc WHERE n = 6 FOR UPDATE;
  obj.setSource('file','DOCDIR','img50.gif');
  obj.import(ctx,FALSE);
  obj.setProperties(ctx, TRUE);
UPDATE tdoc SET doc = obj WHERE n = 6;
COMMIT;
END;
/

-- Create the index using Oracle Text.
--
CREATE INDEX mediaidx ON tdoc(doc.comments) INDEXTYPE IS ctxsys.context;
COMMIT;
```

```
-- As part of the CREATE INDEX statement, you can create a preference,
-- create media attribute sections for each media attribute,
-- that is, format, mimeType, and contentLength.
-- For example,
--
-- 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');

-- Create a section group.
-- Define Media Attribute sections, that is, the XML tags for the attributes
-- or samples.

EXECUTE CTX_DDL.DROP_SECTION_GROUP('MEDIAANN_TAGS');
EXECUTE CTX_DDL.CREATE_SECTION_GROUP('MEDIAANN_TAGS','xml_section_group');
EXECUTE CTX_DDL.ADD_ZONE_SECTION('MEDIAANN_TAGS',
'MEDIAFORMATENCODINGTAG','MEDIA_FORMAT_ENCODING_CODE');
EXECUTE CTX_DDL.ADD_ZONE_SECTION('MEDIAANN_TAGS','MEDIASOURCEMIMETYPETAG',
'MEDIA_SOURCE_MIME_TYPE');
EXECUTE CTX_DDL.ADD_ZONE_SECTION('MEDIAANN_TAGS', 'MEDIASIZETAG','MEDIA_SIZE');
--
-- Add the following PARAMETERS clause to the end of the CREATE INDEX statement:
-- PARAMETERS ('section group MEDIAANN_TAGS'), so the statement appears
-- as follows:

CREATE INDEX mediaidx ON tdoc(doc.comments) INDEXTYPE IS
     CTXSYS.CONTEXT PARAMETERS('stoplist CTXSYS.EMPTY_STOPLIST wordlist
     ANN_WORDLIST filter CTXSYS.NULL_FILTER section group MEDIAANN_TAGS');
COMMIT;
--
-- Now, perform a SELECT statement on the attributes in the doc.comments column.

SELECT n from tdoc;
-- Should display 6 rows.

SELECT n, score(99) from tdoc t WHERE CONTAINS(t.doc.comments, '(MPEG) WITHIN
MEDIAFORMATENCODINGTAG',99)>0;
-- Should find one row for the aud2.mp3 audio file.
```

3.2.15 Scripts for Creating and Populating a Media Table from a BFILE Data Source

The following scripts can be found on the Oracle Technology Network (OTN) Web site: `http://otn.oracle.com/` as an end-to-end script that creates and populates a media table from a BFILE data source. You can get to this site by selecting the Oracle *inter*Media Plugins and Utilities page and from the *inter*Media page, select Sample Code.

The following set of scripts:

1. Creates a tablespace for the media data, creates a user and grants certain privileges to this new user, and creates a media data load directory (create_docuser.sql).
2. Creates a table with two columns, inserts two rows into the table and initializes the object column to empty with a locator (create_doctable.sql).
3. Loads the media data with a SELECT FOR UPDATE operation using an import method to import the data from a BFILE (importdoc.sql).
4. Performs a check of the properties for the loaded data to ensure that it is really there (chkprop.sql).

The fifth script (setup_docschema.sql) automates this entire process by running each script in the required order. The last script (readdoc.sql) creates a stored procedure that performs a SELECT operation to read a specified amount of media data from the BLOB, beginning at a particular offset, until all the media data is read. To successfully load the media data, you must have a *docdir* directory created on your system. This directory contains the aud1.wav and aud2.mp3 files, which are installed in *<ORACLE_HOME>*`/ord/aud/demo` directory; this directory path and disk drive must be specified in the CREATE DIRECTORY statement in the create_docuser.sql file.

Script 1: Create a Tablespace, Create a Media User, Grant Privileges to the Media User, and Create a Media Data Load Directory (create_docuser.sql)

This script creates the docdemo tablespace. It contains a data file named docdemo.dbf of 200MB in size, an initial extent of 64K, and a next extent of 128K, and turns on table logging. Next, the docdemo user is created and given connect, resource, create library, and create directory privileges followed by creating the

media data load directory. Before running this script, you must change the create directory line to point to your data load directory location.

> **Note:** You must edit the create_docuser.sql file and either enter the system password in the connect statement or comment out the connect statement and run this file in the system account. You must specify the disk drive in the CREATE DIRECTORY statement. Also, create the temp temporary tablespace if you have not already created it, otherwise this file will not run.

```
-- create_docuser.sql
-- Connect as admin
connect system/<system password>;

-- Edit this script and either enter your system password here
-- to replace <system password> or comment out this connect
-- statement and connect as system before running this script.

set serveroutput on
set echo on

-- Need system manager privileges to delete a user.
-- Note: There is no need to delete docdemo user if you do not delete
-- the docdemo tablespace, therefore comment out the next line.

-- drop user docdemo cascade;

-- Need system manager privileges to delete a directory. If there is no need to
-- delete it, then comment out the next line.

-- drop directory docdir;

-- Delete then create tablespace.

-- Note: It is better to not delete and create tablespaces,
-- so comment this next line out. The create tablespace statement
-- will fail if it already exists.

-- drop tablespace docdemo including contents;

-- If you uncomment the preceding line and really want to delete the
-- docdemo tablespace, remember to manually delete the docdemo.dbf
-- file to complete this operation. Otherwise, you cannot create
```

```
-- the docdemo tablespace again because the docdemo.dbf file
-- already exists. Therefore, it might be best to create this tablespace
-- once and not delete it.

create tablespace docdemo
       datafile 'docdemo.dbf' size 200M
       minimum extent 64K
       default storage (initial 64K next 128K)
       logging;

-- Create docdemo user.
create user docdemo identified by docdemo
default tablespace docdemo
temporary tablespace temp;

-- Note: If you do not have a temp tablespace already defined, you will have to
-- create it first for this script to work.

grant connect, resource, create library to docdemo;
grant create any directory to docdemo;

-- Note: If this user already exists, you get an error message
-- when you try and create this user again.

-- Connect as docdemo.
connect docdemo/docdemo

-- Create the docdir load directory; this is the directory where the media
-- files are residing.

create or replace directory docdir
       as 'e:\oracle\ord\aud\demo';
grant read on directory docdir to public with grant option;
-- Note for Solaris, the directory specification could be '/user/local'
-- Note: If this directory already exists, an error message
-- is returned stating the operation will fail; ignore the message.
```

Script 2: Create the Media Table and Initialize the Column Object (create_doctable.sql)

This script creates the media table and then performs an insert operation to initialize the column object to empty for two rows. Initializing the column object creates the BLOB locator that is required for populating each row with BLOB data in a subsequent data load operation.

```
--create_doctable.sql

connect docdemo/docdemo;
set serveroutput on
set echo on

drop table doctable;
create table doctable (id number,
       Document ordsys.ordDoc);

-- Insert a row with empty BLOB.
insert into doctable values(1,ORDSYS.ORDDoc.init());

-- Insert a row with empty BLOB.
insert into doctable values(2,ORDSYS.ORDDoc.init());
commit;
```

Script 3: Load the Media Data (importdoc.sql)

This script performs a SELECT FOR UPDATE operation to load the media data by first setting the source for loading the media data from a file, importing the data, setting the properties for the BLOB data, updating the row, and committing the transaction. To successfully run this script, you must copy two media files to your DOCDIR directory using the names specified in this script, or modify this script to match the file names of your media.

```
-- importdoc.sql

set serveroutput on
set echo on
-- Import two files into the database.

DECLARE
  obj ORDSYS.ORDDOC;
  ctx RAW(4000) := NULL;

BEGIN
-- This imports the audio file aud1.wav from the DOCDIR directory
-- on a local file system (srcType=file) and sets the properties.

  select Document into obj from doctable where id = 1 for update;
  obj.setSource('file','DOCDIR','aud1.wav');
  obj.import(ctx,TRUE);
  update doctable set document = obj where id = 1;
  commit;
```

```
-- This imports the audio file aud2.mp3 from the DOCDIR directory
-- on a local file system (srcType=file) and sets the properties.

  select Document into obj from doctable where id = 2 for update;
  obj.setSource('file','DOCDIR','aud2.mp3');
  obj.import(ctx,TRUE);
  update doctable set document = obj where id = 2;
  commit;
END;
/
```

Script 4: Check the Properties of the Loaded Data (chkprop.sql)

This script performs a SELECT operation of the rows of the media table, then gets the media characteristics of the BLOB data to check that the BLOB data is in fact loaded.

```
--chkprop.sql
set serveroutput on;
--Connect docdemo/docdemo
--Query doctable for ORDSYS.ORDDoc.
DECLARE
  document ORDSYS.ORDDoc;
  idnum integer;
  properties_match BOOLEAN;
  ctx RAW(4000) := NULL;

BEGIN
 FOR I IN 1..2 LOOP
  SELECT id, document into idnum, document from doctable where id=I;
   dbms_output.put_line('document id:          '|| idnum);

 dbms_output.put_line('document MIME type:          '|| document.getMimeType());
 dbms_output.put_line('document file format:        '|| document.getFormat());
 dbms_output.put_line('BLOB Length:   '|| TO_CHAR(document.getContentLength()));
dbms_output.put_line('-----------------------------------------------');

 END loop;
END;
/
```

Results from running the script chkprop.sql are the following:

```
SQL> @chkprop.sql
document id:          1
```

```
document MIME type:       audio/xwav
document file format:     WAVE
BLOB Length:            93594
---------------------------------------------
document id:           2
document MIME type:   audio/mpeg
document file format:     MPGA
BLOB Length:            51537
---------------------------------------------
PL/SQL procedure successfully completed.
```

Automated Script (setup_docschema.sql)

This script runs each of the previous four scripts in the correct order to automate this entire process.

```
--setup_docschema.sql
-- Create docdemo user, tablespace, and load directory to
-- hold the media files:
@create_docuser.sql

-- Create Media table:
@create_doctable.sql

--Import 2 media clips and set properties:
@importdoc.sql

--Check the properties of the media clips:
@chkprop.sql

--exit;
```

Read Data from the BLOB (readdoc.sql)

This script creates a stored procedure that performs a select operation to read a specified amount of media data from the BLOB, beginning at a particular offset, until all the media data is read.

```
--readdoc.sql

set serveroutput on
set echo on

create or replace procedure readdocument as

   obj ORDSYS.ORDDoc;
```

```
   buffer RAW (32767);
   numBytes BINARY_INTEGER := 32767;
   startpos integer := 1;
   read_cnt integer := 1;
   ctx RAW(4000) := NULL;

BEGIN

   Select document into obj from doctable where id = 1;

   LOOP
           obj.readFromSource(ctx,startPos,numBytes,buffer);
           DBMS_OUTPUT.PUT_LINE('BLOB Length: '   || TO_CHAR(obj.getContentLength()));
             DBMS_OUTPUT.PUT_LINE('start position: '|| startPos);
             DBMS_OUTPUT.PUT_LINE('doing read: '    || read_cnt);
           startpos := startpos + numBytes;
           read_cnt := read_cnt + 1;
   END LOOP;
-- Note: Add your own code here to process the media data being read;
--       this routine just reads the data into the buffer 32767 bytes
--       at a time, then reads the next chunk, overwriting the first
--       buffer full of data.

EXCEPTION

   WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('End of data ');
   WHEN ORDSYS.ORDSourceExceptions.METHOD_NOT_SUPPORTED THEN
    DBMS_OUTPUT.PUT_LINE('ORDSourceExceptions.METHOD_NOT_SUPPORTED caught');
   WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('EXCEPTION caught');

END;

/
show errors
```

To execute the stored procedure, enter the following SQL statements:

```
SQL> set serveroutput on;
SQL> execute readdocument
Content Length: 93594
start position: 1
doing read: 1
start position: 32768
```

```
doing read: 2
start position: 65535
doing read: 3
---------------
End of data

PL/SQL procedure successfully completed.
```

3.3 Image Data Examples

Image data examples using *inter*Media include the following common operations:

- Adding types to a new or existing table
- Inserting a row using BLOB images
- Populating a row using BLOB images
- Inserting a row using BFILE images
- Populating a row using BFILE images
- Querying a row
- Importing an image from an external file into the database
- Retrieving images (simple read operation; no content-based retrieval)
- Retrieving images similar to a comparison image (content-based retrieval)
- Creating a domain index
- Retrieving images similar to a comparison image using indexing operations (indexed content-based retrieval)
- Copying an image
- Converting an image format
- Copying and converting an image in one step
- Extending *inter*Media with new object types
- Using image types with object views
- Using a set of scripts for creating and populating an image table from a BFILE data source

- Using a set of scripts for creating and populating an image table from an HTTP data source
- Addressing Globalization Support issues

3.3.1 Adding Image Types to an Existing Table

Suppose you have an existing table named 'stockphotos' with the following columns:

```
photo_id      NUMBER
photographer  VARCHAR2(64)
annotation    VARCHAR2(255)
```

To add two new columns to the 'stockphotos' table called 'photo' using the ORDImage type and photo_sig using the ORDImageSignature type, issue the statement in Example 3–25. The photo column will store images and the photo_sig column will store image signatures, so you can later compare these images to a comparison image by means of their image signature.

Example 3–25 adds two new columns of type ORDImage and ORDImageSignature to the stockphotos table.

Example 3–25 Add New Columns of Type ORDImage and ORDImageSignature to the stockphotos Table

```
ALTER TABLE stockphotos
ADD (photo ORDSYS.ORDImage, photo_sig ORDSYS.ORDImageSignature);
```

3.3.2 Adding Image Types to a New Table

Suppose you are creating a new table called 'stockphotos' with the following information:

- Photo ID number
- Photographer's name
- Descriptive annotation
- Photographic image
- Photograph signature

The column for the photograph is for photographs of cloth patterns and uses the ORDImage type, and the column for the photograph signature 'photo_sig' uses the

ORDImageSignature type. The statement in Example 3–26 creates the table and adds ORDImage and ORDImageSignature types to the new table.

Example 3–26 Create the stockphotos Table and Add ORDImage and ORDImageSignature Types

```
CREATE TABLE stockphotos (
  photo_id NUMBER,
  photographer VARCHAR2(64),
  annotation VARCHAR2(255),
  photo ORDSYS.ORDImage,
  photo_sig ORDSYS.ORDImageSignature);
```

3.3.3 Inserting a Row Using BLOB Images

To insert a row into a table that has storage for image content using the ORDImage and ORDImageSignature types, you must populate each type with an initializer. Note that this is different from NULL. Attempting to use the ORDImage or ORDImageSignature types with a NULL value results in an error.

Example 3–27 describes how to insert rows into the table using the ORDImage and ORDImageSignature types. Assume you have a table 'stockphotos' with the following columns:

```
photo_id       NUMBER
photographer   VARCHAR2(64)
annotation     VARCHER2(255)
photo          ORDImage
photo_sig      ORDImageSignature
```

If you are going to store image data in the database (in a binary large object (BLOB)), you must populate the ORDSource.localData attribute with a value and initialize storage for the localData attribute with an empty_blob( ) constructor. To insert a row into the table with empty data in the 'photo' and 'photo_sig' columns, issue the statement in Example 3–27.

Example 3–27 inserts a row into a table with empty data in the ORDImage type column.

Example 3–27 Insert a Row into a Table with Empty Data in the ORDImage Type Column

```
INSERT INTO stockphotos VALUES (
  1, 'John Doe', 'red plaid',
  ORDSYS.ORDImage.init(),
```

```
  ORDSYS.ORDImageSignature.init());
```

3.3.4 Populating a Row Using BLOB Images

Prior to updating a BLOB value, you must lock the row containing the BLOB locator. This is usually done using a SELECT FOR UPDATE statement in SQL and PL/SQL programs, or using an Oracle Call Interface (OCI) pin or lock function in OCI programs.

Example 3–28 populates a row with ORDImage BLOB data and ORDImageSignature data. See Section 3.1.15 for another set of examples for populating rows using BLOB images.

Example 3–28 Populate a Row with ORDImage BLOB Data

```
DECLARE
    -- application variables
    Image ORDSYS.ORDImage;
    ctx RAW(4000) := NULL;
BEGIN
    INSERT INTO stockphotos VALUES (
      1,'John Doe', red plaid,
      ORDSYS.ORDImage.init(),
      ORDSYS.ORDImageSignature.init());
    -- Select the newly inserted row for update
      SELECT photo INTO Image FROM stockphotos
           WHERE photo_id = 1 for UPDATE;
    -- Can use the getContent method to get the LOB locator.
    -- Populate the data with DBMS LOB calls or write an OCI program to
    -- fill in the image BLOB.
    -- This example imports the image file test.gif from the ORDIMGDIR
    -- directory on a local file system
    -- (srcType=FILE) and automatically sets the properties.

    Image.setSource('file','ORDIMGDIR','redplaid.gif');
    Image.import(ctx);

  UPDATE stockphotos SET photo = Image WHERE photo_id = 1;
  COMMIT;
  -- Continue processing
END;
/
```

An UPDATE statement is required to update the property attributes. If you do not use the UPDATE statement now, you can still commit, and the change to the image

will be reflected in the BLOB attribute, but not in the properties. See *Oracle9i Application Developer's Guide - Large Objects (LOBs)* for more information on BLOBs.

3.3.5 Inserting a Row Using BFILE Images

To insert a row into a table that has storage for image content in external files using the ORDImage type, you must populate the type with an initializer. Note that this is different from NULL. Attempting to use the ORDImage type with a NULL value results in an error.

Example 3–29 describes how to insert rows into the table using the ORDImage type. Assume you have a table 'stockphotos' with the following columns:

```
photo_id      NUMBER
photographer  VARCHAR2(64)
annotation    VARCHAR2(255)
photo         ORDImage
photo_sig     ORDImageSignature
```

If you are going to use the ORDImage and ORDImageSignature type columns, you must first populate the columns with a value. To populate the value of the ORDImage type column with an image stored externally in a file, you must populate the row with a file constructor.

Example 3–29 inserts a row into the table with an image called 'redplaid.gif' from the ORDIMGDIR directory.

Example 3–29 Insert a Row into a Table Pointing to an External Image Data File

```
INSERT INTO stockphotos VALUES (
  1,'John Doe','red plaid',
  ORDSYS.ORDImage.init('file','ORDIMGDIR','redplaid.gif'),
  ORDSYS.ORDImageSignature.init());
```

For a description of row insertion into an object type, see Chapter 8, and the *Oracle9i Application Developer's Guide - Large Objects (LOBs)* manual.

The sourceLocation argument 'ORDIMGDIR' is a directory referring to a file system directory. Note that the directory name must be in uppercase. The following sequence creates a directory named ORDIMGDIR:

```
-- Make a directory referring to a file system directory
CREATE DIRECTORY ORDIMGDIR AS '<MYIMAGEDIRECTORY>';
GRANT READ ON DIRECTORY ORDIMGDIR TO <user-or-role>;
```

<MYIMAGEDIRECTORY> is the file system directory, and <user-or-role> is the specific user to whom to grant read access.

3.3.6 Populating a Row Using BFILE Images

Example 3–30 populates the row with ORDImage data stored externally in files.

Example 3–30 Populate a Row with ORDImage External File Data

```
DECLARE
    Image ORDSYS.ORDImage;
BEGIN
    INSERT INTO stockphotos VALUES (1,'John Doe','red plaid',
    ORDSYS.ORDImage.init('file','ORDIMGDIR','redplaid.gif'),
    ORDSYS.ORDImageSignature.init());
    -- Select the newly inserted row for update
    SELECT photo INTO Image FROM stockphotos
           WHERE photo_id = 1 FOR UPDATE;
    -- Set property attributes for the image data
    Image.setProperties;
    UPDATE stockphotos SET photo = Image WHERE photo_id = 1;
    COMMIT;
    -- Continue processing
END;
/
```

3.3.7 Querying a Row

Example 3–31 and Example 3–32 assume you have this table:

```
CREATE TABLE stockphotos (
photo_id NUMBER,
photographer VARCHAR2(64),
annotation VARCHAR2(255),
photo ORDSYS.ORDImage,
photo_sig ORDSYS.ORDImageSignature);
```

Example 3–31 queries the stockphotos table for the photo_id of 1 and the ORDImage data for rows with minimum photo widths (greater than 32 pixels).You must create a table alias (E in this example) when you refer to a type in a SELECT statement.

Example 3–31 Query Rows of ORDImage Data for Widths Greater Than 32 Pixels

```
SELECT photo_id, s.photo.getWidth()
```

```
  FROM stockphotos S
  WHERE photo_id = 1 and
        S.photo.getWidth() > 32;
```

Example 3–32 queries the stockphotos table for photo_id =1 and the ORDImage data for rows with minimum photo widths (greater than 32 pixels) and a minimum content length (greater than 10000 bytes).

Example 3–32 Query Rows of ORDImage Data for Widths Greater Than 32 Pixels and a Minimum Content Length

```
SELECT photo_id, S.photo.getCompressionFormat()
  FROM stockphotos S
  WHERE photo_id = 1 and
        S.photo.getWidth() > 32 and
        S.photo.getContentLength() > 10000;
```

3.3.8 Importing an Image from an External File into the Database

To import an image from an external file into the database, use the ORDImage.import method. Example 3–33 imports image data from an external file into the database. The source type, source location, and source name must be set prior to calling the import( ) method.

Example 3–33 Import an Image from an External File

```
DECLARE
    Image ORDSYS.ORDImage;
    ctx RAW(4000) := NULL;
BEGIN
    SELECT photo
      INTO Image FROM stockphotos
      WHERE photo_id = 1 FOR UPDATE;
    -- Import the image into the database
    Image.import(ctx);
    UPDATE stockphotos SET photo = IMAGE
           WHERE photo_id = 1;
    COMMIT;
END;
/
```

3.3.9 Retrieving an Image

The following examples, Example 3–38 through Example 3–41 use the table definition described in Example 3–34 that includes both an image object and image signature object for content-based retrieval of images.

Example 3–34 Table stockphotos Definition Used for Content-Based Retrieval of Images

```
CREATE TABLE stockphotos(photo_id INTEGER,
                photographer VARCHAR2(64),
                annotation   VARCHAR2(255),
                photo ORDSYS.ORDImage,
                photo_signature ORDSYS.ORDImageSignature);
```

The stockphotos table is loaded with image data as described in Example 3–35.

Example 3–35 Load the stockphotos Table with Image Data

```
DECLARE
    myimg ORDSYS.ORDImage;
    mysig ORDSYS.ORDImageSignature;
    x INTEGER;
    ctx RAW(4000):= NULL;
BEGIN
    -- create 4 plaid patterns, for each get an image from ORDIMGDIR directory
    INSERT INTO stockphotos(photo_id,photographer,annotation,photo,photo_sig)
        VALUES(1,
               'John MacIvor',
               'red plaid',
               ORDSYS.ORDImage.init('file','ORDIMGDIR','redplaid.gif'),
               ORDSYS.ORDImageSignature.init());
    INSERT INTO stockphotos(photo_id,photographer,annotation,photo,photo_sig)
        VALUES(2,
               'Jane Cranston',
               'green plaid',
               ORDSYS.ORDImage.init('file','ORDIMGDIR','greenplaid.gif'),
               ORDSYS.ORDImageSignature.init());
    INSERT INTO stockphotos(photo_id,photographer,annotation,photo,photo_sig)
        VALUES(3,
               'Clark Gordon',
               'blue plaid',
               ORDSYS.ORDImage.init('file','ORDIMGDIR','blueplaid.gif'),
               ORDSYS.ORDImageSignature.init());
    INSERT INTO stockphotos(photo_id,photographer,annotation,photo,photo_sig)
```

```
        VALUES(4,
               'Bruce MacLeod',
               'yellow plaid',
               ORDSYS.ORDImage.init('file','ORDIMGDIR','yellowplaid.gif'),
               ORDSYS.ORDImageSignature.init());

    -- import images and generate signatures
    FOR x in 1..4 LOOP
        SELECT S.photo, S.photo_sig INTO myimg, mysig
            FROM stockphotos S
            WHERE S.photo_id = x FOR UPDATE;
        myimg.import(ctx);
        mysig.generateSignature(myimg);
        UPDATE stockphotos S
            SET S.photo = myimg,
                S.photo_sig = mysig
            WHERE S.photo_id = x;
    END LOOP;
END;
/
```

Rows can be read from the emp table as shown in Example 3–36 to check the contents of the table.

Example 3–36 Check the Contents of the stockphotos Table

```
SELECT photo_id, photographer, annotation
    FROM stockphotos
    ORDER BY photo_id;
```

Finally, Example 3–37 shows how to create the tablespaces needed for index creation by the imageuser user in Example 3–41.

Example 3–37 Create the Tablespaces for the Index

```
CONNECT system/<system-password>;

GRANT CREATE TABLESPACE TO imageuser;
GRANT DROP TABLESPACE TO imageuser;

CONNECT imageuser/imageuser;
CREATE TABLESPACE ordimage_idx_tbs_1
    DATAFILE 'e:\<ORACLE_HOME>\DATABASE\ordimage_idx_tbs_1.dbf' SIZE 1M REUSE;
CREATE TABLESPACE ordimage_idx_tbs_2
    DATAFILE 'e:\<ORACLE_HOME>\DATABASE\ordimage_idx_tbs_2.dbf' SIZE 1M REUSE;
```

Example 3–38 reads an image from the table and prepares it to be passed along, either directly to the end user or to the application for further processing. The program segment selects the desired photograph (where photo_id = 1) and places it in an image storage area.

Example 3–38 Retrieve an Image (Simple Read)

```
SET SERVEROUTPUT ON
SET ECHO ON

DECLARE
   image     ORDSYS.ORDIMAGE;
BEGIN
   -- Select the desired photograph from the stockphotos table.
   SELECT photo INTO image FROM stockphotos
          WHERE photo_id = 1;
END;
/
```

3.3.10 Retrieving Images Similar to a Comparison Image (Content-Based Retrieval)

Example 3–39 performs content-based retrieval; it finds images that are similar to an image chosen for comparison.

The program segment performs the following operations:

1. Defines a cursor to perform the matching. The cursor sets the following weight values:
   - Color: 0.2
   - Texture: 0.1
   - Shape: 0.4
   - Location: 0.3
2. The example assumes that all signatures for images are generated and stored in the `photo_sig` column.
3. Selects all photo signatures in the `photo_sig` column to compare with the comparison image signature (`compare_img`) and where the photo_id is not 1 (photo_id <> 1).
4. Sets the threshold value at 25.
5. Selects the matching images, using the cursor.

Example 3–39 Retrieve Images Similar to a Comparison Image

```
SET SERVEROUTPUT ON
SET ECHO ON

DECLARE
   threshold    NUMBER;
   compare_sig  ORDSYS.ORDImageSignature;
   photographer VARCHAR2(64);
   annotation   VARCHAR2(255);
   photo        ORDSYS.ORDIMAGE;

-- Define cursor for matching. Set weights for the visual attributes.
CURSOR getphotos IS
   SELECT photograpger, annotation, photo FROM stockphotos S
   WHERE ORDSYS.IMGSimilar(S.photo_sig, compare_sig,
                'color="0.2" texture="0.1" shape="0.4"
                 location="0.3"', threshold)=1 AND photo_id <> 1;

BEGIN
 -- select signature of image you want to match against
   SELECT P.photo_sig INTO compare_img FROM stockphotos P
   WHERE P.photo_id = 1;

   -- Set the threshold value.
   threshold := 25;

   -- Retrieve rows for matching images.
   OPEN getphotos;
   LOOP
      FETCH getphotos INTO photographer, annotation, photo;
      EXIT WHEN getphotos%NOTFOUND;
      -- Display or store the results.
      -- .
      -- .
   END LOOP;
   CLOSE getphotos;
END;
/
```

Example 3–40 finds the photo_id and score of the image that is most similar to a comparison image with respect to texture. None of the other image characteristics is considered. This example uses the IMGScore( ) operator, which is an ancillary operator used in conjunction with the IMGSimilar operator. The parameter passed to IMGScore( ) (123 in this example) is an identifier to an IMGSimilar( ) operator,

indicates that the image matching score value returned by an IMGScore( ) operator is the same one used in the corresponding IMGSimilar( ) operator (with label 123). In this example, one of the three images compared to the comparison image were identical to the comparison image and showed a score of zero (0).

Example 3–40 Find photo_id and Score of Similar Image

```
SET SERVEROUTPUT ON
SET ECHO ON

SELECT Q.photo_id,
       ORDSYS.IMGScore(123) SCORE
       FROM stockphotos Q, stockphotos E
       WHERE E.photo_id=1 AND Q.photo_id != E.photo_id AND
             ORDSYS.IMGSimilar(Q.photo_sig, E.photo_sig,
                               'texture=1', 20.0, 123)=1;
PHOTO_ID   SCORE
--------- ---------
        1         0
```

3.3.11 Creating a Domain Index

To improve performance, you can create a domain index on the image signature attribute. Example 3–41 creates an index called imgindex.

Example 3–41 Create an interMedia Index

```
SET SERVEROUTPUT ON
SET ECHO ON

CREATE INDEX imgindex ON stockphotos(photo_sig)
 INDEXTYPE IS ORDSYS.ORDImageIndex
 PARAMETERS('
   ORDIMG_FILTER_TABLESPACE = ordimage_idx_tbs_1,
   ORDIMG_INDEX_TABLESPACE = ordimage_idx_tbs_2');
```

As with any index, the tablespace (ordimage_idx_tbs_1 and ordimage_idx_tbs_2) must be created first.

The following recommendations are good starting points for further index tuning:

- ORDIMG_FILTER_TABLESPACE -- Each signature requires approximately 350 bytes in this tablespace. The tablespace should be at least 350 times the number of signatures in the table.

- ORDIMG_INDEX_TABLESPACE -- The size of the tablespace should be 100 times the size of the initial and final extents specified. For example, if an extent is 10 KB, the tablespace size should be 1 MB. The initial and final extents should be equal to each other. The size of the tablespace should also be approximately equal to the size of ORDIMG_DATA_TABLESPACE.
- Typically, it will be much faster if you create the index after the images are loaded into the database and analyzed.
- Creating an index for large tables can be very time consuming. When importing a large number of images, you should postpone index creation until after the import operation completes. Do this by specifying the following parameters to the IMPORT statement: INDEXES=N and INDEXNAME=<filename>. See *Oracle9i Database Utilities* for details.
- Rollback segments of an appropriate size are required. The size depends on the size of your transactions, such as, how many signatures are indexed at one time.
- Analyze the new index. See Section 2.4.

3.3.12 Retrieving Images Similar to a Comparison Image Using Index Operations (Indexed Content-Based Retrieval)

Queries for indexed and nonindexed comparisons are identical. The Oracle optimizer uses the domain index if it determines that the first argument passed to the IMGSimilar operator is a domain-indexed column. Otherwise, the optimizer invokes a functional implementation of the operator that compares the query signature with the stored signatures, one row at a time.

See Section 3.3.10 for examples of retrieving similar images. As in the example, be sure to specify the query signature as the second parameter.

3.3.13 Copying an Image

To copy an image, use the ORDImage.copy method. Example 3–42 copies image data.

Example 3–42 Copy an Image

```
DECLARE
    Image_1 ORDSYS.ORDImage;
    Image_2 ORDSYS.ORDImage;
BEGIN
    SELECT photo INTO Image_1
```

```
      FROM stockphotos WHERE photo_id = 1;
    SELECT photo INTO Image_2
      FROM stockphotos WHERE photo_id = 1 FOR UPDATE;
    -- Copy the data from Image_1 to Image_2
    Image_1.copy(Image_2);
    -- Continue processing
    UPDATE stockphotos SET photo = Image_2
        WHERE photo_id = 1;
    COMMIT;
END;
/
```

3.3.14 Converting an Image Format

To convert the image data into a different format, use the process( ) method.

> **Note:** The process( ) method processes only into a BLOB, so the image data must be stored locally.

Example 3–43 converts the image data to the TIFF image file format.

Example 3–43 Convert an Image Format

```
DECLARE
    Image ORDSYS.ORDImage;
BEGIN
    SELECT photo INTO Image FROM stockphotos
        WHERE photo_id = 1 FOR UPDATE;
    -- Convert the image to TIFF (in place)
    Image.process('fileFormat=TIFF');
    UPDATE stockphotos SET photo = Image WHERE photo_id = 1;
    COMMIT;
END;
/
```

3.3.15 Copying and Converting in One Step

To make a copy of the image and convert it in one step, use the processCopy( ) method.

Note: The processCopy( ) method processes only into a BLOB, so the destination image must be set to local and the localData attribute in the source must be initialized.

Example 3–44 creates a thumbnail image, converts the image data to the TIFF image file format, copies it to a BLOB, and leaves the original image intact.

Example 3–44 Copy and Convert an Image Format

```
DECLARE
    Image_1 ORDSYS.ORDImage;
    Image_2 ORDSYS.ORDImage;
BEGIN
       SELECT photo INTO Image_1
       FROM stockphotos WHERE photo_id = 1;
    SELECT photo INTO Image_2
       FROM stockphotos WHERE photo_id = 2 FOR UPDATE;
    -- Convert the image to a TIFF thumbnail image and store the
    -- result in Image_2
    Image_1.processCopy('fileFormat=TIFF fixedScale=32 32', Image_2);
    -- Continue processing
    UPDATE stockphotos SET photo = Image_2 WHERE photo_id = 2;
    COMMIT;
END;
/
```

Changes made by the processCopy( ) method can be rolled back. This technique may be useful for a temporary format conversion.

3.3.16 Extending *inter*Media with a New Type

You can use the ORDImage type as the basis for a new type of your own creation as shown in Example 3–45.

Note: When a type is altered any dependent type definitions are invalidated. You will encounter this problem if you define a new type that includes an ORDImage attribute and the *inter*Media ORDImage type is altered, which always occurs during an *inter*Media installation upgrade.

A workaround to this problem is to revalidate all invalid type definitions with the following SQL statement:

```
SQL> ALTER TYPE <type-name> COMPILE;
```

Example 3–45 Extend Oracle interMedia with a New Object Type

```
CREATE TYPE AnnotatedImage AS OBJECT
    ( image ORDSYS.ORDImage,
      description VARCHAR2(2000),
      MEMBER PROCEDURE SetProperties(SELF IN OUT AnnotatedImage),
      MEMBER PROCEDURE Copy(dest IN OUT AnnotatedImage),
      MEMBER PROCEDURE ProcessCopy(command IN VARCHAR2,
                                   dest IN OUT AnnotatedImage)
    );
/

CREATE TYPE BODY AnnotatedImage AS
  MEMBER PROCEDURE SetProperties(SELF IN OUT AnnotatedImage) IS
  BEGIN
    SELF.image.setProperties();
    SELF.description :=
        'This is an example of using Image object as a subtype';
  END SetProperties;
  MEMBER PROCEDURE Copy(dest IN OUT AnnotatedImage) IS
  BEGIN
    SELF.image.copy(dest.image);
    dest.description := SELF.description;
  END Copy;
  MEMBER PROCEDURE ProcessCopy(command IN VARCHAR2,
                               dest IN OUT AnnotatedImage) IS
  BEGIN
    SELF.Image.processCopy(command,dest.image);
    dest.description := SELF.description;
  END ProcessCopy;
END;
/
```

After creating the new type, you can use it as you would any other type. For example:

```
CREATE OR REPLACE DIRECTORY ORDIMGDIR AS 'C:\TESTS';

CREATE TABLE my_example(id NUMBER, an_image AnnotatedImage);
INSERT INTO my_example VALUES (1,
    AnnotatedImage(
        ORDSYS.ORDImage.init('file','ORDIMGDIR','plaid.gif'));
COMMIT;
DECLARE
    myimage AnnotatedImage;
BEGIN
    SELECT an_image INTO myimage FROM my_example;
    myimage.SetProperties;
    DBMS_OUTPUT.PUT_LINE('This image has a description of ');
    DBMS_OUTPUT.PUT_LINE(myimage.description);
    UPDATE my_example SET an_image = myimage;
END;
/
```

3.3.17 Using Image Types with Object Views

Just as a view is a virtual table, an object view is a virtual object table.

Oracle provides object views as an extension of the basic relational view mechanism. By using object views, you can create virtual object tables from data-- of either built-in or user-defined types -- stored in the columns of relational or object tables in the database.

Object views can offer specialized or restricted access to the data and objects in a database. For example, you might use an object view to provide a version of an employee object table that does not have attributes containing sensitive data or a deletion method. Object views also let you try object-oriented programming without permanently converting your tables. Using object views, you can convert data gradually and transparently from relational tables to object-relational tables.

In Example 3–46, consider the following relational table (containing no ORDImage objects):

Example 3–46 Define a Relational Table Containing No ORDImage Object

```
CREATE TABLE flat(
    id              NUMBER,
    localData       BLOB,
```

```
    srcType           VARCHAR2(4000),
    srcLocation       VARCHAR2(4000),
    srcName           VARCHAR2(4000),
    updateTime        DATE,
    local             NUMBER,
    height            INTEGER,
    width             INTEGER,
    contentLength     INTEGER,
    fileFormat        VARCHAR2(4000),
    contentFormat     VARCHAR2(4000),
    compressionFormat VARCHAR2(4000),
    mimeType          VARCHAR2(4000)
    );
```

You can create an object view on the relational table shown in Example 3–46 as follows in Example 3–47.

Example 3–47 Define an Object View Containing an ORDImage Object and Relational Columns

```
CREATE OR REPLACE VIEW object_images_v AS
    SELECT
      id,
      ORDSYS.ORDImage(
        ORDSYS.ORDSource(
          T.localData,
          T.srcType,
          T.srcLocation,
          T.srcName,
          T.updateTime,
          T.local),
        T.height,
        T.width,
        T.contentLength,
        T.fileFormat,
        T.contentFormat,
        T.compressionFormat,
        T.mimeType
        ) IMAGE
      FROM flat T;
```

Object views provide the flexibility of looking at the same relational or object data in more than one way. Thus you can use different in-memory object representations for different applications without changing the way you store the data in the

database. See the *Oracle9i Database Concepts* manual for more information on defining, using, and updating object views.

3.3.18 Scripts for Creating and Populating an Image Table from a BFILE Data Source

The following scripts can be found on the Oracle Technology Network (OTN) Web site: `http://otn.oracle.com/` as end-to-end scripts that create and populate an image table from a BFILE data source. You can get to this site by selecting the Oracle *inter*Media Plugins and Utilities page and from the *inter*Media page, select Sample Code.

The following set of scripts:

1. Creates a tablespace for the image data, creates a user and grants certain privileges to this new user, creates an image data load directory (create_imguser.sql).
2. Creates a table with two columns, inserts two rows into the table and initializes the object column to empty with a locator (create_imgtable.sql).
3. Loads the image data with a SELECT FOR UPDATE operation using an import method to import the data from a BFILE (importimg.sql).
4. Performs a check of the properties for the loaded data to ensure that it is really there (chkprop.sql).

The fifth script (setup_imgschema.sql) automates this entire process by running each script in the required order. The last script (readimage.sql) creates a stored procedure that performs a SELECT operation to read a specified amount of image data from the BLOB beginning at a particular offset until all the image data is read. To successfully load the image data, you must have an *imgdir* directory created on your system containing the img71.gif and img50.gif files, which are installed in the *<ORACLE_HOME>*`/ord/img/demo` directory; this directory path and disk drive must be specified in the CREATE DIRECTORY statement in the create_imguser.sql file.

Script 1: Create a Tablespace, Create an Image User, Grant Privileges to the Image User, and Create an Image Data Load Directory (create_imguser.sql)

This script creates the imgdemo tablespace with a data file named imgdemo.dbf of 200MB in size, with an initial extent of 64K, a next extent of 128K, and turns on table logging. Next, the imgdemo user is created and given connect, resource, create

library, and create directory privileges, followed by creating the image data load directory.

> **Note:** You must edit the create_imguser.sql file and either enter the system password in the connect statement or comment out the connect statement and run this file in the system account. You must specify the disk drive in the CREATE DIRECTORY statement. Also, create the temp temporary tablespace if you have not already created it, otherwise this file will not run.

```
-- create_imguser.sql
-- Connect as admin.
connect system/<system password>;
-- Edit this script and either enter your system password here
-- to replace <system password> or comment out this connect
-- statement and connect as system before running this script.

set serveroutput on
set echo on

-- Need system manager privileges to delete a user.
-- Note: There is no need to delete imgdemo user if you do not delete the
-- imgdemo tablespace, therefore comment out the next line.

-- drop user imgdemo cascade;

-- Need system manager privileges to delete a directory. If threre is
-- no need to really delete it, then comment out the next line.

-- drop directory imgdir;

-- Delete then create the tablespace.

-- Note: It is better to not delete and create tablespaces,
-- so comment this next line out. The create tablespace statement
-- will fail if it already exists.

-- drop tablespace imgdemo including contents;

-- If you uncomment the preceding line and really want to delete the
-- imgdemo tablespace, remember to manually delete the imgdemo.dbf
-- file to complete the operation. Otherwise, you cannot create
-- the imgdemo tablespace again because the imgdemo.dbf file
```

```
-- already exists. Therefore, it might be best to create this
-- tablespace once and not delete it.

-- Create tablespace.
create tablespace imgdemo
       datafile 'imgdemo.dbf' size 200M
       minimum extent 64K
       default storage (initial 64K next 128K)
       logging;

-- Create imgdemo user.
create user imgdemo identified by imgdemo
default tablespace imgdemo
temporary tablespace temp;

-- Note: If you do not have a temp tablespace already defined, you will
-- have to create it first for this script to work.

grant connect, resource, create library to imgdemo;
grant create any directory to imgdemo;

-- Note: If this user already exists, you get an error message when you
-- try and create this user again.

-- Connect as imgdemo.
connect imgdemo/imgdemo

-- Create the imgdir load directory; this is the directory where the image
-- files are residing.

create or replace directory imgdir
       as 'e:\oracle\ord\img\demo';
grant read on directory imgdir to public with grant option;
-- Note: If this directory already exists, an error message
-- is returned stating the operation will fail; ignore the message.
```

Script 2: Create the Image Table and Initialize the Column Object (create_imgtable.sql)

This script creates the image table and then performs an insert operation to initialize the column object to empty for two rows. Initializing the column object creates the BLOB locator that is required for populating each row with BLOB data in a subsequent data load operation.

```
-- create_imgtable.sql
connect imgdemo/imgdemo;
set serveroutput on
set echo on

drop table imgtable;
create table imgtable (id number,
       Image ordsys.ordImage);

-- Insert a row with empty BLOB.
insert into imgtable values(1,ORDSYS.ORDImage.init());

-- Insert a row with empty BLOB.
insert into imgtable values(2,ORDSYS.ORDImage.init());
commit;
```

Script 3: Load the Image Data (importimg.sql)

This script performs a SELECT FOR UPDATE operation to load the image data by first setting the source for loading the image data from a file, importing the data, setting the properties for the BLOB data, updating the row, and committing the transaction. To successfully run this script, you must copy two image files to your IMGDIR directory using the names specified in this script, or modify this script to match the file names of your image files.

```
--importimg.sql
set serveroutput on
set echo on
-- Import the two files into the database.

DECLARE
  obj ORDSYS.ORDIMAGE;
  ctx RAW(4000) := NULL;
BEGIN
-- This imports the image file img71.gif from the IMGDIR directory
-- on a local file system (srcType=file) and sets the properties.

  select Image into obj from imgtable where id = 1 for update;
  obj.setSource('file','IMGDIR','img71.gif');
  obj.import(ctx);

  update imgtable set image = obj where id = 1;
  commit;
```

```
-- This imports the image file img50.gif from the IMGDIR directory
-- on a local file system (srcType=file) and sets the properties.

  select Image into obj from imgtable where id = 2 for update;
  obj.setSource('file','IMGDIR','img50.gif');
  obj.import(ctx);

  update imgtable set image = obj where id = 2;
  commit;
END;
/
```

Script 4: Check the Properties of the Loaded Data (chkprop.sql)

This script performs a SELECT operation of the rows of the image table, then gets the image characteristics of the BLOB data to check that the BLOB data is in fact loaded.

```
-- chkprop.sql
set serveroutput on;
--connect imgdemo/imgdemo
--Query imgtable for ORDSYS.ORDImage.
DECLARE
  image ORDSYS.ORDImage;
  idnum integer;
  properties_match BOOLEAN;

BEGIN
 FOR I IN 1..2 LOOP
  SELECT id into idnum from imgtable where id=I;
   dbms_output.put_line('image id:          '|| idnum);

  SELECT Image into image from imgtable where id=I;

  properties_match := image.checkProperties();
  IF properties_match THEN DBMS_OUTPUT.PUT_LINE('Check Properties Succeeded');
  END IF;

   dbms_output.put_line('image height:      '|| image.getHeight());
   dbms_output.put_line('image width:       '|| image.getWidth());
   dbms_output.put_line('image MIME type:   '|| image.getMimeType());
   dbms_output.put_line('image file format: '|| image.getFileFormat());
   dbms_output.put_line('BLOB Length:     '|| TO_CHAR(image.getContentLength()));
   dbms_output.put_line('-----------------------------------------');
```

```
 END loop;
END;
/
```

Results from running the script chkprop.sql are the following:

```
SQL> @chkprop.sql
image id:          1
Check Properties Succeeded
image height:      15
image width:       43
image MIME type:   image/gif
image file format: GIFF
BLOB Length:       1124
-----------------------------------------
image id:          2
Check Properties Succeeded
image height:      32
image width:       110
image MIME type:   image/gif
image file format: GIFF
BLOB Length:       686
-----------------------------------------

PL/SQL procedure successfully completed.
```

Automated Script (setup_imgschema.sql)

This script runs each of the previous four scripts in the correct order to automate this entire process.

```
-- setup_imgschema.sql
-- Create imgdemo user, tablespace, and load directory to
-- hold image files:
@create_imguser.sql

-- Create image table:
@create_imgtable.sql

--Import 2 images and set properties:
@importimg.sql

--Check the properties of the images:
@chkprop.sql

--exit;
```

Read Data from the BLOB (readimage.sql)

This script performs a SELECT operation to read a specified amount of image data from the BLOB, beginning at a particular offset until all the image data is read.

```
-- readimage.sql

set serveroutput on
set echo on

create or replace procedure readimage as

-- Note: ORDImage has no readFromSource method like ORDAudio
-- and ORDVideo; therefore, you must use the DBMS_LOB package to
-- read image data from a BLOB.

   buffer RAW (32767);
   src BLOB;
   obj ORDSYS.ORDImage;
   amt BINARY_INTEGER := 32767;
   pos integer := 1;
   read_cnt integer := 1;

BEGIN

   Select  t.image.getcontent into src from imgtable t where t.id = 1;
   Select image into obj from imgtable t where t.id = 1;
        DBMS_OUTPUT.PUT_LINE('Content length is: '|| TO_CHAR(obj.getContentLength()));
   LOOP
         DBMS_LOB.READ(src,amt,pos,buffer);
           DBMS_OUTPUT.PUT_LINE('start position: '|| pos);
           DBMS_OUTPUT.PUT_LINE('doing read  '|| read_cnt);
         pos := pos + amt;
         read_cnt := read_cnt + 1;

-- Note: Add your own code here to process the image data being read;
-- this routine just reads data into the buffer 32767 bytes
--  at a time, then reads the next chunk, overwriting the first
--  buffer full of data.
   END LOOP;

EXCEPTION

   WHEN NO_DATA_FOUND THEN
   DBMS_OUTPUT.PUT_LINE('----------------');
   DBMS_OUTPUT.PUT_LINE('End of data ');
```

```
END;

/
show errors
```

To execute the stored procedure, enter the following SQL statements:

```
SQL> set serveroutput on;
SQL> execute readimage(1);
Content length is: 1124
start position: 1
doing read  1
---------------
End of data

PL/SQL procedure successfully completed.
```

3.3.19 Scripts for Populating an Image Table from an HTTP Data Source

The following scripts can be found on the Oracle Technology Network (OTN) Web site: `http://otn.oracle.com/` as end-to-end scripts that create and populate an image table from an HTTP data source. You can get to this site by selecting the Oracle *inter*Media Plugins and Utilities page and from the *inter*Media page, select Sample Code.

> **Note:** Before you run the importimg.sql script described in this section to load image data from an HTTP data source, check to ensure you have already run the create_imguser.sql and create_imgtable.sql scripts described in Section 3.3.18.

The following set of scripts performs a row insert operation and an import operation, then checks the properties of the loaded images to ensure that the images are really loaded.

Initialize the Column Object and Import the Image Data (importimghttp.sql)

This script inserts two rows into the imgtable table, initializing the object column for each row to empty with a locator, and indicating the HTTP source information (source type (HTTP), URL location, and HTTP object name). Within a SELECT FOR UPDATE statement, an import operation loads each image object into the database

followed by an UPDATE statement to update the object attributes for each image, and finally a COMMIT statement to commit the transaction.

To successfully run this script, you must modify this script to point to two images located on your own Web site.

```
--importimghttp.sql
-- Import the two HTTP images from a Web site into the database.
-- Running this script assumes you have already run the
-- create_imguser.sql and create_imgtable.sql scripts.
-- Modify the HTTP URL and object name to point to two images
-- on your own Web site.

set serveroutput on
set echo on

-- Import two images from HTTP source URLs.

connect imgdemo/imgdemo;

-- Insert two rows with empty BLOB.

insert into imgtable values (7,ORDSYS.ORDImage.init(
             'http','your.web.site.com/intermedia','image1.gif'));

insert into imgtable values (8,ORDSYS.ORDImage.init(
             'http','your.web.site.com/intermedia','image2.gif'));

DECLARE
  obj ORDSYS.ORDIMAGE;
  ctx RAW(4000) := NULL;
BEGIN
-- This imports the image file image1.gif from the HTTP source URL
-- (srcType=HTTP), and automatically sets the properties.

  select Image into obj from imgtable where id = 7 for update;
    obj.import(ctx);

  update imgtable set image = obj where id = 7;
  commit;

-- This imports the image file image2.gif from the HTTP source URL
-- (srcType=HTTP), and automatically sets the properties.

  select Image into obj from imgtable where id = 8 for update;
```

```
    obj.import(ctx);

  update imgtable set image = obj where id = 8;
  commit;
END;
/
```

Check the Properties of the Loaded Data

This script performs a SELECT operation of the rows of the image table, then gets the image characteristics of the BLOB data to check that the BLOB data is in fact loaded.

```
--chkprop.sql
set serveroutput on;
--connect imgdemo/imgdemo
--Query imgtable for ORDSYS.ORDImage.
DECLARE
image ORDSYS.ORDImage;
idnum integer;
properties_match BOOLEAN;

BEGIN
  FOR I IN 7..8 LOOP
    SELECT id into idnum from imgtable where id=I;
     dbms_output.put_line('image id: '|| idnum);
    SELECT Image into image from imgtable where id=I for update;
    properties_match := image.checkProperties();
    IF properties_match THEN DBMS_OUTPUT.PUT_LINE('Check Properties Succeeded');
    END IF;
     dbms_output.put_line('image height: '|| image.getHeight());
     dbms_output.put_line('image width: '|| image.getWidth());
     dbms_output.put_line('image MIME type: '|| image.getMimeType());
     dbms_output.put_line('image file format: '|| image.getFileFormat());
     dbms_output.put_line('BLOB length: '|| TO_CHAR(image.getContentLength()));
     dbms_output.put_line('----------------------------------------');
  END loop;
END;
/
```

3.3.20 Addressing Globalization Support Issues

Example 3–48 shows how to use the processCopy( ) method with language settings that use the comma as the decimal point. For example, when the territory is

FRANCE, the decimal point is expected to be a comma. Notice the ",75" specified as the scale factor. This application addresses Globalization Support issues.

Example 3–48 Address a Globalization Support Issue

```
ALTER SESSION SET NLS_LANGUAGE = FRENCH;
ALTER SESSION SET NLS_TERRITORY = FRANCE;
DECLARE
    myimage ORDSYS.ORDImage;
    mylargeimage ORDSYS.ORDImage;
BEGIN
    SELECT photo, large_photo INTO myimage, mylargeimage
      FROM emp FOR UPDATE;
    myimage.setProperties();
    myimage.ProcessCopy('scale=",75"', mylargeimage);
    UPDATE emp SET photo = myimage, large_photo = mylargeimage;
    COMMIT;
END;
/
```

3.4 Video Data Examples

Video data examples using *inter*Media include the following common operations:

- Defining a clip object named clipObject
- Creating an object table named clipsTable
- Creating a list object named clipList that contains a list of clips
- Defining the implementation of the clipList object
- Creating a video object and VideoTable table
- Inserting a video clip into the ClipsTable table
- Inserting a row into the VideoTable table
- Loading a video into the ClipsTable table
- Inserting a reference to a clipObject into the clips list in the video table
- Inserting a reference to a video object into the clip
- Retrieving a video clip from the VideoTable table
- Extending *inter*Media to support a new video data format
- Extending *inter*Media with new object types

- Using video types with object views
- Using a set of scripts for creating and populating a video table from a BFILE data source

The video examples in this section use a table of video clips and a table of videos. For each video clip the following are stored: a videoRef (REF into the video table), clip ID, title, director, category, copyright, producer, awards, time period, rating, duration, cdRef (REF into CdObject for sound tracks), text content (indexed by CONTEXT), cover image (REF into the image table), and video source. For each video the following are stored: an item ID, duration, text content (indexed by CONTEXT), cover image (REF into the image table), and a list of clips on the video.

Reference information on the methods used in these examples is presented in Chapter 9.

3.4.1 Defining a Clip Object

Example 3–49 describes how to define a clip object.

Example 3–49 Define a Clip Object

```
CREATE TYPE clipObject as OBJECT (
  videoRef      REF VideoObject,       -- REF into the video table
  clipId        VARCHAR2(20),          -- Id inside of the clip table
  title         VARCHAR2(4000),
  director      VARCHAR2(4000),
  category      VARCHAR2(20),
  copyright     VARCHAR2(4000),
  producer      VARCHAR2(4000),
  awards        VARCHAR2(4000),
  timePeriod    VARCHAR2(20),
  rating        VARCHAR2(256),
  duration      INTEGER,
  cdRef         REF CdObject,          -- REF into a CdObject(soundtrack)
  txtcontent    CLOB,
  coverImg      REF ORDSYS.ORDImage,   -- REF into the ImageTable
  videoSource   ORDSYS.ORDVideo);
```

3.4.2 Creating an Object Table ClipsTable

Example 3–50 describes how to create an object table named ClipsTable.

Example 3–50 Create a Table Named ClipsTable

```
CREATE TABLE ClipsTable of clipObject (UNIQUE (clipId), clipId NOT NULL);
```

3.4.3 Creating a List Object Containing a List of Clips

Example 3–51 describes how to create a list object containing a list of clips.

Example 3–51 Create a List Object Containing a List of Clips

```
CREATE TYPE clipNstType AS TABLE of REF clipObject;

CREATE TYPE clipList AS OBJECT (clips clipNstType,
       MEMBER PROCEDURE addClip(c IN REF clipObject));
```

3.4.4 Defining the Implementation of the clipList Object

Example 3–52 describes how to define the implementation of the clipList object.

Example 3–52 Define the Implementation of the clipList Object

```
CREATE TYPE BODY clipList AS
  MEMBER PROCEDURE addClip(c IN REF clipObject)
  IS
    pos INTEGER := 0;
  BEGIN
    IF clips IS NULL THEN
      clips := clipNstType(NULL);
      pos := 0;
    ELSE
      pos := clips.count;
    END IF;
    clips.EXTEND;
    clips(pos+1) := c;
  END;
END;
```

3.4.5 Creating a Video Object and a Video Table

This section describes how to create a video object and a video table of video clips that includes, for each video clip, the following information:

- Item ID
- Duration

- Text content
- Cover image
- Clips

Example 3–53 creates a video object named videoObject and a video table named VideoTable that contains the video information.

Example 3–53 Create a Video Table Containing Video Information

```
CREATE TYPE VideoObject as OBJECT (
        itemId        INTEGER,
        duration      INTEGER,
        txtcontent    CLOB,
        coverImg      REF ORDSYS.ORDImage,
        clips         clipList);

CREATE TABLE VideoTable OF VideoObject (UNIQUE(itemId),itemId NOT NULL)
        NESTED TABLE clips.clips STORE AS clip_store_table;
```

3.4.6 Inserting a Video Clip into the ClipsTable Table

Example 3–54 describes how to insert a video clip into the ClipsTable table.

Example 3–54 Insert a Video Clip into the ClipsTable Table

```
-- Insert a Video Clip into the ClipsTable
insert into ClipsTable values (NULL,
                        '11',
                        'Oracle Commercial',
                        'Larry Ellison',
                        'commercial',
                        'Oracle Corporation',
                        '',
                        'no awards',
                        '90s'
                        'no rating',
                        30,
                        NULL,
                        EMPTY_CLOB(),
                        NULL,
                        ORDSYS.ORDVIDEO.init());
```

3.4.7 Inserting a Row into the VideoTable Table

Example 3–55 describes how to insert a row into the VideoTable table.

Example 3–55 Insert a Row into the VideoTable Table

```
-- Insert a row into the VideoTable
insert into VideoTable values (11,
                               30,
                               NULL,
                               NULL,
                               clipList(NULL));
```

3.4.8 Loading a Video into the ClipsTable Table

Example 3–56 describes how to load a video into the ClipsTable table. This example requires a VIDDIR directory to be defined; see the comments in the example.

Example 3–56 Load a Video into the ClipsTable Table

```
-- Load a Video into a clip
-- Create your directory specification below
-- CREATE OR REPLACE DIRECTORY VIDDIR AS '/video/';
DECLARE
     videoObj ORDSYS.ORDVIDEO;
     ctx RAW(4000) := NULL;
BEGIN
     SELECT C.videoSource INTO videoObj
     FROM   ClipsTable C
     WHERE  C.clipId = '11'
     FOR UPDATE;

     videoObj.setDescription('Under Pressure Video Clip');
     videoObj.setSource('file', 'VIDDIR', 'UnderPressure.mov');
     videoObj.import(ctx);
     videoObj.setProperties(ctx,TRUE)

     UPDATE ClipsTable C
        SET    C.videoSource = videoObj
     WHERE  C.clipId = '11';
     COMMIT;
END;

-- Check video insertion
DECLARE
```

```
     videoObj ORDSYS.ORDVideo;
     ctx RAW(4000) := NULL;
BEGIN
     SELECT C.videoSource INTO videoObj
     FROM   ClipsTable C
     WHERE  C.clipId = '11';

     dbms_output.put_line('Content Length: ' ||
                    videoObj.getContentLength(ctx));
     dbms_output.put_line('Content MimeType: ' ||
                    videoObj.getMimeType());
END;
```

3.4.9 Inserting a Reference to a Clip Object into the Clips List in the VideoTable Table

Example 3–57 describes how to insert a reference to a clip object into the clips list in the VideoTable table.

Example 3–57 Insert a Reference to a Clip Object into the Clips List in the VideoTable Table

```
-- Insert a reference to a ClipObject into the Clips List in the VideoTable
DECLARE
     clipRef          REF ClipObject;
     clipListInstance clipList;
BEGIN
     SELECT REF(C) into clipRef
        FROM   ClipsTable C
     where  C.clipId = '11';

     SELECT V.clips INTO clipListInstance
        FROM   VideoTable V
     WHERE  V.itemId = 11
     FOR UPDATE;

     clipListInstance.addClip(clipRef);

     UPDATE VideoTable V
     SET    V.clips = clipListInstance
     WHERE  V.itemId = 11;

     COMMIT;
END;

-- Check insertion of clip ref
```

```
DECLARE
     clip            ClipObject;
     clipRef         REF ClipObject;
     clipListInstance clipList;
     clipType        clipNstType;
BEGIN
     SELECT V.clips INTO clipListInstance
     FROM   VideoTable V
     WHERE  V.itemId = 11;

     SELECT clipListInstance.clips INTO clipType FROM DUAL;
     clipRef := clipType(1);
     SELECT DEREF(clipRef) INTO clip FROM DUAL;

     dbms_output.put_line('Clip Title: ' ||
                          clip.title);
END;
```

3.4.10 Inserting a Reference to a Video Object into the Clip

Example 3–58 describes how to insert a reference to a video object into the clip.

Example 3–58 Insert a Reference to a Video Object into the Clip

```
-- Insert a reference to a video object into the clip
DECLARE
     aVideoRef REF VideoObject;
BEGIN
-- Make a VideoRef an obj to use for update
     SELECT Cp.videoRef INTO aVideoRef
     FROM   ClipsTable Cp
     WHERE  Cp.clipId = '11'
     FOR UPDATE;

-- Change its value
     SELECT REF(V) INTO aVideoRef
     FROM   VideoTable V
     WHERE  V.itemId = 11;

-- Update database
     UPDATE ClipsTable C
     SET    C.videoRef = aVideoRef
     WHERE  C.clipId = '11';

     COMMIT;
```

```
END;
```

3.4.11 Retrieving a Video Clip from the VideoTable Table

Example 3–59 describes how to retrieve a video clip from the VideoTable table and return it as a BLOB. The program segment performs these operations:

1. Defines the retrieveVideo( ) method to retrieve the video clip by its clipId as an ORDVideo BLOB.
2. Selects the desired video clip (where `C.clipId = clipId`) and returns it using the getContent method.

Example 3–59 Retrieve a Video Clip

```
FUNCTION retrieveVideo(clipId IN INTEGER)
RETURN BLOB IS
 obj ORDSYS.ORDVideo;

BEGIN
   -- Select the desired video clip from the ClipTable table.
   SELECT C.videoSource INTO obj from ClipTable C
          WHERE C.clipId = clipId;
   return obj.getContent;
END;
```

3.4.12 Extending *inter*Media to Support a New Video Data Format

This section describes how to extend Oracle *inter*Media to support a new video data format.

To support a new video data format, implement the required interfaces in the ORDX_<format>_VIDEO package in the ORDPLUGINS schema (where <format> represents the name of the new video data format). See Section 9.4.1 for a complete description of the interfaces for the ORDX_DEFAULT_VIDEO package. Use the package body example in Section 9.4.2 as a template to create the video package body.

Then set the new format parameter in the setFormat call to the appropriate format value to indicate to the video object that package ORDPLUGINS.ORDX_<format> _ VIDEO is available as a plug-in.

See Section F.4 for more information on installing your own format plug-in and running the sample scripts provided. See the fplugins.sql and fpluginb.sql files that are installed in the *$ORACLE_HOME*/ord/vid/demo/ directory. These are demonstration

(demo) plug-ins that you can use as a guideline to write any format plug-in that you want to support. See the viddemo.sql file in this same directory to learn how to install your own format plug-in.

3.4.13 Extending *inter*Media with a New Object Type

This section describes how to extend Oracle *inter*Media with a new object type.

You can use the ORDVideo type as the basis for a new type of your own creation.

See Example 3–45 for a more complete example and description.

Note: When a type is altered any dependent type definitions are invalidated. You will encounter this problem if you define a new type that includes an ORDVideo attribute and the *inter*Media ORDVideo type is altered, which always occurs during an *inter*Media installation upgrade.

A workaround to this problem is to revalidate all invalid type definitions with the following SQL statement:

```
SQL> ALTER TYPE <type-name> COMPILE;
```

3.4.14 Using Video Types with Object Views

This section describes how to use video types with object views. Just as a view is a virtual table, an object view is a virtual object table.

Oracle provides object views as an extension of the basic relational view mechanism. By using object views, you can create virtual object tables from data -- of either built-in or user-defined types -- stored in the columns of relational or object tables in the database.

Object views can offer specialized or restricted access to the data and objects in a database. For example, you might use an object view to provide a version of an employee object table that does not have attributes containing sensitive data or a deletion method. Object views also let you try object-oriented programming without permanently converting your tables. Using object views, you can convert data gradually and transparently from relational tables to object-relational tables.

In Example 3–60, consider the following relational table (containing no ORDVideo objects).

Example 3–60 Define a Relational Table Containing No ORDVideo Object

```
create table flat (
   id                number,
   description       VARCHAR2(4000),
   localData         BLOB,
   srcType           VARCHAR2(4000),
   srcLocation       VARCHAR2(4000),
   srcName           VARCHAR2(4000),
   upDateTime        DATE,
   local             NUMBER,
   format            VARCHAR2(31),
   mimeType          VARCHAR2(4000),
   comments          CLOB,
   width             INTEGER,
   height            INTEGER,
   frameResolution   INTEGER,
   frameRate         INTEGER,
   videoDuration     INTEGER,
   numberOfFrames    INTEGER,
   compressionType   VARCHAR2(4000),
   numberOfColors    INTEGER,
   bitRate           INTEGER,
   );
```

You can create an object view on the relational table shown in Example 3–60 as follows in Example 3–61.

Example 3–61 Define an Object View Containing an ORDVideo Object and Relational Columns

```
create or replace view object_video_v as
  select
      id,
      ORDSYS.ORDVideo(
      ORDSYS.ORDSource(
         T.localData, T.srcType, T.srcLocation, T.srcName, T.updateTime,
         T.local),
         T.description,
         T.format,
         T.mimeType,
         T.comments,
         T.width,
         T.height,
         T.frameResolution,
         T.frameRate,
```

```
        T.videoDuration,
        T.numberofFrames,
        T.compressionType,
        T.numberOfColors,
        T.bitRate) VIDEO
from flat T;
```

Object views provide the flexibility of looking at the same relational or object data in more than one way. Therefore, you can use different in-memory object representations for different applications without changing the way you store the data in the database. See the *Oracle9i Database Concepts* manual for more information on defining, using, and updating object views.

3.4.15 Scripts for Creating and Populating a Video Table from a BFILE Data Source

The following scripts can be found on the Oracle Technology Network (OTN) Web site: `http://otn.oracle.com/` as end-to-end scripts that create and populate a video table from a BFILE data source. You can get to this site by selecting the Oracle *inter*Media Plugins and Utilities page and from the *inter*Media page, select Sample Code.

The following set of scripts:

1. Creates a tablespace for the video data, creates a user and grants certain privileges to this new user, creates a video data load directory (create_viduser.sql).
2. Creates a table with two columns, inserts two rows into the table and initializes the object column to empty with a locator (create_vidtable.sql).
3. Loads the video data with a SELECT FOR UPDATE operation using an import method to import the data from a BFILE (importvid.sql).
4. Performs a check of the properties for the loaded data to ensure that it is really there (chkprop.sql).

The fifth script (setup_vidschema.sql) automates this entire process by running each script in the required order. The last script (readvideo.sql) creates a stored procedure that performs a SELECT operation to read a specified amount of video data from the BLOB, beginning at a particular offset, until all the video data is read. To successfully load the video data, you must have a *viddir* directory created on your system containing the vid1.mov and vid2.mov files, which are installed in the *<ORACLE_HOME>*`/ord/vid/demo` directory; this directory path and disk drive must be specified in the CREATE DIRECTORY statement in the create_viduser.sql file.

Script 1: Create a Tablespace, Create a Video User, Grant Privileges to the Video User, and Create a Video Data Load Directory (create_viduser.sql)

This script creates the viddemo tablespace with a data file named viddemo.dbf of 200MB in size, with an initial extent of 64K, a next extent of 128K, and turns on table logging. Next, the viddemo user is created and given connect, resource, create library, and create directory privileges followed by creating the video data load directory.

> **Note:** You must edit the create_viduser.sql file and either enter the system password in the connect statement or comment out the connect statement and run this file in the system account. You must specify the disk drive in the CREATE DIRECTORY statement. Also, create the temp temporary tablespace if you have not already created it, otherwise this file will not run.

```
-- create_viduser.sql

-- Connect as admin.
connect system/<system password>;

-- Edit this script and either enter your system password here
-- to replace <system password> or comment out this connect
-- statement and connect as system before running this script.

set serveroutput on
set echo on

-- Need system manager privileges to delete a user.
-- Note: There is no need to delete viddemo user if you do not
-- delete the viddemo tablespace, therefore comment out the next line.

-- drop user viddemo cascade;

-- Need system manager privileges to delete a directory. If there is no
-- need to really delete it, then comment out the next line.

-- drop directory viddir;

-- Delete then create tablespace.

-- Note: It is better to not delete and create tablespaces,
```

```
-- so comment this next line out. The create tablespace statement
-- will fail if it already exists.

-- drop tablespace viddemo including contents;

-- If you uncomment the previous line and want to delete the
-- viddemo tablespace, remember to manually delete the viddemo.dbf
-- file to complete the operation. Otherwise, you cannot create
-- the viddemo tablespace again because the viddemo.dbf file
-- already exists. Therefore, it might be best to create this
-- tablespace once and not delete it.

-- Create tablespace.
create tablespace viddemo
       datafile 'viddemo.dbf' size 200M
       minimum extent 64K
       default storage (initial 64K next 128K)
       logging;

-- Create viddemo user.
create user viddemo identified by viddemo
default tablespace viddemo
temporary tablespace temp;

-- Note: If you do not have a temp tablespace already defined, you
-- will have to create it first for this script to work.

grant connect, resource, create library to viddemo;
grant create any directory to viddemo;

-- Note: If this user already exists, you get an error message
-- when you try and create this user again.

-- Connect as viddemo.
connect viddemo/viddemo

-- Create the viddir load directory; this is the directory where the video
-- files are residing.

create or replace directory viddir
       as 'e:\oracle\ord\vid\demo';
grant read on directory viddir to public with grant option;

-- Note: If this directory already exists, an error message
-- is returned stating the operation will fail; ignore the message.
```

Script 2: Create the Video Table and Initialize the Column Object (create_vidtable.sql)

This script creates the video table and then performs an insert operation to initialize the column object to empty for two rows. Initializing the column object creates the BLOB locator that is required for populating each row with BLOB data in a subsequent data load operation.

```
--create_vidtable.sql
connect viddemo/viddemo;
set serveroutput on
set echo on

drop table vidtable;
create table vidtable (id number,
       Video ordsys.ordVideo);

-- Insert a row with empty BLOB.
insert into vidtable values(1,ORDSYS.ORDVideo.init());

-- Insert a row with empty BLOB.
insert into vidtable values(2,ORDSYS.ORDVideo.init());
commit;
```

Script 3: Load the Video Data (importvid.sql)

This script performs a SELECT FOR UPDATE operation to load the video data by first setting the source for loading the video data from a file, importing the data, setting the properties for the BLOB data, updating the row, and committing the transaction. To successfully run this script, you must copy two video clips to your VIDDIR directory using the names specified in this script, or modify this script to match the file names of your video clips.

```
-- importvid.sql

set serveroutput on
set echo on
-- Import the two files into the database.

DECLARE
  obj ORDSYS.ORDVIDEO;
  ctx RAW(4000) := NULL;

BEGIN
-- This imports the video file vid1.mov from the VIDDIR directory
-- on a local file system (srcType=file) and sets the properties.
```

```
  select Video into obj from vidtable where id = 1 for update;
  obj.setSource('file','VIDDIR','vid1.mov');
  obj.import(ctx);
  obj.setProperties(ctx);

  update vidtable set video = obj where id = 1;
  commit;

-- This imports the video file vid2.mov from the VIDDIR directory
-- on a local file system (srcType=file) and sets the properties.

  select Video into obj from vidtable where id = 2 for update;
  obj.setSource('file','VIDDIR','vid2.mov');
  obj.import(ctx);
  obj.setProperties(ctx);

  update vidtable set video = obj where id = 2;
  commit;
END;
/
```

Script 4: Check the Properties of the Loaded Data (chkprop.sql)

This script performs a SELECT operation of the rows of the video table, then gets the video characteristics of the BLOB data to check that the BLOB data is in fact loaded.

```
--chkprop.sql
set serveroutput on;
--connect viddemo/viddemo
--Query vidtable for ORDSYS.ORDVideo.
DECLARE
  video ORDSYS.ORDVideo;
  idnum integer;
  properties_match BOOLEAN;
  ctx RAW(4000) := NULL;
  width integer;
  height integer;

BEGIN
 FOR I IN 1..2 LOOP
  SELECT id, video into idnum, video from vidtable where id=I;
   dbms_output.put_line('video id:          '|| idnum);
```

```
  properties_match := video.checkProperties(ctx);
  IF properties_match THEN DBMS_OUTPUT.PUT_LINE('Check Properties Succeeded');
  END IF;

 --dbms_output.put_line('video frame rate:     '|| video.getFrameRate(ctx));
 --dbms_output.put_line('video width & height:   '|| video.getFrameSize(ctx,width,height);
 dbms_output.put_line('video MIME type:       '|| video.getMimeType());
 dbms_output.put_line('video file format:     '|| video.getFormat(ctx));
 dbms_output.put_line('BLOB Length:            '|| TO_CHAR(video.getContentLength(ctx)));
 dbms_output.put_line('---------------------------------------------');
 END loop;
END;
/
```

Results from running the script chkprop.sql are the following:

```
SQL> @chkprop.sql
video id:          1
Check Properties Succeeded
video MIME type:        video/quicktime
video file format:      MOOV
BLOB Length:            4958415
---------------------------------------------
video id:          2
Check Properties Succeeded
video MIME type:        video/quicktime
video file format:      MOOV
BLOB Length:            2891247
---------------------------------------------
```

Automated Script (setup_vidschema.sql)

This script runs each of the previous four scripts in the correct order to automate this entire process.

```
-- setup_vidschema.sql
-- Create viddemo user, tablespace, and load directory to
-- hold the video files:
@create_viduser.sql

-- Create Video table:
@create_vidtable.sql

--Import 2 video clips and set properties:
@importvid.sql

--Check the properties of the video clips:
@chkprop.sql
```

```
--exit;
```

Read Data from the BLOB (readvideo.sql)

This script creates a stored procedure that performs a SELECT operation to read a specified amount of video data from the BLOB, beginning at a particular offset, until all the video data is read.

```
-- readvideo.sql

set serveroutput on
set echo on

create or replace procedure readvideo as

   obj ORDSYS.ORDVideo;
   buffer RAW (32767);
   numbytes BINARY_INTEGER := 32767;
   startpos integer := 1;
   read_cnt integer := 1;
   ctx RAW(4000) := NULL;

BEGIN

   Select video into obj from vidtable where id = 1;

   LOOP
           obj.readFromSource(ctx,startpos,numbytes,buffer);
          DBMS_OUTPUT.PUT_LINE('Content length is: '|| TO_CHAR(obj.getContentLength()));
           DBMS_OUTPUT.PUT_LINE('start position: '|| startpos);
           DBMS_OUTPUT.PUT_LINE('doing read  '|| read_cnt);
          startpos := startpos + numbytes;
          read_cnt := read_cnt + 1;

-- Note: Add your own code here to process the video data being read;
-- this routine just reads the data into the buffer 32767 bytes
-- at a time, then reads the next chunk, overwriting the first
-- buffer full of data.
   END LOOP;

EXCEPTION

   WHEN NO_DATA_FOUND THEN
   DBMS_OUTPUT.PUT_LINE('End of data ');
```

```
    DBMS_OUTPUT.PUT_LINE('----------------');

    WHEN ORDSYS.ORDSourceExceptions.METHOD_NOT_SUPPORTED THEN
     DBMS_OUTPUT.PUT_LINE('ORDSourceExceptions.METHOD_NOT_SUPPORTED caught');
    WHEN OTHERS THEN
     DBMS_OUTPUT.PUT_LINE('EXCEPTION caught');

END;
/
show errors
```

To execute the stored procedure, enter the following SQL statements:

```
SQL> set serveroutput on;
SQL> execute readvideo
Content Length: 4958415
start position: 1
doing read  1
start position: 32768
doing read  2
start position: 65535
.
.
.
doing read  151
start position: 4947818
doing read  152
----------------
End of data

PL/SQL procedure successfully completed.
```

3.5 Extending *inter*Media to Support a New Data Source

This section describes how to extend Oracle *inter*Media to support a new data source.

To support a new data source, implement the required interfaces in the ORDX_<srcType>_SOURCE package in the ORDPLUGINS schema (where <srcType> represents the name of the new external source type). See Section I.3.1 and Section I.3.2 for a complete description of the interfaces for the ORDX_FILE_SOURCE and ORDX_HTTP_SOURCE packages. See Section I.3.4 for an example of modifying the package body listing that is provided. Then set the source type parameter in the setSourceInformation call to the appropriate source type to

indicate to the video object that package ORDPLUGINS.ORDX_<srcType>_SOURCE is available as a plug-in.

4

Ensuring Future Compatibility with Evolving *inter*Media Object Types

The *inter*Media object types may evolve by adding new object attributes in a future release. Client-side applications that want to maintain compatibility with the current release of the *inter*Media object types (ORDAudio, ORDImage, ORDVideo, and ORDSource), even after a server upgrade that includes evolved object types, are advised to do the following:

- Make a call to the compatibility initialization function at the beginning of the application, if necessary (see Section 4.1).
- Use the static constructor functions, init( ), in INSERT statements that are provided beginning with release 8.1.7 (see Section 6.2, Section 8.1.1, and Section 9.2). Do not use the default constructors because INSERT statements using the default constructor will fail if the *inter*Media object types have added new attributes.

> **Note:** If you do not do the preceding recommended actions, you may have to upgrade your client and perhaps even recompile your application when you upgrade to a newer server release with evolved types.

4.1 When and How to Call the Compatibility Initialization Function

Only client-side applications that statically recognize the structure of the *inter*Media object types need to make a call to the compatibility initialization function. Server-side stored procedures will automatically use the newly installed (potentially changed) *inter*Media object types after an upgrade, so you do not need to call the compatibility initialization function from server-side stored procedures.

Client-side applications that do not statically (at compile time) recognize the structure of *inter*Media object types do not need to call the compatibility initialization function. OCI applications that determine the structure of the *inter*Media object types at runtime, through the OCIDescribeAny call, do not need to call the compatibility initialization function.

Client-side applications written in OCI that have been compiled with the C structure of *inter*Media object types (generated by OTT) should make a call to the server-side PL/SQL function, ORDSYS.IM.compatibilityInit( ), at the beginning of the application. See the compatibilityInit( ) method description of this function in this section.

Client-side applications written in Java using the *inter*Media Java Classes for release 8.1.7 or higher, should call the OrdMediaUtil.imCompatibilityInit( ) function after connecting to the Oracle database server.

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

This Java function takes OracleConnection as an argument. The included *inter*Media release 8.1.7 or higher Java API will ensure that your 8.1.7 or higher application will work (without upgrading) with a potential future release of *inter*Media with evolved object types.

There is not yet a way for client-side PL/SQL applications to maintain compatibility with the current release of the *inter*Media object types if the objects add new attributes in a future release.

See the compatibilityInit( ) method in this section, and *Oracle interMedia Java Classes User's Guide and Reference* for further information, and detailed descriptions and examples. This guide is on the Oracle Technology Network, `http://otn.oracle.com/`.

compatibilityInit( )

Format

```
compatibilityInit(release IN    VARCHAR2,
                  errmsg OUT VARCHAR2)
RETURN NUMBER;
```

Description

Allows for compatibly evolving the *inter*Media object types in a future release.

Parameters

release
The release number. This string should be set to '9.0.1' to allow a 9.0.1 application to work (without upgrading) with a potential future release of the Oracle database and *inter*Media with evolved object types.

errmsg
String output parameter. If the function returns a status other than 0, this errmsg string contains the reason for the failure.

Pragmas

None.

Exceptions

None.

Usage Notes

You should begin using the compatibilityInit( ) method as soon as possible to ensure you will not have to upgrade the Oracle software on your client node, or recompile your client application in order to work with a future release of the Oracle database server if the *inter*Media object types change in a future release. See Section 4.1 to determine if you need to call this function.

The compatibility initialization function for *inter*Media is located in the ORDSYS.IM package.

Examples

Using OCI and setting the compatibilityInit( ) method release parameter to release 9.0.1 to allow a 9.0.1 application to work with a future release of the Oracle database and *inter*Media with changed object types; note, that this is not a standalone program in that it assumes that you have allocated handles beforehand:

```
void prepareExecuteStmt( OCIEnv *envHndl,
            OCIStmt **stmtHndl,
            OCIError *errorHndl,
            OCISvcCtx *serviceCtx,
            OCIBind *bindhp[] )
{
   text *statement = (text *)
      "begin :sts := ORDSYS.IM.compatibilityInit( :vers, :errText );
end;";
  sword sts = 0;
  text *vers = (text *)"9.0.1";
  text errText[512];
  sb2 nullInd;

  printf( " Preparing statement\n" );

  OCIHandleAlloc( envHndl, (void **) stmtHndl, OCI_HTYPE_STMT, 0, NULL
);

  OCIStmtPrepare( *stmtHndl, errorHndl, (text *)statement,
         (ub4)strlen( (char *)statement ), OCI_NTV_SYNTAX,
         OCI_DEFAULT );

  printf( " Executing statement\n" );

  OCIBindByPos( *stmtHndl, &bindhp[ 0 ], errorHndl, 1, (void *)&sts,
        sizeof( sts ), SQLT_INT, (void *)0, NULL, 0, 0,
        NULL, OCI_DEFAULT );

  OCIBindByPos( *stmtHndl, &bindhp[ 1 ], errorHndl, 2, vers,
        strlen((char *)vers) + 1, SQLT_STR, (void *)0, NULL,
        0, 0, NULL, OCI_DEFAULT );

  OCIBindByPos( *stmtHndl, &bindhp[ 2 ], errorHndl, 3, errText,
        sizeof( errText ), SQLT_STR, &nullInd, NULL, 0, 0,
        NULL, OCI_DEFAULT);

  OCIStmtExecute( serviceCtx, *stmtHndl, errorHndl, 1, 0,
        (OCISnapshot *)NULL, (OCISnapshot *)NULL, OCI_DEFAULT );
```

```
  printf( " Statement executed\n" );
  if (sts != 0)
    {
    printf( "CompatibilityInit failed with Sts = %d\n", sts );
    printf( "%s\n", errText );
    }


}
```

5

Common Methods for *inter*Media Object Types Reference Information

This chapter presents reference information on the common methods used for the following Oracle *inter*Media data types:

- ORDAudio
- ORDDoc
- ORDImage
- ORDVideo

See Section 5.2 for a list of methods described in this chapter.

The examples in this chapter assume that you have created the test tables as described in Section 6.3.1, Section 7.3.1, Section 8.1.3, and Section 9.3.1.

> **Note:** The *inter*Media methods are designed to be internally consistent. If you use *inter*Media methods (such as import( ) or image process( )) to modify the media data, *inter*Media will ensure that object attributes remain synchronized with the media data. However, if you manipulate the data itself (by either directly modifying the BLOB or changing the external source), then you must ensure that the object attributes stay synchronized and the update time is modified; otherwise, the object attributes will not match the data.

5.1 Important Notes

Methods invoked at the ORDSource level that are handed off to a source plug-in for processing have ctx (RAW(4000)) as the first argument. Before calling any of these methods for the first time, the client must allocate the ctx structure, initialize it to NULL, and invoke the openSource( ) method. At this point, the source plug-in can initialize context for this client. When processing is complete, the client should invoke the closeSource( ) method.

Methods invoked at the ORDAudio, ORDDoc, or ORDVideo level that are handed off to a format plug-in for processing have ctx (RAW(4000)) as the first argument. Before calling any of these methods for the first time, the client must allocate the ctx structure and initialize it to NULL.

> **Note:** In the current release, not all source plug-ins and format plug-ins will use the ctx argument, but if you code as previously described, your application should work with any current or future source and format plug-in.

For ORDAudio, ORDDoc, or ORDVideo object types, you should use any of the individual set methods to set the value of the attribute for an object for formats not natively supported or write a format plug-in and call setProperties( ); otherwise, for formats natively supported, use the setProperties( ) method to populate the attributes of the object.

For ORDImage object types, use the setProperties( ) method to populate the attributes of the object. Use the setProperties( ) for Foreign Images method for foreign image formats.

5.2 Methods

This section presents reference information on the Oracle *inter*Media methods that are common to all object types. These common methods are described in the following groupings. Other methods, which are particular to a particular object type or which are implemented differently for the different object types, are described in Section 6.3, Section 7.3, Section 8.1.2, and Section 9.3.

Common Methods Associated with the updateTime Attribute

- getUpdateTime( ): returns the time when the object was last updated. See "getUpdateTime( )" on page 5-25 for information.
- setUpdateTime( ): sets the update time for the object. This method is called implicitly by methods that modify the media data. See "setUpdateTime( )" on page 5-39 for information.

Common Methods Associated with mimeType Attribute

- setMimeType( ): sets the MIME type of the stored data. This method is called implicitly by methods for natively supported formats. See "setMimeType( )" on page 5-35 for information.
- getMimeType( ): returns the MIME type of the stored data. See "getMimeType( )" on page 5-17 for information.

Common Methods Associated with the source Attribute

- processSourceCommand( ): sends a command and related arguments to the source plug-in. See "processSourceCommand( )" on page 5-29 for information.
- isLocal( ): returns TRUE if the data is stored locally in a BLOB or FALSE if the data is external. See "isLocal( )" on page 5-26 for information.
- setLocal( ): sets a flag to indicate that the data is stored locally in a BLOB. See "setLocal( )" on page 5-34 for information.
- clearLocal( ): clears the flag to indicate that the data is stored externally. See "clearLocal( )" on page 5-5 for information.
- setSource( ): sets the source information to where data is found. See "setSource( )" on page 5-37 for information.
- getSource( ): returns a formatted string containing complete information about the external data source formatted as a URL. See "getSource( )" on page 5-19 for information.

- getSourceLocation( ): returns the external source location of the data. See "getSourceLocation( )" on page 5-21 for information.
- getSourceName( ): returns the external source name of the data. See "getSourceName( )" on page 5-22 for information.
- getSourceType( ): returns the external source type of the data. See "getSourceType( )" on page 5-23 for information.
- export( ): copies data from a local source (localData) within an Oracle database to the specified external data source, and stores source information in the source. See "export( )" on page 5-9 for information.

Note: The export( ) method natively supports only sources of source type file. User-defined sources may support the export( ) method.

- getContent( ): returns the handle to the BLOB used to store contents locally. See "getContent( )" on page 5-15 for information.
- deleteContent( ): deletes the content of the local BLOB. See "deleteContent( )" on page 5-8 for information.
- getBFILE( ): returns the external content as a BFILE. See "getBFILE( )" on page 5-13 for information.

Common Methods Associated with File Operations

- openSource( ): opens a data source or a BLOB. See "openSource( )" on page 5-27 for information.
- closeSource( ): closes a data source or a BLOB. See "closeSource( )" on page 5-6 for information.
- trimSource( ): trims a data source or a BLOB. See "trimSource( )" on page 5-40 for information.
- readFromSource( ): reads a buffer of *n* bytes from a source beginning at a start position. See "readFromSource( )" on page 5-32 for information.
- writeToSource( ): writes a buffer of *n* bytes to a source beginning at a start position. See "writeToSource( )" on page 5-42 for information.

For more information on object types and methods, see *Oracle9i Database Concepts*.

clearLocal( )

Format

clearLocal( );

Description

Resets the local flag to indicate that the data is stored externally. When the local flag is set to clear, media methods look for corresponding data using the srcLocation, srcName, and srcType attributes.

Parameters

None.

Usage Notes

This method sets the local attribute to a 0, meaning the data is stored externally or outside of Oracle9*i*.

Pragmas

None.

Exceptions

None.

Examples

Clear the value of the local flag for the data:

```
DECLARE
 obj ORDSYS.ORDAudio;
BEGIN
 SELECT aud INTO obj FROM TAUD WHERE N = 1 FOR UPDATE;
 obj.clearLocal();
 UPDATE TAUD SET aud=obj WHERE N = 1;
 COMMIT;
END;
/
```

closeSource( )

Format

closeSource(ctx IN OUT RAW) RETURN INTEGER;

Description

Closes a data source.

Parameters

ctx
The source plug-in context information. You must call the openSource( ) method; see Section 5.1 on page 5-2 for more information.

Usage Notes

The return INTEGER is 0 (zero) for success and >0 (for example, 1) for failure. The exact number and the meaning for that number is plug-in defined. For example, for the file plug-in, 1 might mean "File not found," 2 might mean "No such directory," and so forth.

Pragmas

None.

Exceptions

ORDSourceExceptions.INCOMPLETE_SOURCE_INFORMATION

This exception is raised if you call the closeSource( ) method and the value for srcType is NULL and data is not local.

ORDSourceExceptions.METHOD_NOT_SUPPORTED

This exception is raised if you call the closeSource( ) method and this method is not supported by the source plug-in being used.

ORDSourceExceptions.SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the closeSource( ) method within a source plug-in when any other exception is raised.

See Appendix H for more information about these exceptions.

Examples

Close an external data source:

```
DECLARE
  obj ORDSYS.ORDAudio;
  res INTEGER;
  ctx RAW(4000) :=NULL;
BEGIN
  SELECT aud INTO obj FROM taud WHERE N =2 FOR UPDATE;
  res := obj.closeSource(ctx);
  UPDATE TAUD SET aud=obj WHERE N=2 ;
  COMMIT;
  EXCEPTION
   WHEN ORDSYS.ORDSourceExceptions.METHOD_NOT_SUPPORTED THEN
    DBMS_OUTPUT.put_line('ORDSourceExceptions.METHOD_NOT_SUPPORTED caught');
   WHEN OTHERS THEN
    DBMS_OUTPUT.put_line('EXCEPTION caught');
END;
/
```

deleteContent( )

Format

deleteContent( );

Description

Deletes the local data from the current local source (localData).

Parameters

None.

Usage Notes

This method can be called after you export the data from the local source to an external data source and you no longer need this data in the local source.

Call this method when you want to update the object with a new object.

Pragmas

None.

Exceptions

None.

Examples

Delete the local data from the current local source:

```
DECLARE
    Image ORDSYS.ORDImage;
BEGIN
    SELECT large_photo INTO Image FROM emp
      WHERE ename = 'John Doe' FOR UPDATE;
    -- delete the local content of the image
    Image.deleteContent();
    UPDATE emp SET large_photo = Image WHERE ename = 'John Doe';
END;
/
```

export( )

Format

```
export(
    ctx                 IN OUT RAW,
    source_type         IN VARCHAR2,
    source_location IN VARCHAR2,
    source_name         IN VARCHAR2);
```

Description

Copies data from a local source (localData) within an Oracle database to a corresponding external data source.

> **Note:** The export( ) method natively supports only sources of source type file. User-defined sources may support the export( ) method.

Parameters

ctx
The source plug-in context information.

source_type
The source type of the location to where the data is to be exported.

source_location
The location where the data is to be exported.

source_name
The name of the object to where the data is to be exported.

Usage Notes

After exporting data, all attributes remain unchanged and srcType, srcLocation, and srcName are updated with input values. After calling the export( ) method, you can

call the clearLocal( ) method to indicate the data is stored outside the database and call the deleteContent( ) method if you want to delete the content of the local data.

This method is also available for user-defined sources that can support the export method.

The export( ) method for a source type of file is similar to a file copy operation in that the original data stored in the BLOB is not touched other than for reading purposes.

The export( ) method is not an exact mirror operation to the import( ) method in that the clearLocal( ) method is not automatically called to indicate the data is stored outside the database, whereas the import( ) method automatically calls the setLocal( ) method.

Call the deleteContent( ) method after calling the export( ) method to delete the content from the database if you no longer intend to manage the multimedia data within the database.

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

For example, the following grants the user, MEDIAUSER, the permission to write to the file named filename.dat:

```
CALL DBMS_JAVA.GRANT_PERMISSION(
        'MEDIAUSER',
        'java.io.FilePermission',
        '/actual/server/directory/path/filename.dat',
        'write');
```

See the security and performance section in *Oracle9i Java Developer's Guide* for more information.

Invoking this method implicitly calls the setUpdateTime( ) method.

Pragmas

None.

Exceptions

ORDSourceExceptions.INCOMPLETE_SOURCE_INFORMATION

This exception is raised if you call the export( ) method and the value of srcType is NULL.

ORDSourceExceptions.METHOD_NOT_SUPPORTED

This exception is raised if you call the export( ) method and this method is not supported by the source plug-in being used.

ORDSourceExceptions.SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the export( ) method within a source plug-in when any other exception is raised.

See Appendix H for more information about these exceptions.

ORDSourceExceptions.IO_ERROR

This exception is raised if the export( ) method encounters an error writing the BLOB data to the file specified.

Examples

Export data from a local source to an external data source.

> **Note:** You must first create the directory to which you want to export your data. Create this directory using the following SQL statement and then grant read access to PUBLIC to this directory. Change this directory specification to match the location to where you want to export your media files.

```
-- Create the directory to which you
-- want to export your files.

CREATE OR REPLACE DIRECTORY docdir
       as 'e:\<ORACLE_HOME>\ord\doc\demo';
GRANT READ ON DIRECTORY docdir TO PUBLIC WITH GRANT OPTION;

SET ECHO ON;
SET SERVEROUTPUT ON;
CONNECT SYSTEM AS SYSDBA;
BEGIN
  DBMS_JAVA.GRANT_PERMISSION('PUBLIC', 'java.io.FilePermission', 'e:\ORACLE_
HOME>\ord\doc\demo\testdoc.dat',
                              'WRITE');
```

```
  COMMIT;
END;
/
DECLARE
  obj ORDSYS.ORDDoc;
  ctx RAW(4000) :=NULL;
BEGIN
   SELECT doc INTO obj FROM tdoc WHERE N = 1 FOR UPDATE;
   obj.export(ctx,'file','DOCDIR','testdoc.dat');
  EXCEPTION
  WHEN ORDSYS.ORDSourceExceptions.METHOD_NOT_SUPPORTED THEN
   DBMS_OUTPUT.put_line('Source METHOD_NOT_SUPPORTED caught');
  WHEN ORDSYS.ORDSourceExceptions.SOURCE_PLUGIN_EXCEPTION THEN
   DBMS_OUTPUT.put_line('SOURCE PLUGIN EXCEPTION caught');
  WHEN OTHERS THEN
   DBMS_OUTPUT.put_line('OTHER EXCEPTION caught');
END;
/
```

getBFILE( )

Format

getBFILE( ) RETURN BFILE;

Description

Returns the LOB locator of the BFILE containing the media.

Parameters

None.

Usage Notes

This method constructs and returns a BFILE using the stored source.srcLocation and source.srcName attribute information. The source.srcLocation attribute must contain a defined directory object. The source.srcName attribute must be a valid file name and source.srcType must be "file".

Pragmas

PRAGMA RESTRICT_REFERENCES(getBFILE, WNDS, WNPS, RNDS, RNPS)

Exceptions

If the source.srcType attribute value is NULL, calling this method raises an INCOMPLETE_SOURCE_INFORMATION exception.

If the value of srcType is other than file, then calling this method raises an INVALID_SOURCE_TYPE exception.

Examples

Return the BFILE for the stored source directory and file name attributes:

```
DECLARE
    obj ORDSYS.ORDVideo;
    videobfile BFILE;
BEGIN
    SELECT vid INTO obj FROM tvid
      WHERE N=1;
    -- get the video BFILE
```

```
    videobfile := obj.getBFILE();
END;
/
```

getContent( )

Format

getContent( ) RETURN BLOB;

Description

Returns a handle to the local BLOB storage, that is the BLOB within the object.

Parameters

None.

Usage Notes

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getContent, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Examples

A client accesses video data to be put on a Web-based player:

```
DECLARE
  obj ORDSYS.ORDVideo;
BEGIN
  SELECT vid INTO obj FROM TVID  WHERE N=1 FOR UPDATE;
  -- import data
  obj.importFrom(ctx,'file','VIDEODIR','MV1.AVI');
  -- check size
  DBMS_OUTPUT.PUT_LINE(TO_CHAR(obj.getContentLength(ctx)));
  DBMS_OUTPUT.PUT_LINE(TO_CHAR(DBMS_LOB.GETLENGTH(obj.getContent())));
  DBMS_OUTPUT.PUT_LINE(obj.getSource());
  DBMS_OUTPUT.PUT_LINE('deleting contents');
  DBMS_OUTPUT.PUT_LINE('-----------------');
  obj.deleteContent();
  DBMS_OUTPUT.PUT_LINE(TO_CHAR(obj.getContentLength(ctx)));
```

```
  UPDATE TVID SET vid=obj WHERE N=1;
  COMMIT;
END;
/
```

getMimeType( )

Format

getMimeType( ) RETURN VARCHAR2;

Description

Returns the MIME type for the data. This is a simple access method that returns the value of the mimeType attribute.

Parameters

None.

Usage Notes

If the source is an HTTP server, the MIME type information is read from the HTTP header information when the media is imported and stored in the object attribute. If the source is a file or BLOB, the MIME type information is extracted when the setProperties( ) method is called.

For unrecognized file formats, users must call the setMimeType( ) method and specify the MIME type.

Use this method rather than accessing the mimeType attribute directly to protect yourself from potential changes to the internal representation of the object.

Pragmas

PRAGMA RESTRICT_REFERENCES(getMimeType, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Examples

Set the MIME type for some stored image data:

```
DECLARE
    Image ORDSYS.ORDImage;
BEGIN
    SELECT large_photo INTO Image FROM emp
```

```
      WHERE ename = 'John Doe';
    -- set the image mime type
    Image.setMimeType('image/myformat');
END;
/
```

getSource( )

Format

getSource( ) RETURN VARCHAR2;

Description

Returns information about the external location of the data in URL format.

Parameters

None.

Usage Notes

Possible return values are:

- FILE:/ /<DIR OBJECT NAME>/<FILE NAME> for a file source
- HTTP:/ /<URL> for an HTTP source
- User-defined source; for example, TYPE:/ /<USER-DEFINED SOURCE LOCATION>/<USER-DEFINED SOURCE NAME>

Pragmas

PRAGMA RESTRICT_REFERENCES(getSource, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Examples

Get the source of the image data:

```
DECLARE
    Image ORDSYS.ORDImage;
    SourceInfo VARCHAR2(4000);
BEGIN
    SELECT large_photo INTO Image FROM emp
      WHERE ename = 'John Doe';
    -- get the image source information
    SourceInfo := Image.getSource();
```

```
END;
```

getSourceLocation( )

Format

getSourceLocation( ) RETURN VARCHAR2;

Description

Returns a string containing the value of the external data source location.

Parameters

None.

Usage Notes

This method returns a VARCHAR2 string containing the value of the external data location, for example "BFILEDIR".

Pragmas

PRAGMA RESTRICT_REFERENCES(getSourceLocation, WNDS, WNPS, RNDS, RNPS)

Exceptions

ORDSourceExceptions.INCOMPLETE_SOURCE_LOCATION

This exception is raised if you call the getSourceLocation method and the value of srcLocation is NULL.

Examples

Get the source location information about an image data source:

```
DECLARE
    Image ORDSYS.ORDImage;
    SourceLocation VARCHAR2(4000);
BEGIN
    SELECT large_photo INTO Image FROM emp
      WHERE ename = 'John Doe';
    -- get the image source location
    SourceLocation := Image.getSourceLocation();
END;
```

getSourceName( )

Format

getSourceName( ) RETURN VARCHAR2;

Description

Returns a string containing of the name of the external data source.

Parameters

None.

Usage Notes

This method returns a VARCHAR2 string containing the name of the external data source, for example "testimg.dat".

Pragmas

PRAGMA RESTRICT_REFERENCES(getSourceName, WNDS, WNPS, RNDS, RNPS)

Exceptions

ORDSourceExceptions.INCOMPLETE_SOURCE_NAME

This exception is raised if you call the getSourceName( ) method and the value of srcName is NULL.

Examples

Get the source name information about an image data source:

```
DECLARE
    Image ORDSYS.ORDImage;
    SourceName VARCHAR2(4000);
BEGIN
    SELECT large_photo INTO Image FROM emp
      WHERE ename = 'John Doe';
    -- get the image source name
    SourceName := Image.getSourceName();
END;
```

getSourceType( )

Format

getSourceType( ) RETURN VARCHAR2;

Description

Returns a string containing the type of the external data source.

Parameters

None.

Usage Notes

Returns a VARCHAR2 string containing the type of the external data source, for example "file".

Pragmas

PRAGMA RESTRICT_REFERENCES(getSourceType, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Examples

Get the source type information about a media data source:

```
DECLARE
  obj ORDSYS.ORDDoc;
BEGIN
  SELECT doc INTO obj FROM tdoc  WHERE N=1 FOR UPDATE;
  DBMS_OUTPUT.PUT_LINE('setting and getting source');
  DBMS_OUTPUT.PUT_LINE('-------------------------');
  -- set source to a file
  obj.setSource('file','DOCDIR','testdoc.dat');
  -- get source information
  DBMS_OUTPUT.put_line(obj.getSource());
  DBMS_OUTPUT.put_line(obj.getSourceType());
  DBMS_OUTPUT.put_line(obj.getSourceLocation());
  DBMS_OUTPUT.put_line(obj.getSourceName());
```

```
  UPDATE TDOC SET doc=obj WHERE N=1;
  COMMIT;
END;
/
```

getUpdateTime( )

Format

getUpdateTime( ) RETURN DATE;

Description

Returns the time when the object was last updated.

Parameters

None.

Usage Notes

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getUpdateTime, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Examples

Get the updated time of some audio data:

```
DECLARE
 obj ORDSYS.ORDAudio;
BEGIN
 SELECT aud INTO obj FROM TAUD WHERE N = 1 ;
 DBMS_OUTPUT.PUT_LINE(TO_CHAR(obj.getUpdateTime(),'MM-DD-YYYY HH24:MI:SS'));
END;
/
```

isLocal( )

Format

isLocal( ) RETURN BOOLEAN;

Description

Returns TRUE if the data is stored locally in a BLOB or FALSE if the data is stored externally.

Parameters

None.

Usage Notes

If the local attribute is set to 1 or NULL, this method returns TRUE, otherwise this method returns FALSE.

Pragmas

PRAGMA RESTRICT_REFERENCES(getLocal, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Examples

Determine whether or not the audio data is local:

```
DECLARE
 obj ORDSYS.ORDAudio;
BEGIN
 SELECT aud INTO obj FROM TAUD WHERE N = 1 ;
 if(obj.isLocal() = TRUE)  then
    DBMS_OUTPUT.put_line('local is set true');
 else
    DBMS_OUTPUT.put_line('local is set false');
 end if;
END;
/
```

openSource( )

Format

openSource(userArg IN RAW, ctx OUT RAW) RETURN INTEGER;

Description

Opens a data source.

Parameters

userArg
The user argument. This may be used by user-defined source plug-ins.

ctx
The source plug-in context information.

Usage Notes

The return INTEGER is 0 (zero) for success and >0 (for example, 1) for failure. The exact number and the meaning for that number is plug-in defined. For example, for the file plug-in, 1 might mean "File not found," 2 might mean "No such directory," and so forth.

Pragmas

None.

Exceptions

ORDSourceExceptions.INCOMPLETE_SOURCE_INFORMATION

This exception is raised if you call the openSource( ) method and the value for srcType is NULL and data is not local.

ORDSourceExceptions.METHOD_NOT_SUPPORTED

This exception is raised if you call the openSource( ) method and this method is not supported by the source plug-in being used.

ORDSourceExceptions.SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the openSource( ) method within a source plug-in when any other exception is raised.

See Appendix H for more information about these exceptions.

Examples

Open an external data source:

```
DECLARE
  obj ORDSYS.ORDAudio;
  res INTEGER;
  ctx RAW(4000) :=NULL;
  userArg RAW(4000);
BEGIN
  SELECT aud INTO obj FROM taud WHERE N =1 FOR UPDATE;
  res := obj.openSource(userArg, ctx);
  UPDATE taud SET aud=obj WHERE N=1 ;
  COMMIT;
  EXCEPTION
   WHEN ORDSYS.ORDSourceExceptions.METHOD_NOT_SUPPORTED THEN
    DBMS_OUTPUT.put_line('ORDSourceExceptions.METHOD_NOT_SUPPORTED caught');
   WHEN OTHERS THEN
    DBMS_OUTPUT.put_line('EXCEPTION caught');
END;
/
```

processSourceCommand( )

Format

```
processSourceCommand(
                        ctx       IN OUT RAW,
                        cmd       IN VARCHAR2,
                        arguments IN VARCHAR2,
                        result    OUT RAW)
RETURN RAW;
```

Description

Allows you to send any command and its arguments to the source plug-in. This method is available only for user-defined source plug-ins.

Parameters

ctx
The source plug-in context information. This must be allocated. You must call the openSource( ) method; see Section 5.1 on page 5-2 for more information.

cmd
Any command recognized by the source plug-in.

arguments
The arguments of the command.

result
The result of calling this method returned by the source plug-in.

Usage Notes

Use this method to send any command and its respective arguments to the source plug-in. Commands are not interpreted; they are just taken and passed through to be processed.

Pragmas

None.

Exceptions

ORDSourceExceptions.INCOMPLETE_SOURCE_INFORMATION

This exception is raised if you call the processSourceCommand( ) method and the value of srcType is NULL.

ORDSourceExceptions.METHOD_NOT_SUPPORTED

This exception is raised if you call the processSourceCommand( ) method and this method is not supported by the source plug-in being used.

ORDSourceExceptions.SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the processSourceCommand( ) method within a source plug-in when any other exception is raised.

See Appendix H for more information about these exceptions.

Examples

Process some commands:

```
DECLARE
  obj ORDSYS.ORDVideo;
  res RAW(4000);
  result RAW(4000);
  command VARCHAR(4000);
  argList VARCHAR(4000);
  ctx RAW(4000) :=NULL;
BEGIN
select vid into obj from TVID where N =1 for UPDATE;
-- assign command
-- assign argList
res := obj.processSourceCommand(ctx, command, argList, result);
UPDATE TVID SET vid=obj WHERE N=1 ;
COMMIT;
  EXCEPTION
        WHEN ORDSYS.ORDSourceExceptions.METHOD_NOT_SUPPORTED THEN
         DBMS_OUTPUT.put_line('Source METHOD_NOT_SUPPORTED caught');
        WHEN ORDSYS.ORDSourceExceptions.SOURCE_PLUGIN_EXCEPTION THEN
         DBMS_OUTPUT.put_line('SOURCE PLUGIN EXCEPTION caught');
        WHEN ORDSYS.ORDSourceExceptions.INCOMPLETE_SOURCE_INFORMATION THEN
         DBMS_OUTPUT.put_line('SOURCE INCOMPLETE_SOURCE_INFORMATION EXCEPTION
```

```
caught');
        WHEN OTHERS THEN
         DBMS_OUTPUT.PUT_LINE('EXCEPTION caught');
END;
/
```

readFromSource( )

Format

```
readFromSource(
                ctx          IN OUT RAW,
                startPos   IN INTEGER,
                numBytes IN OUT INTEGER,
                buffer       OUT RAW);
```

Description

Allows you to read a buffer of *n* bytes from a source beginning at a start position.

Parameters

ctx
The source plug-in context information. This must be allocated. You must call the openSource( ) method; see Section 5.1 on page 5-2 for more information.

startPos
The start position in the data source.

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

buffer
The buffer into which the data will be read.

Usage Notes

This method is not supported for HTTP sources.

To successfully read HTTP source types, the entire URL source must be requested to be read. If you want to implement a read method for an HTTP source type, you must provide your own implementation for this method in the modified source plug-in for the HTTP source type.

Pragmas

None.

Exceptions

ORDSourceExceptions.INCOMPLETE_SOURCE_INFORMATION

This exception is raised if you call the readFromSource( ) method and the value of srcType is NULL and data is not local.

ORDSourceExceptions.NULL_SOURCE

This exception is raised if you call the readFromSource( ) method and the data is local but localData is NULL.

ORDSourceExceptions.METHOD_NOT_SUPPORTED

This exception is raised if you call the readFromSource( ) method and this method is not supported by the source plug-in being used.

ORDSourceExceptions.SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the readFromSource( ) method within a source plug-in when any other exception is raised.

See Appendix H for more information about these exceptions.

Examples

Read a buffer from the source:

```
DECLARE
  obj ORDSYS.ORDAudio;
  buffer RAW(4000);
  i INTEGER;
  ctx RAW(4000) :=NULL;
BEGIN
  i := 20;
  select aud into obj from TAUD where N =1 ;
  obj.readFromSource(ctx,1,i,buffer);
  DBMS_OUTPUT.PUT_LINE(TO_CHAR(obj.getContentLength(ctx)));
  EXCEPTION
   WHEN ORDSYS.ORDSourceExceptions.METHOD_NOT_SUPPORTED THEN
    DBMS_OUTPUT.put_line('ORDSourceExceptions.METHOD_NOT_SUPPORTED caught');
   WHEN OTHERS THEN
    DBMS_OUTPUT.put_line('EXCEPTION caught');
END;
```

setLocal( )

Format

setLocal( );

Description

Sets the local attribute to indicate that the data is stored internally in a BLOB. When local is set, methods look for corresponding data in the source.localData attribute.

Parameters

None.

Usage Notes

This method sets the local attribute to 1 meaning the data is stored locally in localData.

Pragmas

None.

Exceptions

NULL_LOCAL_DATA

This exception is raised if you call the setLocal method and the source.localData attribute value is NULL.

Examples

Set the flag to local for the data:

```
DECLARE
 obj ORDSYS.ORDAudio;
BEGIN
 SELECT s INTO obj FROM TAUD WHERE N = 1 FOR UPDATE;
 obj.setLocal();
 UPDATE TAUD SET s=obj WHERE N = 1;
 COMMIT;
END;
/
```

setMimeType( )

Format

setMimeType(mime IN VARCHAR2);

Description

Allows you to set the MIME type of the data.

Parameters

mime
The MIME type.

Usage Notes

You can override the automatic setting of MIME information by calling this method with a specified MIME value.

Calling this method implicitly calls the setUpdateTime( ) method.

The method setProperties( ) calls this method implicitly.

For image objects, the methods setProperties( ), process( ), and processCopy( ) call this method implicitly.

Pragmas

None.

Exceptions

INVALID_MIME_TYPE

This exception is raised if you call the setMimeType( ) method and the value for mimeType is NULL.

Examples

Set the MIME type for some stored data:

```
DECLARE
  obj ORDSYS.ORDAudio;
```

```
BEGIN
  SELECT aud INTO obj FROM TAUD  WHERE N=1 FOR UPDATE;
  DBMS_OUTPUT.PUT_LINE('writing mimetype');
  DBMS_OUTPUT.PUT_LINE('----------------');
  obj.setMimeType('audio/basic');
  DBMS_OUTPUT.PUT_LINE(obj.getMimeType);
  UPDATE TAUD SET aud=obj WHERE N=1;
  COMMIT;
END;
/
```

setSource( )

Format

```
setSource(
        source_type      IN VARCHAR2,
        source_location IN VARCHAR2,
        source_name      IN VARCHAR2);
```

Description

Sets or alters information about the external source of the data.

Parameters

source_type
The source type of the external data. See the ORDSource Object Type definition in Appendix I for more information.

source_location
The source location of the external data. See the ORDSource Object Type definition in Appendix I for more information.

source_name
The source name of the external data. See the ORDSource Object Type definition in Appendix I for more information.

Usage Notes

Users can use this method to set the data source to a new BFILE or URL.

You must ensure that the directory exists or is created before you use this method.

Calling this method implicitly calls the setUpdateTime( ) method and the clearLocal( ) method.

Pragmas

None.

Exceptions

ORDSourceExceptions.INCOMPLETE_SOURCE_INFORMATION

This exception is raised if you call the setSource( ) method and the value of srcType is NULL.

Examples

Set the source of the data:

```
DECLARE
  obj ORDSYS.ORDAudio;
BEGIN
  SELECT aud INTO obj FROM TAUD  WHERE N=1 FOR UPDATE;
  DBMS_OUTPUT.PUT_LINE('setting and getting source');
  DBMS_OUTPUT.PUT_LINE('-------------------------');
  obj.setSource('file','AUDIODIR','audio.au');
  DBMS_OUTPUT.PUT_LINE(obj.getSource());
  UPDATE TAUD SET aud=obj WHERE N=1;
  COMMIT;
END;
/
```

setUpdateTime( )

Format

setUpdateTime(current_time DATE);

Description

Sets the time when the data was last updated. Use this method whenever you modify the data. Methods that modify the object attributes and all set media access methods call this method implicitly. For example, the methods setMimeType( ), setSource( ), and deleteContent( ) call setUpdateTime( ) explicitly.

Parameters

current_time
The timestamp to be stored. Defaults to SYSDATE.

Usage Notes

You must invoke this method whenever you modify the data without using object methods.

Pragmas

None.

Exceptions

None.

Examples

Set the updated time of some data:

```
DECLARE
 obj ORDSYS.ORDAudio;
BEGIN
 SELECT aud INTO obj FROM TAUD WHERE N = 1;
 obj.setUpdateTime(SYSDATE);
 UPDATE TAUD SET aud=obj WHERE N = 1;
 COMMIT;
END;
```

trimSource( )

Format

```
trim(ctx       IN OUT RAW,
     newlen IN INTEGER) RETURN INTEGER;
```

Description

Trims a data source.

Parameters

ctx
The source plug-in context information. This must be allocated. You must call the openSource( ) method; see Section 5.1 on page 5-2 for more information.

newlen
The trimmed new length.

Usage Notes

The return INTEGER is 0 (zero) for success and >0 (for example, 1) for failure. The exact number and the meaning for that number is plug-in defined. For example, for the file plug-in, 1 might mean "File not found," 2 might mean "No such directory," and so forth.

Pragmas

None.

Exceptions

ORDSourceExceptions.INCOMPLETE_SOURCE_INFORMATION

This exception is raised if you call the trimSource( ) method and the value for srcType is NULL and data is not local.

ORDSourceExceptions.METHOD_NOT_SUPPORTED

This exception is raised if you call the trimSource( ) method and this method is not supported by the source plug-in being used.

ORDSourceExceptions.SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the trimSource( ) method within a source plug-in when any other exception is raised.

See Appendix H for more information about these exceptions.

Examples

Trim an external data source:

```
DECLARE
  obj ORDSYS.ORDAudio;
  res INTEGER;
  ctx RAW(4000) :=NULL;
BEGIN
  select aud into obj from TAUD where N =1 for UPDATE;
  res := obj.trimSource(ctx,0);
  UPDATE TAUD SET aud=obj WHERE N=1 ;
  COMMIT;
  EXCEPTION
   WHEN ORDSYS.ORDSourceExceptions.METHOD_NOT_SUPPORTED THEN
    DBMS_OUTPUT.put_line('ORDSourceExceptions.METHOD_NOT_SUPPORTED caught');
   WHEN OTHERS THEN
    DBMS_OUTPUT.put_line('EXCEPTION caught');
END;
/
```

writeToSource( )

Format

```
writeToSource(
                ctx          IN OUT RAW,
                startPos     IN INTEGER,
                numBytes IN OUT INTEGER,
                buffer       IN RAW);
```

Description

Allows you to write a buffer of *n* bytes to a source beginning at a start position.

Parameters

ctx
The source plug-in context information. This must be allocated. You must call the openSource( ) method; see Section 5.1 on page 5-2 for more information.

startPos
The start position in the source to where the buffer should be copied.

numBytes
The number of bytes to be written to the source.

buffer
The buffer of data to be written.

Usage Notes

This method assumes that the source allows you to write *n* number of bytes starting at a random byte location. The file and HTTP source types will not permit you to write, and do not support this method. This method will work if data is stored in a local BLOB or is accessible through a user-defined source plug-in.

Pragmas

None.

Exceptions

ORDSourceExceptions.INCOMPLETE_SOURCE_INFORMATION

This exception is raised if you call the writeToSource( ) method and the value of srcType is NULL and data is not local.

ORDSourceExceptions.NULL_SOURCE

This exception is raised if you call the writeToSource( ) method and the data is local but localData is NULL.

ORDSourceExceptions.METHOD_NOT_SUPPORTED

This exception is raised if you call the writeToSource( ) method and this method is not supported by the source plug-in being used.

ORDSourceExceptions.SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the writeToSource( ) method within a source plug-in when any other exception is raised.

See Appendix H for more information about these exceptions.

Examples

Write a buffer to the source:

```
DECLARE
  obj ORDSYS.ORDAudio;
  n INTEGER := 6;
  ctx RAW(4000) :=NULL;
BEGIN
  select aud into obj from TAUD where N =1 for update;
  obj.writeToSource(ctx,1,n,UTL_RAW.CAST_TO_RAW('helloP'));
  DBMS_OUTPUT.PUT_LINE(TO_CHAR(obj.getContentLength(ctx)));
  update  TAUD set aud = obj where  N =1 ;
  COMMIT;
  EXCEPTION
   WHEN ORDSYS.ORDSourceExceptions.METHOD_NOT_SUPPORTED THEN
    DBMS_OUTPUT.put_line('ORDSourceExceptions.METHOD_NOT_SUPPORTED caught');
   WHEN OTHERS THEN
    DBMS_OUTPUT.put_line('EXCEPTION caught');
END;
/
```

6

ORDAudio Reference Information

Oracle *inter*Media contains information about the ORDAudio type:

- Object type -- see Section 6.1.
- Constructors -- see Section 6.2.
- Methods -- see Section 6.3.
- Packages or PL/SQL plug-ins -- see Section 6.4.

The examples in this chapter assume that the test audio table TAUD has been created and filled with data. This table was created using the SQL statements described in Section 6.3.1.

> **Note:** If you manipulate the audio data itself (by either directly modifying the BLOB or changing the external source), then you must ensure that the object attributes stay synchronized and the update time is modified; otherwise, the object attributes will not match the audio data.

Methods invoked at the ORDSource level that are handed off to the source plug-in for processing have ctx(RAW(4000)) as the first argument. Before calling any of these methods for the first time, the client must allocate the ctx structure, initialize it to NULL, and invoke the openSource( ) method. At this point, the source plug-in can initialize context for this client. When processing is complete, the client should invoke the closeSource( ) method.

Methods invoked from a source plug-in call have the first argument as ctx (RAW(4000)).

Methods invoked at the ORDAudio level that are handed off to the format plug-in for processing have ctx (RAW(4000)) as the first argument. Before calling any of these methods for the first time, the client must allocate the ctx structure and initialize it to NULL.

Note: In the current release, not all source or format plug-ins will use the ctx argument, but if you code as previously described, your application should work with any current or future source or format plug-in.

You should use any of the individual set methods to set the value of the attribute for an object for formats not natively supported; otherwise, for formats natively supported, use the setProperties( ) method to populate the attributes of the object.

6.1 Object Types

Oracle *inter*Media describes the ORDAudio object type, which supports the storage and management of audio data.

ORDAudio Object Type

The ORDAudio object type supports the storage and management of audio data. This object type is defined as follows:

```
CREATE OR REPLACE TYPE ORDAudio
AS OBJECT
(
  -- ATTRIBUTES
description         VARCHAR2(4000),
source              ORDSource,
format              VARCHAR2(31),
mimeType            VARCHAR2(4000),
comments            CLOB,
  -- AUDIO RELATED ATTRIBUTES
encoding            VARCHAR2(256),
numberOfChannels    INTEGER,
samplingRate        INTEGER,
sampleSize          INTEGER,
compressionType     VARCHAR2(4000),
audioDuration       INTEGER,

  -- METHODS
-- CONSTRUCTORS
--
STATIC FUNCTION init( ) RETURN ORDAudio,
STATIC FUNCTION init(srcType     IN VARCHAR2,
                     srcLocation IN VARCHAR2,
                     srcName     IN VARCHAR2) RETURN ORDAudio,
-- Methods associated with the date attribute
MEMBER FUNCTION getUpdateTime( ) RETURN DATE,
PRAGMA RESTRICT_REFERENCES(getUpdateTime, WNDS, WNPS, RNDS, RNPS),
MEMBER PROCEDURE setUpdateTime(current_time DATE),
-- Methods associated with the description attribute
MEMBER PROCEDURE setDescription(user_description IN VARCHAR2),
MEMBER FUNCTION getDescription( ) RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getDescription, WNDS, WNPS, RNDS, RNPS),

-- Methods associated with the mimeType attribute
MEMBER PROCEDURE setMimeType(mime IN VARCHAR2),
MEMBER FUNCTION getMimeType( ) RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getMimeType, WNDS, WNPS, RNDS, RNPS),
```

```
-- Methods associated with the source attribute
MEMBER FUNCTION processSourceCommand(
                                    ctx       IN OUT RAW,
                                    cmd       IN VARCHAR2,
                                    arguments IN VARCHAR2,
                                    result    OUT RAW)
               RETURN RAW,

MEMBER FUNCTION  isLocal( ) RETURN BOOLEAN,
PRAGMA RESTRICT_REFERENCES(isLocal, WNDS, WNPS, RNDS, RNPS),

MEMBER PROCEDURE  setLocal( ),
MEMBER PROCEDURE  clearLocal( ),

MEMBER PROCEDURE setSource(
                          source_type     IN VARCHAR2,
                          source_location IN VARCHAR2,
                          source_name     IN VARCHAR2),
MEMBER FUNCTION getSource( ) RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getSource, WNDS, WNPS, RNDS, RNPS),

MEMBER FUNCTION getSourceType( ) RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getSourceType, WNDS, WNPS, RNDS, RNPS),

MEMBER FUNCTION getSourceLocation( ) RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getSourceLocation, WNDS, WNPS, RNDS, RNPS),

MEMBER FUNCTION getSourceName( ) RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getSourceName, WNDS, WNPS, RNDS, RNPS),

MEMBER PROCEDURE import(ctx IN OUT RAW),
MEMBER PROCEDURE importFrom(
                           ctx             IN OUT RAW,
                           source_type     IN VARCHAR2,
                           source_location IN VARCHAR2,
                           source_name     IN VARCHAR2),
MEMBER PROCEDURE export(
                       ctx             IN OUT RAW,
                       source_type     IN VARCHAR2,
                       source_location IN VARCHAR2,
                       source_name     IN VARCHAR2),
MEMBER FUNCTION getContentLength(ctx IN OUT RAW) RETURN INTEGER,
PRAGMA RESTRICT_REFERENCES(getContentLength, WNDS, WNPS, RNDS, RNPS),

MEMBER PROCEDURE getContentInLob(
```

```
                                ctx      IN OUT RAW,
                                dest_lob IN OUT NOCOPY BLOB,
                                mimeType OUT VARCHAR2,
                                format   OUT VARCHAR2),

MEMBER FUNCTION getContent( ) RETURN BLOB,
PRAGMA RESTRICT_REFERENCES(getContent, WNDS, WNPS, RNDS, RNPS),

MEMBER PROCEDURE deleteContent( ),

MEMBER FUNCTION getBFILE( ) RETURN BFILE,
PRAGMA RESTRICT_REFERENCES(getBFILE, WNDS, WNPS, RNDS, RNPS),

-- Methods associated with file operations on the source
MEMBER FUNCTION openSource(userArg IN RAW, ctx OUT RAW) RETURN INTEGER,
MEMBER FUNCTION closeSource(ctx IN OUT RAW) RETURN INTEGER,
MEMBER FUNCTION trimSource(ctx     IN OUT RAW,
                           newlen  IN INTEGER) RETURN INTEGER,
MEMBER PROCEDURE readFromSource(
                                ctx      IN OUT RAW,
                                startPos IN INTEGER,
                                numBytes IN OUT INTEGER,
                                buffer   OUT RAW),
MEMBER PROCEDURE writeToSource(
                               ctx      IN OUT RAW,
                               startPos IN INTEGER,
                               numBytes IN OUT INTEGER,
                               buffer   IN RAW),

-- Methods associated with audio attributes accessors
MEMBER PROCEDURE setFormat(knownformat IN VARCHAR2),
MEMBER FUNCTION getFormat( ) RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getFormat, WNDS, WNPS, RNDS, RNPS),

MEMBER PROCEDURE setEncoding(knownEncoding IN VARCHAR2),
MEMBER FUNCTION getEncoding( ) RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getEncoding, WNDS, WNPS, RNDS, RNPS),

MEMBER PROCEDURE setNumberOfChannels(knownNumberOfChannels IN INTEGER),
MEMBER FUNCTION getNumberOfChannels( ) RETURN INTEGER,
PRAGMA RESTRICT_REFERENCES(getNumberOfChannels, WNDS, WNPS, RNDS, RNPS),

MEMBER PROCEDURE setSamplingRate(knownSamplingRate IN INTEGER),
MEMBER FUNCTION getSamplingRate( ) RETURN INTEGER,
PRAGMA RESTRICT_REFERENCES(getSamplingRate, WNDS, WNPS, RNDS, RNPS),
```

```
MEMBER PROCEDURE setSampleSize(knownSampleSize IN INTEGER),
MEMBER FUNCTION getSampleSize( ) RETURN INTEGER,
PRAGMA RESTRICT_REFERENCES(getSampleSize, WNDS, WNPS, RNDS, RNPS),

MEMBER PROCEDURE setCompressionType(knownCompressionType IN VARCHAR2),
MEMBER FUNCTION getCompressionType( ) RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getCompressionType, WNDS, WNPS, RNDS, RNPS),

MEMBER PROCEDURE setAudioDuration(knownAudioDuration IN INTEGER),
MEMBER FUNCTION getAudioDuration( ) RETURN INTEGER,
PRAGMA RESTRICT_REFERENCES(getAudioDuration, WNDS, WNPS, RNDS, RNPS),

MEMBER PROCEDURE setKnownAttributes(
                                  knownFormat IN VARCHAR2,
                                  knownEncoding IN VARCHAR2,
                                  knownNumberOfChannels IN INTEGER,
                                  knownSamplingRate IN INTEGER,
                                  knownSampleSize IN INTEGER,
                                  knownCompressionType IN VARCHAR2,
                                  knownAudioDuration IN INTEGER),

-- Methods associated with setting all the properties
MEMBER PROCEDURE setProperties(ctx         IN OUT RAW,
                               setComments IN BOOLEAN),
MEMBER FUNCTION checkProperties(ctx IN OUT RAW) RETURN BOOLEAN,

MEMBER FUNCTION getAttribute(
                           ctx  IN OUT RAW,
                           name IN VARCHAR2) RETURN VARCHAR2,

MEMBER PROCEDURE getAllAttributes(
                                ctx        IN OUT RAW,
                                attributes IN OUT NOCOPY CLOB),

-- Methods associated with audio processing
MEMBER FUNCTION processAudioCommand(
                                  ctx       IN OUT RAW,
                                  cmd       IN VARCHAR2,
                                  arguments IN VARCHAR2,
                                  result    OUT RAW)
              RETURN RAW
);
```

where:

- description: the description of the audio object.
- source: the ORDSource where the audio data is to be found.
- format: the format in which the audio data is stored.
- mimeType: the MIME type information.
- comments: the comment information of the audio object.
- encoding: the encoding type of the audio data.
- numberOfChannels: the number of audio channels in the audio data.
- samplingRate: the rate in Hz at which the audio data was recorded.
- sampleSize: the sample width or number of samples of audio in the data.
- compressionType: the compression type of the audio data.
- audioDuration: the total duration of the audio data stored.

Note: The comments attribute is populated by setProperties( ) when the setComments parameter is TRUE and by the Oracle *inter*Media Annotator utility. Oracle Corporation recommends that you not write to this attribute directly.

6.2 Constructors

This section describes the constructor functions.

The *inter*Media constructor functions are as follows:

- init( )
- init(srcType,srcLocation,srcName)

init( )

Format

init( ) RETURN ORDAudio;

Description

Allows for easy initialization of instances of the ORDAudio object type.

Parameters

None.

Pragmas

None.

Exceptions

None.

Usage Notes

This static method initializes all the ORDAudio attributes to NULL with the following exceptions:

- source.updateTime is set to SYSDATE
- source.local is set to 1 (local)
- source.localData is set to empty_blob

You should begin using the init( ) method as soon as possible to allow you to more easily initialize the ORDAudio object type, especially if the ORDAudio type evolves and attributes are added in a future release. INSERT statements left unchanged using the default constructor (which initializes each object attribute), will fail under these circumstances.

Examples

Initialize the ORDAudio object attributes:

```
BEGIN
INSERT INTO taud VALUES (ORDSYS.ORDAudio.init());
```

```
END;
/
```

init(srcType,srcLocation,srcName)

Format

```
init(srcType     IN VARCHAR2,
   srcLocation IN VARCHAR2,
   srcName     IN VARCHAR2)
 RETURN ORDAudio;
```

Description

Allows for easy initialization of instances of the ORDAudio object type.

Parameters

srcType
The source type of the audio data.

srcLocation
The source location of the audio data.

srcName
The source name of the audio data.

Pragmas

None.

Exceptions

None.

Usage Notes

This static method initializes all the ORDAudio attributes to NULL with the following exceptions:

- source.updateTime is set to SYSDATE
- source.local is set to 0
- source.localData is set to empty_blob

- source.srcType is set to the input value
- source.srcLocation is set to the input value
- source.srcName is set to the input value

You should begin using the init( ) method as soon as possible to allow you to more easily initialize the ORDAudio object type, especially if the ORDAudio type evolves and attributes are added in a future release. INSERT statements left unchanged using the default constructor (which initializes each object attribute), will fail under these circumstances.

Examples

Initialize the ORDAudio object attributes:

```
BEGIN
INSERT INTO taud VALUES (ORDSYS.ORDAudio.init('file','AUDDIR','audio1.au'));
END;
/
```

6.3 Methods

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

ORDAudio Methods Associated with the updateTime Attribute

- getUpdateTime( ): returns the time when the audio object was last updated. See "getUpdateTime( )" on page 5-25 for information.
- setUpdateTime( ): sets the update time for the audio object. This method is called implicitly by methods that modify natively supported audio formats. See "setUpdateTime( )" on page 5-39 for information.

ORDAudio Methods Associated with the description Attribute

- setDescription( ): sets the description of the audio data. See "setDescription( )" on page 6-44.
- getDescription( ): returns the description of the audio data. See "getDescription( )" on page 6-28.

ORDAudio Methods Associated with mimeType Attribute

- setMimeType( ): sets the MIME type of the stored audio data. This method is called implicitly by any method that modifies natively supported audio formats. See "setMimeType( )" on page 5-35 for information.
- getMimeType( ): returns the MIME type of the stored audio data. See "getMimeType( )" on page 5-17 for information.

ORDAudio Methods Associated with the source Attribute

- processSourceCommand( ): sends a command and related arguments to the source plug-in. See "processSourceCommand( )" on page 5-29 for information.
- isLocal( ): returns TRUE if the data is stored locally in a BLOB or FALSE if the data is external. See "isLocal( )" on page 5-26 for information.
- setLocal( ): sets a flag to indicate that the data is stored locally in a BLOB. See "setLocal( )" on page 5-34 for information.
- clearLocal( ): clears the flag to indicate that the data is stored externally. See "clearLocal( )" on page 5-5 for information.

- setSource( ): sets the source information to where audio data is found. See "setSource( )" on page 5-37 for information.
- getSource( ): returns a formatted string containing complete information about the external data source formatted as a URL. See "getSource( )" on page 5-19 for information.
- getSourceType( ): returns the external source type of the audio data. See "getSourceType( )" on page 5-23 for information.
- getSourceLocation( ): returns the external source location of the audio data. See "getSourceLocation( )" on page 5-21 for information.
- getSourceName( ): returns the external source name of the audio data. See "getSourceName( )" on page 5-22 for information.
- import( ): transfers data from an external data source (specified by calling setSourceInformation( )) to the local source (localData) within an Oracle database, setting the value of the local attribute to "1", meaning local and updating the timestamp. See "import( )" on page 6-36.
- importFrom( ): transfers data from the specified external data source (source type, location, name) to the local source (localData) within an Oracle database, setting the value of the local attribute to "1", meaning local and updating the timestamp. See "importFrom( )" on page 6-36.
- export( ): copies data from a local source (localData) within an Oracle database to the specified external data source, and stores source information in the source. See "export( )" on page 5-9 for information.

  Note: The export( ) method natively supports only sources of source type file. User-defined sources may support the export( ) method.

- getContentLength( ): returns the length of the data source (as number of bytes). See "getContentLength( )" on page 6-24.
- getContentInLob( ): returns content into a temporary LOB. See "getContentInLob( )" on page 6-26 for information.
- getContent( ): returns the handle to the BLOB used to store contents locally. See "getContent( )" on page 5-15 for information.
- deleteContent( ): deletes the content of the local BLOB. See "deleteContent( )" on page 5-8 for information.

- getBFILE( ): returns the external content as a BFILE. See "getBFILE( )" on page 5-13 for information.

ORDAudio Methods Associated with File Operations

- openSource( ): opens a data source or a BLOB. See "openSource( )" on page 5-27 for information.
- closeSource( ): closes a data source or a BLOB. See "closeSource( )" on page 5-6 for information.
- trimSource( ): trims a data source or a BLOB. See "trimSource( )" on page 5-40 for information.
- readFromSource( ): reads a buffer of *n* bytes from a source beginning at a start position. See "readFromSource( )" on page 5-32 for information.
- writeToSource( ): writes a buffer of *n* bytes to a source beginning at a start position. See "writeToSource( )" on page 5-42 for information.

ORDAudio Methods Associated with Audio Attributes Accessors

- setFormat( ): sets the object attribute value of the format of the audio data. See "setFormat( )" on page 6-47 for information.
- getFormat( ): returns the object attribute value of the format in which the audio data is stored. See "getFormat( )" on page 6-30.
- setEncoding( ): sets the object attribute value of the encoding type of the audio data. See "setEncoding( )" on page 6-46.
- getEncoding( ): returns the object attribute value of the encoding type of the audio data. See "getEncoding( )" on page 6-29.
- setNumberOfChannels( ): sets the object attribute value of the number of audio channels of the audio data. See "setNumberOfChannels( )" on page 6-51.
- getNumberOfChannels( ): returns the object attribute value of the number of audio channels in the audio data. See "getNumberOfChannels( )" on page 6-31.
- setSamplingRate( ): sets the object attribute value of the sampling rate of the audio data. See "setSamplingRate( )" on page 6-54.
- getSamplingRate( ): returns the object attribute value of the sampling rate in samples per second at which the audio data was recorded. See "getSamplingRate( )" on page 6-33.

- setSampleSize( ): sets the object attribute value of the sample width or number of samples of audio in the data. See "setSampleSize( )" on page 6-55.
- getSampleSize( ): returns the object attribute value of the sample width or number of samples of audio in the data. See "getSampleSize( )" on page 6-32.
- setCompressionType( ): sets the object attribute value of the compression type of the audio data. See "setCompressionType( )" on page 6-43.
- getCompressionType( ): returns the object attribute value of the compression type of the audio data. See "getCompressionType( )" on page 6-25.
- setAudioDuration( ): sets the object attribute value of the total time value for the time required to play the audio data. See "setAudioDuration( )" on page 6-42.
- getAudioDuration( ): returns the object attribute value of the total time required to play the audio data. See "getAudioDuration( )" on page 6-23.
- setKnownAttributes( ): sets known audio attributes including format, encoding type, number of channels, sampling rate, sample size, compression type, and audio duration of the audio data. The parameters are passed in with this call. See "setKnownAttributes( )" on page 6-49.
- setProperties( ): reads the audio data to get the values of the object attributes and then stores them in the object. If the value for the setComments parameter is TRUE, then the comments field of the object will be populated with a rich set of format and application properties of the audio object in XML form, identical to what is provided by the *inter*Media Annotator utility. See "setProperties( )" on page 6-52.

  For the known attributes that ORDAudio understands, it sets the properties for these attributes. These include: format, encoding type, data type, number of channels, sampling rate, and sample size of the audio data. See "setMimeType( )" on page 5-35 for information.
- checkProperties( ): calls the format plug-in to check the properties including format, encoding type, number of channels, sampling rate, and sample size of the audio data, and returns a Boolean value TRUE if the properties stored in object attributes match those in the audio data. See "checkProperties( )" on page 6-17.
- getAttribute( ): returns the value of the requested attribute. This method is only available for user-defined format plug-ins. See "getAttribute( )" on page 6-21.
- getAllAttributes( ): returns a formatted string for convenient client access. For natively supported formats, the string includes the following list of audio data

attributes separated by a comma (,): FileFormat, Encoding, NumberOfChannels, SamplingRate, and SampleSize. Different format plug-ins can return data in any format in this call. For user-defined formats, the string is defined by the format plug-in. See "getAllAttributes( )" on page 6-19.

ORDAudio Methods Associated with Processing Audio Data

- processAudioCommand( ): sends commands and related arguments to the format plug-in for processing. This method is available only for user-defined format plug-ins. See "processAudioCommand( )" on page 6-39.

For more information on object types and methods, see *Oracle9i Database Concepts*.

6.3.1 Example Table Definitions

The methods described in this reference chapter show examples based on a test audio table TAUD. Refer to the TAUD table definition that follows when reading through the examples:

TAUD Table Definition

```
CREATE TABLE TAUD(n NUMBER, aud ORDSYS.ORDAUDIO)
storage (initial 100K next 100K pctincrease 0);

INSERT INTO TAUD VALUES(1, ORDSYS.ORDAudio.init());
INSERT INTO TAUD VALUES(2, ORDSYS.ORDAudio.init());
```

checkProperties( )

Format

checkProperties(ctx IN OUT RAW) RETURN BOOLEAN;

Description

Checks the properties of the stored audio data, including the following audio attributes: sample size, sample rate, number of channels, format, and encoding type.

Parameters

ctx
The format plug-in context information.

Usage Notes

If the format is set to NULL, then the checkProperties( ) method uses the default format plug-in; otherwise, it uses the plug-in specified by the format.

The checkProperties( ) method does not check the MIME type because a file can have multiple correct MIME types and this is not well defined.

Pragmas

None.

Exceptions

AUDIO_PLUGIN_EXCEPTION

This exception is raised if you call the checkProperties( ) method and the audio plug-in raises an exception.

Examples

Check property information for known audio attributes:

```
DECLARE
  obj ORDSYS.ORDAudio;
  ctx RAW(4000) :=NULL;
BEGIN
  select aud into obj from TAUD where N =1 for update;
```

```
  if( obj.checkProperties(ctx) =  TRUE ) then
  DBMS_OUTPUT.put_line('true');
  else
  DBMS_OUTPUT.put_line('false');
  end if;
  EXCEPTION
   WHEN ORDSYS.ORDAudioExceptions.AUDIO_PLUGIN_EXCEPTION THEN
    DBMS_OUTPUT.put_line('ORDAudioExceptions.AUDIO_PLUGIN_EXCEPTION caught');
   WHEN OTHERS THEN
    DBMS_OUTPUT.put_line('EXCEPTION caught');
END;
/
```

getAllAttributes( )

Format

```
getAllAttributes(
        ctx         IN OUT RAW,
        attributes IN OUT NOCOPY CLOB);
```

Description

Returns a formatted string for convenient client access. For natively supported formats, the string includes the following list of audio data attributes separated by a comma (,): fileFormat, mimeType, encoding, numberOfChannels, samplingRate, sampleSize, compressionType, and audioDuration. For user-defined formats, the string is defined by the format plug-in.

Parameters

ctx
The format plug-in context information.

attributes
The attributes.

Usage Notes

These audio data attributes are available from the header of the formatted audio data.

Pragmas

None.

Exceptions

AUDIO_PLUGIN_EXCEPTION

This exception is raised if you call the getAllAttributes( ) method and the audio plug-in raises an exception.

Examples

Return all audio attributes for audio data stored in the database:

```
DECLARE
  obj ORDSYS.ORDAudio;
  tempLob   CLOB;
  ctx RAW(4000) :=NULL;
BEGIN
  SELECT aud INTO obj FROM TAUD  WHERE N=1;
  DBMS_OUTPUT.PUT_LINE('getting comma separated list of all attribs');
  DBMS_OUTPUT.PUT_LINE('-----------------------------------------');
  DBMS_LOB.CREATETEMPORARY(tempLob, FALSE, DBMS_LOB.CALL);
  obj.getAllAttributes(ctx,tempLob);
  DBMS_OUTPUT.put_line(DBMS_LOB.substr(tempLob, DBMS_LOB.getLength(tempLob) , 1));
  EXCEPTION
  WHEN ORDSYS.ORDAudioExceptions.AUDIO_PLUGIN_EXCEPTION THEN
  DBMS_OUTPUT.PUT_LINE('ORDAudioExceptions.AUDIO_PLUGIN_EXCEPTION caught');
END;
/
```

getAttribute( )

Format

```
getAttribute(
        ctx      IN OUT RAW,
        name IN VARCHAR2)
RETURN VARCHAR2;
```

Description

Returns the value of the requested attribute from audio data for user-defined formats only.

Parameters

ctx
The format plug-in context information.

name
The name of the attribute.

Usage Notes

The audio data attributes are available from the header of the formatted audio data.

Audio data attribute information can be extracted from the audio data itself. You can extend support to a format not understood by the ORDAudio object by implementing an ORDPLUGINS.ORDX_<format>_AUDIO package that supports that format. See Section 3.4.13 for more information.

Pragmas

None.

Exceptions

AUDIO_PLUGIN_EXCEPTION

This exception is raised if you call the getAttribute( ) method and the audio plug-in raises an exception.

Examples

Return information for the specified audio attribute for audio data stored in the database:

```
DECLARE
  obj ORDSYS.ORDAudio;
  res VARCHAR2(4000);
  ctx RAW(4000) :=NULL;
BEGIN
  SELECT aud INTO obj FROM TAUD  WHERE N=1;
  DBMS_OUTPUT.PUT_LINE('getting audio sample size');
  DBMS_OUTPUT.PUT_LINE('---------------------');
  res := obj.getAttribute(ctx,'sample_size');
  EXCEPTION
        WHEN ORDSYS.ORDSourceExceptions.METHOD_NOT_SUPPORTED THEN
         DBMS_OUTPUT.put_line('Source METHOD_NOT_SUPPORTED caught');
        WHEN ORDSYS.ORDSourceExceptions.SOURCE_PLUGIN_EXCEPTION THEN
         DBMS_OUTPUT.put_line('SOURCE PLUGIN EXCEPTION caught');
        WHEN ORDSYS.ORDAudioExceptions.METHOD_NOT_SUPPORTED THEN
         DBMS_OUTPUT.put_line('AUDIO METHOD_NOT_SUPPORTED EXCEPTION caught');
        WHEN ORDSYS.ORDAudioExceptions.AUDIO_PLUGIN_EXCEPTION THEN
         DBMS_OUTPUT.put_line('AUDIO PLUGIN EXCEPTION caught');
        WHEN OTHERS THEN
         DBMS_OUTPUT.PUT_LINE('EXCEPTION caught');
END;
/
```

getAudioDuration( )

Format

getAudioDuration( ) RETURN INTEGER;

Description

Returns the value of the audioDuration attribute of the audio object.

Parameters

None.

Usage Notes

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getAudioDuration, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Examples

See the example in setProperties( ) on page 6-52.

getContentLength( )

Format

getContentLength(ctx IN OUT RAW) RETURN INTEGER;

Description

Returns the length of the audio data content stored in the source.

Parameters

ctx
The source plug-in context information.

Usage Notes

This method is not supported for all source types. For example, HTTP type sources do not support this method. If you want to implement this call for HTTP type sources, you must define your own modified HTTP source type and implement this method on it.

Pragmas

PRAGMA RESTRICT_REFERENCES(getContentLength, WNDS, WNPS, RNDS, RNPS)

Exceptions

ORDSourceExceptions.INCOMPLETE_SOURCE_INFORMATION

This exception is raised if you call the getContentLength( ) method and the value of srcType is NULL and data is not stored locally in the BLOB.

ORDSourceExceptions.SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the getContentLength( ) method within a source plug-in when any other exception is raised.

See Appendix H for more information about these exceptions.

Examples

See the example in import( ) on page 6-35.

getCompressionType( )

Format

getCompressionType( ) RETURN VARCHAR2;

Description

Returns the value of the compressionType attribute of the audio object.

Parameters

None.

Usage Notes

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getCompressionType, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Examples

See the example in setProperties( ) on page 6-52.

getContentInLob( )

Format

```
getContentInLob(
                ctx          IN OUT RAW,
                dest_lob     IN OUT NOCOPY BLOB,
                mimeType     OUT VARCHAR2,
                format       OUT VARCHAR2);
```

Description

Copies data from a data source into the specified BLOB. The BLOB must not be the BLOB in source.localData.

Parameters

ctx
The source plug-in context information.

dest_lob
The LOB in which to receive data.

mimeType
The MIME type of the data; this may or may not be returned.

format
The format of the data; this may or may not be returned.

Usage Notes

None.

Pragmas

None.

Exceptions

ORDSourceExceptions.INCOMPLETE_SOURCE_INFORMATION

This exception is raised if you call the getContentInLob( ) method and the value of srcType is NULL.

ORDSourceExceptions.METHOD_NOT_SUPPORTED

This exception is raised if you call the getContentInLob( ) method and this method is not supported by the source plug-in being used.

ORDSourceExceptions.SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the getContentInLob( ) method within a source plug-in when any other exception is raised.

See Appendix H for more information about these exceptions.

Examples

Get data from a data source and put it into the specified BLOB:

```
DECLARE
 obj ORDSYS.ORDAudio;
 tempBLob BLOB;
 mimeType VARCHAR2(4000);
 format VARCHAR2(4000);
 ctx RAW(4000) :=NULL;
BEGIN
 SELECT aud INTO obj FROM TAUD WHERE N = 1 ;
 if(obj.isLocal) then
   DBMS_OUTPUT.put_line('local is true');
 end if;
 DBMS_LOB.CREATETEMPORARY(tempBLob, true, 10);
 obj.getContentInLob(ctx,tempBLob, mimeType,format);
 DBMS_OUTPUT.PUT_LINE(TO_CHAR(DBMS_LOB.getLength(tempBLob)));
 EXCEPTION
  WHEN ORDSYS.ORDSourceExceptions.METHOD_NOT_SUPPORTED THEN
   DBMS_OUTPUT.put_line('ORDSourceExceptions.METHOD_NOT_SUPPORTED caught');
  WHEN OTHERS THEN
   DBMS_OUTPUT.put_line('EXCEPTION caught');
END;
/
```

getDescription( )

Format

getDescription( ) RETURN VARCHAR2;

Description

Returns the description of the audio data.

Parameters

None.

Usage Notes

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getDescription, WNDS, WNPS, RNDS, RNPS)

Exceptions

DESCRIPTION_IS_NOT_SET

This exception is raised if you call the getDescription( ) method and the description is not set.

Examples

See the example in setDescription( ) on page 6-44.

getEncoding( )

Format

getEncoding( ) RETURN VARCHAR2;

Description

Returns the value of the encoding attribute of the audio object.

Parameters

None.

Usage Notes

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getEncoding, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Examples

See the example in setProperties( ) on page 6-52.

getFormat( )

Format

getFormat( ) RETURN VARCHAR2;

Description

Returns the value of the format attribute of the audio object.

Parameters

None.

Usage Notes

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getFormat, WNDS, WNPS, RNDS, RNPS)

Exceptions

AUDIO_FORMAT_IS_NULL

This exception is raised if you call the getFormat( ) method and the value for format is NULL.

Examples

See the example in setProperties( ) on page 6-52.

getNumberOfChannels( )

Format

getNumberOfChannels( ) RETURN INTEGER;

Description

Returns the value of the numberOfChannels attribute of the audio object.

Parameters

None.

Usage Notes

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getNumberOfChannels, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Examples

See the example in setProperties( ) on page 6-52.

getSampleSize( )

Format

getSampleSize( ) RETURN INTEGER;

Description

Returns the value of the sampleSize attribute of the audio object.

Parameters

None.

Usage Notes

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getSampleSize, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Examples

See the example in setProperties( ) on page 6-52.

getSamplingRate( )

Format

getSamplingRate( ) IN INTEGER;

Description

Returns the value of the samplingRate attribute of the audio object. The unit is Hz.

Parameters

None.

Usage Notes

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getSamplingRate, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Examples

See the example in setProperties( ) on page 6-52.

import( )

Format

```
import(ctx IN OUT RAW);
```

Description

Transfers audio data from an external audio data source to a local source (localData) within an Oracle database.

Parameters

ctx
The source plug-in context information.This must be allocated. You must call the openSource( ) method; see the introduction to this chapter for more information.

Usage Notes

Use the setSource( ) method to set the external source type, location, and name prior to calling the import( ) method.

You must ensure that the directory exists or is created before you use this method for srcType 'file'.

After importing data from an external audio data source to a local source (within an Oracle database), the source information remains unchanged (that is, pointing to the source from where the data was imported).

Invoking this method implicitly calls the setUpdateTime( ) and setLocal methods.

Pragmas

None.

Exceptions

ORDSourceExceptions.INCOMPLETE_SOURCE_INFORMATION

This exception is raised if you call the import( ) method and the value of srcType is NULL.

ORDSourceExceptions.NULL_SOURCE

This exception is raised if you call the import( ) method and the value of dlob is NULL.

ORDSourceExceptionsMETHOD_NOT_SUPPORTED

This exception is raised if you call the import( ) method and the import( ) method is not supported by the source plug-in being used.

ORDSourceExceptions.SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the import( ) method within a source plug-in when any other exception is raised.

See Appendix H for more information about these exceptions.

Examples

Import audio data from an external audio data source into the local source:

```
DECLARE
  obj ORDSYS.ORDAudio;
  ctx RAW(4000) :=NULL;
BEGIN
  SELECT aud INTO obj FROM TAUD  WHERE N=1 FOR UPDATE;
  DBMS_OUTPUT.PUT_LINE('setting and getting source');
  DBMS_OUTPUT.PUT_LINE('--------------------------');
  -- set source to a file
  obj.setSource('file','AUDIODIR','testaud.dat');
  -- get source information
  DBMS_OUTPUT.PUT_LINE(obj.getSource());
  -- import data
  obj.import(ctx);
  -- check size
  DBMS_OUTPUT.PUT_LINE(TO_CHAR(obj.getContentLength(ctx)));
  DBMS_OUTPUT.PUT_LINE(obj.getSource());
  DBMS_OUTPUT.PUT_LINE('deleting contents');
  DBMS_OUTPUT.PUT_LINE('-----------------');
  obj.deleteContent();
  DBMS_OUTPUT.PUT_LINE(TO_CHAR(obj.getContentLength(ctx)));
  UPDATE TAUD SET aud=obj WHERE N=1;
  COMMIT;
END;
/
```

importFrom( )

Format

```
importFrom(ctx                IN OUT RAW,
           source_type        IN VARCHAR2,
           source_location    IN VARCHAR2,
           source_name        IN VARCHAR2);
```

Description

Transfers audio data from the specified external audio data source to a local source (localData) within an Oracle database.

Parameters

ctx
The source plug-in context information. This must be allocated. You must call the openSource( ) method; see the introduction to this chapter for more information.

source_type
The source type of the audio data.

source_location
The location from where the audio data is to be imported.

source_name
The name of the audio data.

Usage Notes

This method is similar to the import( ) method except the source information is specified as parameters to the method instead of separately.

You must ensure that the directory exists or is created before you use this method for srcType 'file'.

After importing data from an external audio data source to a local source (within an Oracle database), the source information (that is, pointing to the source from where the data was imported) is set to the input values.

Invoking this method implicitly calls the setUpdateTime( ) and setLocal methods.

Pragmas

None.

Exceptions

ORDSourceExceptions.NULL_SOURCE

This exception is raised if you call the importFrom( ) method and the value of dlob is NULL.

ORDSourceExceptionsMETHOD_NOT_SUPPORTED

This exception is raised if you call the importFrom( ) method and this method is not supported by the source plug-in being used.

ORDSourceExceptions.SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the importFrom( ) method within a source plug-in when any other exception is raised.

See Appendix H for more information about these exceptions.

Examples

Import audio data from the specified external data source into the local source:

```
DECLARE
  obj ORDSYS.ORDAudio;
  ctx RAW(4000) :=NULL;
BEGIN
  SELECT aud INTO obj FROM TAUD  WHERE N=1 FOR UPDATE;
  DBMS_OUTPUT.PUT_LINE('setting and getting source');
  DBMS_OUTPUT.PUT_LINE('-------------------------');
  -- set source to a file
  -- import data
  obj.importFrom(ctx,'file','AUDIODIR','testaud.dat');
  -- check size
  DBMS_OUTPUT.PUT_LINE(TO_CHAR(obj.getContentLength(ctx)));
  DBMS_OUTPUT.PUT_LINE(obj.getSource());
  DBMS_OUTPUT.PUT_LINE('deleting contents');
  DBMS_OUTPUT.PUT_LINE('----------------');
  obj.deleteContent();
  DBMS_OUTPUT.PUT_LINE(TO_CHAR(obj.getContentLength(ctx)));
  DBMS_OUTPUT.PUT_LINE(TO_CHAR(DBMS_LOB.GETLENGTH(obj.getContent())));
```

```
  UPDATE TAUD SET aud=obj WHERE N=1;
  COMMIT;
  EXCEPTION
        WHEN ORDSYS.ORDAudioExceptions.METHOD_NOT_SUPPORTED THEN
        DBMS_OUTPUT.PUT_LINE('Source not specified');
END;
/
```

processAudioCommand( )

Format

```
processAudioCommand(
                    ctx        IN OUT RAW,
                    cmd        IN VARCHAR2,
                    arguments IN VARCHAR2,
                    result     OUT RAW)
RETURN RAW;
```

Description

Allows you to send a command and related arguments to the format plug-in for processing.

> **Note:** This method is supported only for user-defined format plug-ins.

Parameters

ctx
The format plug-in context information.

cmd
Any command recognized by the format plug-in.

arguments
The arguments of the command.

result
The result of calling this function returned by the format plug-in.

Usage Notes

Use this method to send any audio commands and their respective arguments to the format plug-in. Commands are not interpreted; they are taken and passed through to a format plug-in to be processed.

If the format is set to NULL, then the processAudioCommand( ) method uses the default format plug-in; otherwise, it uses your user-defined format plug-in.

You can extend support to a format that is not understood by the ORDAudio object by preparing an ORDPLUGINS.ORDX_<format>_AUDIO package that supports that format. See Section 3.4.13 for more information.

Pragmas

None.

Exceptions

AUDIO_PLUGIN_EXCEPTION

This exception is raised if you call the processAudioCommand( ) method and the audio plug-in raises an exception.

Examples

Process a set of commands:

```
DECLARE
  obj ORDSYS.ORDAudio;
  res RAW(4000);
  result RAW(4000);
  command VARCHAR(4000);
  argList VARCHAR(4000);
  ctx RAW(4000) :=NULL;
BEGIN
  select aud into obj from TAUD where N =1 for UPDATE;
  -- assign command
  -- assign argList
  res := obj.processAudioCommand (ctx, command, argList, result);
  UPDATE TAUD SET aud=obj WHERE N=1 ;
  COMMIT;
  EXCEPTION
        WHEN ORDSYS.ORDSourceExceptions.METHOD_NOT_SUPPORTED THEN
         DBMS_OUTPUT.put_line('Source METHOD_NOT_SUPPORTED caught');
        WHEN ORDSYS.ORDSourceExceptions.SOURCE_PLUGIN_EXCEPTION THEN
         DBMS_OUTPUT.put_line('SOURCE PLUGIN EXCEPTION caught');
        WHEN ORDSYS.ORDAudioExceptions.METHOD_NOT_SUPPORTED THEN
         DBMS_OUTPUT.put_line('AUDIO METHOD_NOT_SUPPORTED EXCEPTION caught');
        WHEN ORDSYS.ORDAudioExceptions.AUDIO_PLUGIN_EXCEPTION THEN
         DBMS_OUTPUT.put_line('AUDIO PLUGIN EXCEPTION caught');
        WHEN OTHERS THEN
```

```
        DBMS_OUTPUT.put_line('EXCEPTION caught');
END;
/
```

setAudioDuration( )

Format

setAudioDuration(knownAudioDuration IN INTEGER);

Description

Sets the value of the audioDuration attribute of the audio object.

Parameters

knownAudioDuration
A known audio duration.

Usage Notes

Calling this method implicitly calls the setUpdateTime( ) method.

Pragmas

None.

Exceptions

NULL_INPUT_VALUE

This exception is raised if you call the setAudioDuration( ) method and the value for the knownAudioDuration parameter is NULL.

Examples

See the example in setProperties( ) on page 6-52.

setCompressionType( )

Format

setCompressionType(knownCompressionType IN VARCHAR2);

Description

Sets the value of the compressionType attribute of the audio object.

Parameters

knownCompressionType
A known compression type.

Usage Notes

The value of the compressionType always matches that of the encoding value because in many audio formats, encoding and compression type are tightly integrated. See Appendix A for more information.

Calling this method implicitly calls the setUpdateTime( ) method.

Pragmas

None.

Exceptions

NULL_INPUT_VALUE

This exception is raised if you call the setCompressionType( ) method and the value for the knownCompressionType parameter is NULL.

Examples

See the example in setProperties( ) on page 6-52.

setDescription( )

Format

setDescription (user_description IN VARCHAR2);

Description

Sets the description of the audio data.

Parameters

user_description
The description of the audio data.

Usage Notes

Each audio object may need a description to help some client applications. For example, a Web-based client can show a list of audio descriptions from which a user can select one to access the audio data.

Web-access components and other client components provided with Oracle *inter*Media make use of this description attribute to present audio data to users.

Calling this method implicitly calls the setUpdateTime( ) method.

Pragmas

None.

Exceptions

None.

Examples

Set the description attribute for some audio data:

```
DECLARE
  obj ORDSYS.ORDAudio;
BEGIN
  SELECT aud INTO obj FROM TAUD  WHERE N=1 FOR UPDATE;
  DBMS_OUTPUT.PUT_LINE('writing title');
  DBMS_OUTPUT.PUT_LINE('-------------');
```

```
  obj.setDescription('audio1.wav');
  DBMS_OUTPUT.PUT_LINE(obj.getDescription());
  UPDATE TAUD SET aud=obj WHERE N=1;
  COMMIT;
END;
/
```

setEncoding( )

Format

setEncoding(knownEncoding IN VARCHAR2);

Description

Sets the value of the encoding attribute of the audio object.

Parameters

knownEncoding
A known encoding type.

Usage Notes

The value of encoding always matches that of the compressionType value because in many audio formats, encoding and compression type are tightly integrated. See Appendix A for more information.

Calling this method implicitly calls the setUpdateTime( ) method.

Pragmas

None.

Exceptions

NULL_INPUT_VALUE

This exception is raised if you call the setEncoding( ) method and the value for the knownEncoding parameter is NULL.

Examples

See the example in setProperties( ) on page 6-52.

setFormat( )

Format

setFormat(knownFormat IN VARCHAR2);

Description

Sets the format attribute of the audio object.

Parameters

knownFormat
The known format of the audio data to be set in the audio object.

Usage Notes

Calling this method implicitly calls the setUpdateTime( ) method.

Pragmas

None.

Exceptions

NULL_INPUT_VALUE

This exception is raised if you call the setFormat( ) method and the value for the knownFormat parameter is NULL.

Examples

Set the format for some audio data:

```
DECLARE
  obj ORDSYS.ORDAudio;
BEGIN
  select aud into obj from TAUD where N =1 for update;
  obj.setFormat('AUFF');
  obj.setEncoding('MULAW');
  obj.setNumberOfChannels(1);
  obj.setSamplingRate(8);
  obj.setSampleSize(8);
  obj.setCompressionType('8BITMONOAUDIO');
  obj.setAudioDuration(16);
```

```
  DBMS_OUTPUT.put_line('format: ' || obj.getformat);
  DBMS_OUTPUT.put_line('encoding: ' || obj.getEncoding);
  DBMS_OUTPUT.put_line('numberOfChannels: ' || TO_CHAR(obj.getNumberOfChannels));
  DBMS_OUTPUT.put_line('samplingRate: ' || TO_CHAR(obj.getSamplingRate));
  DBMS_OUTPUT.put_line('sampleSize: ' || TO_CHAR(obj.getSampleSize));
  DBMS_OUTPUT.put_line('compressionType : ' || obj.getCompressionType);
  DBMS_OUTPUT.put_line('audioDuration: ' || TO_CHAR(obj.getAudioDuration));
  COMMIT;
  EXCEPTION
   WHEN ORDSYS.ORDAudioExceptions.NULL_INPUT_VALUE THEN
    DBMS_OUTPUT.put_line('ORDAudioExceptions.NULL_INPUT_VALUE caught');
   WHEN OTHERS THEN
    DBMS_OUTPUT.put_line('EXCEPTION caught');
END;
/
```

setKnownAttributes( )

Format

```
setKnownAttributes(
                    knownFormat              IN VARCHAR2,
                    knownEncoding            IN VARCHAR2,
                    knownNumberOfChannels IN INTEGER,
                    knownSamplingRate        IN INTEGER,
                    knownSamleSize           IN INTEGER,
                    knownCompressionType  IN VARCHAR2,
                    knownAudioDuration       IN INTEGER);
```

Description

Sets the known audio attributes for the audio object.

Parameters

knownFormat
The known format.

knownEncoding
The known encoding type.

knownNumberOfChannels
The known number of channels.

knownSamplingRate
The known sampling rate.

knownSampleSize
The known sample size.

knownCompressionType
The known compression type.

knownAudioDuration
The known audio duration.

Usage Notes

Calling this method implicitly calls the setUpdateTime( ) method.

Pragmas

None.

Exceptions

None.

Examples

Set the known attributes for the audio data.

```
DECLARE
  obj ORDSYS.ORDAudio;
BEGIN
  select aud into obj from TAUD where N =1 for update;
  obj.setKnownAttributes('AUFF','MULAW', 1, 8, 8, '8BITMONOAUDIO',16);
  DBMS_OUTPUT.put_line('format: ' || obj.getformat());
  DBMS_OUTPUT.put_line('encoding: ' || obj.getEncoding());
  DBMS_OUTPUT.put_line('numberOfChannels: ' || TO_CHAR(obj.getNumberOfChannels()));
  DBMS_OUTPUT.put_line('samplingRate: ' || TO_CHAR(obj.getSamplingRate()));
  DBMS_OUTPUT.put_line('sampleSize: ' || TO_CHAR(obj.getSampleSize()));
  DBMS_OUTPUT.put_line('compressionType : ' || obj.getCompressionType());
  DBMS_OUTPUT.put_line('audioDuration: ' || TO_CHAR(obj.getAudioDuration()));
  COMMIT;
  EXCEPTION
   WHEN ORDSYS.ORDAudioExceptions.METHOD_NOT_SUPPORTED THEN
    DBMS_OUTPUT.put_line('ORDAudioExceptions.METHOD_NOT_SUPPORTED caught');
   WHEN OTHERS THEN
    DBMS_OUTPUT.put_line('EXCEPTION caught');
END;
/
```

setNumberOfChannels( )

Format

setNumberOfChannels(knownNumberOfChannels IN INTEGER);

Description

Sets the value of the numberOfChannels attribute for the audio object.

Parameters

knownNumberOfChannels
A known number of channels.

Usage Notes

Calling this method implicitly calls the setUpdateTime( ) method.

Pragmas

None.

Exceptions

NULL_INPUT_VALUE

This exception is raised if you call the setNumberOfChannels( ) method and the value for the knownNumberOfChannels parameter is NULL.

Examples

See the example in setProperties( ) on page 6-52.

setProperties( )

Format

```
setProperties(ctx              IN OUT RAW,
              setComments IN BOOLEAN);
```

Description

Reads the audio data to get the values of the object attributes and then stores them in the object attributes. This method sets the properties for the following attributes of the audio data: format, encoding type, number of channels, sampling rate, and sample size. It populates the comments field of the object with a rich set of format and application properties in XML form if the value of the setComments parameter is TRUE.

Parameters

ctx
The format plug-in context information.

setComments
If the value is TRUE, then the comments field of the object is populated with a rich set of format and application properties of the audio object in XML form, identical to what is provided by the *inter*Media Annotator utility; otherwise, if the value is FALSE, the comments field of the object remains unpopulated. The default value is FALSE.

Usage Notes

If the property cannot be extracted from the media source, then the respective attribute is set to NULL.

If the format is set to NULL, then the setProperties( ) method uses the default format plug-in; otherwise, it uses the plug-in specified by the format.

Pragmas

None.

Exceptions

AUDIO_PLUGIN_EXCEPTION

This exception is raised if you call the setProperties( ) method and the audio plug-in raises an exception.

Examples

Set the property information for known audio attributes:

```
DECLARE
  obj ORDSYS.ORDAudio;
  ctx RAW(4000) :=NULL;
BEGIN
  select aud into obj from TAUD where N =1 for update;
  obj.setProperties(ctx,FALSE);
  --DBMS_OUTPUT.put_line('format: ' || obj.getformat());
  DBMS_OUTPUT.put_line('encoding: ' || obj.getEncoding());
  DBMS_OUTPUT.put_line('numberOfChannels: ' || TO_CHAR(obj.getNumberOfChannels()));
  DBMS_OUTPUT.put_line('samplingRate: ' || TO_CHAR(obj.getSamplingRate()));
  DBMS_OUTPUT.put_line('sampleSize: ' || TO_CHAR(obj.getSampleSize()));
  update  TAUD set aud = obj where  N =1 ;
  COMMIT;
  EXCEPTION
   WHEN ORDSYS.ORDAudioExceptions.METHOD_NOT_SUPPORTED THEN
    DBMS_OUTPUT.put_line('ORDAudioExceptions.METHOD_NOT_SUPPORTED caught');
   WHEN OTHERS THEN
    DBMS_OUTPUT.put_line('EXCEPTION caught');
END;
/
```

setSamplingRate( )

Format

setSamplingRate(knownSamplingRate IN INTEGER);

Description

Sets the value of the samplingRate attribute of the audio object. The unit is Hz.

Parameters

knownSamplingRate
A known sampling rate.

Usage Notes

Calling this method implicitly calls the setUpdateTime( ) method.

Pragmas

None.

Exceptions

NULL_INPUT_VALUE

This exception is raised if you call the setSamplingRate( ) method and the value for the knownSamplingRate parameter is NULL.

Examples

See the example in setProperties( ) on page 6-52.

setSampleSize( )

Format

setSampleSize(knownSampleSize IN INTEGER);

Description

Sets the value of the sampleSize attribute of the audio object.

Parameters

knownSampleSize
A known sample size.

Usage Notes

Calling this method implicitly calls the setUpdateTime( ) method.

Pragmas

None.

Exceptions

NULL_INPUT_VALUE

This exception is raised if you call the setSampleSize( ) method and the value for the knownSampleSize parameter is NULL.

Examples

See the example in setProperties( ) on page 6-52.

6.4 Packages or PL/SQL Plug-ins

This section presents reference information on the packages or PL/SQL plug-ins provided. Table 6–1 describes the PL/SQL plug-in packages provided in the ORDPLUGINS schema.

Table 6–1 PL/SQL Plug-ins Provided in the ORDPLUGINS Schema

PL/SQL Plug-in Packages	Audio Format	MIME Type
ORDPLUGINS.ORDX_DEFAULT_AUDIO	<format>	Dependent on file format
ORDPLUGINS.ORDX_AUFF_AUDIO	AUFF	audio/basic
ORDPLUGINS.ORDX_AIFF_AUDIO	AIFF	audio/x-aiff
ORDPLUGINS.ORDX_AIFC_AUDIO	AIFC	audio/x-aiff
ORDPLUGINS.ORDX_WAVE_AUDIO	WAVE	audio/x-wave
ORDPLUGINS.ORDX_MPGA_AUDIO	MPGA	audio/mpeg

Section 6.4.1 describes the ORDPLUGINS.ORDX_DEFAULT_AUDIO package, the methods supported, and the level of support. Note that the methods supported and the level of support for the other PL/SQL plug-in packages described in Table 6–1 are identical for all plug-in packages, therefore, refer to Section 6.4.1.

6.4.1 ORDPLUGINS.ORDX_DEFAULT_AUDIO Package

Use the following provided ORDPLUGINS.ORDX_DEFAULT_AUDIO package as a guide in developing your own ORDPLUGINS.ORDX_<format>_AUDIO audio format package. This package sets the mimeType field in the setProperties( ) method with a MIME type value that is dependent on the file format.

```
CREATE OR REPLACE PACKAGE ORDX_DEFAULT_AUDIO
authid current_user
AS
--AUDIO ATTRIBUTES ACCESSORS
--Deprecated Functions Deprecated in Release 8.1.6 Begin Here
FUNCTION getFormat(ctx IN OUT RAW, obj IN ORDSYS.ORDAudio)
                RETURN VARCHAR2;
FUNCTION getEncoding(ctx IN OUT RAW, obj IN ORDSYS.ORDAudio)
                RETURN VARCHAR2;
FUNCTION getNumberOfChannels(ctx IN OUT RAW, obj IN ORDSYS.ORDAudio)
                RETURN INTEGER;
FUNCTION getSamplingRate(ctx IN OUT RAW, obj IN ORDSYS.ORDAudio)
                RETURN INTEGER;
```

```
FUNCTION getSampleSize(ctx IN OUT RAW, obj IN ORDSYS.ORDAudio)
                RETURN INTEGER;
FUNCTION getCompressionType(ctx IN OUT RAW, obj IN ORDSYS.ORDAudio)
                RETURN VARCHAR2;
FUNCTION getAudioDuration(ctx IN OUT RAW, obj IN ORDSYS.ORDAudio)
                RETURN INTEGER;
--Deprecated Functions Deprecated in Release 8.1.6 End Here

PROCEDURE setProperties(ctx IN OUT RAW,
                        obj IN OUT NOCOPY ORDSYS.ORDAudio,
                        setComments IN NUMBER := 0);
FUNCTION checkProperties(ctx IN OUT RAW, obj IN OUT ORDSYS.ORDAudio)
                RETURN NUMBER;
FUNCTION getAttribute(ctx IN OUT RAW,
                      obj IN ORDSYS.ORDAudio,
                      name IN VARCHAR2) RETURN VARCHAR2;
PROCEDURE getAllAttributes(ctx IN OUT RAW,
                           obj IN ORDSYS.ORDAudio,
                           attributes IN OUT NOCOPY CLOB);
--AUDIO PROCESSING METHODS
FUNCTION processCommand(ctx       IN OUT RAW,
                        obj       IN OUT NOCOPY ORDSYS.ORDAudio,
                        cmd       IN VARCHAR2,
                        arguments IN VARHAR2,
                        result    OUT RAW)
        RETURN RAW;

PRAGMA RESTRICT_REFERENCES(getFormat, WNDS, WNPS, RNDS, RNPS);
PRAGMA RESTRICT_REFERENCES(getEncoding, WNDS, WNPS, RNDS, RNPS);
PRAGMA RESTRICT_REFERENCES(getNumberOfChennels, WNDS, WNPS, RNDS, RNPS);
PRAGMA RESTRICT_REFERENCES(getSamplingRate, WNDS, WNPS, RNDS, RNPS);
PRAGMA RESTRICT_REFERENCES(getSampleSize, WNDS, WNPS, RNDS, RNPS);
PRAGMA RESTRICT_REFERENCES(getCompressionType, WNDS, WNPS, RNDS, RNPS);
PRAGMA RESTRICT_REFERENCES(getAttribute, WNDS, WNPS, RNDS, RNPS);
PRAGMA RESTRICT_REFERENCES(getAudioDuration, WNDS, WNPS, RNDS, RNPS);

END;
/
```

Table 6–2 shows the methods supported in the ORDPLUGINS.ORDX_DEFAULT_AUDIO package and the exceptions raised if you call a method that is not supported.

Table 6–2 Methods Supported in the ORDPLUGINS.ORDX_DEFAULT_AUDIO Package

Name of Method	Level of Support
getFormat	Supported; if the source is local, get the attribute and return the file format, but if the source is NULL, raise an ORDSYS.ORDSourceExceptions.EMPTY_SOURCE exception; otherwise, if the source is external, raise an ORDSYS.ORDAudioExceptions.LOCAL_DATA_SOURCE_REQUIRED exception.
getEncoding	Supported; if the source is local, get the attribute and return the encoding, but if the source is NULL, raise an ORDSYS.ORDSourceExceptions.EMPTY_SOURCE exception; otherwise, if the source is external, raise an ORDSYS.ORDAudioExceptions.LOCAL_DATA_SOURCE_REQUIRED exception.
getNumberOfChannels	Supported; if the source is local, get the attribute and return the number of channels, but if the source is NULL, raise an ORDSYS.ORDSourceExceptions.EMPTY_SOURCE exception; otherwise, if the source is external, raise an ORDSYS.ORDAudioExceptions.LOCAL_DATA_SOURCE_REQUIRED exception.
getSamplingRate	Supported; if the source is local, get the attribute and return the sampling rate, but if the source is NULL, raise an ORDSYS.ORDSourceExceptions.EMPTY_SOURCE exception; otherwise, if the source is external, raise an ORDSYS.ORDAudioExceptions.LOCAL_DATA_SOURCE_REQUIRED exception.
getSampleSize	Supported; if the source is local, get the attribute and return the sample size, but if the source is NULL, raise an ORDSYS.ORDSourceExceptions.EMPTY_SOURCE exception; otherwise, if the source is external, raise an ORDSYS.ORDAudioExceptions.LOCAL_DATA_SOURCE_REQUIRED exception.
getCompressionType	Supported; if the source is local, get the attribute and return the compression type, but if the source is NULL, raise an ORDSYS.ORDSourceExceptions.EMPTY_SOURCE exception; otherwise, if the source is external, raise an ORDSYS.ORDAudioExceptions.LOCAL_DATA_SOURCE_REQUIRED exception.
getAudioDuration	Supported; if the source is local, get the attribute and return the audio duration, but if the source is NULL, raise an ORDSYS.ORDSourceExceptions.EMPTY_SOURCE exception; otherwise, if the source is external, raise an ORDSYS.ORDAudioExceptions.LOCAL_DATA_SOURCE_REQUIRED exception.

Table 6–2 Methods Supported in the ORDPLUGINS.ORDX_DEFAULT_AUDIO Package(Cont.)

Name of Method	Level of Support
setProperties	Supported; if the source is local, process the local data and set the properties, but if the source is NULL, raise an ORDSYS.ORDSourceExceptions.EMPTY_SOURCE exception; if the source is a BFILE, then process the BFILE and set the properties; if the source is neither local nor a BFILE, get the media content into a temporary LOB, process the data, and set the properties.
checkProperties	Supported; if the source is local, process the local data and check the properties, but if the source is NULL, raise an ORDSYS.ORDSourceExceptions.EMPTY_SOURCE exception; if the source is a BFILE, then process the BFILE and check the properties; if the source is neither local nor a BFILE, get the media content into a temporary LOB, process the data, and check the properties.
getAttribute	Not supported - raises exceptions: METHOD_NOT_SUPPORTED and AUDIO_PLUGIN_EXCEPTION.
getAllAttributes	Supported; if the source is local, get the attributes and return them, but if the source is NULL, raise an ORDSYS.ORDSourceExceptions.EMPTY_SOURCE exception; otherwise, if the source is external, raise an ORDSYS.ORDAudioExceptions.LOCAL_DATA_SOURCE_REQUIRED exception.
processCommand	Not supported - raises exceptions: METHOD_NOT_SUPPORTED and AUDIO_PLUGIN_EXCEPTION.

6.4.2 Extending *inter*Media to Support a New Audio Data Format

Extending *inter*Media to support a new audio data format consists of four steps:

1. Design your new audio data format.
2. Implement your new audio data format and name it, for example, ORDX_MY_AUDIO.SQL.
3. Install your new ORDX_MY_AUDIO.SQL plug-in in the ORDPLUGINS schema.
4. Grant EXECUTE privileges on your new plug-in, for example, ORDX_MY_AUDIO.SQL plug-in, to PUBLIC.

Section 3.1.12 briefly describes how to extend *inter*Media to support a new audio data format and describes the interface. A package body listing is provided in Example 6–1 to assist you in this operation. Add your variables to the places that say "--Your variables go here" and add your code to the places that say "--Your code goes here".

See Section F.1 for more information on installing your own audio format plug-in and running the sample scripts provided.

Example 6–1 Show the Package Body for Extending Support to a New Audio Data Format

```
CREATE OR REPLACE PACKAGE BODY ORDX_MY_AUDIO
AS
  --AUDIO ATTRIBUTES ACCESSORS
  FUNCTION  getFormat(ctx IN OUT RAW, obj IN ORDSYS.ORDAudio)
  RETURN VARCHAR2
  IS
--Your variables go here
  BEGIN
--Your code goes here
  END;
  FUNCTION  getEncoding(ctx IN OUT RAW, obj IN ORDSYS.ORDAudio)
  RETURN VARCHAR2
  IS
--Your variables go here
  BEGIN
--Your code goes here
  END;
  FUNCTION  getNumberOfChannels(ctx IN OUT RAW, obj IN ORDSYS.ORDAudio)
  RETURN INTEGER
  IS
--Your variables go here
  BEGIN
--Your code goes here
  END;
  FUNCTION  getSamplingRate(ctx IN OUT RAW, obj IN ORDSYS.ORDAudio)
  RETURN INTEGER
  IS
--Your variables go here
  BEGIN
--Your code goes here
  END;
  FUNCTION  getSampleSize(ctx IN OUT RAW, obj IN ORDSYS.ORDAudio)
  RETURN INTEGER
  IS
--Your variables go here
  BEGIN
--Your code goes here
  END;
  FUNCTION  getCompressionType(ctx IN OUT RAW, obj IN ORDSYS.ORDAudio)
```

```
  RETURN VARCHAR2
  IS
--Your variables go here
  BEGIN
--Your code goes here
  END;
  FUNCTION  getAudioDuration(ctx IN OUT RAW,
                             obj IN ORDSYS.ORDAudio)
  RETURN INTEGER
  IS
--Your variables go here
  BEGIN
--Your code goes here
  END;
  PROCEDURE setProperties(ctx IN OUT RAW,
                          obj IN OUT NOCOPY ORDSYS.ORDAudio,
                          setComments IN NUMBER :=0)
  IS
--Your variables go here
  BEGIN
--Your code goes here
  END;
  FUNCTION checkProperties(ctx IN OUT RAW, obj IN OUT ORDSYS.ORDAudio)
  RETURN NUMBER
  IS
--Your variables go here
  BEGIN
--Your code goes here
  END;
  FUNCTION  getAttribute(ctx IN OUT RAW,
                         obj IN ORDSYS.ORDAudio,
                         name IN VARCHAR2)
  RETURN VARCHAR2
  IS
--Your variables go here
  BEGIN
--Your code goes here
  END;
  PROCEDURE getAllAttributes(ctx IN OUT RAW,
                             obj IN ORDSYS.ORDAudio,
                             attributes IN OUT NOCOPY CLOB)
  IS
--Your variables go here
  BEGIN
--Your code goes here
```

```
  END;
  -- AUDIO PROCESSING METHODS
  FUNCTION  processCommand(
                                ctx       IN OUT RAW,
                                obj       IN OUT NOCOPY ORDSYS.ORDAudio,
                                cmd       IN VARCHAR2,
                                arguments IN VARCHAR2,
                                result    OUT RAW)
  RETURN RAW
  IS
--Your variables go here
  BEGIN
--Your code goes here
  END;
END;
/
show errors;
```

7

ORDDoc Reference Information

Oracle *inter*Media contains information about the ORDDoc type:

- Object type -- see Section 7.1.
- Constructors -- see Section 7.2.
- Methods -- see Section 7.3.
- Packages or PL/SQL plug-ins -- see Section 7.4.

The examples in this chapter assume that the test media table TDOC has been created and filled with data. This table was created using the SQL statements described in Section 7.3.1.

> **Note:** If you manipulate the media data itself (by either directly modifying the BLOB or changing the external source), then you must ensure that the object attributes stay synchronized and the update time is modified; otherwise, the object attributes will not match the media data.

Methods invoked at the ORDSource level that are handed off to the source plug-in for processing have ctx(RAW(4000)) as the first argument. Before calling any of these methods for the first time, the client must allocate the ctx structure, initialize it to NULL, and invoke the openSource( ) method. At this point, the source plug-in can initialize context for this client. When processing is complete, the client should invoke the closeSource( ) method.

Methods invoked from a source plug-in call have the first argument as ctx (RAW(4000)).

Methods invoked at the ORDDoc level that are handed off to the format plug-in for processing have ctx (RAW(4000)) as the first argument. Before calling any of these methods for the first time, the client must allocate the ctx structure and initialize it to NULL.

Note: In the current release, not all source or format plug-ins will use the ctx argument, but if you code as previously described, your application should work with any current or future source or format plug-in.

You should use any of the individual set methods to set the value of the attribute for an object for formats not natively supported; otherwise, for formats natively supported, use the setProperties( ) method to populate the attributes of the object.

7.1 Object Types

Oracle *inter*Media describes the ORDDoc object type, which supports the storage and management of any media data including text, image, audio, and video.

ORDDoc Object Type

The ORDDoc object type supports the storage and management of media data. This object type is defined as follows:

```
CREATE OR REPLACE TYPE ORDDoc
AS OBJECT
(
  -- ATTRIBUTES
source              ORDSource,
format              VARCHAR(80),
mimeType            VARCHAR(80),
contentLength       INTEGER,
comments            CLOB,

  -- METHODS
-- CONSTRUCTORS
--
STATIC FUNCTION init( ) RETURN ORDDoc,
STATIC FUNCTION init(srcType     IN VARCHAR2,
                     srcLocation IN VARCHAR2,
                     srcName     IN VARCHAR2) RETURN ORDDoc,
-- Methods associated with the mimeType attribute
MEMBER FUNCTION getMimeType RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getMimeType, WNDS, WNPS, RNDS, RNPS),
MEMBER PROCEDURE setMimeType(mime IN VARCHAR2),

-- Methods associated with the date attribute
MEMBER FUNCTION getUpdateTime RETURN DATE,
PRAGMA RESTRICT_REFERENCES(getUpdateTime, WNDS, WNPS, RNDS, RNPS),
MEMBER PROCEDURE setUpdateTime(current_time DATE),

-- Methods associated with the format attribute
MEMBER FUNCTION getFormat RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getFormat, WNDS, WNPS, RNDS, RNPS),
MEMBER PROCEDURE setFormat(format IN VARCHAR2),

-- Methods associated with the source attribute
MEMBER FUNCTION  isLocal RETURN BOOLEAN,
PRAGMA RESTRICT_REFERENCES(isLocal, WNDS, WNPS, RNDS, RNPS),
MEMBER PROCEDURE  setLocal,
MEMBER PROCEDURE  clearLocal,
```

```
MEMBER PROCEDURE setSource(source_type     IN VARCHAR2,
                           source_location IN VARCHAR2,
                           source_name     IN VARCHAR2),
MEMBER FUNCTION getSource RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getSource, WNDS, WNPS, RNDS, RNPS),

MEMBER FUNCTION getSourceType RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getSourceType, WNDS, WNPS, RNDS, RNPS),

MEMBER FUNCTION getSourceLocation RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getSourceLocation, WNDS, WNPS, RNDS, RNPS),

MEMBER FUNCTION getSourceName RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getSourceName, WNDS, WNPS, RNDS, RNPS),

MEMBER PROCEDURE setProperties(ctx         IN OUT RAW,
                               setComments IN BOOLEAN),

MEMBER FUNCTION getBFILE RETURN BFILE,
PRAGMA RESTRICT_REFERENCES(getBFILE, WNDS, WNPS, RNDS, RNPS),

MEMBER PROCEDURE import(ctx      IN OUT RAW,
                        set_prop IN BOOLEAN),
MEMBER PROCEDURE importFrom(ctx             IN OUT RAW,
                            source_type     IN VARCHAR2,
                            source_location IN VARCHAR2,
                            source_name     IN VARCHAR2
                            set_prop        IN BOOLEAN),
MEMBER PROCEDURE export(ctx             IN OUT RAW,
                        source_type     IN VARCHAR2,
                        source_location IN VARCHAR2,
                        source_name     IN VARCHAR2),

MEMBER FUNCTION openSource(userArg IN RAW, ctx OUT RAW) RETURN INTEGER,
MEMBER FUNCTION closeSource(ctx IN OUT RAW) RETURN INTEGER,
MEMBER FUNCTION trimSource(ctx     IN OUT RAW,
                           newlen  IN INTEGER) RETURN INTEGER,
MEMBER PROCEDURE readFromSource(ctx      IN OUT RAW,
                                startPos IN INTEGER,
                                numBytes IN OUT INTEGER,
                                buffer   OUT RAW),
MEMBER PROCEDURE writeToSource(ctx      IN OUT RAW,
                               startPos IN INTEGER,
                               numBytes IN OUT INTEGER,
                               buffer   IN RAW),
```

```
MEMBER FUNCTION getContentLength RETURN INTEGER,
PRAGMA RESTRICT_REFERENCES(getContentLength, WNDS, WNPS, RNDS, RNPS),

MEMBER PROCEDURE getContentInLob(ctx      IN OUT RAW,
                                 dest_lob IN OUT NOCOPY BLOB,
                                 mimeType OUT VARCHAR2,
                                 format   OUT VARCHAR2),

MEMBER FUNCTION getContent RETURN BLOB,
PRAGMA RESTRICT_REFERENCES(getContent, WNDS, WNPS, RNDS, RNPS),

MEMBER PROCEDURE deleteContent,

MEMBER FUNCTION processSourceCommand(ctx       IN OUT RAW,
                                     cmd       IN VARCHAR2,
                                     arguments IN VARCHAR2,
                                     result    OUT RAW)
                RETURN RAW

);
```

where:

- source: the ORDSource where the media data is found.
- format: the format in which the media data is stored.
- mimeType: the MIME type information.
- contentLength: the length of the media data stored in the source.
- comments: the metadata information of the media object.

7.2 Constructors

This section describes the constructor functions.

The *inter*Media constructor functions are as follows:

- init( )
- init(srcType,srcLocation,srcName)

init( )

Format

init( ) RETURN ORDDoc;

Description

Allows for easy initialization of instances of the ORDDoc object type.

Parameters

None.

Pragmas

None.

Exceptions

None.

Usage Notes

This static method initializes all the ORDDoc attributes to NULL with the following exceptions:

- source.updateTime is set to SYSDATE
- source.local is set to 1 (local)
- source.localData is set to empty_blob

You should begin using the init( ) method as soon as possible to allow you to more easily initialize the ORDDoc object type, especially if the ORDDoc type evolves and attributes are added in a future release. INSERT statements left unchanged using the default constructor (which initializes each object attribute), will fail under these circumstances.

Examples

Initialize the ORDDoc object attributes:

```
BEGIN
   INSERT INTO tdoc VALUES (1,ORDSYS.ORDDoc.init());
```

```
END;
/
```

init(srcType,srcLocation,srcName)

Format

```
init(srcType     IN VARCHAR2,
   srcLocation IN VARCHAR2,
   srcName     IN VARCHAR2)
 RETURN ORDDoc;
```

Description

Allows for easy initialization of instances of the ORDDoc object type.

Parameters

srcType
The source type of the media data.

srcLocation
The source location of the media data.

srcName
The source name of the mediamedia data.

Pragmas

None.

Exceptions

None.

Usage Notes

This static method initializes all the ORDDoc attributes to NULL with the following exceptions:

- source.updateTime is set to SYSDATE
- source.local is set to 0
- source.localData is set to empty_blob

- source.srcType is set to the input value
- source.srcLocation is set to the input value
- source.srcName is set to the input value

You should begin using the init( ) method as soon as possible to allow you to more easily initialize the ORDDoc object type, especially if the ORDDoc type evolves and attributes are added in a future release. INSERT statements left unchanged using the default constructor (which initializes each object attribute), will fail under these circumstances.

Examples

Initialize the ORDDoc object attributes.

> **Note:** You must first create the DOCDIR directory; this is the directory where your media files reside. Create this directory using the following SQL statement and then grant read access to PUBLIC to this directory. Change this directory specification to match the location of your media files.

```
-- Create the DOCDIR load directory; this is the directory where the media
-- files reside.

CREATE OR REPLACE DIRECTORY docdir
       as 'e:\oracle\ord\doc\demo';
GRANT READ ON DIRECTORY docdir TO PUBLIC WITH GRANT OPTION;

BEGIN
  INSERT INTO tdoc VALUES (2, ORDSYS.ORDDoc.init('file','DOCDIR','doc1.pdf');
END;
/
```

7.3 Methods

This section presents reference information on the Oracle *inter*Media methods used for media data manipulation. These methods are described in the following groupings:

ORDDoc Methods Associated with mimeType Attribute

- getMimeType: returns the MIME type of the stored media data. See "getMimeType( )" on page 5-17 for information.
- setMimeType( ): sets the MIME type of the stored media data. This method is called implicitly by any method that modifies natively supported media formats. See "setMimeType( )" on page 5-35 for information.

ORDDoc Methods Associated with the updateTime Attribute

- getUpdateTime: returns the time when the media object was last updated. See "getUpdateTime( )" on page 5-25 for information.
- setUpdateTime( ): sets the update time for the media object. This method is called implicitly by methods that modify natively supported media formats. See "setUpdateTime( )" on page 5-39 for information.

ORDDoc Methods Associated with the format Attribute

- setFormat( ): sets the object attribute value of the format of the media data. See "setFormat( )" on page 7-24 for information.
- getFormat: returns the object attribute value of the format in which the media data is stored. See "getFormat" on page 7-17.

ORDDoc Methods Associated with the source Attribute

- isLocal: returns TRUE if the data is stored locally in a BLOB or FALSE if the data is external. See "isLocal( )" on page 5-26 for information.
- setLocal: sets a flag to indicate that the data is stored locally in a BLOB. See "setLocal( )" on page 5-34 for information.
- clearLocal: clears the flag to indicate that the data is stored externally. See "clearLocal( )" on page 5-5 for information.
- setSource( ): sets the source information to where media data is found. See "setSource( )" on page 5-37 for information.

- getSource: returns a formatted string containing complete information about the external data source formatted as a URL. See "getSource( )" on page 5-19 for information.
- getSourceType: returns the external source type of the media data. See "getSourceType( )" on page 5-23 for information.
- getSourceLocation: returns the external source location of the media data. See "getSourceLocation( )" on page 5-21 for information.
- getSourceName: returns the external source name of the media data. See "getSourceName( )" on page 5-22 for information.
- setProperties( ): reads the media data to get the values of the object attributes and then stores them in the object for known format types. If the value for the setComments parameter is TRUE, then the comments field of the object will be populated with an extensive set of format and application properties of the media object in XML form, identical to what is provided by the *inter*Media Annotator utility. For the known attributes that ORDDoc understands, it sets the properties for these attributes. These include the format of the media data. This method also automatically sets the content length of the media and sets the update time. See "setProperties( )" on page 7-26.
- import( ): transfers data from an external data source (specified by calling setSourceInformation( )) to the local source (localData) within an Oracle database, setting the value of the local attribute to "1", meaning local, and updating the timestamp. See "import( )" on page 7-18.
- importFrom( ): transfers data from the specified external data source (source type, location, name) to the local source (localData) within an Oracle database, setting the value of the local attribute to "1", meaning local and updating the timestamp. See "importFrom( )" on page 7-21.
- export( ): copies data from a local source (localData) within an Oracle database to the specified external data source, and stores source information in the source. See "export( )" on page 5-9 for information.

  Note: The export( ) method natively supports only sources of source type file. User-defined sources may support the export( ) method.

- getContentLength( ): returns the length of the data source (as number of bytes). See "getContentLength( )" on page 7-16.

- getContentInLob( ): returns content into a temporary LOB. See "getContentInLob( )" on page 7-14 for information.
- getContent: returns the handle to the BLOB used to store contents locally. See "getContent( )" on page 5-15 for information.
- deleteContent: deletes the content of the local BLOB. See "deleteContent( )" on page 5-8 for information.
- getBFILE: returns the external content as a BFILE. See "getBFILE( )" on page 5-13 for information.
- processSourceCommand( ): sends a command and related arguments to the source plug-in. See "processSourceCommand( )" on page 5-29 for information.

ORDDoc Methods Associated with File Operations

- openSource( ): opens a data source or a BLOB. See "openSource( )" on page 5-27 for information.
- closeSource( ): closes a data source or a BLOB. See "closeSource( )" on page 5-6 for information.
- trimSource( ): trims a data source or a BLOB. See "trimSource( )" on page 5-40 for information.
- readFromSource( ): reads a buffer of *n* bytes from a source beginning at a start position. See "readFromSource( )" on page 5-32 for information.
- writeToSource( ): writes a buffer of *n* bytes to a source beginning at a start position. See "writeToSource( )" on page 5-42 for information.

For more information on object types and methods, see *Oracle9i Database Concepts*.

7.3.1 Example Table Definitions

The methods described in this reference chapter show examples based on a test media table TDOC. Refer to the TDOC table definition that follows when reading through the examples:

TDOC Table Definition

```
CREATE TABLE TDOC(n NUMBER CONSTRAINT n_pk PRIMARY KEY,
                  doc ORDSYS.ORDDOC)
STORAGE (INITIAL 100K NEXT 100K PCTINCREASE 0);

INSERT INTO tdoc VALUES(1, ORDSYS.ORDDoc.init());
```

```
INSERT INTO tdoc VALUES(2, ORDSYS.ORDDoc.init());
```

getContentInLob( )

Format

```
getContentInLob(
        ctx          IN OUT RAW,
        dest_lob     IN OUT NOCOPY BLOB,
        mimeType     OUT VARCHAR2,
        format       OUT VARCHAR2);
```

Description

Copies data from a data source into the specified BLOB. The BLOB must not be the BLOB in source.localData.

Parameters

ctx
The source plug-in context information.

dest_lob
The LOB in which to receive data.

mimeType
The MIME type of the data; this may or may not be returned.

format
The format of the data; this may or may not be returned.

Usage Notes

None.

Pragmas

None.

Exceptions

ORDSourceExceptions.INCOMPLETE_SOURCE_INFORMATION

This exception is raised if you call the getContentInLob( ) method and the value of srcType is NULL.

ORDSourceExceptions.METHOD_NOT_SUPPORTED

This exception is raised if you call the getContentInLob( ) method and this method is not supported by the source plug-in being used.

ORDSourceExceptions.SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the getContentInLob( ) method within a source plug-in when any other exception is raised.

See Appendix H for more information about these exceptions.

Examples

Get data from a data source and put it into the specified BLOB:

```
DECLARE
 obj ORDSYS.ORDDoc;
 tempBLob BLOB;
 mimeType VARCHAR2(4000);
 format VARCHAR2(4000);
 ctx RAW(4000) :=NULL;
BEGIN
 SELECT doc INTO obj FROM tdoc WHERE N = 1 ;
 if(obj.isLocal()) then
   DBMS_OUTPUT.put_line('local is true');
 end if;
 DBMS_LOB.CREATETEMPORARY(tempBLob, true, 10);
 obj.getContentInLob(ctx,tempBLob, mimeType,format);
 DBMS_OUTPUT.PUT_LINE(TO_CHAR(DBMS_LOB.getLength(tempBLob)));
 EXCEPTION
  WHEN ORDSYS.ORDSourceExceptions.METHOD_NOT_SUPPORTED THEN
   DBMS_OUTPUT.put_line('ORDSourceExceptions.METHOD_NOT_SUPPORTED caught');
  WHEN OTHERS THEN
   DBMS_OUTPUT.put_line('EXCEPTION caught');
END;
/
```

getContentLength( )

Format

```
getContentLength RETURN INTEGER;
```

Description

Returns the length of the media data content stored in the source.

Parameters

None.

Usage Notes

This method is not supported for all source types. For example, HTTP type sources do not support this method. If you want to implement this call for HTTP type sources, you must define your own modified HTTP source type and implement this method on it.

Pragmas

PRAGMA RESTRICT_REFERENCES(getContentLength, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Examples

See the example in import( ) on page 7-19.

getFormat

Format

getFormat RETURN VARCHAR2;

Description

Returns the value of the format attribute of the media object.

Parameters

None.

Usage Notes

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getFormat, WNDS, WNPS, RNDS, RNPS)

Exceptions

INVALID_FORMAT_TYPE

This exception is raised if you call the getFormat( ) method and the value for format is NULL.

Examples

See the example in setProperties( ) on page 7-26.

import( )

Format

```
import(ctx        IN OUT RAW
       set_prop IN BOOLEAN);
```

Description

Transfers media data from an external media data source to a local source (localData) within an Oracle database.

Parameters

ctx
The source plug-in context information.This must be allocated. You must call the openSource( ) method; see the introduction to this chapter for more information.

set_prop
If the value is TRUE, then the setProperties( ) method is called to read the media data to get the values of the object attributes and store them in the object attributes, otherwise, if the value is FALSE, the set Properties( ) method is not called. The default value is FALSE.

Usage Notes

Use the setSource( ) method to set the external source type, location, and name prior to calling the import( ) method.

You must ensure that the directory exists or is created before you use this method.

After importing data from an external media data source to a local source (within an Oracle database), the source information remains unchanged (that is, pointing to the source from where the data was imported).

Invoking this method implicitly calls the setUpdateTime( ) and setLocal methods.

Pragmas

None.

Exceptions

ORDSourceExceptions.INCOMPLETE_SOURCE_INFORMATION

This exception is raised if you call the import( ) method and the value of srcType is NULL.

ORDSourceExceptions.NULL_SOURCE

This exception is raised if you call the import( ) method and the value of dlob is NULL.

ORDSourceExceptionsMETHOD_NOT_SUPPORTED

This exception is raised if you call the import( ) method and the import( ) method is not supported by the source plug-in being used.

ORDSourceExceptions.SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the import( ) method within a source plug-in when any other exception is raised.

ORDSYS.ORDDocExceptions.DOC_PLUGIN_EXCEPTION

This exception is raised if you call the import( ) method and the setProperties( ) method raises an exception from within the media plug-in.

See Appendix H for more information about these exceptions.

Examples

Import media data from an external media data source into the local source:

```
DECLARE
  obj ORDSYS.ORDDoc;
  ctx RAW(4000) :=NULL;
BEGIN
  SELECT doc INTO obj FROM tdoc  WHERE N=1 FOR UPDATE;
  DBMS_OUTPUT.PUT_LINE('setting and getting source');
  DBMS_OUTPUT.PUT_LINE('-------------------------');
  -- set source to a file
  obj.setSource('file','DOCDIR','testdoc.dat');
  -- get source information
  DBMS_OUTPUT.PUT_LINE(obj.getSource());
  -- import data
  obj.import(ctx,FALSE);
  -- check size
  DBMS_OUTPUT.PUT_LINE(TO_CHAR(DBMS_LOB.getLength(obj.getContent())));
  DBMS_OUTPUT.PUT_LINE(obj.getSource());
```

```
  DBMS_OUTPUT.PUT_LINE('deleting contents');
  DBMS_OUTPUT.PUT_LINE('-----------------');
  obj.deleteContent();
  DBMS_OUTPUT.PUT_LINE(TO_CHAR(DBMS_LOB.getLength(obj.getContent())));
  UPDATE tdoc SET doc=obj WHERE N=1;
  COMMIT;
  EXCEPTION
    WHEN ORDSYS.ORDSourceExceptions.METHOD_NOT_SUPPORTED THEN
      DBMS_OUTPUT.PUT_LINE('ORDSourceExceptions.METHOD_NOT_SUPPORTED caught');
    WHEN ORDSYS.ORDDocExceptions.DOC_PLUGIN_EXCEPTION THEN
      DBMS_OUTPUT.put_line('DOC PLUGIN EXCEPTION caught');
END;
/
```

importFrom( )

Format

```
importFrom(ctx                IN OUT RAW,
           source_type        IN VARCHAR2,
           source_location    IN VARCHAR2,
           source_name        IN VARCHAR2
           set_prop           IN BOOLEAN);
```

Description

Transfers media data from the specified external media data source to a local source (localData) within an Oracle database.

Parameters

ctx
The source plug-in context information. This must be allocated. You must call the openSource( ) method; see the introduction to this chapter for more information.

source_type
The source type of the media data.

source_location
The location from where the media data is to be imported.

source_name
The name of the media data.

set_prop
If the value is TRUE, then the setProperties( ) method is called to read the media data to get the values of the object attributes and store them in the object attributes, otherwise, if the value is FALSE, the set Properties( ) method is not called. The default value is FALSE.

Usage Notes

This method is similar to the import( ) method except the source information is specified as parameters to the method instead of separately.

You must ensure that the directory exists or is created before you use this method.

After importing data from an external media data source to a local source (within an Oracle database), the source information (that is, pointing to the source from where the data was imported) is set to the input values.

Invoking this method implicitly calls the setUpdateTime( ) and setLocal methods.

Pragmas

None.

Exceptions

ORDSourceExceptions.INCOMPLETE_SOURCE_INFORMATION

This exception is raised if you call the import( ) method and the value of srcType is NULL.

ORDSourceExceptionsMETHOD_NOT_SUPPORTED

This exception is raised if you call the importFrom( ) method and this method is not supported by the source plug-in being used.

ORDSourceExceptions.SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the importFrom( ) method within a source plug-in when any other exception is raised.

ORDSYS.ORDDocExceptions.DOC_PLUGIN_EXCEPTION

This exception is raised if you call the import( ) method and the setProperties( ) method raises an exception from within the media plug-in.

See Appendix H for more information about these exceptions.

Examples

Import media data from the specified external data source into the local source:

```
DECLARE
  obj ORDSYS.ORDDoc;
  ctx RAW(4000) :=NULL;
BEGIN
```

```
  SELECT doc INTO obj FROM tdoc WHERE N=1 FOR UPDATE FOR UPDATE;
  DBMS_OUTPUT.PUT_LINE('setting and getting source');
  DBMS_OUTPUT.PUT_LINE('--------------------------');
  -- set source to a file
  -- import data
  obj.importFrom(ctx,'file','DOCDIR','testdoc.dat',FALSE);
  -- check size
  DBMS_OUTPUT.PUT_LINE(TO_CHAR(DBMS_LOB.GETLENGTH(obj.getContent())));
  DBMS_OUTPUT.PUT_LINE(obj.getSource());
  DBMS_OUTPUT.PUT_LINE('deleting contents');
  DBMS_OUTPUT.PUT_LINE('-----------------');
  obj.deleteContent();
  DBMS_OUTPUT.PUT_LINE(TO_CHAR(DBMS_LOB.GETLENGTH(obj.getContent())));
  UPDATE tdoc SET doc=obj WHERE N=1;
  COMMIT;
  EXCEPTION
    WHEN ORDSYS.ORDSourceExceptions.METHOD_NOT_SUPPORTED THEN
      DBMS_OUTPUT.PUT_LINE('ORDSourceExceptions.METHOD_NOT_SUPPORTED caught');
    WHEN ORDSYS.ORDDocExceptions.DOC_PLUGIN_EXCEPTION THEN
      DBMS_OUTPUT.put_line('DOC PLUGIN EXCEPTION caught');
END;
/
```

setFormat( )

Format

setFormat(knownFormat IN VARCHAR2);

Description

Sets the format attribute of the media object.

Parameters

knownFormat
The known format of the data to be set in the corresponding media object.

Usage Notes

Calling this method implicitly calls the setUpdateTime( ) method.

Pragmas

None.

Exceptions

NULL_INPUT_VALUE

This exception is raised if you call the setFormat method and the value for the knownFormat parameter is NULL.

Examples

Set the format for some media data:

```
DECLARE
  obj ORDSYS.ORDDoc;
BEGIN
  select doc into obj from tdoc where N =1 for update;
  obj.setFormat('PDF');
  DBMS_OUTPUT.put_line('format: ' || obj.getformat());
  COMMIT;
  EXCEPTION
   WHEN ORDSYS.ORDAudioExceptions.NULL_INPUT_VALUE THEN
    DBMS_OUTPUT.put_line('ORDSourceExceptions.NULL_INPUT_VALUE caught');
   WHEN OTHERS THEN
```

```
    DBMS_OUTPUT.put_line('EXCEPTION caught');
END;
/
```

setProperties( )

Format

```
setProperties(ctx          IN OUT RAW,
              setComments IN BOOLEAN);
```

Description

Reads the media data to get the values of the object attributes and then stores them in the object attributes. This method sets the properties for the following attributes of the media data: format, MIME type, and content length. It populates the comments field of the object with an extensive set of format and application properties in XML form if the value of the setComments parameter is TRUE.

> **Note:** Some audio, image, and video formats are supported, but no media formats are supported. This method works for only natively supported formats. See Appendix A, Appendix B, and Appendix C for information on natively supported media formats.

Parameters

ctx
The format plug-in context information.

setComments
If the value is TRUE, then the comments field of the object is populated with an extensive set of format and application properties of the media object in XML form, identical to what is provided by the *inter*Media Annotator utility; otherwise, if the value is FALSE, the comments field of the object remains unpopulated. The default value is FALSE.

Usage Notes

If the property cannot be extracted from the media source, then the respective attribute is set to NULL.

If the format is set to NULL, then the setProperties( ) method uses the default format plug-in; otherwise, it uses the plug-in specified by the format.

Pragmas

None.

Exceptions

DOC_PLUGIN_EXCEPTION

This exception is raised if you call the setProperties( ) method and the media plug-in raises an exception.

Examples

Example 1: Set the property information for known media attributes:

```
DECLARE
  obj ORDSYS.ORDDoc;
  ctx RAW(4000) :=NULL;
BEGIN
  SELECT doc INTO obj FROM tdoc WHERE N =1 FOR UPDATE;
  obj.setProperties(ctx,FALSE);
  --DBMS_OUTPUT.put_line('format: ' || obj.getformat());
  UPDATE  tdoc SET doc = obj WHERE  N =1 ;
  COMMIT;
  EXCEPTION
   WHEN ORDSYS.ORDDocExceptions.DOC_PLUGIN_EXCEPTION THEN
    DBMS_OUTPUT.put_line('DOC PLUGIN EXCEPTION caught');
   WHEN OTHERS THEN
    DBMS_OUTPUT.put_line('EXCEPTION caught');
END;
/
```

Example 2: Set the property information for known media attributes and store the format and application properties in the comments attribute. Create an extensible index on the contents of the comments attribute using Oracle Text.

```
DECLARE
  obj ORDSYS.ORDDoc;
  ctx RAW(4000) :=NULL;
BEGIN
  SELECT doc INTO obj FROM tdoc WHERE N =1 FOR UPDATE;
  obj.setProperties(ctx,TRUE);
  --DBMS_OUTPUT.put_line('format: ' || obj.getformat());
  UPDATE  tdoc SET doc = obj WHERE  N =1 ;
  COMMIT;
  EXCEPTION
   WHEN ORDSYS.ORDDocExceptions.DOC_PLUGIN_EXCEPTION THEN
    DBMS_OUTPUT.put_line('DOC PLUGIN EXCEPTION caught');
   WHEN OTHERS THEN
```

```
    DBMS_OUTPUT.put_line('EXCEPTION caught');
END;
/
-- Must have interMedia text installed on your system.
CREATE INDEX mediaindex ON tdoc(doc.comments) INDEXTYPE IS ctxsys.context;
```

7.4 Packages or PL/SQL Plug-ins

This section presents reference information on the packages or PL/SQL plug-ins provided. Table 7–1 describes the PL/SQL plug-in packages provided in the ORDPLUGINS schema.

Table 7–1 PL/SQL Plug-ins Provided in the ORDPLUGINS Schema

PL/SQL Plug-in Packages	Media Format	MIME Type
ORDPLUGINS.ORDX_DEFAULT_DOC	<format>	Dependent on file format

Section 7.4.1 describes the ORDPLUGINS.ORDX_DEFAULT_DOC package, the method supported, and the level of support. The method supported and the level of support for the PL/SQL plug-in package is described in Table 7–2.

7.4.1 ORDPLUGINS.ORDX_DEFAULT_DOC Package

Use the following provided ORDPLUGINS.ORDX_DEFAULT_DOC package as a guide in developing your own ORDPLUGINS.ORDX_<format>_DOC media format package. This package sets the mimeType field in the setProperties( ) method with a MIME type value that is dependent on the file format.

```
CREATE OR REPLACE PACKAGE ORDX_DEFAULT_DOC
authid current_user
AS

PROCEDURE setProperties(ctx IN OUT RAW,
                        obj IN OUT NOCOPY ORDSYS.ORDDoc,
                        setComments IN NUMBER := 0);

END;
/
```

Table 7–2 shows the method supported in the ORDPLUGINS.ORDX_DEFAULT_DOC package and the exception raised if the source is null.

Table 7–2 Method Supported in the ORDPLUGINS.ORDX_DEFAULT_DOC Package

Name of Method	Level of Support
setProperties	Supported; if the source is local, process the local data and set the properties, but if the source is NULL, raise an ORDSYS.ORDSourceExceptions.EMPTY_SOURCE exception; if the source is a BFILE, then process the BFILE and set the properties; if the source is neither local nor a BFILE, get the media content into a temporary LOB, process the data, and set the properties.

7.4.2 Extending *inter*Media to Support a New Media Data Format

Extending *inter*Media to support a new media data format consists of four steps:

1. Design your new media data format.
2. Implement your new media data format and name it, for example, ORDX_MY_DOC.SQL.
3. Install your new ORDX_MY_DOC.SQL plug-in in the ORDPLUGINS schema.
4. Grant EXECUTE privileges on your new plug-in, for example, ORDX_MY_DOC.SQL plug-in, to PUBLIC.

Section 3.2.11 briefly describes how to extend *inter*Media to support a new media data format and describes the interface. A package body listing is provided in Example 7–1 to assist you in this operation. Add your variables to the places that say "--Your variables go here" and add your code to the places that say "--Your code goes here".

See Section F.2 for more information on installing your own media format plug-in and running the sample scripts provided.

Example 7–1 Show the Package Body for Extending Support to a New Media Data Format

```
CREATE OR REPLACE PACKAGE BODY ORDX_MY_DOC
AS
  --DOCUMENT ATTRIBUTES ACCESSORS
  PROCEDURE setProperties(ctx IN OUT RAW,
                          obj IN OUT NOCOPY ORDSYS.ORDDoc,
                          setComments IN NUMBER :=FALSE)
  IS
--Your variables go here
  BEGIN
--Your code goes here
  END;
END;
/
show errors;
```

8

Image Object Types Reference Information

Oracle *inter*Media contains the following information about the ORDImage type and the ORDImageSignature type:

- ORDImage Object type -- see Section 8.1.
- ORDImage Constructors -- see Section 8.1.1.
- ORDImage Methods -- see Section 8.1.2.
- ORDImageSignature Object type -- see Section 8.2.
- ORDImageSignature Constructor -- see Section 8.2.1.
- ORDImageSignature Methods -- see Section 8.2.2.
- ORDImageSignature Operators -- see Section 8.2.3.

The examples in this chapter assume that the test image table EMP has been created and filled with data. These tables were created using the SQL statements described in Section 8.1.3.

> **Note:** If you manipulate the image data itself (by either directly modifying the BLOB or changing the external source), then you must ensure that the object attributes stay synchronized and the update time is modified; otherwise, the object attributes will not match the image data.

Methods invoked at the ORDSource level that are handed off to the source plug-in for processing have ctx (RAW(4000)) as the first argument. Before calling any of these methods for the first time, the client must allocate the ctx structure, initialize it to NULL, and invoke the source.open( ) method. At this point, the source plug-in

can initialize the context for this client. When processing is complete, the client should invoke the source.close( ) method.

Methods invoked from a source plug-in call have the first argument as ctx (RAW(4000)).

Note: In the current release, not all source plug-ins will use the ctx argument, but if you code as previously described, your application should work with any current or future source plug-in.

8.1 ORDImage Object Types

Oracle *inter*Media describes the ORDImage object type, which supports the storage, management, and manipulation of image data and the ORDImageSignature object type, which supports content-based retrieval (image matching).

ORDImage Object Type

The ORDImage object type supports the storage and management of image data. This object type is defined as follows:

```
CREATE OR REPLACE TYPE ORDImage
AS OBJECT
(
  ------------------
  -- TYPE ATTRIBUTES
  ------------------
  source              ORDSource,
  height              INTEGER,
  width               INTEGER,
  contentLength       INTEGER,
  fileFormat          VARCHAR2(4000),
  contentFormat       VARCHAR2(4000),
  compressionFormat   VARCHAR2(4000),
  mimeType            VARCHAR2(4000),

---------------------
-- METHOD DECLARATION
---------------------
-- CONSTRUCTORS
--
STATIC FUNCTION init( ) RETURN ORDImage,
STATIC FUNCTION init(srcType     IN VARCHAR2,
                     srcLocation IN VARCHAR2,
                     srcName     IN VARCHAR2) RETURN ORDImage,

  -- Methods associated with copy operations
  MEMBER PROCEDURE copy(dest IN OUT ORDImage),

  -- Methods associated with image process operations
  MEMBER PROCEDURE process(command IN VARCHAR2),

  MEMBER PROCEDURE processCopy(command IN     VARCHAR2,
                               dest    IN OUT ORDImage),

  -- Methods associated with image property set and check operations
  MEMBER PROCEDURE setProperties,

  MEMBER PROCEDURE setProperties(description IN VARCHAR2),
```

```
MEMBER FUNCTION checkProperties RETURN BOOLEAN,

-- Methods associated with image attributes accessors
MEMBER FUNCTION getHeight RETURN INTEGER,
PRAGMA RESTRICT_REFERENCES(getHeight, WNDS, WNPS, RNDS, RNPS),

MEMBER FUNCTION getWidth RETURN INTEGER,
PRAGMA RESTRICT_REFERENCES(getWidth, WNDS, WNPS, RNDS, RNPS),

MEMBER FUNCTION getContentLength RETURN INTEGER,
PRAGMA RESTRICT_REFERENCES(getContentLength, WNDS, WNPS, RNDS, RNPS),

MEMBER FUNCTION getFileFormat RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getFileFormat, WNDS, WNPS, RNDS, RNPS),

MEMBER FUNCTION getContentFormat RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getContentFormat, WNDS, WNPS, RNDS, RNPS),

MEMBER FUNCTION getCompressionFormat RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getCompressionFormat, WNDS, WNPS, RNDS, RNPS),

-- Methods associated with the local attribute
MEMBER PROCEDURE  setLocal,
MEMBER PROCEDURE  clearLocal,
MEMBER FUNCTION  isLocal RETURN BOOLEAN,
PRAGMA RESTRICT_REFERENCES(isLocal, WNDS, WNPS, RNDS, RNPS),

-- Methods associated with the date attribute
MEMBER FUNCTION  getUpdateTime RETURN DATE,
PRAGMA RESTRICT_REFERENCES(getUpdateTime, WNDS, WNPS, RNDS, RNPS),
MEMBER PROCEDURE setUpdateTime(current_time DATE),

-- Methods associated with the mimeType attribute
MEMBER FUNCTION  getMimeType RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getMimeType, WNDS, WNPS, RNDS, RNPS),
MEMBER PROCEDURE setMimeType(mime IN VARCHAR2),

-- Methods associated with the source attribute
MEMBER FUNCTION  getContent RETURN BLOB,
PRAGMA RESTRICT_REFERENCES(getContent, WNDS, WNPS, RNDS, RNPS),

MEMBER FUNCTION getBFILE RETURN BFILE,
PRAGMA RESTRICT_REFERENCES(getBFILE, WNDS, WNPS, RNDS, RNPS),
```

```
  MEMBER PROCEDURE deleteContent,

  MEMBER PROCEDURE setSource(source_type     IN VARCHAR2,
                             source_location IN VARCHAR2,
                             source_name     IN VARCHAR2),
  MEMBER FUNCTION  getSource RETURN VARCHAR2,
  PRAGMA RESTRICT_REFERENCES(getSource, WNDS, WNPS, RNDS, RNPS),

  MEMBER FUNCTION  getSourceType RETURN VARCHAR2,
  PRAGMA RESTRICT_REFERENCES(getSourceType, WNDS, WNPS, RNDS, RNPS),

  MEMBER FUNCTION  getSourceLocation RETURN VARCHAR2,
  PRAGMA RESTRICT_REFERENCES(getSourceLocation, WNDS, WNPS, RNDS, RNPS),

  MEMBER FUNCTION  getSourceName RETURN VARCHAR2,
  PRAGMA RESTRICT_REFERENCES(getSourceName, WNDS, WNPS, RNDS, RNPS),

  MEMBER PROCEDURE import(ctx IN OUT RAW),
  MEMBER PROCEDURE importFrom(ctx         IN OUT RAW,
                              source_type     IN VARCHAR2,
                              source_location IN VARCHAR2,
                              source_name     IN VARCHAR2),
  MEMBER PROCEDURE export(ctx             IN OUT RAW,
                              source_type     IN VARCHAR2,
                              source_location IN VARCHAR2,
                              source_name     IN VARCHAR2),

);
```

where:

- source: the source of the stored image data.
- height: the height of the image in pixels.
- width: the width of the image in pixels.
- contentLength: the size of the *on-disk* image file in bytes.
- fileFormat: the file type or format in which the image data is stored (TIFF, JIFF, and so forth.).
- contentFormat: the type of image (monochrome and so forth).
- compressionFormat: the compression algorithm used on the image data.
- mimeType: the MIME type information.

8.1.1 Constructors

This section describes the constructor functions.

The *inter*Media constructor functions are as follows:

- init( ) for ORDImage
- init(srcType,srcLocation,srcName) for ORDImage

init( ) for ORDImage

Format

init( ) RETURN ORDImage;

Description

Allows for easy initialization of instances of the ORDImage object type.

Parameters

None.

Pragmas

None.

Exceptions

None.

Usage Notes

This static method initializes all the ORDImage attributes to NULL with the following exceptions:

- source.updateTime is set to SYSDATE
- source.local is set to 1 (local)
- source.localData is set to empty_blob

You should begin using the init( ) method as soon as possible to allow you to more easily initialize the ORDImage object type, especially if the ORDImage type evolves and attributes are added in a future release. INSERT statements left unchanged using the default constructor (which initializes each object attribute), will fail under these circumstances.

Examples

Initialize the ORDImage object attributes:

```
BEGIN
  INSERT INTO emp VALUES (ORDSYS.ORDImage.init());
```

```
END;
/
```

init(srcType,srcLocation,srcName) for ORDImage

Format

```
init(srcType     IN VARCHAR2,
   srcLocation IN VARCHAR2,
   srcName     IN VARCHAR2)
 RETURN ORDImage;
```

Description

Allows for easy initialization of instances of the ORDImage object type.

Parameters

srcType
The source type of the image data.

srcLocation
The source location of the image data.

srcName
The source name of the image data.

Pragmas

None.

Exceptions

None.

Usage Notes

This static method initializes all the ORDImage attributes to NULL with the following exceptions:

- source.updateTime is set to SYSDATE
- source.local is set to 0
- source.localData is set to empty_blob

- source.srcType is set to the input value
- source.srcLocation is set to the input value
- source.srcName is set to the input value

You should begin using the init( ) method as soon as possible to allow you to more easily initialize the ORDImage object type, especially if the ORDImage type evolves and attributes are added in a future release. INSERT statements left unchanged using the default constructor (which initializes each object attribute), will fail under these circumstances.

Examples

Initialize the ORDImage object attributes:

```
BEGIN
  INSERT INTO emp VALUES (ORDSYS.ORDImage.init('file','IMGDIR','image1.gif'));
END;
/
```

8.1.2 Methods

This section presents reference information on the Oracle *inter*Media methods used for image data manipulation. These methods are described in the following groupings:

ORDImage Methods Associated with copy Operations

- copy( ): creates a copy of an image in another ORDImage. See "copy( )" on page 8-16.

ORDImage Methods Associated with process Operations

- process( ): performs in-place image processing on an image stored in a BLOB. See "process( )" on page 8-29.
- processCopy( ): performs image processing while copying an image to another ORDImage BLOB data type. See "processCopy( )" on page 8-34.

ORDImage Methods Associated with properties set and check Operations

- setProperties: fills in the attribute fields of an image for native image formats. See "setProperties" on page 8-36.

- setProperties( ): fills in the attribute fields of an image and includes a description parameter for foreign image formats. See "setProperties( ) for Foreign Images" on page 8-38 for a description of what a foreign image is.
- checkProperties: verifies the stored image attributes match the actual image. See "checkProperties" on page 8-15.

ORDImage Methods Associated with image Attributes

- getHeight: returns the height of the image in pixels. See "getHeight" on page 8-22.
- getWidth: returns the width of the image in pixels. See "getWidth" on page 8-23.
- getContentLength: returns the size of the image in bytes. See "getContentLength" on page 8-20.
- getFileFormat: returns the file type of an image. See "getFileFormat" on page 8-21.
- getContentFormat: returns the format of the image. See "getContentFormat" on page 8-19.
- getCompressionFormat: returns the type of compression used on the image. See "getCompressionFormat" on page 8-18.

ORDImage Methods Associated with the local Attribute

- setLocal: sets a flag to indicate that the data is stored locally in a BLOB. See "setLocal( )" on page 5-34 for information.
- clearLocal: clears the flag to indicate that the data is stored externally. See "clearLocal( )" on page 5-5 for information.
- isLocal: returns TRUE if the data is stored locally in a BLOB or FALSE if the data is external. See "isLocal( )" on page 5-26 for information.

ORDImage Methods Associated with the date Attribute

- getUpdateTime: returns the time when the image object was last updated. See "getUpdateTime( )" on page 5-25 for information.
- setUpdateTime( ): sets the update time for the image object. This method is called implicitly by methods that modify natively supported images. See "setUpdateTime( )" on page 5-39 for information.

ORDImage Methods Associated with the mimeType Attribute

- getMimeType: returns the MIME type of the stored image data. See "getMimeType( )" on page 5-17 for information.
- setMimeType( ): sets the MIME type of the stored image data. This method is called implicitly by any method that modifies natively supported images. See "setMimeType( )" on page 5-35 for information.

ORDImage Methods Associated with the source Attribute

- processSourceCommand( ): sends a command and related arguments to the source plug-in. See "processSourceCommand( )" on page 5-29 for information.
- getContent: returns the content of the local data. See "getContent( )" on page 5-15 for information.
- getBFILE: returns the external content as a BFILE. See "getBFILE( )" on page 5-13 for information.
- deleteContent: deletes the content of the local data. See "deleteContent( )" on page 5-8 for information.
- setSource( ): sets the source information to where external image data is to be found. See "setSource( )" on page 5-37 for information.
- getSource: returns a string containing complete information about the external data source formatted as a URL. See "getSource( )" on page 5-19 for information.
- getSourceType: returns the external source type of the image data. See "getSourceType( )" on page 5-23 for information.
- getSourceLocation: returns the external source location of the image data. See "getSourceLocation( )" on page 5-21 for information.
- getSourceName: returns the external source name of the image data. See "getSourceName( )" on page 5-22 for information.
- import( ): transfers data from an external data source (specified by calling setSourceInformation( )) to the local source (localData) within an Oracle database, setting the value of the local attribute to "1", meaning local, and updating the timestamp and image attributes. See "import( )" on page 8-24.
- importFrom( ): transfers data from the specified external data source (source type, location, name) to the local source (localData) within an Oracle database, setting the value of the local attribute to "1", meaning local, and updating the timestamp and image attributes. See "importFrom( )" on page 8-26.

- export( ): copies data from a local source (localData) within an Oracle database to the specified external data source, setting source information to parameters supplied, and leaving all attributes unchanged. See "export( )" on page 5-9 for information.

> **Note:** The export( ) method natively supports only sources of source type file. User-defined sources may support the export( ) method.

ORDImage Methods Associated with File Operations

- openSource( ): opens a data source or a BLOB. See "openSource( )" on page 5-27 for information.
- closeSource( ): closes a data source or a BLOB. See "closeSource( )" on page 5-6 for information.
- trimSource( ): trims a data source or a BLOB. See "trimSource( )" on page 5-40 for information.
- readFromSource( ): reads a buffer of *n* bytes from a source beginning at a start position. See "readFromSource( )" on page 5-32 for information.
- writeToSource( ): writes a buffer of *n* bytes to a source beginning at a start position. See "writeToSource( )" on page 5-42 for information.

8.1.3 Example Table Definitions

The methods described in this chapter show examples based on a test image table EMP. Refer to the EMP table definition that follows when reading through the examples:

EMP Table Definition

```
CREATE TABLE emp (
  ename VARCHAR2(50),
  salary NUMBER,
  job VARCHAR2(50),
  department INTEGER,
  photo ORDSYS.ORDImage,
  large_photo ORDSYS.ORDImage);
DECLARE
  Image ORDSYS.ORDImage;
BEGIN
```

```
  INSERT INTO emp VALUES ('John Doe', 24000, 'Technical Writer', 123,
      ORDSYS.ORDImage.init('file','ORDIMGDIR','jdoe.gif'));
  INSERT INTO emp VALUES ('Jane Doe', 24000, 'Technical Writer', 456,
      ORDSYS.ORDImage.init('file','ORDIMGDIR','jadoe.gif'));
  SELECT large_photo INTO Image FROM emp
        WHERE ename = 'John Doe' FOR UPDATE;
  Image.setProperties;
  UPDATE emp SET large_photo = Image WHERE ename = 'John Doe';
  COMMIT;
  SELECT large_photo INTO Image FROM emp
        WHERE ename = 'Jane Doe' FOR UPDATE;
  Image.setProperties;
  UPDATE emp SET large_photo = Image WHERE ename = 'Jane Doe';
  COMMIT;
END;
/
```

checkProperties

Format

checkProperties RETURN BOOLEAN;

Description

Verifies that the properties stored in attributes of the image object match the properties of the image. This method should not be used for foreign images (those formats not natively supported by *inter*Media).

Parameters

None.

Usage Notes

Use this method to verify that the image attributes match the actual image.

Pragmas

None.

Exceptions

None.

Examples

Check the image attributes:

```
DECLARE
    Image ORDSYS.ORDImage;
    properties_match BOOLEAN;
BEGIN
    SELECT large_photo INTO Image FROM emp
        WHERE ename = 'John Doe' FOR UPDATE;
    -- check that properties match the image
    properties_match := Image.checkProperties();
    IF properties_match THEN
        DBMS_OUTPUT.PUT_LINE('Check Properties succeeded');
    END IF;
END;
```

copy( )

Format

copy(dest IN OUT ORDImage);

Description

Copies an image without changing it.

Parameters

dest
The destination of the new image.

Usage Notes

This method copies the image data, as is, including all source and image attributes, into the supplied local destination image.

If the data is stored locally in the source, then calling this method copies the BLOB to the destination source.localData attribute.

Calling this method copies the external source information to the external source information of the new image whether or not source data is stored locally.

Calling this method implicitly calls the setUpdateTime( ) method on the destination object to update its timestamp information.

Pragmas

None.

Exceptions

NULL_LOCAL_DATA

This exception is raised if you call the copy( ) method and the destination source.localData attribute is not initialized.

This exception is raised if you call the copy( ) method and the source.isLocal attribute value is 1 and the source.localData attribute value is NULL.

Examples

Create a copy of the image:

```
DECLARE
    Image_1 ORDSYS.ORDImage;
    Image_2 ORDSYS.ORDImage;
BEGIN
    SELECT photo, large_photo
       INTO Image_2, Image_1
       FROM emp
       WHERE ename = 'John Doe' FOR UPDATE;
    -- copy the data from Image_1 to Image_2
    Image_1.copy(Image_2);
    UPDATE emp SET photo = Image_2
        WHERE ename = 'John Doe';
END;
/
```

getCompressionFormat

Format

getCompressionFormat RETURN VARCHAR2;

Description

Returns the compression type of an image. This method does not actually read the image, it is a simple access method that returns the value of the compressionFormat attribute.

Parameters

None.

Usage Notes

Use this method rather than accessing the compressionFormat attribute directly to protect yourself from potential changes to the internal representation of the ORDImage object.

Pragmas

PRAGMA RESTRICT_REFERENCES(getCompressionFormat, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Examples

Get the compression type of an image:

```
DECLARE
    Image ORDSYS.ORDImage;
    CompressionFormat VARCHAR2(4000);
BEGIN
    SELECT large_photo INTO Image FROM emp
      WHERE ename = 'John Doe';
    -- get the image compression format
    CompressionFormat := Image.getCompressionFormat();
END;
```

getContentFormat

Format

getContentFormat RETURN VARCHAR2;

Description

Returns the content type of an image (such as monochrome). This method does not actually read the image; it is a simple access method that returns the value of the contentFormat attribute.

Parameters

None.

Usage Notes

Use this method rather than accessing the contentFormat attribute directly to protect yourself from potential changes to the internal representation of the ORDImage object.

Pragmas

PRAGMA RESTRICT_REFERENCES(getContentFormat, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Examples

Get the type of an image:

```
DECLARE
    Image ORDSYS.ORDImage;
    ContentFormat VARCHAR2(4000);
BEGIN
    SELECT large_photo INTO Image FROM emp
      WHERE ename = 'John Doe';
    -- get the image content format
    ContentFormat := Image.getContentFormat();
END;
```

getContentLength

Format

getContentLength RETURN INTEGER;

Description

Returns the length of the image data content stored in the source. This method does not actually read the image; it is a simple access method that returns the value of the content length attribute.

Parameters

None.

Usage Notes

Use this method rather than accessing the contentLength attribute directly to protect from potential future changes to the internal representation of the ORDImage object.

Pragmas

PRAGMA RESTRICT_REFERENCES(getContentLength, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Examples

Get the length of the image data content stored in the source:

```
DECLARE
    Image ORDSYS.ORDImage;
    ContentLength INTEGER;
BEGIN
    SELECT large_photo INTO Image FROM emp
      WHERE ename = 'John Doe';
    -- get the image size
    ContentLength := Image.getContentLength();
END;
```

getFileFormat

Format

getFileFormat RETURN VARCHAR2;

Description

Returns the file type of an image (such as TIFF or JFIF). This method does not actually read the image; it is a simple access method that returns the value of the fileFormat attribute.

Parameters

None.

Usage Notes

Use this method rather than accessing the fileFormat attribute directly to protect yourself from potential changes to the internal representation of the ORDImage object.

Pragmas

PRAGMA RESTRICT_REFERENCES(getFileFormat, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Examples

Get the file type of an image:

```
DECLARE
    Image ORDSYS.ORDImage;
    FileFormat VARCHAR2(4000);
BEGIN
    SELECT large_photo INTO Image FROM emp
      WHERE ename = 'John Doe';
    -- get the image file format
    FileFormat := Image.getFileFormat();
END;
```

getHeight

Format

getHeight RETURN INTEGER;

Description

Returns the height of an image in pixels. This method does not actually read the image; it is a simple access method that returns the value of the height attribute.

Parameters

None.

Usage Notes

Use this method rather than accessing the height attribute directly to protect yourself from potential changes to the internal representation of the ORDImage object.

Pragmas

PRAGMA RESTRICT_REFERENCES(getHeight, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Examples

Get the height of an image:

```
DECLARE
    Image ORDSYS.ORDImage;
    Height INTEGER;
BEGIN
    SELECT large_photo INTO Image FROM emp
      WHERE ename = 'John Doe';
    -- get the image height
    Height := Image.getHeight();
END;
/
```

getWidth

Format

getWidth RETURN INTEGER;

Description

Returns the width of an image in pixels. This method does not actually read the image; it is a simple access method that returns the value of the width attribute.

Parameters

None.

Usage Notes

Use this method rather than accessing the width attribute directly to protect yourself from potential changes to the internal representation of the ORDImage object.

Pragmas

PRAGMA RESTRICT_REFERENCES(getWidth, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Examples

Get the width of an image:

```
DECLARE
    Image ORDSYS.ORDImage;
    Width INTEGER;
BEGIN
    SELECT large_photo INTO Image FROM emp
      WHERE ename = 'John Doe';
    -- get the image width
    Width := Image.getWidth();
END;
/
```

import( )

Format

```
MEMBER PROCEDURE import(ctx IN OUT RAW);
```

Description

Transfers image data from an external image data source to a local source (localData) within an Oracle database.

Parameters

ctx
The source plug-in context information. This must be allocated. You must call the source.open( ) method; see the introduction to this chapter for more information.

Usage Notes

Use the setSource( ) method to set the external source type, location, and name prior to calling the import( ) method.

You must ensure that the directory exists or is created before you use this method for srcType 'file'.

After importing data from an external image data source to a local source (within an Oracle database), the source information remains unchanged (that is, pointing to the source from where the data was imported).

Invoking this method implicitly calls the setUpdateTime( ) and setLocal methods.

If the file format of the imported image is not previously set to a string beginning with "OTHER", the setProperties( ) method is also called. Set the file format to a string preceded by "OTHER" for foreign image formats; calling the setProperties( ) method for Foreign Images does this for you.

Pragmas

None.

Exceptions

ORDSourceExceptions.INCOMPLETE_SOURCE_INFORMATION

This exception is raised if you call the import( ) method and the value of srcType is NULL.

ORDSourceExceptions.NULL_SOURCE

This exception is raised if you call the import( ) method and the value of dlob is NULL.

ORDSourceExceptionsMETHOD_NOT_SUPPORTED

This exception is raised if you call the import( ) method and this method is not supported by the source plug-in being used.

See Appendix H for more information about these exceptions.

Examples

Import image data from an external image data source into the local source:

```
DECLARE
    Image ORDSYS.ORDImage;
    ctx RAW(4000) :=NULL;
BEGIN
    -- select the image to be imported
    SELECT large_photo INTO Image FROM emp
           WHERE ename = 'John Doe' FOR UPDATE;
    -- import the image into the database
    Image.import(ctx);
    -- update the image object
    UPDATE emp SET large_photo = Image WHERE ename = 'John Doe';
END;
/
```

importFrom( )

Format

```
importFrom(ctx                IN OUT RAW,
           source_type        IN VARCHAR2,
           source_location IN VARCHAR2,
           source_name        IN VARCHAR2);
```

Description

Transfers image data from the specified external image data source to a local source (localData) within an Oracle database.

Parameters

ctx
The source plug-in context information. This must be allocated. You must call the source.open( ) method; see the introduction to this chapter for more information.

source_type
The source type of the image data.

source_location
The location from where the image data is to be imported.

source_name
The name of the image data.

Usage Notes

This method is similar to the import( ) method except the source information is specified as parameters to the method instead of separately.

You must ensure that the directory exists or is created before you use this method for srcType 'file'.

After importing data from an external image data source to a local source (within an Oracle database), the source information (that is, pointing to the source from where the data was imported) is set to the input values.

Invoking this method implicitly calls the setUpdateTime( ) and setLocal methods.

If the file format of the imported image is not previously set to a string beginning with "OTHER", the setProperties( ) method is also called. Set the file format to a string preceded by "OTHER" for foreign image formats; calling the setProperties( ) method for Foreign Images does this for you.

Pragmas

None.

Exceptions

ORDSourceExceptions.NULL_SOURCE

This exception is raised if you call the importFrom( ) method and the value of dlob is NULL.

ORDSourceExceptions.METHOD_NOT_SUPPORTED

This exception is raised if you call the importFrom( ) method and this method is not supported by the source plug-in being used.

ORDSourceExceptions.SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the importFrom( ) method within a source plug-in when any other exception is raised.

See Appendix H for more information about these exceptions.

Examples

Import image data from the specified external data source into the local source:

```
DECLARE
    Image ORDSYS.ORDImage;
    ctx RAW(4000) :=NULL;
BEGIN
    -- select the image to be imported
    SELECT large_photo INTO Image FROM emp
           WHERE ename = 'John Doe' FOR UPDATE;
    -- import the image into the database
    Image.importFrom(ctx,
                     'file',
                     'ORDIMGDIR',
                     'jdoe.gif');
    -- update the image object
```

```
    UPDATE emp SET large_photo = Image WHERE ename = 'John Doe';
END;
/
```

process( )

Format

process(command IN VARCHAR2);

Description

Performs one or more image processing operations on a BLOB, writing the image back onto itself.

Parameters

command
A list of image processing operations to perform on the image.

Usage Notes

You can change one or more of the image attributes shown in Table 8–1. Table 8–2 shows additional changes that can be made only to raw pixel and foreign images.

Table 8–1 Image Processing Operators

Operator Name	Usage	Values
compressionFormat	Compression type / format; coerces output to specified compression format if supported by file format.	JPEG, SUNRLE, BMPRLE,TARGARLE, LZW, LZWHDIFF, FAX3, FAX4, HUFFMAN3, PACKBITS, GIFLZW, ASCII, RAW, DEFLATE, NONE
compressionQuality	Compression quality; determines quality of lossy compression; JPEG only.	MAXCOMPRATIO, MAXINTEGRITY, LOWCOMP, MEDCOMP, HIGHCOMP
contentFormat	Image type / pixel / data format MONOCHROME | nBIT[BIP | BIL | BSQ] {LUT[RGB | GRAY] | {[DRCT]RGB | GREY}}; coerces output to specified content format.	See Figure 8–1 for use and syntax flow of the following values: MONOCHROME, 1BIT, 2BIT, 4BIT, 8BIT, 12BIT, 16BIT, 24BIT, 32BIT, 48BIT, BIP, BIL, BSQ, LUT, DRCT, RGB, GRAY [SCALE], GREY [SCALE]
cut	Window to cut or crop (origin.x origin.y width height); first pixel in x or y is 0 (zero); must define a window inside image.	positive INTEGER INTEGER INTEGER INTEGER maximum value is 2147483648

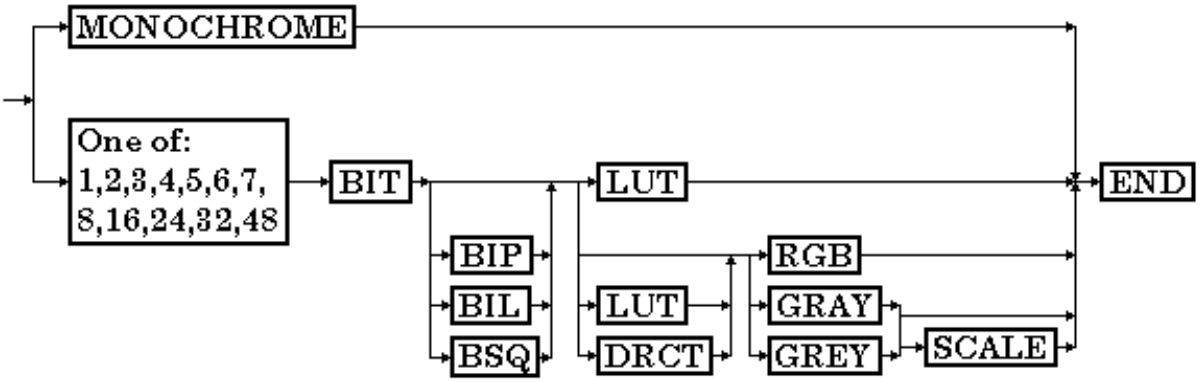
Table 8–1 Image Processing Operators(Cont.)

Operator Name	Usage	Values
fileFormat	File format of the image; coerces output to specified file format.	BMPF, CALS, GIFF, JFIF, PBMF, PGMF, PICT, PNGF, PNMF, PPMF, RASF, RPIX, TGAF, TIFF, WBMP
fixedScale	Scale to a specific size in pixels (width, height); may not be combined with other scale verbs.	positive INTEGER INTEGER
maxScale	Scale to a specific size in pixels, while maintaining the aspect ratio (maxWidth, maxHeight); may not be combined with other scale verbs.	positive INTEGER INTEGER
scale	Scale factor (for example, 0.5 or 2.0); uniformly scales image; may not be combined with other scale verbs.	<FLOAT> positive
xScale	X-axis scale factor (default is 1); non-uniformly scales image; may be combined only with the yScale operator; may not be combined with any other scale verbs.	<FLOAT> positive
yScale	Y-axis scale factor (default is 1); non-uniformly scales image; may only be combined with the xScale operator; may not be combined with any other scale verbs.	<FLOAT> positive

Table 8–2 Additional Image Processing Operators for Raw Pixel and Foreign Images

Operator Name	Usage	Values
channelOrder	Indicates the relative position of the red, green, and blue channels (bands) within the image; changes order of output channels. Only for RPIX.	RGB (default), RBG, GRB, GBR, BRG, BGR
inputChannels	For multiband images, specify either one (grayscale) or three integers indicating which channels to assign to red (first), green (second), and blue (third). Note that this parameter affects the source image, not the destination; RPIX only.	INTEGER or INTEGER INTEGER INTEGER
interleave (deprecated in release 9.0.1; functions moved to ContentFormat operator)	Controls band layout within the image: Band Interleaved by Pixel Band Interleaved by Line Band Sequential Coerces output to be BIP, BIL, or BSQ; RPIX only.	BIP (default), BIL, BSQ
pixelOrder	If NORMAL, then the leftmost pixel appears first in the image; coerces pixel direction. RPIX only.	NORMAL (default), REVERSE
scanlineOrder	If NORMAL, then the top scanline appears first in the image; coerces scanline direction. RPIX and BMPF only.	NORMAL (default), INVERSE
Dither	See Section D.4.6 for more information.	ERRORDIFFUSION, ORDEREDDITHER
Page	Selects a page from a multipage file; for use with TIFF only; first page is 0 (zero).	positive INTEGER
Tiled	No arguments; forces output image to be tiled; for use with TIFF only.	

Figure 8–1 Use and Syntax Flow Diagram for the contentFormat Operator Values



Note: When specifying values that include floating-point numbers, you must use double quotation marks ("") around the value. If you do not, the wrong values may be passed and you will get incorrect results.

There is no implicit import( ) or importFrom( ) call performed when you call this method; if data is external, you must first call import( ) or importFrom( ) to make the data local before you can process it.

Implicit setProperties( ), setUpdateTime( ), and setMimeType( ) methods are done after the process( ) method is called.

See Appendix D for more information on process( ) method operators.

Pragmas

None.

Exceptions

DATA_NOT_LOCAL

This exception is raised if you call the process( ) method and the data is not local or the source.localData attribute is not initialized.

Examples

Example 1: Change the file format of image1 to GIF:

```
image1.process('fileFormat=GIFF');
```

Example 2: Change image1 to use lower quality JPEG compression and double the length of the image along the X-axis:

```
image1.process('compressionFormat=JPEG, compressionQuality=MAXCOMPRATIO,
xScale="2.0"');
```

Note that changing the length on only one axis (for example, xScale=2.0) does not affect the length on the other axis, and would result in image distortion. Also, only the xScale and yScale parameters can be combined in a single operation. Any other combinations of scale operators result in an error.

The maxScale and fixedScale operators are especially useful for creating thumbnail images from various-sized originals. The following line creates at most a 32-by-32 pixel thumbnail image, preserving the original aspect ratio:

```
image1.process('maxScale=32 32');
```

Example 3: Convert the image to TIFF:

```
DECLARE
    Image ORDSYS.ORDImage;
BEGIN
    SELECT photo INTO Image FROM emp
        WHERE ename = 'John Doe' FOR UPDATE;
        Image.process('fileFormat=TIFF');
    UPDATE emp SET photo = Image WHERE ename = 'John Doe';
END;
/
```

Example 4: Change the content format to 8BIT, BIP pixel layout, LUT interpretation, and RGB color space:

```
DECLARE
    Image ORDSYS.ORDImage;
BEGIN
    SELECT photo INTO Image FROM emp
        WHERE ename = 'John Doe' FOR UPDATE;
        Image.process('fileFormat=TIFF','contentFormat=8BITBIPLUTRGB');
    UPDATE emp SET photo = Image WHERE ename = 'John Doe';
END;
/
```

processCopy( )

Format

```
processCopy(command IN     VARCHAR2,
            dest    IN OUT ORDImage);
```

Description

Copies an image stored internally or externally to another image stored internally in a BLOB.

Parameters

command
A list of image processing changes to make for the image in the new copy.

dest
The destination of the new image.

Usage Notes

See Table 8–1, "Image Processing Operators" and Table 8–2, "Additional Image Processing Operators for Raw Pixel and Foreign Images".

You cannot specify the same BLOB as both the source and destination.

Calling this method processes the image into the destination BLOB from any source (local or external).

Implicit setProperties( ), setUpdateTime( ), and setMimeType( ) methods are done on the destination image after the processCopy( ) method is called.

See Appendix D for more information on processCopy operators.

Pragmas

None.

Exceptions

NULL_DESTINATION

This exception is raised if you call the processCopy( ) method and the value of dest is NULL.

DATA_NOT_LOCAL

This exception is raised if you call the processCopy( ) method and the dest.source.isLocal attribute value is FALSE, (the destination image must be local).

NULL_LOCAL_DATA

This exception is raised if you call the processCopy( ) method and the dest.source.localData attribute value is NULL (destination image must be initialized).

This exception is raised if you call the processCopy( ) method and the source.isLocal attribute value is 1 and the source.localData attribute value is NULL.

Examples

Copy an image, changing the file format, compression format, and data format in the destination image:

```
DECLARE
    Image_1 ORDSYS.ORDImage;
    Image_2 ORDSYS.ORDImage;
    mycommand VARCHAR2(400);
BEGIN
    SELECT photo, large_photo
        INTO Image_2, Image_1
        FROM emp
        WHERE ename = 'John Doe' FOR UPDATE;
    mycommand := 'fileFormat=tiff compressionFormat=packbits
                 contentFormat = 8bitlut';
    Image_1.processCopy(mycommand, Image_2);
    UPDATE emp SET photo = Image_2 WHERE ename = 'John Doe';
END;
/
```

setProperties

Format

setProperties;

Description

Reads the image data to get the values of the object attributes, then stores them into the appropriate attribute fields. The image data can be stored in the database in a BLOB, or externally in a BFILE or URL. If the data is stored externally in anything other than a BFILE, the data is read into a temporary BLOB in order to determine the image characteristics.

This method should not be called for foreign images. Use the setProperties(description) method for foreign images.

Parameters

None.

Usage Notes

After you have copied, stored, or processed a native format image, call this method to set the current characteristics of the new content, except when this method is called implicitly.

This method sets the following information about an image:

- Height in pixels
- Width in pixels
- Data size of the on-disk image in bytes
- File type (TIFF, JFIF, and so forth)
- Image type (monochrome and so forth)
- Compression type (JPEG, LZW, and so forth)
- MIME type (generated based on file format)

Calling this method implicitly calls the setUpdateTime( ) and the setMimeType( ) methods.

Pragmas

None.

Exceptions

NULL_LOCAL_DATA

This exception is raised if you call the setProperties( ) method and the source.isLocal attribute value is 1 and the source.localData attribute value is NULL.

Examples

Select the image, and then set the attributes using the setProperties method:

```
DECLARE
   Image ORDSYS.ORDImage;
BEGIN
   INSERT INTO emp VALUES ('John Doe', 24000, 'Technical Writer', 123,
      ORDSYS.ORDImage.init('file','ORDIMGDIR','jdoe.gif');
   -- select the newly inserted row for update
   SELECT large_photo INTO Image FROM emp
          WHERE ename = 'John Doe' FOR UPDATE;
   -- set property attributes for the image data
   Image.setProperties();
   DBMS_OUTPUT.PUT_LINE('image width = ' || image.getWidth());
   DBMS_OUTPUT.PUT_LINE('image height = ' || image.getHeight());
   DBMS_OUTPUT.PUT_LINE('image size = ' || image.getContentLength());
   DBMS_OUTPUT.PUT_LINE('image file type = ' || image.getFileFormat());
   DBMS_OUTPUT.PUT_LINE('image type = ' || image.getContentFormat());
   DBMS_OUTPUT.PUT_LINE('image compression = ' || image.getCompressionFormat());
   DBMS_OUTPUT.PUT_LINE('image mime type = ' || image.getMimeType());
   UPDATE emp SET large_photo = Image WHERE ename = 'John Doe';
 END;
/
```

Example output:

```
image width = 360
image height = 490
image size = 66318
image file type = JFIF
image type = 24BITRGB
image compression = JPEG
image mime type = image/jpeg
```

setProperties( ) for Foreign Images

Format

setProperties(description IN VARCHAR2);

Description

Allows you to write the characteristics of a foreign image into the appropriate attribute fields.

Parameters

description
Specifies the image characteristics to set for the foreign image.

Usage Notes

Note: Once you have set the properties for a foreign image, it is up to you to keep the properties consistent. If *inter*Media detects an unknown file format, it will not implicitly set the properties.

After you have copied, stored, or processed a foreign image, call this method to set the characteristics of the new image content. Unlike the native image types described in Appendix E, foreign images either do not contain information on how to interpret the bits in the file or, *inter*Media does not understand the information. In this case, you must set the information explicitly.

You can set the following image characteristics for foreign images, as shown in Table 8–3.

Table 8–3 **Image Characteristics for Foreign Files**

Field	Data Type	Description
CompressionFormat	STRING	Value must be CCITG3, CCITG4, or NONE (default).
DataOffset	INTEGER	The offset allows the image to have a header that *inter*Media does not try to interpret. Set the offset to avoid any potential header. The value must be a positive integer less than the LOB length. Default is zero.
DefaultChannelSelection	INTEGER	For multiband images, specify either one (grayscale) or three integers indicating which channels to assign to red (first), green (second), and blue (third). For example, DefaultChannelSelection = 1 for single-band images and DefaultChannelSelection = 1, 2, 3 for triple-band images.
Height	INTEGER	Height of the image in pixels. Value must be a positive integer. There is no default, thus a value must be specified.
Interleaving	STRING	Band layout within the image. Valid styles are: ■ BIP (default) Band Interleaved by Pixel ■ BIL Band Interleaved by Line ■ BSQ Band Sequential
NumberOfBands	INTEGER	Value must be a positive integer less than 255 describing the number of color bands in the image. Default is 3.
PixelOrder	STRING	If NORMAL (default), the leftmost pixel appears first in the file. If REVERSE, the rightmost pixel appears first.
ScanlineOrder	STRING	If NORMAL (default), the top scanline appears first in the file. If INVERSE, then the bottom scanline appears first.
UserString	STRING	A 4-character descriptive string. If used, the string is stored in the fileFormat field, appended to the file format ("OTHER:"). Default is blank and fileFormat is set to "OTHER".
Width	INTEGER	Width of the image in pixels. Value must be a positive integer. There is no default, thus a value must be specified.
MimeType	STRING	Value must be a MIME type, such as img/gif.

The values supplied to setProperties( ) are written to the existing ORDImage data attributes. The fileFormat is set to "OTHER" and includes the user string, if supplied; for example, 'OTHER: LANDSAT'.

Pragmas

None.

Exceptions

NULL_PROPERTIES_DESCRIPTION

This exception is raised if you call the setProperties( ) method for Foreign Images and the description attribute value is NULL.

Examples

Select the foreign image and then set the properties for the image:

```
DECLARE
    Image ORDSYS.ORDImage;
BEGIN
    SELECT large_photo INTO Image FROM emp
           WHERE ename = 'John Doe' FOR UPDATE;
    -- set property attributes for the image data
    Image.setProperties('width=123 height=321 compressionFormat=NONE' ||
                        ' userString=DJM dataOffset=128' ||
                        ' scanlineOrder=INVERSE pixelOrder=REVERSE' ||
                        ' interleaving=BIL numberOfBands=1' ||
                        ' defaultChannelSelection=1');
    UPDATE emp SET large_photo = Image WHERE ename = 'John Doe';
END;
/
```

8.2 ORDImageSignature Object Type

Oracle *inter*Media describes the ORDImageSignature object type, which supports content-based retrieval (image matching).

The examples in this section assume that a table called stockphotos has been created and filled with some photographic images. The table was created using the following SQL statement:

```
CREATE TABLE stockphotos (photo_id NUMBER,
                          photographer VARCHAR2(64),
                          annotation VARCHAR2(255),
                          photo ORDSYS.ORDImage,
                          photo_sig ORDSYS.ORDImageSignature);
```

When you are storing or copying images, you must first create an empty BLOB in the table. The following method invocation creates an empty ORDImageSignature object:

```
ORDSYS.ORDImageSignature.init();
```

ORDImageSignature Object Type

The ORDImageSignature object type supports content-based retrieval or image matching. This object type is defined as follows:

```
CREATE OR REPLACE TYPE ORDImageSignature
AS OBJECT
(
  -- Signature of the image.  Contains color, texture
  -- and shape information of the image.  It is stored
  -- in a BLOB.

  signature BLOB,

-----------------------
-- METHOD DECLARATION

-- Makes the callout

STATIC FUNCTION init RETURN ORDImageSignature,

STATIC FUNCTION evaluateScore(sig1    IN ORDImageSignature,
                              sig2    IN ORDImageSignature,
                              weights IN VARCHAR2)
  RETURN FLOAT,

STATIC FUNCTION isSimilar(sig1      IN ORDImageSignature,
                          sig2      IN ORDImageSignature,
                          weights   IN VARCHAR2,
                          threshold IN FLOAT)
  RETURN INTEGER,

MEMBER PROCEDURE generateSignature(image  IN ORDImage)
);
```

where:

- signature: holds the signature of the stored image data.

8.2.1 Constructors

This section describes the constructor functions.

The *inter*Media constructor functions are as follows:

- init( ) for ORDImageSignature

init( ) for ORDImageSignature

Format

init( ) RETURN ORDImageSignature;

Description

Allows for easy initialization of instances of the ORDImageSignature object type.

Parameters

None.

Pragmas

None.

Exceptions

None.

Usage Notes

This static method initializes the ORDImageSignature signature attribute to empty_blob.

You should always use the init( ) method to initialize the ORDImageSignature object type, which will be especially useful if the ORDImageSignature type evolves and attributes are added in a future release.

Examples

Initialize the ORDImageSignature object attribute:

```
BEGIN
INSERT INTO stockphotos VALUES (
5,Al MacFarlane,'red plaid',
ORDSYS.ORDImage.init('file','ORDIMGDIR','macfarlane.gif'),
ORDSYS.ORDImageSignature.init( ));
END;
/
```

8.2.2 Methods

This section presents reference information on the Oracle *inter*Media methods used for image data manipulation. These methods are described in the following groupings:

ORDImage Signature Methods Associated with Signature Operations

- evaluateScore( ): evaluates the distance between two input signatures based on the influence of specified attributes in the weights parameter.
- isSimilar( ): computes the distance between two input signatures based on the influence of specified attributes in the weights parameter and the specified threshold value.
- generateSignature( ): generates a signature for the specified ORDImage object.

evaluateScore( )

Format

```
evaluateScore(
        sig1 IN ORDImageSignature,
        sig2 IN ORDImageSignature,
        weights IN VARCHAR2)
RETURN FLOAT;
```

Description

A static method of the ORDImageSignature object type that evaluates the distance between two input signatures based on the influence of the specified attributes in the weights parameter.

Parameters

sig1
Signature object.

sig2
Signature object.

weights
A string consisting of matching attribute names followed by values. The matching attributes refer to the weights assigned by the user to the different attributes that influences the kind of match. The string can have all or some of the following attributes. Attributes not specified by the user have a default value of 0. At least one of the attributes, color, texture, and shape, must have a value greater than 0.

color: The importance of the feature color. It is a value between 0.0 and 1.0. DEFAULT: 0

texture: The importance of the feature texture. It is a value between 0.0 and 1.0. DEFAULT: 0

shape: The importance of the feature shape. It is a value between 0.0 and 1.0. DEFAULT: 0

location: The importance of the location of the regions in the image. It is a value between 0.0 and 1.0. DEFAULT: 0. Location weight string cannot be specified alone, it must be used with another weight string.

Usage Notes

None.

Pragmas

None.

Exceptions

None.

Examples

Compare two signatures and evaluate the score between them:

```
DECLARE
    t_image     ORDSYS.ORDImage;
    image_sig   ORDSYS.ORDImageSignature;
    compare_sig ORDSYS.ORDImageSignature;
BEGIN
    SELECT photo, photo_sig INTO t_image, image_sig FROM stockphotos
      WHERE photo_id=1 FOR UPDATE;
    -- evaluate the signature of two images
    ORDSYS.ORDImageSignature.evaluateScore(image_sig,compare_sig,
             'color=1.0,texture=0,shape=0,location=0');
    UPDATE stockphotos SET photo = t_image WHERE photo_id=1;
END;
/
```

generateSignature( )

Format

generateSignature (image IN ORDImage);

Description

Generates a signature for a given input image that is passed back as the signature object.

Parameters

image
The image object whose signature is to be generated.

Usage Notes

None.

Pragmas

None.

Exceptions

None.

Examples

Generate a signature for an image object:

```
DECLARE
    t_image   ORDSYS.ORDImage;
    image_sig ORDSYS.ORDImageSignature;
BEGIN
    SELECT photo, photo_sig INTO t_image, image_sig FROM stockphotos
       WHERE photo_id=1 FOR UPDATE;
    -- generate a signature
    image_sig.generateSignature(t_image);
    UPDATE stockphotos SET photo_sig = image_sig WHERE photo_id =1;
END;
/
```

isSimilar( )

Format

```
isSimilar(
                sig1 IN ORDImageSignature,
                sig2 IN ORDImageSignature,
                weights IN VARCHAR2,
                threshold IN FLOAT)
RETURN INTEGER;
```

Description

A static method of the ORDImageSignature object type that compares two signatures and computes the distance between them based on the influence of the specified attributes in the weights parameter and the specified threshold value. If the distance is less than the specified threshold, a value of 1 is returned, otherwise a value of 0 is returned.

Parameters

sig1
Signature object.

sig2
Signature object.

weights
A string consisting of matching attribute names followed by values. The matching attributes refer to the weights assigned by the user to the different attributes that influences the kind of match. The string can have all or some of the following attributes. Attributes not specified by the user have a default value of 0. At least one of the attributes, color, texture, or shape, must have a value greater than 0.

color: The importance of the feature color. It is a value between 0.0 and 1.0. DEFAULT: 0

texture: The importance of the feature texture. It is a value between 0.0 and 1.0. DEFAULT: 0

shape: The importance of the feature shape. It is a value between 0.0 and 1.0. DEFAULT: 0

location: The importance of the location of the regions in the image. It is a value between 0.0 and 1.0. DEFAULT: 0. This attribute must be specified with one other attribute; it cannot be specified by itself.

threshold
The degree of the match that the user desires. For example, if the value is specified as 10, then only those images whose signatures are a distance of 10 or less (score of 10 or less) from the query signature will be returned. The value of the threshold ranges from 0 to 100, which is the range of the distance.

Usage Notes

You can use this method to compare two signatures not stored in the database or when you must perform a comparison within a PL/SQL construct.

Pragmas

None.

Exceptions

None.

Examples

Compute the distance between two signatures:

```
DECLARE
    t_image   ORDSYS.ORDImage;
    image_sig ORDSYS.ORDImageSignature;
    imagesig2 ORDSYS.ORDImageSignature;
BEGIN
    SELECT photo, photo_sig INTO t_image, image_sig FROM stockphotos
      WHERE photo_id = 1 FOR UPDATE;
    -- compute the distance between two signatures
ORDSYS.ORDImageSignature.isSimilar(image_sig,imagesig2,'color=1.0,texture=0,shape=0,location=0',10);
    UPDATE stockphotos SET photo = t_image WHERE photo_id = 1;
END;
/
```

8.2.3 ORDImageSignature Operators

The following ORDImageSignature operators are schema level operators and do not reside within a package. These operators use the domain index, if it exists.

- ORDSYS.IMGSimilar( ): Compares the signature of a query image with the signatures of images stored in a table and determines whether or not the images match, based on the weights and threshold. Returns 1 if the computed distance measure (weighted average) is less than or equal to the threshold value, and returns 0 when the distance between the two images is more than the threshold.
- ORDSYS.IMGScore( ): IMGScore( ) is an ancillary operator to IMGSimilar( ) and returns the score of similarity value computed by the primary operator, IMGSimilar( ).

IMGSimilar Operator

Format

```
...IMGSimilar (ORDSYS.ORDImageSignature,
               ORDSYS.ORDImageSignature,
               VARCHAR2,
               FLOAT
               [ ,referencetoScore IN NUMBER] )...;
```

Description

Determines whether or not two images match. Specifically, the operator compares the signatures of two images, computes a weighted sum of the distance between the two images using user-supplied weight values for the visual attributes, compares the weighted sum with the threshold value, and returns the integer value 1 if the weighted sum is less than or equal to the threshold value. Otherwise, the operator returns 0.

Parameters

ORDSYS.ORDImageSignature (signature)
The signature of the image to which you are comparing the query image. Data type is ORDImageSignature. To use the domain index for the comparison, this first parameter must be the signature column on which the domain index has been created. Otherwise, Oracle9*i* uses the non-indexed implementation of query evaluation.

ORDSYS.ORDImageSignature (query signature)
The signature of the query or test image. Data type is ORDImageSignature.

VARCHAR2 (weightstring)
A list of weights to apply to each visual attribute. Data type is VARCHAR2. The following attributes can be specified, with a value of 0.0 specifying no importance

and a value of 1.0 specifying highest importance. You must specify a value greater than zero for at least one of the attributes, not including location.

Attribute	Description
color	The weight value (0.0 to 1.0) assigned to the color visual attribute. Data type is number. Default is 0.0.
texture	The weight value (0.0 to 1.0) assigned to the texture visual attribute. Data type is number. Default is 0.0.
shape	The weight value (0.0 to 1.0) assigned to the shape visual attribute. Data type is number. Default is 0.0.
location	The weight value (0.0 to 1.0) assigned to the location visual attribute. Data type is number. Default is 0.0. This attribute must be specified with one other attribute; it cannot be specified by itself.

Note: When specifying parameter values that include floating-point numbers, you should use double quotation marks (" ") around the value. If you do not, this may result in incorrect values being passed, and you will get incorrect results.

FLOAT (threshold)
The threshold value with which the weighted sum of the distances is to be compared. If the weighted sum is less than or equal to the threshold value, the images are considered to match. The range of this parameter is from 0.0 to 100.0.

referencetoScore
An optional parameter used when ancillary data (score of similarity) is required elsewhere in the query. Set this parameter to the same value here as used in the IMGScore( ) operator. Data type is NUMBER.

Return Value

Returns an integer value of 0 (not similar) or 1 (match).

Pragmas

None.

Exceptions

None.

Usage Notes

Before the IMGSimilar operator can be used, the image signatures must be created with the generateSignature( ) method. Also, to use the domain index, the index of type ORDImageIndex must have already been created. See Section 2.4 for information on creating and using the index and see Section 2.5 for additional performance tips.

The IMGSimilar( ) operator returns Boolean values to indicate whether two images match (true, if their image matching score is below the threshold). If you want to know the score value itself, you can use the IMGScore( ) operator in conjunction with the IMGSimilar( ) operator to retrieve the score computed in the IMGSimilar( ) operator.

The IMGSimilar( ) operator is useful when the application needs a simple Yes or No for whether or not two images match. The IMGScore( ) operator is useful when an application wants to make finer distinctions about matching or to perform special processing based on the degree of similarity between images.

The weights supplied for the four visual attributes are normalized prior to processing such that they add up to 1.0.

You must specify at least one of the three image attributes color, texture, or shape in the weightstring.

Examples

Using the IMG index, find all images similar to the query image using a threshold value of 25 and the following weights for the visual attributes:

- Color: 0.2
- Texture: 0.1
- Shape: 0.5
- Location: 0.2

This example assumes you already used the generateSignature( ) method to generate a signature for the query image. If an index exists on the signature column, it will be used automatically. See the IMGScore( ) operator for an example that uses the referenceToScore parameter.

```
DECLARE
   t_img           ORDSYS.ORDImage;
   i               INTEGER;
   image_sig       ORDSYS.ORDImageSignature;
   query_signature ORDSYS.ORDImageSignature;
```

```
BEGIN
   SELECT photo_id, photo, photo_sig
       INTO i, t_img, image_sig FROM stockphotos WHERE
       ORDSYS.IMGSimilar(photo_sig, query_signature,
       'color="0.2" texture="0.1" shape="0.5" location="0.2"', 25) = 1;
END;
/
```

IMGScore Operator

Format

...IMGScore (NUMBER)...;

Description

Compares the signatures of two images and returns a number representing the weighted sum of the distances for the visual attributes. IMGScore( ) is an ancillary operator, used only in conjunction with the primary operator, IMGSimilar( ) to retrieve the score computed in the IMGSimilar( ) operator. Each IMGScore( ) and IMGSimilar( ) operator shares the same reference number.

Parameters

NUMBER (referencetoSimilar)
Identifier to an IMGSimilar( ) operator. This identifier indicates that the image matching score value returned by the IMGScore( ) operator is the same one used in the corresponding IMGSimilar( ) operator. This parameter can also be used to maintain references for multiple invocations of the IMGSimilar( ) operator. Data type is NUMBER.

Return Value

This function returns a FLOAT value between 0.0 and 100.0, where 0.0 means the images are identical and 100.0 means the images are completely different.

Pragmas

None.

Exceptions

None.

Usage Notes

Before the IMGScore( ) operator can be used, the image signatures must be created with the generateSignature( ) method. Also, if you want the comparison to use the domain index, the index of type ORDImageIndex must have already been created.

See Section 2.4 for information on creating and using the index, and see Section 2.5 for additional performance tips.

The IMGScore( ) operator can be useful when an application wants to make finer distinctions about matching than the simple Yes or No returned by IMGSimilar( ). For example, using the score returned by IMGScore( ), the application might assign each image being compared to one of several categories, such as Definite Matches, Probable Matches, Possible Matches, and Nonmatches. The IMGScore( ) operator can also be useful if the application needs to perform special processing based on the degree of similarity between images.

Examples

Example 1

Find the weighted sum of the distances between a test image and the other images in the stockphotos table, using a threshold of 50 and the following weights for the visual attributes:

- Color: 0.2
- Texture: 0.1
- Shape: 0.5
- Location: 0.2

This example assumes that the signatures were already created using the generateSignature( ) method and they are stored in the database. Notice that both IMGScore( ) and IMGSimilar( ) are using 123 as the reference number in this example.

```
DECLARE
   img_score       NUMBER;
   i               INTEGER;
   query_signature ORDSYS.ORDImageSignature;
   image_sig       ORDSYS.ORDImageSignature;
   t_img           ORDSYS.ORDImage;

BEGIN
SELECT photo_id, ORDSYS.IMGScore(123), photo, photo_sig
     INTO i, img_score, t_img, image_sig FROM stockphotos
WHERE
     ORDSYS.IMGSimilar(image_sig, query_signature,
               'color="0.2"  texture="0.1" shape="0.5" location="0.2"', 50, 123) = 1
END;
/
```

The following shows possible results from this example. The first image has the lowest score, and therefore is the best match of the test image. Changing the weights used in the scoring would lead to different results.

```
ENAME      SCORE
---------- ------------
1          10.5
  1 row selected.
```

Example 2

The following example demonstrates the use of reference numbers to refer to scores evaluated in different IMGSimilar calls. In this example, a query is searching for a stock image in the stockphotos table that is similar in color to query image 1 and similar in shape and location to query image 2.

```
DECLARE
img_score NUMBER;
i INTEGER;
query_sig1 ORDSYS.ORDImageSignature;
query_sig2 ORDSYS.ORDImageSignature;
image1_sig ORDSYS.ORDImageSignature;
t_img ORDSYS.ORDImage;
BEGIN
SELECT photo_id, ORDSYS.IMGScore(1), ORDSYS.IMGScore(2), photo,
photo_sig
INTO i, img_score, t_img, image_sig FROM stockphotos
WHERE
ORDSYS.IMGSimilar(image_sig, query_sig1,
'color="1.0"', 50, 1) = 1
AND
ORDSYS.IMGSimilar(image_sig, query_sig2,
'shape="0.5" location="0.2"', 50, 2) = 1
END;
/
```

9

ORDVideo Reference Information

Oracle *inter*Media contains the following information about the ORDVideo type:

- Object type -- see Section 9.1.
- Constructors -- see Section 9.2.
- Methods -- see Section 9.3.
- Packages or PL/SQL plug-ins -- see Section 9.4.

The examples in this chapter assume that the test video table TVID has been created and filled with data. This table was created using the SQL statements described in Section 9.3.1.

> **Note:** If you manipulate the video data itself (by either directly modifying the BLOB or changing the external source), then you must ensure that the object attributes stay synchronized and the update time is modified; otherwise, the object attributes will not match the video data.

Methods invoked at the ORDSource level that are handed off to the source plug-in for processing have ctx (RAW(4000)) as the first argument. Before calling any of these methods for the first time, the client must allocate the ctx structure, initialize it to NULL, and invoke the openSource( ) method. At this point, the source plug-in can initialize context for this client. When processing is complete, the client should invoke the closeSource( ) method.

Methods invoked from a source plug-in call have the first argument as ctx (RAW(4000)).

Methods invoked at the ORDVideo level that are handed off to the format plug-in for processing have ctx (RAW(4000)) as the first argument. Before calling any of these methods for the first time, the client must allocate the ctx structure and initialize it to NULL.

Note: In the current release, not all source or format plug-ins will use the ctx argument, but if you code as previously described, your application should work with any current or future source or format plug-in.

You should use any of the individual set methods to set the value of the attribute for an object for formats not natively supported; otherwise, for formats natively supported, use the setProperties( ) method to populate the attributes of the object.

9.1 Object Types

Oracle *inter*Media describes the ORDVideo object type, which supports the storage and management of video data.

ORDVideo Object Type

The ORDVideo object type supports the storage and management of video data. This object type is defined as follows:

```
CREATE OR REPLACE TYPE ORDVideo
AS OBJECT
(
  -- ATTRIBUTES
description         VARCHAR2(4000),
source              ORDSource,
format              VARCHAR2(31),
mimeType            VARCHAR2(4000),
comments            CLOB,
  -- VIDEO RELATED ATTRIBUTES
width               INTEGER,
height              INTEGER,
frameResolution     INTEGER,
frameRate           INTEGER,
videoDuration       INTEGER,
numberOfFrames      INTEGER,
compressionType     VARCHAR2(4000),
numberOfColors      INTEGER,
bitRate             INTEGER,

  -- METHODS
-- CONSTRUCTORS
--
STATIC FUNCTION init( ) RETURN ORDVideo,
STATIC FUNCTION init(srcType     IN VARCHAR2,
                     srcLocation IN VARCHAR2,
                     srcName     IN VARCHAR2) RETURN ORDVideo,
-- Methods associated with the date attribute
MEMBER FUNCTION getUpdateTime RETURN DATE,
PRAGMA RESTRICT_REFERENCES(getUpdateTime, WNDS, WNPS, RNDS, RNPS),
MEMBER PROCEDURE setUpdateTime(current_time DATE),
-- Methods associated with the description attribute
MEMBER PROCEDURE setDescription(user_description IN VARCHAR2),
MEMBER FUNCTION getDescription RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getDescription, WNDS, WNPS, RNDS, RNPS),

-- Methods associated with the mimeType attribute
MEMBER PROCEDURE setMimeType(mime IN VARCHAR2),
```

```
MEMBER FUNCTION getMimeType RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getMimeType, WNDS, WNPS, RNDS, RNPS),

-- Methods associated with the source attribute
MEMBER FUNCTION processSourceCommand(
                                    ctx       IN OUT RAW,
                                    cmd       IN VARCHAR2,
                                    arguments IN VARCHAR2,
                                    result    OUT RAW)
                RETURN RAW,

MEMBER FUNCTION  isLocal RETURN BOOLEAN,
PRAGMA RESTRICT_REFERENCES(isLocal, WNDS, WNPS, RNDS, RNPS),

MEMBER PROCEDURE  setLocal,
MEMBER PROCEDURE  clearLocal,

MEMBER PROCEDURE setSource(
                           source_type     IN VARCHAR2,
                           source_location IN VARCHAR2,
                           source_name     IN VARCHAR2),
MEMBER FUNCTION getSource RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getSource, WNDS, WNPS, RNDS, RNPS),

MEMBER FUNCTION getSourceType RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getSourceType, WNDS, WNPS, RNDS, RNPS),

MEMBER FUNCTION getSourceLocation RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getSourceLocation, WNDS, WNPS, RNDS, RNPS),

MEMBER FUNCTION getSourceName RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getSourceName, WNDS, WNPS, RNDS, RNPS),

MEMBER PROCEDURE import(ctx IN OUT RAW),
MEMBER PROCEDURE importFrom(
                           ctx             IN OUT RAW,
                           source_type     IN VARCHAR2,
                           source_location IN VARCHAR2,
                           source_name     IN VARCHAR2),
MEMBER PROCEDURE export(
                       ctx             IN OUT RAW,
                       source_type     IN VARCHAR2,
                       source_location IN VARCHAR2,
                       source_name     IN VARCHAR2),
```

```
MEMBER FUNCTION getContentLength(ctx IN OUT RAW) RETURN INTEGER,
PRAGMA RESTRICT_REFERENCES(getContentLength, WNDS, WNPS, RNDS, RNPS),

MEMBER PROCEDURE getContentInLob(
                              ctx      IN OUT RAW,
                              dest_lob IN OUT NOCOPY BLOB,
                              mimeType OUT VARCHAR2,
                              format   OUT VARCHAR2),

MEMBER FUNCTION getContent RETURN BLOB,
PRAGMA RESTRICT_REFERENCES(getContent, WNDS, WNPS, RNDS, RNPS),

MEMBER PROCEDURE deleteContent,

MEMBER FUNCTION getBFILE RETURN BFILE,
PRAGMA RESTRICT_REFERENCES(getBFILE, WNDS, WNPS, RNDS, RNPS),

-- Methods associated with file operations on the source
MEMBER FUNCTION openSource(userArg IN RAW, ctx OUT RAW) RETURN INTEGER,
MEMBER FUNCTION closeSource(ctx IN OUT RAW) RETURN INTEGER,
MEMBER FUNCTION trimSource(ctx     IN OUT RAW,
                           newlen  IN INTEGER) RETURN INTEGER,
MEMBER PROCEDURE readFromSource(
                              ctx      IN OUT RAW,
                              startPos IN INTEGER,
                              numBytes IN OUT INTEGER,
                              buffer   OUT RAW),
MEMBER PROCEDURE writeToSource(
                             ctx      IN OUT RAW,
                             startPos IN INTEGER,
                             numBytes IN OUT INTEGER,
                             buffer   IN RAW),

-- Methods associated with the video attributes accessors
MEMBER PROCEDURE setFormat(knownformat IN VARCHAR2),
MEMBER FUNCTION getFormat RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getFormat, WNDS, WNPS, RNDS, RNPS),

MEMBER PROCEDURE setFrameSize(knownWidth IN INTEGER, knownHeight IN INTEGER),
MEMBER PROCEDURE getFrameSize(retWidth OUT INTEGER, retHeight OUT INTEGER),
PRAGMA RESTRICT_REFERENCES(getFrameSize, WNDS, WNPS, RNDS, RNPS),

MEMBER PROCEDURE setFrameResolution(knownFrameResolution IN INTEGER),
MEMBER FUNCTION getFrameResolution RETURN INTEGER,
PRAGMA RESTRICT_REFERENCES(getFrameResolution, WNDS, WNPS, RNDS, RNPS),
```

```
MEMBER PROCEDURE setFrameRate(knownFrameRate IN INTEGER),
MEMBER FUNCTION getFrameRate RETURN INTEGER,
PRAGMA RESTRICT_REFERENCES(getFrameRate, WNDS, WNPS, RNDS, RNPS),

MEMBER PROCEDURE setVideoDuration(knownVideoDuration IN INTEGER),
MEMBER FUNCTION getVideoDuration RETURN INTEGER,
PRAGMA RESTRICT_REFERENCES(getVideoDuration, WNDS, WNPS, RNDS, RNPS),

MEMBER PROCEDURE setNumberOfFrames(knownNumberOfFrames IN INTEGER),
MEMBER FUNCTION getNumberOfFrames RETURN INTEGER,
PRAGMA RESTRICT_REFERENCES(getNumberOfFrames, WNDS, WNPS, RNDS, RNPS),

MEMBER PROCEDURE setCompressionType(knownCompressionType IN VARCHAR2),
MEMBER FUNCTION getCompressionType RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getCompressionType, WNDS, WNPS, RNDS, RNPS),

MEMBER PROCEDURE setNumberOfColors(knownNumberOfColors IN INTEGER),
MEMBER FUNCTION getNumberOfColors RETURN INTEGER,
PRAGMA RESTRICT_REFERENCES(getNumberOfColors, WNDS, WNPS, RNDS, RNPS),

MEMBER PROCEDURE setBitRate(knownBitRate IN INTEGER),
MEMBER FUNCTION getBitRate RETURN INTEGER,
PRAGMA RESTRICT_REFERENCES(getBitRate, WNDS, WNPS, RNDS, RNPS),

MEMBER PROCEDURE setKnownAttributes(
                                  knownFormat IN VARCHAR2,
                                  knownWidth IN INTEGER,
                                  knownHeight IN INTEGER,
                                  knownFrameResolution IN INTEGER,
                                  knownFrameRate IN INTEGER,
                                  knownVideoDuration IN INTEGER
                                  knownNumberOfFrames IN INTEGER,
                                  knownCompressionType IN VARCHAR2,
                                  knownNumberOfColors IN INTEGER,
                                  knownBitRate IN INTEGER),

-- Methods associated with setting all the properties
MEMBER PROCEDURE setProperties(ctx         IN OUT RAW,
                               setComments IN BOOLEAN),
MEMBER FUNCTION checkProperties(ctx IN OUT RAW) RETURN BOOLEAN,

MEMBER FUNCTION getAttribute(
                            ctx  IN OUT RAW,
                            name IN VARCHAR2) RETURN VARCHAR2,
```

```
MEMBER PROCEDURE getAllAttributes(
                                  ctx        IN OUT RAW,
                                  attributes IN OUT NOCOPY CLOB),

-- Methods associated with video processing
MEMBER FUNCTION processVideoCommand(
                                    ctx       IN OUT RAW,
                                    cmd       IN VARCHAR2,
                                    arguments IN VARCHAR2,
                                    result    OUT RAW)
                    RETURN RAW
);
```

where:

- description: the description of the video object.
- source: the ORDSource where the video data is to be found.
- format: the format in which the video data is stored.
- mimeType: the MIME type information.
- comments: the metadata information of the video object.
- width: the width of each frame of the video data.
- height: the height of each frame of the video data.
- frameResolution: the frame resolution of the video data.
- frameRate: the frame rate of the video data.
- videoDuration: the total duration of the video data stored.
- numberOfFrames: the number of frames in the video data.
- compressionType: the compression type of the video data.
- numberOfColors: the number of colors in the video data.
- bitRate: the bit rate of the video data.

Note: The comments attribute is populated by setProperties( ) when the setComments parameter is TRUE and by the Oracle *inter*Media Annotator utility. Oracle Corporation recommends that you not write to this attribute directly.

9.2 Constructors

This section describes the constructor functions.

The *inter*Media constructor functions are as follows:

- init( )
- init(srcType,srcLocation,srcName)

init( )

Format

init( ) RETURN ORDVideo;

Description

Allows for easy initialization of instances of the ORDVideo object type.

Parameters

None.

Pragmas

None.

Exceptions

None.

Usage Notes

This static method initializes all the ORDVideo attributes to NULL with the following exceptions:

- source.updateTime is set to SYSDATE
- source.local is set to 1 (local)
- source.localData is set to empty_blob

You should begin using the init( ) method as soon as possible to allow you to more easily initialize the ORDVideo object type, especially if the ORDVideo type evolves and attributes are added in a future release. INSERT statements left unchanged using the default constructor (which initializes each object attribute), will fail under these circumstances.

Examples

Initialize the ORDVideo object attributes:

```
BEGIN
  INSERT INTO tvid VALUES (ORDSYS.ORDVideo.init( ));
```

```
END;
/
```

init(srcType,srcLocation,srcName)

Format

```
init(srcType      IN VARCHAR2,
     srcLocation IN VARCHAR2,
     srcName     IN VARCHAR2)
 RETURN ORDVideo;
```

Description

Allows for easy initialization of instances of the ORDVideo object type.

Parameters

srcType
The source type of the video data.

srcLocation
The source location of the video data.

srcName
The source name of the video data.

Pragmas

None.

Exceptions

None.

Usage Notes

This static method initializes all the ORDVideo attributes to NULL with the following exceptions:

- source.updateTime is set to SYSDATE
- source.local is set to 0
- source.localData is set to empty_blob

- source.srcType is set to the input value
- source.srcLocation is set to the input value
- source.srcName is set to the input value

You should begin using the init( ) method as soon as possible to allow you to more easily initialize the ORDVideo object type, especially if the ORDVideo type evolves and attributes are added in a future release. INSERT statements left unchanged using the default constructor (which initializes each object attribute), will fail under these circumstances.

Examples

Initialize the ORDVideo object attributes:

```
BEGIN
  INSERT INTO tvid VALUES (ORDSYS.ORDVideo.init('file','VIDDIR','video1.rm'));
END;
/
```

9.3 Methods

This section presents reference information on the Oracle *inter*Media methods used for video data manipulation. These methods are described in the following groupings:

ORDVideo Methods Associated with the updateTime Attribute

- getUpdateTime( ): returns the time when the video object was last updated. See "getUpdateTime( )" on page 5-25 for information.
- setUpdateTime( ): sets the update time for the video object. This method is called implicitly by methods that modify natively supported video formats. See "setUpdateTime( )" on page 5-39 for information.

ORDVideo Methods Associated with the description Attribute

- setDescription( ): sets the description of the video data. See "setDescription( )" on page 9-49.
- getDescription: returns the description of the video data. See "getDescription" on page 9-29.

ORDVideo Methods Associated with mimeType Attribute

- setMimeType( ): sets the MIME type of the stored video data. This method is called implicitly by any method that modifies natively supported video formats. See "setMimeType( )" on page 5-35 for information.
- getMimeType( ): returns the MIME type for video data. See "getMimeType( )" on page 5-17 for information.

ORDVideo Methods Associated with the source Attribute

- processSourceCommand( ): sends a command and related arguments to the source plug-in. See "processSourceCommand( )" on page 5-29 for information.
- isLocal( ): returns TRUE if the data is stored locally in a BLOB or FALSE if the data is external. See "isLocal( )" on page 5-26 for information.
- setLocal( ): sets a flag to indicate that the data is stored locally in a BLOB. See "setLocal( )" on page 5-34 for information.
- clearLocal( ): clears the flag to indicate that the data is stored externally. See "clearLocal( )" on page 5-5 for information.

- setSource( ): sets the source information to where video data is to be found. See "setSource( )" on page 5-37 for information.
- getSource( ): returns a formatted string containing complete information about the external data source formatted as a URL. See "getSource( )" on page 5-19 for information.
- getSourceType( ): returns the external source type of the video data. See "getSourceType( )" on page 5-23 for information.
- getSourceLocation( ): returns the external source location of the video data. See "getSourceLocation( )" on page 5-21 for information.
- getSourceName( ): returns the external source name of the video data. See "getSourceName( )" on page 5-22 for information.
- import( ): transfers data from an external data source (specified by calling setSourceInformation( )) to the local source (localData) within an Oracle database, setting the value of the local attribute to "1", meaning local and updating the timestamp. See "import( )" on page 9-39.
- importFrom( ): transfers data from the specified external data source (source type, location, name) to the local source (localData) within an Oracle database, setting the value of the local attribute to "1", meaning local and updating the timestamp. See "importFrom( )" on page 9-41.
- export( ): copies data from a local source (localData) within an Oracle database to the specified external data source, and stores source information in the source. See "export( )" on page 5-9 for information.

  Note: The export( ) method natively supports only sources of source type file. User-defined sources may support the export( ) method.

- getContentLength( ): returns the length of the data source (as number of bytes). See "getContentLength( )" on page 9-28.
- getContentInLob( ): returns content into a temporary LOB. See "getContentInLob( )" on page 9-26 for information.
- getContent( ): returns the handle to the BLOB used to store contents locally. See "getContent( )" on page 5-15 for information.
- deleteContent( ): deletes the content of the local BLOB. See "deleteContent( )" on page 5-8 for information.

- getBFILE: returns the external content as a BFILE. See "getBFILE( )" on page 5-13 for information.

ORDVideo Methods Associated with File Operations

- openSource( ): opens a data source or a BLOB. See "openSource( )" on page 5-27 for information.
- closeSource( ): closes a data source or a BLOB. See "closeSource( )" on page 5-6 for information.
- trimSource( ): trims a data source or a BLOB. See "trimSource( )" on page 5-40 for information.
- readFromSource( ): reads a buffer of *n* bytes from a source beginning at a start position. See "readFromSource( )" on page 5-32 for information.
- writeToSource( ): writes a buffer of *n* bytes to a source beginning at a start position. See "writeToSource( )" on page 5-42 for information.

ORDVideo Methods Associated with Video Attributes Accessors

- setFormat( ): sets the object attribute value of the format of the video data. See "setFormat( )" on page 9-51 for information.
- getFormat: returns the object attribute value of the format in which the video data is stored. See "getFormat" on page 9-30.
- setFrameSize( ): sets the object attribute value of the width and height in pixels of each frame in the video data. See "setFrameSize( )" on page 9-55.
- getFrameSize: returns the object attribute value of the width and height in pixels of each frame in the video data. See "getFrameSize( )" on page 9-34.
- setFrameResolution( ): sets the object attribute value of the number of pixels per inch of frames in the video data. See "setFrameResolution( )" on page 9-54.
- getFrameResolution: returns the object attribute value of the number of pixels per inch of frames in the video data. See "getFrameResolution" on page 9-33.
- setFrameRate( ): sets the object attribute value of the rate in frames per second at which the video data was recorded. See "setFrameRate( )" on page 9-53.
- getFrameRate: returns the object attribute value of the rate in frames per second at which the video data was recorded. See "getFrameRate" on page 9-32.
- setVideoDuration( ): sets the object attribute value of the total time it takes to play the entire video data. See "setVideoDuration( )" on page 9-64.

- getVideoDuration: returns the object attribute value of the total time it takes to play the entire video data. See "getVideoDuration" on page 9-38.
- setNumberOfFrames( ): sets the object attribute value of the total number of frames in the video data. See "setNumberOfFrames( )" on page 9-61.
- getNumberOfFrames: returns the object attribute value of the total number of frames in the video data. See "getNumberOfFrames" on page 9-37.
- setCompressionType( ): sets the value of the compression type attribute of the video object. See "setCompressionType( )" on page 9-48.
- getCompressionType: returns the object attribute value of the compression type in the video data. See "getCompressionType" on page 9-25.
- setNumberOfColors( ): sets the object attribute value of the number of colors in the video data. See "setNumberOfColors( )" on page 9-60.
- getNumberOfColors: returns the object attribute value of the number of colors in the video data. See "getNumberOfColors" on page 9-36.
- setBitRate( ): sets the object attribute value of the bit rate in the video data. See "setBitRate( )" on page 9-47.
- getBitRate: returns the object attribute value of the bit rate in the video data. See "getBitRate" on page 9-24.
- setKnownAttributes( ): sets known video attributes including format, frame size, frame resolution, frame rate, video duration, number of frames, compression type, number of colors, and bit rate of the video data. The parameters are passed in with this call. See "setKnownAttributes( )" on page 9-57.
- setProperties( ): reads the video data to get the values of the object attributes and then stores them in the object. If the value for the setComments parameter is TRUE, then the comments field of the object will be populated with a rich set of format and application properties of the video object in XML form, identical to what is provided by the *inter*Media Annotator utility. For the known attributes that ORDVideo understands, it sets the properties for these attributes, which include: format, frame size, frame resolution, frame rate, video duration, number of frames, compression type, number of colors, and bit rate of the video data. See "setProperties( )" on page 9-62.
- checkProperties( ): calls the format plug-in to check the properties including format, frame size, frame resolution, frame rate, video duration, number of frames, compression type, number of colors, and bit rate of the video data; it

returns a Boolean value TRUE if the properties stored in object attributes match those in the video data. See "checkProperties( )" on page 9-18.

- getAttribute( ): returns the value of the requested attribute. This method is only available for user-defined format plug-ins. See "getAttribute( )" on page 9-22.
- getAllAttributes( ): returns a formatted string for convenient client access. For natively supported formats, the string includes the following list of video data attributes separated by a comma (,): format, frameSize, frameResolution, frameRate, videoDuration, numberOfFrames, compressionType, numberOfColors, and bitRate. The string is defined by the user-defined format plug-in. See "getAllAttributes( )" on page 9-20.

ORDVideo Methods Associated with Processing Video Data

- processVideoCommand( ): sends commands and related arguments to the format plug-in for processing. See "processVideoCommand( )" on page 9-44.

For more information on object types and methods, see *Oracle9i Database Concepts*.

9.3.1 Example Table Definitions

The methods described in this reference chapter show examples based on a test video table TVID. Refer to the TVID table definition that follows when reading through the examples:

TVID Table Definition

```
CREATE TABLE TVID(n NUMBER, vid ORDSYS.ORDVideo)
storage (initial 100K next 100K pctincrease 0);

INSERT INTO TVID VALUES(1, ORDSYS.ORDVideo.init());
INSERT INTO TVID VALUES(2, ORDSYS.ORDVideo.init());
```

checkProperties( )

Format

checkProperties(ctx IN OUT RAW) RETURN BOOLEAN;

Description

Checks all the properties of the stored video data, including the following video attributes: format, frame size, frame resolution, frame rate, video duration, number of frames, compression type, number of colors, and bit rate.

Parameters

ctx
The format plug-in context information.

Usage Notes

The checkProperties( ) method does not check the MIME type because a file can have multiple correct MIME types and this is not well defined.

Pragmas

None.

Exceptions

VIDEO_PLUGIN_EXCEPTION

This exception is raised if you call the checkProperties( ) method and the video plug-in raises an exception when calling this method.

Examples

Check property information for known video attributes:

```
DECLARE
  obj ORDSYS.ORDVideo;
  ctx RAW(4000) :=NULL;
BEGIN
  select vid into obj from TVID where N =1 ;
  if (obj.checkProperties(ctx)) then
```

```
  DBMS_OUTPUT.put_line('check Properties returned true');
else
  DBMS_OUTPUT.put_line('check Properties returned false');
  end if;
  EXCEPTION
    WHEN OTHERS THEN
     DBMS_OUTPUT.put_line('exception raised');
END;
/
```

getAllAttributes( )

Format

```
getAllAttributes(
        ctx         IN OUT RAW,
        attributes IN OUT NOCOPY CLOB);
```

Description

Returns a formatted string for convenient client access. For natively supported formats, the string includes the following list of audio data attributes separated by a comma (,): width, height, format, frameResolution, frameRate, videoDuration, numberOfFrames, compressionType, numberOfColors, and bitRate. For user-defined formats, the string is defined by the format plug-in.

Parameters

ctx
The format plug-in context information.

attributes
The attributes.

Usage Notes

These video data attributes are available from the header of the formatted video data.

Video data attribute information can be extracted from the video data itself. You can extend support to a video format that is not understood by the ORDVideo object by implementing an ORDPLUGINS.ORDX_<format>_VIDEO package that supports that format. See Section 3.4.13 for more information.

Pragmas

None.

Exceptions

METHOD_NOT_SUPPORTED

This exception is raised if you call the getAllAttributes( ) method and the video plug-in raises an exception when calling this method.

Examples

Return all video attributes for video data stored in the database:

```
DECLARE
  obj ORDSYS.ORDVideo;
  tempLob   CLOB;
  ctx RAW(4000) :=NULL;
BEGIN

  SELECT vid INTO obj FROM TVID  WHERE N=1;
  DBMS_OUTPUT.PUT_LINE('getting comma separated list of all attributes');
  DBMS_OUTPUT.PUT_LINE('-------------------------------------------');

  DBMS_LOB.CREATETEMPORARY(tempLob, FALSE, DBMS_LOB.CALL);
  obj.getAllAttributes(ctx,tempLob);
 DBMS_OUTPUT.put_line(DBMS_LOB.substr(tempLob, DBMS_LOB.getLength(tempLob), 1));
  EXCEPTION
  WHEN ORDSYS.ORDSourceExceptions.METHOD_NOT_SUPPORTED THEN
   DBMS_OUTPUT.put_line('Source METHOD_NOT_SUPPORTED caught');
  WHEN ORDSYS.ORDSourceExceptions.SOURCE_PLUGIN_EXCEPTION THEN
   DBMS_OUTPUT.put_line('SOURCE PLUGIN EXCEPTION caught');
  WHEN ORDSYS.ORDVideoExceptions.METHOD_NOT_SUPPORTED THEN
   DBMS_OUTPUT.put_line('VIDEO METHOD_NOT_SUPPORTED EXCEPTION caught');
  WHEN ORDSYS.ORDVideoExceptions.VIDEO_PLUGIN_EXCEPTION THEN
   DBMS_OUTPUT.put_line('VIDEO PLUGIN EXCEPTION caught');
  WHEN OTHERS THEN
   DBMS_OUTPUT.put_line('EXCEPTION CAUGHT');
END;
/
```

getAttribute( )

Format

```
getAttribute(
        ctx     IN OUT RAW,
        name IN VARCHAR2)
RETURN VARCHAR2;
```

Description

Returns the value of the requested attribute from video data for user-defined formats only.

Parameters

ctx
The format plug-in context information.

name
The name of the attribute.

Usage Notes

The video data attributes are available from the header of the formatted video data.

Pragmas

None.

Exceptions

VIDEO_PLUGIN_EXCEPTION

This exception is raised if you call the getAttribute( ) method and the video plug-in raises an exception when calling this method.

Examples

Return information for the specified video attribute for video data stored in the database:

```
DECLARE
  obj ORDSYS.ORDVideo;
  res VARCHAR2(4000);
  ctx RAW(4000) :=NULL;
BEGIN
  SELECT vid INTO obj FROM TVID  WHERE N=1;
  DBMS_OUTPUT.PUT_LINE('getting video duration');
  DBMS_OUTPUT.PUT_LINE('---------------------');
  res := obj.getAttribute(ctx,'video_duration');
  EXCEPTION
        WHEN ORDSYS.ORDSourceExceptions.METHOD_NOT_SUPPORTED THEN
         DBMS_OUTPUT.put_line('Source METHOD_NOT_SUPPORTED caught');
        WHEN ORDSYS.ORDSourceExceptions.SOURCE_PLUGIN_EXCEPTION THEN
         DBMS_OUTPUT.put_line('SOURCE PLUGIN EXCEPTION caught');
        WHEN ORDSYS.ORDVideoExceptions.METHOD_NOT_SUPPORTED THEN
         DBMS_OUTPUT.put_line('VIDEO METHOD_NOT_SUPPORTED EXCEPTION caught');
        WHEN ORDSYS.ORDVideoExceptions.VIDEO_PLUGIN_EXCEPTION THEN
         DBMS_OUTPUT.put_line('VIDEO PLUGIN EXCEPTION caught');
        WHEN OTHERS THEN
         DBMS_OUTPUT.PUT_LINE('EXCEPTION caught');
END;
/
```

getBitRate

Format

getBitRate RETURN INTEGER;

Description

Returns the value of the bitRate attribute of the video object.

Parameters

None.

Usage Notes

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getBitRate, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Examples

Return the object attribute value of the bitRate attribute of the video object:

```
DECLARE
  obj ORDSYS.ORDVideo;
  res INTEGER;
BEGIN
  SELECT vid INTO obj FROM TVID  WHERE N=1 ;
  res := obj.getBitRate();
  DBMS_OUTPUT.put_line('bit rate : ' || res );
END;
/
```

getCompressionType

Format

getCompressionType RETURN VARCHAR2;

Description

Returns the value of the compressionType attribute of the video object.

Parameters

None.

Usage Notes

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getCompressionType, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Examples

Return the object attribute value of the compressionType attribute of the video object:

```
DECLARE
  obj ORDSYS.ORDVideo;
  res VARCHAR2(4000);
BEGIN
  SELECT vid INTO obj FROM TVID  WHERE N=1 ;
  res := obj.getCompressionType();
  DBMS_OUTPUT.put_line('compression type: ' ||res);
END;
/
```

getContentInLob( )

Format

```
getContentInLob(
        ctx             IN OUT RAW,
        dest_lob        IN OUT NOCOPY BLOB,
        mimeType  OUT VARCHAR2,
        format          OUT VARCHAR2);
```

Description

Copies data from a data source into the specified BLOB. The BLOB must not be the BLOB in source.localData.

Parameters

ctx
The source plug-in context information.

dest_lob
The LOB in which to receive data.

mimeType
The MIME type of the data; this may or may not be returned.

format
The format of the data; this may or may not be returned.

Usage Notes

None.

Pragmas

None.

Exceptions

ORDSourceExceptions.INCOMPLETE_SOURCE_INFORMATION

This exception is raised if you call the getContentInLob( ) method and the value of srcType is NULL.

ORDSourceExceptions.METHOD_NOT_SUPPORTED

This exception is raised if you call the getContentInLob( ) method and this method is not supported by the source plug-in being used.

ORDSourceExceptions.SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the getContentInLob( ) method and within a source plug-in when any other exception is raised.

See Appendix H for more information about these exceptions.

Examples

Get data from a data source into the specified BLOB on the local source:

```
DECLARE
  obj ORDSYS.ORDVideo;
  tempBLob BLOB;
  mimeType VARCHAR2(4000);
  format VARCHAR2(4000);
  ctx RAW(4000) :=NULL;
BEGIN
 SELECT vid INTO obj FROM TVID WHERE N = 1 ;
 if(obj.isLocal) then
   DBMS_OUTPUT.put_line('local is true');
 end if;
 DBMS_LOB.CREATETEMPORARY(tempBLob, true, 10);
 obj.getContentInLob(ctx,tempBLob, mimeType,format);
 DBMS_OUTPUT.PUT_LINE(TO_CHAR(DBMS_LOB.getLength(tempBLob)));
EXCEPTION
WHEN ORDSYS.ORDSourceExceptions.METHOD_NOT_SUPPORTED THEN
DBMS_OUTPUT.put_line('ORDSourceExceptions.METHOD_NOT_SUPPORTED caught');
WHEN OTHERS THEN
DBMS_OUTPUT.put_line('EXCEPTION caught');
END;
/
```

getContentLength( )

Format

getContentLength(ctx IN OUT RAW) RETURN INTEGER;

Description

Returns the length of the video data content stored in the source.

Parameters

ctx
The source plug-in context information.

Usage Notes

This method is not supported for all source types. For example, HTTP type sources do not support this method. If you want to implement this call for HTTP type sources, you must define your own modified HTTP source type and implement this method on it.

Pragmas

PRAGMA RESTRICT_REFERENCES(getContentLength, WNDS, WNPS, RNDS, RNPS)

Exceptions

ORDSourceExceptions.INCOM-PLETE_SOURCE_INFORMATION

This exception is raised if you call the getContentLength( ) method and the value of srcType is NULL and data is not stored locally in the BLOB.

ORDSourceExceptions.SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the getContentLength( ) method within a source plug-in when any other exception is raised.

See Appendix H for more information about this exception.

Examples

See the example in "import( )" on page 9-40.

getDescription

Format

getDescription RETURN VARCHAR2;

Description

Returns the description of the video data.

Parameters

None.

Usage Notes

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getDescription, WNDS, WNPS, RNDS, RNPS)

Exceptions

DESCRIPTION_IS_NOT_SET

This exception is raised if you call the getDescription method and the description is not set.

Examples

See the example in setDescription( ) on page 9-49.

getFormat

Format

getFormat RETURN VARCHAR2;

Description

Returns the value of the format attribute of the video object.

Parameters

None.

Usage Notes

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getStoredFormat, WNDS, WNPS, RNDS, RNPS)

Exceptions

VIDEO_FORMAT_IS_NULL

This exception is raised if you call the getFormat( ) method and the value for format is NULL.

Examples

Set the format and then get it for some stored video data:

```
DECLARE
  obj ORDSYS.ORDVideo;
BEGIN
  SELECT vid INTO obj FROM TVID  WHERE N=1 FOR UPDATE;
  DBMS_OUTPUT.PUT_LINE('writing format');
  DBMS_OUTPUT.PUT_LINE('--------------');
  obj.setFormat('avi');
  DBMS_OUTPUT.PUT_LINE(obj.getFormat());
  UPDATE TVID SET vid=obj WHERE N=1;
  COMMIT;
```

```
END;
/
```

getFrameRate

Format

getFrameRate RETURN INTEGER;

Description

Returns the value of the frameRate attribute of the video object.

Parameters

None.

Usage Notes

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getFrameRate, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Examples

Return the object attribute value of the frame rate for video data stored in the database:

```
DECLARE
  obj ORDSYS.ORDVideo;
  res INTEGER;
BEGIN
  SELECT vid INTO obj FROM TVID  WHERE N=1 ;
  res := obj.getFrameRate();
  DBMS_OUTPUT.put_line('frame rate : ' ||res);
END;
/
```

getFrameResolution

Format

getFrameResolution RETURN INTEGER;

Description

Returns the value of the frameResolution attribute of the video object.

Parameters

None.

Usage Notes

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getFrameResolution, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Examples

Return the value of the frame resolution for the video data:

```
DECLARE
  obj ORDSYS.ORDVideo;
  res INTEGER;
BEGIN
  SELECT vid INTO obj FROM TVID  WHERE N=1 ;
  res := obj.getFrameResolution();
  DBMS_OUTPUT.put_line('resolution : ' ||res);
END;
/
```

getFrameSize( )

Format

```
getFrameSize(
        retWidth  OUT INTEGER,
        retHeight OUT INTEGER);
```

Description

Returns the value of the height and width attributes of the video object.

Parameters

retWidth
The frame width in pixels.

retHeight
The frame height in pixels.

Usage Notes

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getFrameSize, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Examples

Return the frame size for video data:

```
DECLARE
  obj ORDSYS.ORDVideo;
  width INTEGER;
  height INTEGER;
BEGIN
  SELECT vid INTO obj FROM TVID  WHERE N=1 ;
```

```
  obj.getFrameSize(width, height);
  DBMS_OUTPUT.put_line('width :' || width);
  DBMS_OUTPUT.put_line('height :' || height);
END;
/
```

getNumberOfColors

Format

getNumberOfColors RETURN INTEGER;

Description

Returns the value of the numberOfColors attribute of the video object.

Parameters

None.

Usage Notes

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getNumberOfColors, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Examples

Return the object attribute value of the numberOfColors attribute of the video object:

```
DECLARE
  obj ORDSYS.ORDVideo;
  res INTEGER;
BEGIN
  SELECT vid INTO obj FROM TVID  WHERE N=1 ;
  res := obj.getNumberOfColors();
  DBMS_OUTPUT.put_line('number of colors: ' ||res);
END;
/
```

getNumberOfFrames

Format

getNumberOfFrames RETURN INTEGER;

Description

Returns the value of the numberOfFrames attribute of the video object.

Parameters

None.

Usage Notes

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getNumberOfFrames, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Examples

Return the object attribute value of the total number of frames in the video data:

```
DECLARE
  obj ORDSYS.ORDVideo;
  res INTEGER;
BEGIN
  SELECT vid INTO obj FROM TVID  WHERE N=1 ;
  res := obj.getNumberOfFrames();
  DBMS_OUTPUT.put_line('number of frames : ' ||res);
END;
/
```

getVideoDuration

Format

getVideoDuration RETURN INTEGER;

Description

Returns the value of the videoDuration attribute of the video object.

Parameters

None.

Usage Notes

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getVideoDuration, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Examples

Return the total time to play the video data:

```
DECLARE
  obj ORDSYS.ORDVideo;
  res INTEGER;
BEGIN
  SELECT vid INTO obj FROM TVID  WHERE N=1 ;
  res := obj.getVideoDuration();
  DBMS_OUTPUT.put_line('video duration : ' ||res);
END;
/
```

import( )

Format

import(ctx IN OUT RAW);

Description

Transfers video data from an external video data source to a local source (localData) within an Oracle database.

Parameters

ctx
The source plug-in context information. This must be allocated. You must call the openSource( ) method; see the introduction to this chapter for more information.

Usage Notes

Use the setSource( ) method to set the external source type, location, and name prior to calling import.

You must ensure that the directory exists or is created before you use this method for srcType 'file'.

After importing data from an external video data source to a local source (within an Oracle database), the source information remains unchanged (that is, pointing to the source from where the data was imported).

Invoking this method implicitly calls the setUpdateTime( ) and setLocal methods.

Pragmas

None.

Exceptions

ORDSourceExceptions.INCOMPLETE_SOURCE_INFORMATION

This exception is raised if you call the import( ) method and the value of srcType is NULL.

ORDSourceExceptions.NULL_SOURCE

This exception is raised if you call the import( ) method and the value of dlob is NULL.

ORDSourceExceptions.METHOD_NOT_SUPPORTED

This exception is raised if you call the import( ) method and this method is not supported by the source plug-in being used.

ORDSourceExceptions.SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the import( ) method within a source plug-in when any other exception is raised.

See Appendix H for more information about these exceptions.

Examples

Import video data by first setting the source and then importing it:

```
DECLARE
  obj ORDSYS.ORDVideo;
  ctx RAW(4000) :=NULL;
BEGIN
  SELECT vid INTO obj FROM TVID  WHERE N=1 FOR UPDATE;
  DBMS_OUTPUT.PUT_LINE('setting and getting source');
  DBMS_OUTPUT.PUT_LINE('-------------------------');
  -- set source to a file
  obj.setSource('file','VIDEODIR','testvid.dat');
  -- get source information
  DBMS_OUTPUT.PUT_LINE(obj.getSource());
  -- import data
  obj.import(ctx);
  -- check size
  DBMS_OUTPUT.PUT_LINE(TO_CHAR(obj.getContentLength(ctx)));
  DBMS_OUTPUT.PUT_LINE(obj.getSource());
  DBMS_OUTPUT.PUT_LINE('deleting contents');
  DBMS_OUTPUT.PUT_LINE('-----------------');
  obj.deleteContent();
  DBMS_OUTPUT.PUT_LINE(TO_CHAR(obj.getContentLength(ctx)));
  UPDATE TVID SET vid=obj WHERE N=1;
  COMMIT;
END;
/
```

importFrom( )

Format

```
importFrom(ctx IN OUT RAW,
           source_type      IN VARCHAR2,
           source_location  IN VARCHAR2,
           source_name      IN VARCHAR2);
```

Description

Transfers video data from the specified external video data source to a local source (localData) within an Oracle database.

Parameters

ctx
The source plug-in context information. This must be allocated. You must call the openSource( ) method; see the introduction to this chapter for more information.

source_type
The source type of the video data.

source_location
The location from where the video data is to be imported.

source_name
The name of the video data.

Usage Notes

This method is similar to the import( ) method except the source information is specified as parameters to the method instead of separately.

You must ensure that the directory exists or is created before you use this method for srcType 'file'.

After importing data from an external video data source to a local source (within an Oracle database), the source information (that is, pointing to the source from where the data was imported) is set to the input values.

Invoking this method implicitly calls the setUpdateTime( ) and setLocal methods.

Pragmas

None.

Exceptions

ORDSourceExceptions.NULL_SOURCE exception

This exception is raised if you call the importFrom( ) method and the value dlob is NULL.

ORDSourceExceptions.METHOD_NOT_SUPPORTED

This exception is raised if you call the importFrom( ) method and this method is not supported by the source plug-in being used.

ORDSourceExceptions.SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the importFrom( ) method within a source plug-in when any other exception is raised.

See Appendix H for more information about these exceptions.

Examples

Import video data from the specified external data source into the local source:

```
DECLARE
  obj ORDSYS.ORDVideo;
  ctx RAW(4000) :=NULL;
BEGIN
  SELECT vid INTO obj FROM TVID  WHERE N=1 FOR UPDATE;
  DBMS_OUTPUT.PUT_LINE('setting and getting source');
  DBMS_OUTPUT.PUT_LINE('-------------------------');
  -- import data
  obj.importFrom(ctx,'file','VIDEODIR','MV1.AVI');
  -- check size
  DBMS_OUTPUT.PUT_LINE(TO_CHAR(obj.getContentLength(ctx)));
  DBMS_OUTPUT.PUT_LINE(TO_CHAR(DBMS_LOB.GETLENGTH(obj.getContent)));
  DBMS_OUTPUT.PUT_LINE(obj.getSource());
  DBMS_OUTPUT.PUT_LINE('deleting contents');
  DBMS_OUTPUT.PUT_LINE('----------------');
  obj.deleteContent();
  DBMS_OUTPUT.PUT_LINE(TO_CHAR(obj.getContentLength(ctx)));
  UPDATE TVID SET vid=obj WHERE N=1;
  COMMIT;
```

```
  EXCEPTION
        WHEN ORDSYS.ORDSourceExceptions.METHOD_NOT_SUPPORTED THEN
         DBMS_OUTPUT.put_line('Source METHOD_NOT_SUPPORTED caught');
        WHEN ORDSYS.ORDSourceExceptions.SOURCE_PLUGIN_EXCEPTION THEN
         DBMS_OUTPUT.put_line('SOURCE PLUGIN EXCEPTION caught');
        WHEN ORDSYS.ORDVideoExceptions.METHOD_NOT_SUPPORTED THEN
         DBMS_OUTPUT.put_line('VIDEO METHOD_NOT_SUPPORTED EXCEPTION caught');
        WHEN ORDSYS.ORDVideoExceptions.VIDEO_PLUGIN_EXCEPTION THEN
         DBMS_OUTPUT.put_line('VIDEO PLUGIN EXCEPTION caught');
        WHEN OTHERS THEN
         DBMS_OUTPUT.PUT_LINE('EXCEPTION Caught');
END;
/
```

processVideoCommand( )

Format

```
processVideoCommand(
                        ctx          IN OUT RAW,
                        cmd          IN VARCHAR2,
                        arguments IN VARCHAR2,
                        result       OUT RAW)
RETURN RAW;
```

Description

Allows you to send a command and related arguments to the format plug-in for processing.

Note: This method is supported only for user-defined format plug-ins.

Parameters

ctx
The format plug-in context information.

cmd
Any command recognized by the format plug-in.

arguments
The arguments of the command.

result
The result of calling this function returned by the format plug-in.

Usage Notes

Use this method to send any video commands and their respective arguments to the format plug-in. Commands are not interpreted; they are taken and passed through to a format plug-in to be processed.

If the format is set to NULL, then the processVideoCommand( ) method uses the default format plug-in; otherwise, it uses your user-defined format plug-in.

You can extend support to a format that is not understood by the ORDVideo object by preparing an ORDPLUGINS.ORDX_<format>_VIDEO package that supports that format. See Section 3.4.13 for more information.

Pragmas

None.

Exceptions

METHOD_NOT_SUPPORTED or VIDEO_PLUGIN_EXCEPTION

Either exception is raised if you call the ProcessVideoCommand( ) method and the video plug-in raises an exception when calling this method.

Examples

Process a set of commands:

```
DECLARE
  obj ORDSYS.ORDVideo;
  res RAW(4000);
  result RAW(4000);
  command VARCHAR(4000);
  argList VARCHAR(4000);
  ctx RAW(4000) :=NULL;
BEGIN
select vid into obj from TVID where N =1 for UPDATE;
-- assign command
-- assign argList
res := obj.processVideoCommand(ctx, command, argList, result);
UPDATE TVID SET vid=obj WHERE N=1 ;
COMMIT;
EXCEPTION
        WHEN ORDSYS.ORDSourceExceptions.METHOD_NOT_SUPPORTED THEN
         DBMS_OUTPUT.put_line('Source METHOD_NOT_SUPPORTED caught');
        WHEN ORDSYS.ORDSourceExceptions.SOURCE_PLUGIN_EXCEPTION THEN
         DBMS_OUTPUT.put_line('SOURCE PLUGIN EXCEPTION caught');
        WHEN ORDSYS.ORDVideoExceptions.METHOD_NOT_SUPPORTED THEN
         DBMS_OUTPUT.put_line('VIDEO METHOD_NOT_SUPPORTED EXCEPTION caught');
        WHEN ORDSYS.ORDVideoExceptions.VIDEO_PLUGIN_EXCEPTION THEN
         DBMS_OUTPUT.put_line('VIDEO PLUGIN EXCEPTION caught');
        WHEN OTHERS THEN
```

```
            DBMS_OUTPUT.put_line('EXCEPTION caught');
END;
/
```

setBitRate( )

Format

setBitRate(knownBitRate IN INTEGER);

Description

Sets the value of the bitRate attribute of the video object.

Parameters

knownBitRate
The bit rate.

Usage Notes

Calling this method implicitly calls the setUpdateTime( ) method.

Pragmas

None.

Exceptions

NULL_INPUT_VALUE

This exception is raised if you call the setBitRate( ) method and the value for the knownBitRate parameter is NULL.

Examples

See the example in "setFrameSize( )" on page 9-55.

setCompressionType( )

Format

setCompressionType(knownCompressionType IN VARCHAR2);

Description

Sets the value of the compressionType attribute of the video object.

Parameters

knownCompressionType
A known compression type.

Usage Notes

Calling this method implicitly calls the setUpdateTime( ) method.

Pragmas

None.

Exceptions

NULL_INPUT_VALUE

This exception is raised if you call the setCompressionType( ) method and the value for the knownCompressionType parameter is NULL.

Examples

See the example in "setFrameSize( )" on page 9-55.

setDescription( )

Format

setDescription (user_description IN VARCHAR2);

Description

Sets the description of the video data.

Parameters

user_description
The description of the video data.

Usage Notes

Each video object may need a description to help some client applications. For example, a Web-based client can show a list of video descriptions from which a user can select one to access the video data.

Web access components and other client components provided with Oracle *inter*Media make use of this description attribute to present video data to users.

Calling this method implicitly calls the setUpdateTime( ) method.

Pragmas

None.

Exceptions

None.

Examples

Set the description attribute for some video data:

```
DECLARE
  obj ORDSYS.ORDVideo;
BEGIN
  SELECT vid INTO obj FROM TVID  WHERE N=1 FOR UPDATE;
  DBMS_OUTPUT.PUT_LINE('writing description');
  DBMS_OUTPUT.PUT_LINE('-------------');
```

```
  obj.setDescription('video1');
  DBMS_OUTPUT.PUT_LINE(obj.getDescription());
  UPDATE TVID SET vid=obj WHERE N=1;
  COMMIT;
END;
/
```

setFormat( )

Format

setFormat(knownFormat IN VARCHAR2);

Description

Sets the format attribute of the video object.

Parameters

knownFormat
The known format of the video data to be set in the video object.

Usage Notes

Calling this method implicitly calls the setUpdateTime( ) method.

Pragmas

None.

Exceptions

NULL_INPUT_VALUE

This exception is raised if you call the setFormat( ) method and the value for the knownFormat parameter is NULL.

Examples

Set the format for some stored video data:

```
DECLARE
  obj ORDSYS.ORDVideo;
BEGIN
  SELECT vid INTO obj FROM TVID  WHERE N=1 FOR UPDATE;
  DBMS_OUTPUT.PUT_LINE('writing format');
  DBMS_OUTPUT.PUT_LINE('--------------');
  obj.setFormat('avi');
  DBMS_OUTPUT.PUT_LINE(obj.getFormat);
  UPDATE TVID SET vid=obj WHERE N=1;
  COMMIT;
```

```
  EXCEPTION
    WHEN ORDSYS.ORDVideoExceptions.NULL_INPUT_VALUE THEN
      DBMS_OUTPUT.put_line('ORDVideoExceptions.NULL_INPUT_VALUE caught');
    WHEN OTHERS THEN
      DBMS_OUTPUT.put_line('EXCEPTION caught');
END;
/
```

setFrameRate( )

Format

setFrameRate(knownFrameRate IN INTEGER);

Description

Sets the value of the frameRate attribute of the video object.

Parameters

knownFrameRate
The frame rate.

Usage Notes

Calling this method implicitly calls the setUpdateTime( ) method.

Pragmas

None.

Exceptions

NULL_INPUT_VALUE

This exception is raised if you call the setFrameRate( ) method and the value for the knownFrameRate parameter is NULL.

Examples

See the example in "setFrameSize( )" on page 9-55.

setFrameResolution( )

Format

setFrameResolution(knownFrameResolution IN INTEGER);

Description

Sets the value of the frameResolution attribute of the video object.

Parameters

knownFrameResolution
The known frame resolution in pixels per inch.

Usage Notes

Calling this method implicitly calls the setUpdateTime( ) method.

Pragmas

None.

Exceptions

NULL_INPUT_VALUE

This exception is raised if you call the setFrameResolution( ) method and the value for the knownFrameResolution parameter is NULL.

Examples

See the example in "setFrameSize( )" on page 9-55.

setFrameSize( )

Format

```
setFrameSize(
        knownWidth  IN INTEGER,
        knownHeight IN INTEGER);
```

Description

Sets the value of the height and width attributes of the video object.

Parameters

knownWidth
The frame width in pixels.

knownHeight
The frame height in pixels.

Usage Notes

Calling this method implicitly calls the setUpdateTime( ) method.

Pragmas

None.

Exceptions

NULL_INPUT_VALUE

This exception is raised if you call the setFrameSize( ) method and the value for either the knownWidth or knownHeight parameter is NULL.

Examples

Set the frame size for video data:

```
DECLARE
  obj ORDSYS.ORDVideo;
BEGIN
```

```
  select vid into obj from TVID where N =1 for update;
  obj.setFrameSize(1,2);
  obj.setFrameResolution(4);
  obj.setFrameRate(5);
  obj.setVideoDuration(20);
  obj.setNumberOfFrames(8);
  obj.setCompressionType('Cinepak');
  obj.setBitRate(1500);
  obj.setNumberOfColors(256);
  update TVID set vid = obj where N = 1;
  COMMIT;
END;
/
```

setKnownAttributes( )

Format

```
setKnownAttributes(
                    knownFormat          IN VARCHAR2,
                    knownWidth           IN INTEGER,
                    knownHeight          IN INTEGER,
                    knownFrameResolution IN INTEGER,
                    knownFrameRate       IN INTEGER,
                    knownVideoDuration   IN INTEGER,
                    knownNumberOfFrames  IN INTEGER,
                    knownCompressionType IN VARCHAR2,
                    knownNumberOfColors  IN INTEGER,
                    knownBitRate         IN INTEGER);
```

Description

Sets the known video attributes for the video data.

Parameters

knownFormat
The known format.

knownWidth
The known width.

knownHeight
The known height.

knownFrameResolution
The known frame resolution.

knownFrameRate
The known frame rate.

knownVideoDuration
The known video duration.

knownNumberOfFrames
The known number of frames.

knownCompressionType
The known compression type.

knownNumberOfColors
The known number of colors.

knownBitRate
The known bit rate.

Usage Notes

Calling this method implicitly calls the setUpdateTime( ) method.

Pragmas

None.

Exceptions

None.

Examples

Set the property information for all known attributes for video data:

```
DECLARE
  obj ORDSYS.ORDVideo;
  width integer;
  height integer;
BEGIN
 select vid into obj from TVID where N =1 for update;
 obj.setKnownAttributes('MOOV',1,2,4,5,20,8,'Cinepak', 256, 1500);
 obj.getFrameSize(width, height);
 DBMS_OUTPUT.put_line('width: '  || TO_CHAR(width));
 DBMS_OUTPUT.put_line('height: ' || TO_CHAR(height));
 DBMS_OUTPUT.put_line('format: ' || obj.getFormat());
 DBMS_OUTPUT.put_line('frame resolution: ' ||TO_CHAR(obj.getFrameResolution()));
 DBMS_OUTPUT.put_line('frame rate: ' || TO_CHAR(obj.getFrameRate()));
 DBMS_OUTPUT.put_line('video duration: ' || TO_CHAR(obj.getVideoDuration()));
```

```
 DBMS_OUTPUT.put_line('number of frames: ' || TO_CHAR(obj.getNumberOfFrames()));
 DBMS_OUTPUT.put_line('compression type: ' || obj.getCompressionType());
 DBMS_OUTPUT.put_line('bite rate: ' || TO_CHAR(obj.getBitRate()));
 DBMS_OUTPUT.put_line('number of colors: ' || TO_CHAR(obj.getNumberOfColors()));
 update TVID set vid = obj where N = 1;
 COMMIT;
END;
/
```

setNumberOfColors( )

Format

setNumberOfColors(knownNumberOfColors RETURN INTEGER);

Description

Sets the value of the numberOfColors attribute of the video object.

Parameters

knownNumberOfColors
A known number of colors.

Usage Notes

Calling this method implicitly calls the setUpdateTime( ) method.

Pragmas

None.

Exceptions

NULL_INPUT_VALUE

This exception is raised if you call the setNumberOfColors( ) method and the value for the knownNumberOfColors parameter is NULL.

Examples

See the example in "setFrameSize( )" on page 9-55.

setNumberOfFrames( )

Format

setNumberOfFrames(knownNumberOfFrames RETURN INTEGER);

Description

Sets the value of the numberOfFrames attribute of the video object.

Parameters

knownNumberOfFrames
A known number of frames.

Usage Notes

Calling this method implicitly calls the setUpdateTime( ) method.

Pragmas

None.

Exceptions

NULL_INPUT_VALUE

This exception is raised if you call the setNumberOfFrames( ) method and the value for the knownNumberOfFrames parameter is NULL.

Examples

See the example in "setFrameSize( )" on page 9-55.

setProperties( )

Format

```
setProperties(ctx IN OUT RAW,
              setComments IN BOOLEAN);
```

Description

Reads the video data to get the values of the object attributes and then stores them in the object. For the known attributes that ORDVideo understands, it sets the properties for these attributes, which include: format, frame size, frame resolution, frame rate, video duration, number of frames, compression type, number of colors, and bit rate. It populates the comments field of the object with a rich set of format and application properties in XML form if the value of the setComments parameter is TRUE.

Parameters

ctx
The format plug-in context information.

setComments
If the value is TRUE, then the comments field of the object is populated with a rich set of format and application properties of the video object in XML form, identical to what is provided by the *inter*Media Annotator utility; otherwise, if the value is FALSE, the comments field of the object remains unpopulated. The default value is FALSE.

Usage Notes

If the property cannot be extracted from the media source, then the respective attribute is set to NULL.

If the format is set to NULL, then the setProperties( ) method uses the default format plug-in; otherwise, it uses your user-defined format plug-in.

Pragmas

None.

Exceptions

VIDEO_PLUGIN_EXCEPTION

This exception is raised if you call the setProperties( ) method and the video plug-in raises an exception when calling this method.

Examples

Set the property information for known video attributes:

```
DECLARE
  obj ORDSYS.ORDVideo;
  ctx RAW(4000) :=NULL;
BEGIN
  select vid into obj from TVID where N =1 for update;
  obj.setProperties(ctx,FALSE);
  update TVID set vid = obj where N = 1;
  COMMIT;
  EXCEPTION
    WHEN OTHERS THEN
     DBMS_OUTPUT.put_line('exception raised');
END;
/
```

setVideoDuration( )

Format

setVideoDuration(knownVideoDuration RETURN INTEGER);

Description

Sets the value of the videoDuration attribute of the video object.

Parameters

knownVideoDuration
A known video duration.

Usage Notes

Calling this method implicitly calls the setUpdateTime( ) method.

Pragmas

None.

Exceptions

NULL_INPUT_VALUE

This exception is raised if you call the setVideoDuration( ) method and the value for the knownVideoDuration parameter is NULL.

Examples

See the example in "setFrameSize( )" on page 9-55.

9.4 Packages or PL/SQL Plug-ins

This section presents reference information on the packages or PL/SQL plug-ins provided. Table 9–1 describes the PL/SQL plug-in packages provided in the ORDPLUGINS schema.

Table 9–1 PL/SQL Plug-ins Provided in the ORDPLUGINS Schema

PL/SQL Plug-in Packages	Audio Format	MIME Type
ORDPLUGINS.ORDX_DEFAULT_VIDEO	<format>	Dependent on file format
ORDPLUGINS.ORDX_AVI_VIDEO	AVI	video/x-msvideo
ORDPLUGINS.ORDX_MOOV_VIDEO	MOOV	video/quicktime
ORDPLUGINS.ORDX_RMFF_VIDEO	RMFF	audio/x-pn-realaudio

Section 9.4.1 describes the ORDPLUGINS.ORDX_DEFAULT_VIDEO package, the methods supported, and the level of support. Note that the methods supported and the level of support for the other PL/SQL plug-in packages described in Table 9–1 are identical for all plug-in packages, therefore, refer to Section 9.4.1.

9.4.1 ORDPLUGINS.ORDX_DEFAULT_VIDEO Package

Use the following provided ORDPLUGINS.ORDX_DEFAULT_VIDEO package as a guide in developing your own ORDPLUGINS.ORDX_<format>_VIDEO video format package. This package sets the mimeType field in the setProperties( ) method with a MIME type value that is dependent on the file format.

```
CREATE OR REPLACE PACKAGE ORDX_DEFAULT_VIDEO
authid current_user
AS
--VIDEO ATTRIBUTES ACCESSORS
FUNCTION  getFormat(ctx IN OUT RAW, obj IN ORDSYS.ORDVideo)
          RETURN VARCHAR2;
FUNCTION  getAttribute(ctx IN OUT RAW,
                       obj IN ORDSYS.ORDVideo,
                       name IN VARCHAR2)
          RETURN VARCHAR2;
PROCEDURE getFrameSize(ctx IN OUT RAW,
                       obj IN ORDSYS.ORDVideo,
                       width OUT INTEGER,
                       height OUT INTEGER);
FUNCTION  getFrameResolution(ctx IN OUT RAW, obj IN ORDSYS.ORDVideo)
            RETURN INTEGER;
```

```
FUNCTION  getFrameRate(ctx IN OUT RAW, obj IN ORDSYS.ORDVideo)
          RETURN INTEGER;
FUNCTION  getVideoDuration(ctx IN OUT RAW, obj IN ORDSYS.ORDVideo)
          RETURN INTEGER;
FUNCTION  getNumberOfFrames(ctx IN OUT RAW, obj IN ORDSYS.ORDVideo)
          RETURN INTEGER;
FUNCTION  getCompressionType(ctx IN OUT RAW, obj IN ORDSYS.ORDVideo)
          RETURN VARCHAR2;
FUNCTION  getNumberOfColors(ctx IN OUT RAW, obj IN ORDSYS.ORDVideo)
          RETURN INTEGER;
FUNCTION  getBitRate(ctx IN OUT RAW, obj IN ORDSYS.ORDVideo)
          RETURN INTEGER;
PROCEDURE setProperties(ctx IN OUT RAW,
                        obj IN OUT NOCOPY ORDSYS.ORDVideo,
                        setComments IN NUMBER := 0);
FUNCTION checkProperties(ctx IN OUT RAW,obj IN ORDSYS.ORDVideo) RETURN NUMBER;

-- must return name=value; name=value; ...  pairs
PROCEDURE getAllAttributes(ctx IN OUT RAW,
                           obj IN ORDSYS.ORDVideo,
                           attributes IN OUT NOCOPY CLOB);
-- VIDEO PROCESSING METHODS
FUNCTION  processCommand(
                          ctx       IN OUT RAW,
                          obj       IN OUT NOCOPY ORDSYS.ORDVideo,
                          cmd       IN VARCHAR2,
                          arguments IN VARCHAR2,
                          result    OUT RAW)
         RETURN RAW;
PRAGMA RESTRICT_REFERENCES(getFormat, WNDS, WNPS, RNDS, RNPS);
PRAGMA RESTRICT_REFERENCES(getAttribute, WNDS, WNPS, RNDS, RNPS);
PRAGMA RESTRICT_REFERENCES(getFrameSize, WNDS, WNPS, RNDS, RNPS);
PRAGMA RESTRICT_REFERENCES(getFrameResolution, WNDS, WNPS, RNDS, RNPS);
PRAGMA RESTRICT_REFERENCES(getFrameRate, WNDS, WNPS, RNDS, RNPS);
PRAGMA RESTRICT_REFERENCES(getVideoDuration, WNDS, WNPS, RNDS, RNPS);
PRAGMA RESTRICT_REFERENCES(getNumberOfFrames, WNDS, WNPS, RNDS, RNPS);
PRAGMA RESTRICT_REFERENCES(getCompressionType, WNDS, WNPS, RNDS, RNPS);
PRAGMA RESTRICT_REFERENCES(getNumberOfColors, WNDS, WNPS, RNDS, RNPS);
PRAGMA RESTRICT_REFERENCES(getBitRate, WNDS, WNPS, RNDS, RNPS);

END;
/
```

Table 9–2 shows the methods supported in the ORDPLUGINS.ORDX_DEFAULT_VIDEO package and the exceptions raised if you call a method that is not supported.

Table 9–2 Methods Supported in the ORDPLUGINS.ORDX_DEFAULT_VIDEO Package

Name of Method	Level of Support
getFormat	Supported; if the source is local, get the attribute and return the file format, but if the source is NULL, raise an ORDSYS.ORDSourceExceptions.EMPTY_SOURCE exception; otherwise, if the source is external, raise an ORDSYS.ORDAudioExceptions.LOCAL_DATA_SOURCE_REQUIRED exception.
getAttribute	Not supported - raises exceptions: METHOD_NOT_SUPPORTED and VIDEO_PLUGIN_EXCEPTION
getFrameSize	Supported; if the source is local, get the attribute and return the frame size, but if the source is NULL, raise an ORDSYS.ORDSourceExceptions.EMPTY_SOURCE exception; otherwise, if the source is external, raise an ORDSYS.ORDAudioExceptions.LOCAL_DATA_SOURCE_REQUIRED exception.
getFrameResolution	Not supported - raises exceptions: METHOD_NOT_SUPPORTED and VIDEO_PLUGIN_EXCEPTION
getFrameRate	Supported; if the source is local, get the attribute and return the frame rate, but if the source is NULL, raise an ORDSYS.ORDSourceExceptions.EMPTY_SOURCE exception; otherwise, if the source is external, raise an ORDSYS.ORDAudioExceptions.LOCAL_DATA_SOURCE_REQUIRED exception.
getVideoDuration	Supported; if the source is local, get the attribute and return the video duration, but if the source is NULL, raise an ORDSYS.ORDSourceExceptions.EMPTY_SOURCE exception; otherwise, if the source is external, raise an ORDSYS.ORDAudioExceptions.LOCAL_DATA_SOURCE_REQUIRED exception.
getNumberOfFrames	Supported; if the source is local, get the attribute and return the number of frames, but if the source is NULL, raise an ORDSYS.ORDSourceExceptions.EMPTY_SOURCE exception; otherwise, if the source is external, raise an ORDSYS.ORDAudioExceptions.LOCAL_DATA_SOURCE_REQUIRED exception.
getCompressionType	Supported; if the source is local, get the attribute and return the compression type, but if the source is NULL, raise an ORDSYS.ORDSourceExceptions.EMPTY_SOURCE exception; otherwise, if the source is external, raise an ORDSYS.ORDAudioExceptions.LOCAL_DATA_SOURCE_REQUIRED exception.

Table 9–2 Methods Supported in the ORDPLUGINS.ORDX_DEFAULT_VIDEO Package (Cont.)

Name of Method	Level of Support
getNumberOfColors	Supported; if the source is local, get the attribute and return the number of colors, but if the source is NULL, raise an ORDSYS.ORDSourceExceptions.EMPTY_SOURCE exception; otherwise, if the source is external, raise an ORDSYS.ORDAudioExceptions.LOCAL_DATA_SOURCE_REQUIRED exception.
getBitRate	Supported; if the source is local, get the attribute and return the bit rate, but if the source is NULL, raise an ORDSYS.ORDSourceExceptions.EMPTY_SOURCE exception; otherwise, if the source is external, raise an ORDSYS.ORDAudioExceptions.LOCAL_DATA_SOURCE_REQUIRED exception.
setProperties	Supported; if the source is local, process the local data and set the properties, but if the source is NULL, raise an ORDSYS.ORDSourceExceptions.EMPTY_SOURCE exception; if the source is a BFILE, then process the BFILE and set the properties; if the source is neither local nor a BFILE, get the media content into a temporary LOB, process the data, and set the properties.
checkProperties	Supported; if the source is local, process the local data and set the properties, but if the source is NULL, raise an ORDSYS.ORDSourceExceptions.EMPTY_SOURCE exception; if the source is a BFILE, then process the BFILE and set the properties; if the source is neither local nor a BFILE, get the media content into a temporary LOB, process the data, and set the properties.
getAllAttributes	Supported; if the source is local, get the attributes and return them, but if the source is NULL, raise an ORDSYS.ORDSourceExceptions.EMPTY_SOURCE exception; otherwise, if the source is external, raise an ORDSYS.ORDAudioExceptions.LOCAL_DATA_SOURCE_REQUIRED exception.
processCommand	Not supported - raises exceptions: METHOD_NOT_SUPPORTED and VIDEO_PLUGIN_EXCEPTION

9.4.2 Extending *inter*Media to Support a New Video Data Format

Extending *inter*Media to support a new video data format consists of four steps:

1. Design your new video data format.
2. Implement your new video data format and name it, for example, ORDX_MY_VIDEO.SQL.
3. Install your new ORDX_MY_VIDEO.SQL plug-in in the ORDPLUGINS schema.
4. Grant EXECUTE privileges on your new plug-in, for example, ORDX_MY_VIDEO.SQL plug-in, to PUBLIC.

Section 3.4.12 briefly describes how to extend *inter*Media to support a new video data format and describes the interface. A package body listing is provided in Example 9–1 to assist you in this operation. Add your variables to the places that say "--Your variables go here" and add your code to the places that say "--Your code goes here".

See Section F.4 for more information on installing your own video format plug-in and running the sample scripts provided.

Example 9–1 Show the Package Body for Extending Support to a New Video Data Format

```
CREATE OR REPLACE PACKAGE BODY ORDX_MY_VIDEO
AS
  --VIDEO ATTRIBUTES ACCESSORS
  FUNCTION  getFormat(ctx IN OUT RAW, obj IN ORDSYS.ORDVideo)
  RETURN VARCHAR2
  IS
--Your variables go here
  BEGIN
--Your code goes here
  END;
  FUNCTION  getAttribute(ctx IN OUT RAW,
                         obj IN ORDSYS.ORDVideo,
                         name IN VARCHAR2)
  RETURN VARCHAR2
  IS
--Your variables go here
  BEGIN
--Your code goes here
  END;
  PROCEDURE getFrameSize(ctx IN OUT RAW,
                         obj IN ORDSYS.ORDVideo,
                         width OUT INTEGER,
                         height OUT INTEGER)
  IS
--Your variables go here
  BEGIN
--Your code goes here
  END;
  FUNCTION  getFrameResolution(ctx IN OUT RAW, obj IN ORDSYS.ORDVideo)
  RETURN INTEGER
  IS
--Your variables go here
  BEGIN
```

```
--Your code goes here
  END;
  FUNCTION  getFrameRate(ctx IN OUT RAW, obj IN ORDSYS.ORDVideo)
  RETURN INTEGER
  IS
--Your variables go here
  BEGIN
--Your code goes here
  END;
  FUNCTION  getVideoDuration(ctx IN OUT RAW, obj IN ORDSYS.ORDVideo)
  RETURN INTEGER
  IS
--Your variables go here
  BEGIN
--Your code goes here
  END;
  FUNCTION  getNumberOfFrames(ctx IN OUT RAW, obj IN ORDSYS.ORDVideo)
  RETURN INTEGER
  IS
--Your variables go here
  BEGIN
--Your code goes here
  END;
  FUNCTION  getCompressionType(ctx IN OUT RAW, obj IN ORDSYS.ORDVideo)
  RETURN VARCHAR2
  IS
--Your variables go here
  BEGIN
--Your code goes here
  END;
  FUNCTION  getNumberOfColors(ctx IN OUT RAW, obj IN ORDSYS.ORDVideo)
  RETURN INTEGER
  IS
--Your variables go here
  BEGIN
--Your code goes here
  END;
  FUNCTION  getBitRate(ctx IN OUT RAW, obj IN ORDSYS.ORDVideo)
  RETURN INTEGER
  IS
--Your variables go here
  BEGIN
--Your code goes here
  END;
```

```
PROCEDURE setProperties(ctx IN OUT RAW,
          obj IN OUT NOCOPY ORDSYS.ORDVideo,
          setComments IN NUMBER :=0)
  IS
--Your variables go here
  BEGIN
--Your code goes here
  END;
  FUNCTION checkProperties(ctx IN OUT RAW, obj IN ORDSYS.ORDVideo) RETURN NUMBER
  IS
  IS
--Your variables go here
  BEGIN
--Your code goes here
  END;
  PROCEDURE getAllAttributes(ctx IN OUT RAW,
                             obj IN ORDSYS.ORDVideo,
                             attributes IN OUT NOCOPY CLOB)
  IS
--Your variables go here
  BEGIN
--Your code goes here
  END;
  -- VIDEO PROCESSING METHODS
  FUNCTION  processCommand(
                                ctx       IN OUT RAW,
                                obj       IN OUT NOCOPY ORDSYS.ORDVideo,
                                cmd       IN VARCHAR2,
                                arguments IN VARCHAR2,
                                result OUT RAW)
  RETURN RAW
  IS
--Your variables go here
  BEGIN
--Your code goes here
  END;
END;
/
show errors;
```

10

*inter*Media Relational Interface Reference

Application developers, who created multimedia applications without using the *inter*Media object types to store and manage media data in relational tables, and who do not want to migrate their existing multimedia applications to use *inter*Media objects, can use the *inter*Media relational interface for managing their media data. The *inter*Media relational interface consists of a set of methods for:

- Extracting information directly from their media data as either an XML string or as XML and individual attributes
- Processing and copying image data
- Loading media data into the Oracle database
- Exporting media data from the Oracle database into operating system files

The primary benefit of using the *inter*Media relational interface is to let application developers take advantage of *inter*Media functions with only minimal changes to their applications, and all without having to change their schemas to the *inter*Media objects to store their data.

The Oracle *inter*Media relational interface consists of a set of static methods (see Section 10.1) for the *inter*Media objects: ORDAudio, ORDDoc, ORDImage, and ORDVideo. Because these are static methods, no object is instantiated. Data is passed by method arguments rather than by object attributes.

The examples in this chapter assume that each of the media tables described in the respective sections of this chapter has been created and filled with data.

Methods related to the source of the media have ctx(RAW(4000)) as the first argument. Before calling any of these methods for the first time, the client must allocate the ctx structure and initialize it to NULL.

ORDAudio, ORDDoc, and ORDVideo methods related to media parsing have ctx (RAW(4000)) as the first argument. Before calling any of these methods for the first time, the client must allocate the ctx structure and initialize it to NULL.

10.1 Static Methods for the Relational Interface

This section presents reference information on the static methods for the relational interface. It is divided into subsections that describe those static methods (export( ), import( ), and importFrom( )) that are common to all object types and those static methods that are unique to a particular object type or implemented differently for the different object type.

10.1.1 Static Methods Common to All Object Types

The following static methods common to all object types for the relational interface are all associated with the source of the media.

export( ) -- copies data from a local source within an Oracle database to the specified external data source.

importFrom( ) -- transfers data from the specified external data source to the specified local source within an Oracle database.

importFrom( ) (all attributes) -- transfers data (all attributes) from the specified external data source to the specified local source within an Oracle database and returns format and mimeType if available.

10.1.2 Static Methods Uniquely Associated with Each Object Type

The following static methods (grouped by object type) for the relational interface are either unique to a particular object type or are implemented differently for each object type.

ORDAudio

getProperties( ) for BLOBs -- reads the BLOB data to get the values of the media attributes and then stores them in the input CLOB in XML form.

getProperties( ) for BLOBs (all attributes) -- reads the BLOB data to get the values of the media attributes and then stores them in the input CLOB in XML form and returns them as explicit parameters.

getProperties( ) for BFILES -- reads the BFILE data to get the values of the media attributes and then stores them in the input CLOB in XML form.

getProperties( ) for BFiles (all attributes) -- reads the BFILE data to get the values of the media attributes and then stores them in the input CLOB in XML form and returns them as explicit parameters.

ORDDoc

getProperties( ) for BLOBs -- reads the BLOB data to get the values of the media attributes and then stores them in the input CLOB in XML form.

getProperties( ) for BLOBs (all attributes) -- reads the BLOB data to get the values of the media attributes and then stores them in the input CLOB in XML form and returns them as explicit parameters.

getProperties( ) for BFILES -- reads the BFILE data to get the values of the media attributes and then stores them in the input CLOB in XML form.

getProperties( ) for BFILES (all attributes) -- reads the BFILE data to get the values of the media attributes and then stores them in the input CLOB in XML form and returns them as explicit parameters.

ORDImage

getProperties( ) for BLOBs -- reads the BLOB data to get the values of the media attributes and then stores them in the input CLOB in XML form.

getProperties( ) for BLOBs (all attributes) -- reads the BLOB data to get the values of the media attributes and then stores them in the input CLOB in XML form and returns them as explicit parameters.

getProperties( ) for BFILES -- reads the BFILE data to get the values of the media attributes and then stores them in the input CLOB in XML form.

getProperties( ) for BFILES (all attributes) -- reads the BFILE data to get the values of the media attributes and then stores them in the input CLOB in XML form and returns them as explicit parameters.

process( ) -- performs in-place image processing on an image stored in a BLOB.

processCopy( ) for BLOBs -- copies an image from a BLOB to the destination BLOB while performing image processing on the destination BLOB.

processCopy( ) for BFILEs -- copies an image from a BFILE to the destination BLOB while performing image processing on the destination BLOB.

ORDVideo

getProperties( ) for BLOBs -- reads the BLOB data to get the values of the media attributes and then stores them in the input CLOB in XML form.

getProperties( ) for BLOBs (all attributes) -- reads the BLOB data to get the values of the media attributes and then stores them in the input CLOB in XML form and returns them as explicit parameters.

getProperties( ) for BFILES -- reads the BFILE data to get the values of the media attributes and then stores them in the input CLOB in XML form.

getProperties( ) for BFILES (all attributes) -- reads the BFILE data to get the values of the media attributes and then stores them in the input CLOB in XML form and returns them as explicit parameters.

10.2 Static Methods Common to All Object Types

The examples in this section assume that you have created the test tables as described in Section 10.3.1, Section 10.4.1, Section 10.5.1, and Section 10.6.1, respectively for each object type.

This section presents reference information on the Oracle *inter*Media common static methods used for the relational interface.

export( )

Format

```
export(
    ctx              IN OUT RAW,
    local_data       IN BLOB,
    source_type      IN VARCHAR2,
    source_location  IN VARCHAR2,
    source_name      IN VARCHAR2);
```

Description

Copies data from a local source (local_data) within an Oracle database to an external data source.

> **Note:** The export( ) method natively supports only sources of source type file. User-defined sources may support the export( ) method.

Parameters

ctx
The source plug-in context information.

local_data
The BLOB location that is being exported.

source_type
The source type of the location to where the data is to be exported.

source_location
The location where the data is to be exported.

source_name
The name of the object to where the data is to be exported.

Usage Notes

After calling the export( ) method, you can issue a SQL DELETE statement or call the DBMS_LOB.TRIM procedure to delete the content stored locally, if desired.

The export( ) method for a source type of 'file' is similar to a file copy operation in that the original data stored in the BLOB is not touched other than for reading purposes.

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

For example, the following grants the user, MEDIAUSER, the permission to write to the file named filename.dat:

```
CALL DBMS_JAVA.GRANT_PERMISSION(
        'MEDIAUSER',
        'java.io.FilePermission',
        '/actual/server/directory/path/filename.dat',
        'write');
```

See the security and performance section in *Oracle9i Java Developer's Guide* for more information.

Pragmas

None.

Exceptions

ORDSourceExceptions.INCOMPLETE_SOURCE_INFORMATION

This exception is raised if you call the export( ) method and the value of srcType is NULL.

ORDSourceExceptions.METHOD_NOT_SUPPORTED

This exception is raised if you call the export( ) method and this method is not supported by the source plug-in being used.

ORDSourceExceptions.SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the export( ) method within a source plug-in when any other exception is raised.

ORDSourceExceptions.IO_ERROR

This exception is raised if the export( ) method encounters an error writing the BLOB data to the file specified.

See Appendix H for more information about these exceptions.

Examples

Export data from a local source to an external audio data source:

> **Note:** <*ORACLE_HOME*> must be replaced with your Oracle home and <*system-password*> with the system password.

```
CONNECT SYSTEM/<system-password>;
CREATE OR REPLACE DIRECTORY AUDIODIR AS 'e:\<ORACLE_HOME>\ord\aud\demo';
GRANT READ ON DIRECTORY AUDIODIR TO PUBLIC WITH GRANT OPTION;

CALL DBMS_JAVA.GRANT_PERMISSION(
    'MEDIAUSER',
    'java.io.FilePermission',
    'e:\<ORACLE_HOME>\ord\aud\demo\testaud.dat',
    'write');
CONNECT MEDIAUSER/MEDIAUSER;
DECLARE
  audio_data BLOB;
  ctx RAW(4000) :=NULL;
BEGIN
  SELECT aud INTO audio_data FROM taud WHERE N = 1;
  ORDSYS.ORDAudio.export(ctx,audio_data,'file','AUDIODIR','testaud.dat');
EXCEPTION
  WHEN OTHERS THEN
  RAISE;
END;
/
```

importFrom( )

Format

```
importFrom(ctx              IN OUT RAW,
           local_data       IN OUT NOCOPY BLOB,
           source_type      IN VARCHAR2,
           source_location  IN VARCHAR2,
           source_name      IN VARCHAR2);
```

Description

Transfers data from the specified external data source to a local source (local_data) within an Oracle database.

Parameters

ctx
The source plug-in context information.

local_data
The BLOB location to receive the data.

source_type
The source type of the data.

source_location
The location from where the data is to be imported.

source_name
The name of the data.

Usage Notes

You must ensure that the directory exists or is created before you use this method for file sources.

Pragmas

None.

Exceptions

ORDSourceExceptions.NULL_SOURCE

This exception is raised if you call the importFrom( ) method and the value of local_ data is NULL or has not been initialized.

ORDSourceExceptions.METHOD_NOT_SUPPORTED

This exception is raised if you call the importFrom( ) method and this method is not supported by the source plug-in being used.

ORDSourceExceptions.SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the importFrom( ) method within a source plug-in when any other exception is raised.

See Appendix H for more information about these exceptions.

Examples

Import document data from the specified external data source into the local source:

> **Note:** <*ORACLE_HOME*> must be replaced with your Oracle home.

```
CONNECT system/<system-password>;
CREATE OR REPLACE DIRECTORY DOCDIR AS 'e:\<ORACLE_HOME>\ord\doc\demo';
GRANT READ ON DIRECTORY DOCDIR TO PUBLIC WITH GRANT OPTION;

CONNECT MEDIAUSER/MEDIAUSER;

DECLARE
  document_data BLOB;
  ctx RAW(4000) :=NULL;
BEGIN
   SELECT document INTO document_data FROM tdoc WHERE N = 1 FOR UPDATE;
   ORDSYS.ORDDoc.importFrom(ctx,document_data,'file','DOCDIR','testimg.dat');
   UPDATE tdoc SET document = document_data WHERE N = 1;
   COMMIT;
   SELECT document INTO document_data FROM tdoc WHERE N = 2 FOR UPDATE;
   ORDSYS.ORDDoc.importFrom(ctx,document_data,'file','DOCDIR','testaud.dat');
   UPDATE tdoc SET document = document_data WHERE N = 2;
   COMMIT;
   SELECT document INTO document_data FROM tdoc WHERE N = 3 FOR UPDATE;
   ORDSYS.ORDDoc.importFrom(ctx,document_data,'file','DOCDIR','testvid.dat');
   UPDATE tdoc SET document = document_data WHERE N = 3;
```

```
   COMMIT;
EXCEPTION
   WHEN OTHERS THEN
   RAISE;
END;
/
```

importFrom( ) (all attributes)

Format

```
importFrom(ctx              IN OUT RAW,
           local_data       IN OUT NOCOPY BLOB,
           source_type      IN VARCHAR2,
           source_location  IN VARCHAR2,
           source_name      IN VARCHAR2,
           format           OUT VARCHAR2,
           mime_type        OUT VARCHAR2);
```

Description

Transfers data from the specified external data source to a local source (local_data) within an Oracle database.

Parameters

ctx
The source plug-in context information.

local_data
The BLOB location to receive the data.

source_type
The source type of the data.

source_location
The location from where the data is to be imported.

source_name
The name of the data.

format
The format of the data. The value is returned if it is available (from HTTP sources).

mime_type
The MIME type of the data. The value is returned if it is available (from HTTP sources).

Usage Notes

You must ensure that the directory exists or is created before you use this method for file sources.

Pragmas

None.

Exceptions

ORDSourceExceptions.NULL_SOURCE

This exception is raised if you call the importFrom( ) method and the value local_ data is NULL or has not been initialized.

ORDSourceExceptions.METHOD_NOT_SUPPORTED

This exception is raised if you call the importFrom( ) method and this method is not supported by the source plug-in being used.

ORDSourceExceptions.SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the importFrom( ) method within a source plug-in when any other exception is raised.

See Appendix H for more information about these exceptions.

Examples

Import image data from the specified external data source into the local source:

Note: <*ORACLE_HOME*> must be replaced with your Oracle home.

```
CONNECT system/<system-password>;
CREATE OR REPLACE DIRECTORY IMAGEDIR AS 'e:\<ORACLE_HOME>\ord\img\demo';
GRANT READ ON DIRECTORY IMAGEDIR TO PUBLIC WITH GRANT OPTION;

DECLARE
  image_data BLOB;
  ctx RAW(4000) :=NULL;
  img_format     VARCHAR2(32)  := NULL;
```

```
  img_mime_type  VARCHAR2(80);
BEGIN
   SELECT img INTO image_data FROM timg WHERE N = 1 FOR UPDATE;
   ORDSYS.ORDImage.importFrom(ctx,image_data,'file','IMAGEDIR','testimg.dat',img_format,img_mime_type);
   UPDATE timg SET img = image_data WHERE N = 1;
   COMMIT;
EXCEPTION
   WHEN OTHERS THEN
   RAISE;
END;
/
```

10.3 Static Methods Unique to the ORDAudio Object Type Relational Interface

This section presents reference information on the Oracle *inter*Media static methods unique to the ORDAudio relational interface.

The relational interface adds *inter*Media support to audio data stored in BLOBs and BFILEs rather than in the ORDAudio type. The following interface is defined in the ordaspec.sql file:

```
.
.
.
  -- Static Methods for the relational interface
  STATIC PROCEDURE export(ctx             IN OUT RAW,
                          local_data      IN BLOB,
                          source_type     IN VARCHAR2,
                          source_location IN VARCHAR2,
                          source_name     IN VARCHAR2),
  --
  STATIC PROCEDURE importFrom(ctx             IN OUT RAW,
                              local_data      IN OUT NOCOPY BLOB,
                              source_type     IN VARCHAR2,
                              source_location IN VARCHAR2,
                              source_name     IN VARCHAR2),
  --
  STATIC PROCEDURE importFrom(ctx             IN OUT RAW,
                              local_data      IN OUT NOCOPY BLOB,
                              source_type     IN VARCHAR2,
                              source_location IN VARCHAR2,
                              source_name     IN VARCHAR2,
                              format          OUT VARCHAR2,
```

```
                              mime_type       OUT VARCHAR2),
  --
  STATIC PROCEDURE getProperties(ctx            IN OUT RAW,
                                 audioBlob      IN BLOB,
                                 attributes     IN OUT NOCOPY CLOB,
                                 format         IN VARCHAR2),
  --
  STATIC PROCEDURE getProperties(ctx              IN OUT RAW,
                                 audioBlob        IN BLOB,
                                 attributes       IN OUT NOCOPY CLOB,
                                 mimeType         OUT VARCHAR2,
                                 format           IN OUT VARCHAR2,
                                 encoding         OUT VARCHAR2,
                                 numberOfChannels OUT INTEGER,
                                 samplingRate     OUT INTEGER,
                                 sampleSize       OUT INTEGER,
                                 compressionType  OUT VARCHAR2,
                                 audioDuration    OUT INTEGER),
  --
  STATIC PROCEDURE getProperties(ctx            IN OUT RAW,
                                 audioBfile     IN OUT NOCOPY BFILE,
                                 attributes     IN OUT NOCOPY CLOB,
                                 format         IN VARCHAR2),
  --
  STATIC PROCEDURE getProperties(ctx              IN OUT RAW,
                                 audioBfile       IN OUT NOCOPY BFILE,
                                 attributes       IN OUT NOCOPY CLOB,
                                 mimeType         OUT VARCHAR2,
                                 format           IN OUT VARCHAR2,
                                 encoding         OUT VARCHAR2,
                                 numberOfChannels OUT INTEGER,
                                 samplingRate     OUT INTEGER,
                                 sampleSize       OUT INTEGER,
                                 compressionType  OUT VARCHAR2,
                                 audioDuration    OUT INTEGER),
.
.
.
```

10.3.1 Example Table Definitions

The methods described in this section show examples based on a test audio table TAUD. Refer to the TAUD table definition that follows when reading through the examples:

TAUD Table Definition

```
CREATE TABLE taud(n                NUMBER,
                  aud              BLOB,
                  attributes       CLOB,
                  mimetype         VARCHAR2(4000),
                  format           VARCHAR2(31),
                  encoding         VARCHAR2(256),
                  numberofchannels INTEGER,
                  samplingrate     INTEGER,
                  samplesize       INTEGER,
                  compressiontype  VARCHAR2(4000),
                  audioduration    INTEGER)
STORAGE (INITIAL 100K NEXT 100K PCTINCREASE 0);

INSERT INTO taud VALUES(1,EMPTY_BLOB(),EMPTY_CLOB(), NULL, NULL, NULL, NULL,
NULL, NULL, NULL, NULL);
INSERT INTO taud VALUES(2,EMPTY_BLOB(),EMPTY_CLOB(), NULL, NULL, NULL, NULL,
NULL, NULL, NULL, NULL);
COMMIT;
```

getProperties( ) for BLOBs

Format

```
getProperties(ctx          IN OUT RAW,
              audioBlob    IN BLOB,
              attributes   IN OUT NOCOPY CLOB,
              format       IN VARCHAR2);
```

Description

Reads the audio BLOB data to get the values of the media attributes for supported formats and then stores them in the input CLOB. This method populates the CLOB with an extensive set of format and application properties in XML form.

Parameters

ctx
The format plug-in context information.

audioBlob
The audio data represented as a BLOB.

attributes
The CLOB to hold the XML attribute information generated by the getProperties( ) method. This CLOB is populated with an extensive set of format and application properties of the audio BLOB data in XML form.

format
The optional format of the audio data. If this parameter is specified, then the format plug-in for this format type is invoked.

Usage Notes

None.

Pragmas

None.

Exceptions

AUDIO_PLUGIN_EXCEPTION

This exception is raised if you call the getProperties( ) method and the audio plug-in raises an exception.

ORDSourceExceptions.EMPTY_SOURCE

This exception is raised when the source is local but the source is NULL.

Examples

Get the property information for known audio attributes:

```
DECLARE
  aud_attrib CLOB;
  ctx RAW(4000) :=NULL;
  aud_data BLOB;
  aud_format VARCHAR2(160) := NULL;
BEGIN
  SELECT aud,attributes INTO aud_data,aud_attrib FROM taud WHERE N =1 FOR UPDATE;
  ORDSYS.ORDAudio.getProperties(ctx,aud_data,aud_attrib,aud_format);
  DBMS_OUTPUT.put_line('Size of XML Annotations: ' ||
                TO_CHAR(DBMS_LOB.GETLENGTH(aud_attrib)));
  UPDATE taud SET aud=aud_data, attributes=aud_attrib WHERE N=1;
  COMMIT;
EXCEPTION
  WHEN OTHERS THEN
  RAISE;
END;
/
```

getProperties( ) (all attributes) for BLOBs

Format

```
getProperties(ctx                IN OUT RAW,
              audioBlob          IN BLOB,
              attributes         IN OUT NOCOPY CLOB,
              mimeType           OUT VARCHAR2,
              format             IN OUT VARCHAR2
              encoding           OUT VARCHAR2,
              numberOfChannels   OUT INTEGER,
              samplingRate       OUT INTEGER,
              sampleSize         OUT INTEGER,
              compressionType    OUT VARCHAR2,
              audioDuration      OUT INTEGER);
```

Description

Reads the audio BLOB data to get the values of the media attributes for supported formats and then stores them in the input CLOB as explicit parameters. This method gets the properties for the following attributes of the audio data: duration, MIME type, compression type, format, encoding type, number of channels, sampling rate, and sample size. It populates the CLOB with an extensive set of format and application properties in XML form.

Parameters

ctx
The format plug-in context information.

audioBlob
The audio data represented as a BLOB.

attributes
The CLOB to hold the XML attribute information generated by the getProperties( ) method. This CLOB is populated with an extensive set of format and application properties of the audio BLOB data in XML form.

mimeType
The MIME type of the audio data.

format
The optional format of the audio data. If this parameter is specified, then the format plug-in for this format type is invoked. If not specified, the derived format value is returned.

encoding
The encoding type of the audio data.

numberOfChannels
The number of channels in the audio data.

samplingRate
The sampling rate in samples per second at which the audio data was recorded.

sampleSize
The sample width or number of samples of audio in the data.

compressionType
The compression type of the audio data.

audioDuration
The total time required to play the audio data.

Usage Notes

If the property cannot be extracted from the media source, then the respective parameter is set to NULL.

Pragmas

None.

Exceptions

AUDIO_PLUGIN_EXCEPTION

This exception is raised if you call the getProperties( ) method and the audio plug-in raises an exception.

ORDSourceExceptions.EMPTY_SOURCE

This exception is raised when the source is local but the source is NULL.

Examples

Get the property information for known audio attributes:

```
DECLARE
  aud_attrib        CLOB;
  ctx               RAW(4000) :=NULL;
  aud_data          BLOB;
  mimeType          VARCHAR2(80);
  format            VARCHAR2(32);
  encoding          VARCHAR2(160);
  numberOfChannels  NUMBER;
  samplingRate      NUMBER;
  sampleSize        NUMBER;
  compressionType   VARCHAR2(160);
  audioDuration     NUMBER;
BEGIN
  SELECT aud, attributes, mimetype, format, encoding, numberofchannels, samplingrate,
samplesize, compressiontype, audioduration INTO aud_data, aud_attrib, mimeType, format,
encoding, numberOfChannels, samplingRate, sampleSize, compressionType, audioDuration FROM
taud WHERE N = 1 FOR UPDATE;

  ORDSYS.ORDAudio.getProperties(ctx, aud_data, aud_attrib, mimeType, format, encoding,
             numberOfChannels, samplingRate, sampleSize, compressionType, audioDuration);

  DBMS_OUTPUT.put_line('Size of XML Annotations ' ||
                  TO_CHAR(DBMS_LOB.GETLENGTH(aud_attrib)));
  DBMS_OUTPUT.put_line('mimeType: ' || mimeType );
  DBMS_OUTPUT.put_line('format: ' || format );
  DBMS_OUTPUT.put_line('encoding: ' || encoding );
  DBMS_OUTPUT.put_line('numberOfChannels: ' || numberOfChannels );
  DBMS_OUTPUT.put_line('samplingRate: ' || samplingRate );
  DBMS_OUTPUT.put_line('sampleSize: ' || sampleSize );
  DBMS_OUTPUT.put_line('compressionType: ' || compressionType );
  DBMS_OUTPUT.put_line('audioDuration: ' || audioDuration );
  UPDATE taud SET
              aud=aud_data,
              attributes=aud_attrib,
              mimetype=mimeType,
              format=format,
              encoding=encoding,
              numberofchannels=numberOfChannels,
              samplingrate=samplingRate,
```

```
              samplesize=sampleSize,
              compressiontype=compressionType,
              audioduration=audioDuration
            WHERE n=1;
  COMMIT;
EXCEPTION
  WHEN OTHERS THEN
  RAISE;
END;
/
```

getProperties( ) for BFILEs

Format

```
getProperties(ctx          IN OUT RAW,
              audioBfile   IN OUT NOCOPY BFILE,
              attributes   IN OUT NOCOPY CLOB,
              format       IN VARCHAR2);
```

Description

Reads the audio BFILE data to get the values of the media attributes for supported formats and then stores them in the input CLOB. This method populates the CLOB with an extensive set of format and application properties in XML form.

Parameters

ctx
The format plug-in context information.

audioBfile
The audio data represented as a BFILE.

attributes
The CLOB to hold the XML attribute information generated by the getProperties( ) method. This CLOB is populated with an extensive set of format and application properties of the audio BFILE data in XML form.

format
The optional format of the audio data. If this parameter is specified, then the format plug-in for this format type is invoked.

Usage Notes

None.

Pragmas

None.

Exceptions

AUDIO_PLUGIN_EXCEPTION

This exception is raised if you call the getProperties( ) method and the audio plug-in raises an exception.

ORDSourceExceptions.EMPTY_SOURCE

This exception is raised when the source is local but the source is NULL.

Examples

Get the property information for known audio attributes:

```
DECLARE
  aud_attrib CLOB;
  ctx RAW(4000) :=NULL;
  aud_data BFILE := BFILENAME('AUDIODIR','testaud.dat');
  aud_format VARCHAR2(160) := NULL;
BEGIN
  DBMS_LOB.CREATETEMPORARY(aud_attrib, FALSE, DBMS_LOB.CALL);
  ORDSYS.ORDAudio.getProperties(ctx, aud_data, aud_attrib, aud_format);

  DBMS_OUTPUT.put_line('Size of XML Annotations ' ||
                  TO_CHAR(DBMS_LOB.GETLENGTH(aud_attrib)));
EXCEPTION
  WHEN OTHERS THEN
  RAISE;
END;
/
```

getProperties( ) (all attributes) for BFILEs

Format

```
getProperties(ctx                IN OUT RAW,
              audioBfile         IN OUT NOCOPY BFILE,
              attributes         IN OUT NOCOPY CLOB,
              mimeType           OUT VARCHAR2,
              format             IN OUT VARCHAR2
              encoding           OUT VARCHAR2,
              numberOfChannels OUT INTEGER,
              samplingRate       OUT INTEGER,
              sampleSize         OUT INTEGER,
              compressionType    OUT VARCHAR2,
              audioDuration      OUT INTEGER);
```

Description

Reads the audio BFILE data to get the values of the media attributes for supported formats and then stores them in the input CLOB as explicit parameters. This method gets the properties for the following attributes of the audio data: duration, MIME type, compression type, format, encoding type, number of channels, sampling rate, and sample size. It populates the CLOB with an extensive set of format and application properties in XML form.

Parameters

ctx
The format plug-in context information.

audioBfile
The audio data represented as a BFILE.

attributes
The CLOB to hold the XML attribute information generated by the getProperties( ) method. This CLOB is populated with an extensive set of format and application

properties of the audio BFILE data in XML form, identical to what is provided by the *inter*Media Annotator utility.

mimeType
The MIME type of the audio data.

format
The optional format of the audio data. If this parameter is specified, then the format plug-in for this format type is invoked. If not specified, the derived format value is returned.

encoding
The encoding type of the audio data.

numberOfChannels
The number of channels in the audio data.

samplingRate
The sampling rate in samples per second at which the audio data was recorded.

sampleSize
The sample width or number of samples of audio in the data.

compressionType
The compression type of the audio data.

audioDuration
The total time required to play the audio data.

Usage Notes

If the property cannot be extracted from the media source, then the respective parameter is set to NULL.

Pragmas

None.

Exceptions

AUDIO_PLUGIN_EXCEPTION

This exception is raised if you call the getProperties( ) method and the audio plug-in raises an exception.

ORDSourceExceptions.EMPTY_SOURCE

This exception is raised when the source is local but the source is NULL.

Examples

Get the property information for known audio attributes:

```
DECLARE
  aud_attrib        CLOB;
  ctx               RAW(4000) :=NULL;
  data              BFILE := BFILENAME('AUDIODIR','testaud.dat');
  mimeType          VARCHAR2(80);
  format            VARCHAR2(32);
  encoding          VARCHAR2(160);
  numberOfChannels  NUMBER;
  samplingRate      NUMBER;
  sampleSize        NUMBER;
  compressionType   VARCHAR2(160);
  audioDuration     NUMBER;
BEGIN
  DBMS_LOB.CREATETEMPORARY(aud_attrib, FALSE, DBMS_LOB.CALL);

  ORDSYS.ORDAudio.getProperties(ctx, data, aud_attrib, mimeType, format, encoding,
              numberOfChannels, samplingRate, sampleSize, compressionType, audioDuration);

  DBMS_OUTPUT.put_line('Size of XML Annotations ' ||
                   TO_CHAR(DBMS_LOB.GETLENGTH(aud_attrib)));
  DBMS_OUTPUT.put_line('mimeType: ' || mimeType );
  DBMS_OUTPUT.put_line('format: ' || format );
  DBMS_OUTPUT.put_line('encoding: ' || encoding );
  DBMS_OUTPUT.put_line('numberOfChannels: ' || numberOfChannels );
  DBMS_OUTPUT.put_line('samplingRate: ' || samplingRate );
  DBMS_OUTPUT.put_line('sampleSize: ' || sampleSize );
  DBMS_OUTPUT.put_line('compressionType: ' || compressionType );
  DBMS_OUTPUT.put_line('audioDuration: ' || audioDuration );
EXCEPTION
  WHEN OTHERS THEN
  RAISE;
END;
/
```

10.4 Static Methods Unique to the ORDDoc Object Type Relational Interface

This section presents reference information on the Oracle *inter*Media static methods unique to the ORDDoc relational interface.

The relational interface adds *inter*Media support to audio, document, image, and video data stored in BLOBs and BFILEs rather than in the ORDDoc type. The following interface is defined in the orddspec.sql file:

```
.
.
.
  -- Static Methods for the relational interface
  STATIC PROCEDURE export(ctx             IN OUT RAW,
                          local_data      IN BLOB,
                          source_type     IN VARCHAR2,
                          source_location IN VARCHAR2,
                          source_name     IN VARCHAR2),
  --
  STATIC PROCEDURE importFrom(ctx             IN OUT RAW,
                              local_data      IN OUT NOCOPY BLOB,
                              source_type     IN VARCHAR2,
                              source_location IN VARCHAR2,
                              source_name     IN VARCHAR2),
  --
  STATIC PROCEDURE importFrom(ctx             IN OUT RAW,
                              local_data      IN OUT NOCOPY BLOB,
                              source_type     IN VARCHAR2,
                              source_location IN VARCHAR2,
                              source_name     IN VARCHAR2,
                              format          OUT VARCHAR2,
                              mime_type       OUT VARCHAR2),
  --
  STATIC PROCEDURE getProperties(ctx            IN OUT RAW,
                                 docBlob        IN BLOB,
                                 attributes     IN OUT NOCOPY CLOB,
                                 format         IN VARCHAR2),
  --
  STATIC PROCEDURE getProperties(ctx              IN OUT RAW,
                                 docBlob          IN BLOB,
                                 attributes       IN OUT NOCOPY CLOB,
                                 mimeType         OUT VARCHAR2,
                                 format           IN OUT VARCHAR2,
                                 contentLength    OUT INTEGER),
 --
  STATIC PROCEDURE getProperties(ctx            IN OUT RAW,
                                 docBfile       IN OUT NOCOPY BFILE,
                                 attributes     IN OUT NOCOPY CLOB,
                                 format         IN VARCHAR2),
  --
```

```
  STATIC PROCEDURE getProperties(ctx               IN OUT RAW,
                                 docBfile          IN OUT NOCOPY BFILE,
                                 attributes        IN OUT NOCOPY CLOB,
                                 mimeType          OUT VARCHAR2,
                                 format            IN OUT VARCHAR2,
                                 contentLength     OUT INTEGER),
.
.
.
```

10.4.1 Example Table Definitions

The methods described in this section show examples based on a test document table TDOC. Refer to the TDOC table definition that follows when reading through the examples:

TDOC Table Definition

```
CREATE TABLE tdoc(n             NUMBER,
                  document      BLOB,
                  attributes    CLOB,
                  mimetype      VARCHAR2(80),
                  format        VARCHAR2(80),
                  contentlength INTEGER)
STORAGE (INITIAL 100K NEXT 100K PCTINCREASE 0);

INSERT INTO tdoc VALUES(1, EMPTY_BLOB(), EMPTY_CLOB(), NULL, NULL, NULL);
INSERT INTO tdoc VALUES(2, EMPTY_BLOB(), EMPTY_CLOB(), NULL, NULL, NULL);
INSERT INTO tdoc VALUES(3, EMPTY_BLOB(), EMPTY_CLOB(), NULL, NULL, NULL);
INSERT INTO tdoc VALUES(4, EMPTY_BLOB(), EMPTY_CLOB(), NULL, NULL, NULL);
COMMIT;
```

getProperties( ) for BLOBs

Format

```
getProperties(ctx          IN OUT RAW,
              docBlob      IN BLOB,
              attributes   IN OUT NOCOPY CLOB,
              format       IN VARCHAR2);
```

Description

Reads the document BLOB data to get the values of the media attributes and then stores them in the input CLOB. This method populates the CLOB with an extensive set of format and application properties in XML form.

Parameters

ctx
The format plug-in context information.

docBlob
The document data represented as a BLOB.

attributes
The CLOB to hold the XML attribute information generated by the getProperties( ) method. This CLOB is populated with an extensive set of format and application properties of the document BLOB data in XML form.

format
The optional format of the document data. If this parameter is specified, then the format plug-in for this format type is invoked.

Usage Notes

None.

Pragmas

None.

Exceptions

DOC_PLUGIN_EXCEPTION

This exception is raised if you call the getProperties( ) method and the document plug-in raises an exception.

ORDSourceExceptions.EMPTY_SOURCE

This exception is raised when the source is local but the source is NULL.

Examples

Get the property information for known document attributes:

```
DECLARE
  doc_attrib CLOB;
  ctx RAW(4000) :=NULL;
  doc_data BLOB;
  doc_format VARCHAR2(160) := NULL;

BEGIN
    SELECT document,attributes INTO doc_data,doc_attrib FROM tdoc WHERE N = 1 FOR UPDATE;
   ORDSYS.ORDDoc.getProperties(ctx, doc_data, doc_attrib, doc_format);

   DBMS_OUTPUT.put_line('Size of XML Annotations ' ||
                   TO_CHAR(DBMS_LOB.GETLENGTH(doc_attrib)));
   UPDATE tdoc SET document=doc_data, attributes=doc_attrib WHERE N=1;
   COMMIT;
   EXCEPTION
      WHEN OTHERS THEN
      RAISE;
END;
/
```

getProperties( ) (all attributes) for BLOBs

Format

```
getProperties(ctx             IN OUT RAW,
              docBlob         IN BLOB,
              attributes      IN OUT NOCOPY CLOB,
              mimeType        OUT VARCHAR2,
              format          IN OUT VARCHAR2,
              contentLength   OUT INTEGER);
```

Description

Reads the document BLOB data to get the values of the media attributes and then stores them in the input CLOB as explicit parameters. This method gets the properties for the following attributes of the document data: MIME type, content length, and format. It populates the CLOB with an extensive set of format and application properties in XML form.

Parameters

ctx
The format plug-in context information.

docBlob
The document data represented as a BLOB.

attributes
The CLOB to hold the XML attribute information generated by the getProperties( ) method. This CLOB is populated with an extensive set of format and application properties of the document BLOB data in XML form.

mimeType
The MIME type of the document data.

format
The optional format of the document data. If this parameter is specified, then the format plug-in for this format type is invoked.

contentLength
The length in bytes of the content.

Usage Notes

If the property cannot be extracted from the media source, then the respective parameter is set to NULL.

Pragmas

None.

Exceptions

DOC_PLUGIN_EXCEPTION

This exception is raised if you call the getProperties( ) method and the document plug-in raises an exception.

ORDSourceExceptions.EMPTY_SOURCE

This exception is raised when the source is local but the source is NULL.

Examples

Get the property information for known document attributes:

```
DECLARE
  doc_attrib        CLOB;
  ctx               RAW(4000) :=NULL;
  doc_data          BLOB;
  doc_mimeType      VARCHAR2(80);
  doc_format        VARCHAR2(32);
  doc_contentLength NUMBER;
BEGIN
   SELECT document, attributes, mimetype, format, contentlength INTO doc_data, doc_attrib,
doc_mimeType, doc_format, doc_contentLength FROM tdoc WHERE N = 1 FOR UPDATE;

   ORDSYS.ORDDoc.getProperties(ctx, doc_data, doc_attrib,
                 doc_mimeType, doc_format, doc_contentLength);

   DBMS_OUTPUT.put_line('Size of XML Annotations ' ||
                  TO_CHAR(DBMS_LOB.GETLENGTH(doc_attrib)));
   DBMS_OUTPUT.put_line('mimeType: ' || doc_mimeType );
   DBMS_OUTPUT.put_line('format: ' || doc_format );
   DBMS_OUTPUT.put_line('contentLength: ' || doc_contentLength );
   UPDATE tdoc SET
     document=doc_data,
     attributes=doc_attrib,
```

```
      mimetype=doc_mimeType,
      format=doc_format,
      contentlength=doc_contentLength
     WHERE N=1;
    COMMIT;
EXCEPTION
    WHEN OTHERS THEN
    RAISE;
END;
/
```

getProperties( ) for BFILEs

Format

```
getProperties(ctx          IN OUT RAW,
              docBfile     IN OUT NOCOPY BFILE,
              attributes   IN OUT NOCOPY CLOB,
              format       IN VARCHAR2);
```

Description

Reads the document BFILE data to get the values of the media attributes and then stores them in the input CLOB. It populates the CLOB with an extensive set of format and application properties in XML form.

Parameters

ctx
The format plug-in context information.

docBfile
The document data represented as a BFILE.

attributes
The CLOB to hold the XML attribute information generated by the getProperties( ) method. This CLOB is populated with an extensive set of format and application properties of the document BFILE data in XML form.

format
The optional format of the document data. If this parameter is specified, then the format plug-in for this format type is invoked.

Usage Notes

None.

Pragmas

None.

Exceptions

DOC_PLUGIN_EXCEPTION

This exception is raised if you call the getProperties( ) method and the document plug-in raises an exception.

ORDSourceExceptions.EMPTY_SOURCE

This exception is raised when the source is local but the source is NULL.

Examples

Get the property information for known document attributes:

```
DECLARE
  doc_attrib CLOB;
  ctx RAW(4000) :=NULL;
  doc_data BFILE := BFILENAME('DOCDIR','testvid.dat');
  doc_format VARCHAR2(160) := NULL;
BEGIN
   DBMS_LOB.CREATETEMPORARY(doc_attrib, FALSE, DBMS_LOB.CALL);
   ORDSYS.ORDDoc.getProperties(ctx, doc_data, doc_attrib, doc_format);

   DBMS_OUTPUT.put_line('Size of XML Annotations ' ||
                   TO_CHAR(DBMS_LOB.GETLENGTH(doc_attrib)));
EXCEPTION
   WHEN OTHERS THEN
   RAISE;
END;
/
```

getProperties( ) (all attributes) for BFILEs

Format

```
getProperties(ctx              IN OUT RAW,
              docBfile         IN OUT NOCOPY BFILE,
              attributes       IN OUT NOCOPY CLOB,
              mimeType         OUT VARCHAR2,
              format           IN OUT VARCHAR2,
              contentLength    OUT INTEGER);
```

Description

Reads the document BFILE data to get the values of the media attributes for supported formats and then stores them in the input CLOB as explicit parameters. This method gets the properties for the following attributes of the document data: MIME type, content length, and format. It populates the CLOB with an extensive set of format and application properties in XML form.

Parameters

ctx
The format plug-in context information.

docBfile
The document data represented as a BFILE.

attributes
The CLOB to hold the XML attribute information generated by the getProperties( ) method. This CLOB is populated with an extensive set of format and application properties of the document BFILE data in XML form.

mimeType
The MIME type of the document data.

format
The optional format of the document data. If this parameter is specified, then the format plug-in for this format type is invoked. If not specified, the derived format is returned.

contentLength
The length in bytes of the content.

Usage Notes

If the property cannot be extracted from the media source, then the respective parameter is set to NULL.

Pragmas

None.

Exceptions

DOC_PLUGIN_EXCEPTION

This exception is raised if you call the getProperties( ) method and the document plug-in raises an exception.

ORDSourceExceptions.EMPTY_SOURCE

This exception is raised when the source is local but the source is NULL.

Examples

Get the property information for known document attributes:

```
DECLARE
  doc_attrib        CLOB;
  ctx               RAW(4000) :=NULL;
  doc_data          BFILE := BFILENAME('DOCDIR','testimg.dat');
  doc_mimeType      VARCHAR2(80);
  doc_format        VARCHAR2(32);
  doc_contentLength NUMBER;
BEGIN
   DBMS_LOB.CREATETEMPORARY(doc_attrib, FALSE, DBMS_LOB.CALL);
   ORDSYS.ORDDoc.getProperties(ctx, doc_data, doc_attrib,
                  doc_mimeType, doc_format, doc_contentLength);
   DBMS_OUTPUT.put_line('Size of XML Annotations ' ||
                   TO_CHAR(DBMS_LOB.GETLENGTH(doc_attrib)));
   DBMS_OUTPUT.put_line('mimeType: ' || doc_mimeType );
   DBMS_OUTPUT.put_line('format: ' || doc_format );
   DBMS_OUTPUT.put_line('contentLength: ' || doc_contentLength );
```

```
EXCEPTION
   WHEN OTHERS THEN
   RAISE;
END;
/
```

10.5 Static Methods Unique to the ORDImage Object Type Relational Interface

This section presents reference information on the Oracle *inter*Media static methods unique to the ORDImage relational interface.

The relational interface adds *inter*Media support to image data stored in BLOBs and BFILEs rather than in the ORDImage type. The following interface is defined in the ordispec.sql file:

```
.
.
.
  -- Static Methods for the relational interface
  STATIC PROCEDURE export(ctx             IN OUT RAW,
                          local_data      IN BLOB,
                          source_type     IN VARCHAR2,
                          source_location IN VARCHAR2,
                          source_name     IN VARCHAR2),
  --
  STATIC PROCEDURE importFrom(ctx             IN OUT RAW,
                              local_data      IN OUT NOCOPY BLOB,
                              source_type     IN VARCHAR2,
                              source_location IN VARCHAR2,
                              source_name     IN VARCHAR2),
  --
  STATIC PROCEDURE importFrom(ctx             IN OUT RAW,
                              local_data      IN OUT NOCOPY BLOB,
                              source_type     IN VARCHAR2,
                              source_location IN VARCHAR2,
                              source_name     IN VARCHAR2,
                              format          OUT VARCHAR2,
                              mime_type       OUT VARCHAR2),
  --
  STATIC PROCEDURE getProperties(imageBlob         IN BLOB,
                                 attributes        IN OUT NOCOPY CLOB,
                                 mimeType          OUT VARCHAR2,
                                 width             OUT INTEGER,
                                 height            OUT INTEGER,
```

```
                                fileFormat        OUT VARCHAR2,
                                contentFormat     OUT VARCHAR2,
                                compressionFormat OUT VARCHAR2,
                                contentLength     OUT INTEGER),
  --
  STATIC PROCEDURE getProperties(imageBlob         IN BLOB,
                                attributes        IN OUT NOCOPY CLOB),
  --
  STATIC PROCEDURE getProperties(imageBfile       IN OUT NOCOPY BFILE,
                                attributes        IN OUT NOCOPY CLOB,
                                mimeType          OUT VARCHAR2,
                                width             OUT INTEGER,
                                height            OUT INTEGER,
                                fileFormat        OUT VARCHAR2,
                                contentFormat     OUT VARCHAR2,
                                compressionFormat OUT VARCHAR2,
                                contentLength     OUT INTEGER),
  --
  STATIC PROCEDURE getProperties(imageBfile IN OUT NOCOPY BFILE,
                                attributes IN OUT NOCOPY CLOB),
  --
  STATIC PROCEDURE process(imageBlob IN OUT NOCOPY BLOB,
                           command   IN VARCHAR2),
  --
  STATIC PROCEDURE processCopy(imageBlob IN OUT NOCOPY BLOB,
                               command   IN VARCHAR2,
                               dest      IN OUT NOCOPY BLOB),
  --
  STATIC PROCEDURE processCopy(imageBfile IN OUT BFILE,
                               command    IN VARCHAR2,
                               dest       IN OUT NOCOPY BLOB),
.
.
.
```

10.5.1 Example Table Definitions

The methods described in this section show examples based on a test image table TIMG. Refer to the TIMG table definition that follows when reading through the examples:

TIMG Table Definition

```
CREATE TABLE timg(n NUMBER,
                  img BLOB,
```

```
                    attributes CLOB,
                    mimetype VARCHAR2(4000),
                    width INTEGER,
                    height INTEGER,
                    fileformat VARCHAR2(4000),
                    contentformat VARCHAR2(4000),
                    compressionformat VARCHAR2(4000),
                    contentlength INTEGER)
STORAGE (INITIAL 100K NEXT 100K PCTINCREASE 0);

INSERT INTO timg VALUES(1, EMPTY_BLOB(), EMPTY_CLOB(), NULL,
                        NULL, NULL, NULL, NULL, NULL, NULL);
INSERT INTO timg VALUES(2, EMPTY_BLOB(), EMPTY_CLOB(), NULL,
                        NULL, NULL, NULL, NULL, NULL, NULL);
COMMIT;
```

getProperties( ) for BLOBs

Format

```
getProperties(imageBlob      IN BLOB,
              attributes     IN OUT NOCOPY CLOB);
```

Description

Reads the image BLOB data to get the values of the media attributes for supported formats and then stores them in the input CLOB. This method populates the CLOB with a set of format properties in XML form.

Parameters

imageBlob
The image data represented as a BLOB.

attributes
The CLOB to hold the XML attribute information generated by the getProperties( ) method. This CLOB is populated with a set of format properties of the image BLOB data in XML form.

Usage Notes

None.

Pragmas

None.

Exceptions

ORDImageExceptions.NULL_CONTENT

This exception is raised when the content attribute is NULL.

Examples

Get the property information for known image attributes:

```
DECLARE
  img_attrib CLOB;
```

```
  img_data BLOB;
BEGIN
   SELECT img, attributes INTO img_data, img_attrib FROM timg WHERE N = 1 FOR UPDATE;
   ORDSYS.ORDImage.getProperties(img_data, img_attrib);

   DBMS_OUTPUT.put_line('Size of XML Annotations ' ||
                  TO_CHAR(DBMS_LOB.GETLENGTH(img_attrib)));
   UPDATE timg SET img=img_data, attributes=img_attrib WHERE N=1;
   COMMIT;
EXCEPTION
   WHEN OTHERS THEN
   RAISE;
END;
/
```

getProperties( ) (all attributes) for BLOBs

Format

```
getProperties(imageBlob          IN BLOB,
              attributes         IN OUT NOCOPY CLOB,
              mimeType           OUT VARCHAR2,
              width              OUT INTEGER,
              height             OUT INTEGER,
              fileFormat         OUT VARCHAR2,
              contentFormat      OUT VARCHAR2,
              compressionFormat  OUT VARCHAR2,
              contentLength      OUT INTEGER);
```

Description

Reads the image BLOB data to get the values of the media attributes for supported formats and then stores them in the input CLOB as explicit parameters. This method gets the properties for the following attributes of the image data: MIME type, width, height, file format, content format, compression format, and content length. It populates the CLOB with a set of format properties in XML form.

Parameters

imageBlob
The image data represented as a BLOB.

attributes
The CLOB to hold the XML attribute information generated by the getProperties( ) method. This CLOB is populated with a set of format properties of the image BLOB data in XML form.

mimeType
The MIME type of the image data.

width
The width of the image in pixels.

height
The height of the image in pixels.

fileFormat
The format of the image data.

contentFormat
The type of image (monochrome, and so forth).

compressionFormat
The compression algorithm used on the image data.

contentLength
The size of the on-disk image file in bytes.

Usage Notes

If the property cannot be extracted from the media source, then the respective parameter is set to NULL.

Pragmas

None.

Exceptions

ORDImageExceptions.NULL_CONTENT

This exception is raised when the content attribute is NULL.

Examples

Get the property information for known image attributes:

```
DECLARE
  img_data             BLOB;
  img_attrib        CLOB;
  mimeType          VARCHAR2(4000);
  width             NUMBER;
  height            NUMBER;
  fileFormat        VARCHAR2(32);
  contentFormat     VARCHAR2(4000);
  compressionFormat VARCHAR2(4000);
  contentLength     NUMBER;
BEGIN
   SELECT img, attributes, mimetype, width, height, fileformat, contentformat,
compressionformat, contentlength INTO img_data, img_attrib, mimeType, width, height,
```

```
fileFormat, contentFormat, compressionFormat, contentLength FROM timg WHERE N = 1 FOR
UPDATE;

  ORDSYS.ORDImage.getProperties(img_data, img_attrib,
        mimeType, width, height, fileFormat,
        contentFormat, compressionFormat, contentLength);

   DBMS_OUTPUT.put_line('Size of XML Annotations ' ||
                 TO_CHAR(DBMS_LOB.GETLENGTH(img_attrib)));
   DBMS_OUTPUT.put_line('mimeType: ' || mimeType );
   DBMS_OUTPUT.put_line('width: ' || width );
   DBMS_OUTPUT.put_line('height: ' || height );
   DBMS_OUTPUT.put_line('fileFormat: ' || fileFormat );
   DBMS_OUTPUT.put_line('contentFormat: ' || contentFormat );
   DBMS_OUTPUT.put_line('compressionFormat: ' || compressionFormat );
   DBMS_OUTPUT.put_line('contentLength: ' || contentLength );
   UPDATE timg SET
       img=img_data,
       attributes=img_attrib,
       mimetype=mimeType,
       width=width,
       height=height,
       fileformat=fileFormat,
       contentformat=contentFormat,
       compressionformat=compressionFormat,
       contentlength=contentLength
    WHERE N=1;
   COMMIT;
EXCEPTION
   WHEN OTHERS THEN
   RAISE;
END;
/
```

getProperties( ) for BFILEs

Format

```
getProperties(imageBfile      IN OUT NOCOPY BFILE,
              attributes      IN OUT NOCOPY CLOB);
```

Description

Reads the image BFILE data to get the values of the media attributes for supported formats and then stores them in the input CLOB. This method populates the CLOB with a set of format properties in XML form.

Parameters

imageBfile
The image data represented as a BFILE.

attributes
The CLOB to hold the XML attribute information generated by the getProperties( ) method. This CLOB is populated with a set of format properties of the image BFILE data in XML form.

Usage Notes

None.

Pragmas

None.

Exceptions

ORDImageExceptions.NULL_CONTENT

This exception is raised when the content attribute is NULL.

Examples

Get the property information for known image attributes:

```
DECLARE
  img_attrib CLOB;
```

```
  data BFILE := BFILENAME('IMAGEDIR','testimg.dat');
BEGIN
   DBMS_LOB.CREATETEMPORARY(img_attrib, FALSE, DBMS_LOB.CALL);
   ORDSYS.ORDImage.getProperties(data, img_attrib);

   DBMS_OUTPUT.put_line('Size of XML Annotations ' ||
                  TO_CHAR(DBMS_LOB.GETLENGTH(img_attrib)));
EXCEPTION
   WHEN OTHERS THEN
   RAISE;
END;
/
```

getProperties( ) (all attributes) for BFILEs

Format

```
getProperties(imageBfile          IN OUT NOCOPY BFILE,
              attributes          IN OUT NOCOPY CLOB,
              mimeType            OUT VARCHAR2,
              width               OUT INTEGER,
              height              OUT INTEGER,
              fileFormat          OUT VARCHAR2,
              contentFormat       OUT VARCHAR2,
              compressionFormat   OUT VARCHAR2,
              contentLength       OUT INTEGER);
```

Description

Reads the image BFILE data to get the values of the media attributes for supported formats and then stores them in the input CLOB as explicit parameters. This method gets the properties for the following attributes of the image data: MIME type, width, height, file format, content format, compression format, and content length. It populates the CLOB with a set of format properties in XML form.

Parameters

imageBfile
The image data represented as a BFILE.

attributes
The CLOB to hold the XML attribute information generated by the getProperties( ) method. This CLOB is populated with a set of format properties of the image BFILE data in XML form.

mimeType
The MIME type of the image data.

width
The width of the image in pixels.

height
The height of the image in pixels.

fileFormat
The format of the image data.

contentFormat
The type of image (monochrome, and so forth).

compressionFormat
The compression algorithm used on the image data.

contentLength
The size of the on-disk image file in bytes.

Usage Notes

If the property cannot be extracted from the media source, then the respective parameter is set to NULL.

Pragmas

None.

Exceptions

ORDImageExceptions.NULL_CONTENT

This exception is raised when the content attribute is NULL.

Examples

Get the property information for known image attributes:

```
DECLARE
  img_data          BFILE := BFILENAME('IMAGEDIR','testimg.dat');
  img_attrib        CLOB;
  mimeType          VARCHAR2(80);
  width             NUMBER;
  height            NUMBER;
  fileFormat        VARCHAR2(32);
  contentFormat     VARCHAR2(4000);
  compressionFormat VARCHAR2(4000);
  contentLength     NUMBER;
BEGIN
   DBMS_LOB.CREATETEMPORARY(img_attrib, FALSE, DBMS_LOB.CALL);
```

```
  ORDSYS.ORDImage.getProperties(img_data, img_attrib,
         mimeType, width, height, fileFormat,
         contentFormat, compressionFormat, contentLength);

   DBMS_OUTPUT.put_line('Size of XML Annotations ' ||
                  TO_CHAR(DBMS_LOB.GETLENGTH(img_attrib)));
   DBMS_OUTPUT.put_line('mimeType: ' || mimeType );
   DBMS_OUTPUT.put_line('width: ' || width );
   DBMS_OUTPUT.put_line('height: ' || height );
   DBMS_OUTPUT.put_line('fileFormat: ' || fileFormat );
   DBMS_OUTPUT.put_line('contentFormat: ' || contentFormat );
   DBMS_OUTPUT.put_line('compressionFormat: ' || compressionFormat );
   DBMS_OUTPUT.put_line('contentLength: ' || contentLength );
EXCEPTION
   WHEN OTHERS THEN
   RAISE;
END;
/
```

process( )

Format

```
process(imageBlob IN OUT NOCOPY BLOB,
        command IN VARCHAR2);
```

Description

Performs one or more image processing operations on a BLOB, writing the image back onto itself.

Parameters

imageBlob
The image data represented as a BLOB.

command
A list of image processing operations to perform on the image.

Usage Notes

You can change one or more of the image attributes shown in Table 8–1. Table 8–2 shows additional changes that can be made only to raw pixel and foreign images. See Appendix D for more information on process( ) method operators.

The process( ) method changes image attributes, therefore if you are storing image attributes, you should call the getProperties( ) method after calling the process( ) method.

Pragmas

None.

Exceptions

DATA_NOT_LOCAL

This exception is raised if you call the process( ) method and the imageBlob parameter is not initialized.

Examples

Example 1: Change the file format of an image in the image_data BLOB to GIF:

```
ORDSYS.ORDImage.process(image_data, 'fileFormat=GIFF');
```

Example 2: Change the image in the image_data BLOB to use higher quality JPEG compression and double the length of the image along the X-axis:

```
ORDSYS.ORDImage.process(image_data, 'compressionFormat=JPEG,
          compressionQuality=MAXCOMPRATIO, xScale="2.0"');
```

Note that changing the length on only one axis (for example, xScale=2.0) does not affect the length on the other axis, and would result in image distortion. Also, only the xScale and yScale parameters can be combined in a single scale operation. Any other combinations of scale operators result in an error.

Example 3: The maxScale and fixedScale operators are especially useful for creating thumbnail images from various-sized originals. The following line creates at most a 32-by-32 pixel thumbnail image, preserving the original aspect ratio:

```
ORDSYS.ORDImage.process(image_data, 'maxScale=32 32');
```

Example 4: Convert the image to TIFF:

```
DECLARE
img_attrib CLOB;
image_data BLOB;
BEGIN
   SELECT img, attributes INTO image_data, img_attrib FROM timg WHERE N = 1 FOR
UPDATE;
   ORDSYS.ORDImage.process(image_data, 'fileFormat=TIFF');
   ORDSYS.ORDImage.getProp(image_data, img_attrib);
   UPDATE timg SET img = image_data, attributes=img_attrib WHERE N = 1;
   COMMIT;
EXCEPTION
   WHEN OTHERS THEN
   RAISE;
END;
/
```

processCopy( ) for BLOBs

Format

```
processCopy(imageBlob IN BLOB,
            command  IN VARCHAR2,
            dest     IN OUT NOCOPY  BLOB);
```

Description

Copies an image stored internally to another image stored internally in a BLOB and processes the destination image.

Parameters

imageBlob
The source image data represented as a BLOB.

command
A list of image processing changes to make for the image in the new copy.

dest
The destination of the new image.

Usage Notes

See Table 8–1, "Image Processing Operators" and Table 8–2, "Additional Image Processing Operators for Raw Pixel and Foreign Images".

You cannot specify the same BLOB as both the source and destination.

Calling this method processes the image into the destination BLOB from any source BLOB.

The processCopy( ) method changes image attributes, therefore if you are storing image attributes, you should call the getProperties( ) method after calling the processCopy( ) method.

See Appendix D for more information on processCopy operators.

Pragmas

None.

Exceptions

DATA_NOT_LOCAL

This exception is raised if you call the processCopy( ) method and the imageBlob parameter is not initialized.

Examples

Copy an image, changing the file format, compression format, and content format in the destination image:

```
DECLARE
  dest_attrib      CLOB;
  image_data       BLOB;
  destination_data BLOB;
  the_Command      VARCHAR2(4000);
BEGIN
   SELECT img INTO image_data FROM timg WHERE N = 1;
   SELECT img, attributes INTO destination_data, dest_attrib FROM timg
      WHERE N = 2 FOR UPDATE;

   the_Command := 'fileFormat=tiff, compressionFormat=packbits,
           contentFormat=8bitlut';
   ORDSYS.ORDImage.processCopy(image_data, the_Command, destination_data);
   ORDSYS.ORDImage.getProperties(destination_data, dest_attrib);
   UPDATE timg SET img = destination_data, attributes=dest_attrib WHERE N = 2;
   COMMIT;
EXCEPTION
   WHEN OTHERS THEN
   RAISE;
END;
/
```

processCopy( ) for BFILEs

Format

```
processCopy(imageBfile IN OUT NOCOPY BFILE,
            command  IN VARCHAR2,
            dest     IN OUT NOCOPY BLOB);
```

Description

Copies an image stored externally to another image stored internally in a BLOB and processes the destination image.

Parameters

imageBfile
The image data represented as a BFILE.

command
A list of image processing changes to make for the image in the new copy.

dest
The destination of the new image.

Usage Notes

See Table 8–1, "Image Processing Operators" and Table 8–2, "Additional Image Processing Operators for Raw Pixel and Foreign Images".

Calling this method processes the image into the destination BLOB from any source BFILE.

The processCopy( ) method changes image attributes, therefore if you are storing image attributes, you should call the getProperties( ) method after calling the processCopy( ) method.

See Appendix D for more information on processCopy operators.

Pragmas

None.

Exceptions

NULL_DESTINATION

This exception is raised if you call the processCopy( ) method and the value of dest is NULL.

NULL_LOCAL_DATA

This exception is raised when source.localData is NULL.

Examples

Copy an image, generating a thumbnail of, at most, 32 x 32 pixels in the destination image:

```
DECLARE
  dest_attrib      CLOB;
  image_data       BFILE := BFILENAME('IMAGEDIR','testimg.dat');
  destination_data BLOB;
  the_Command      VARCHAR2(4000);
BEGIN
   SELECT img, attributes INTO destination_data, dest_attrib FROM timg
       WHERE N = 2 FOR UPDATE;

   the_Command := 'maxScale=32 32';
   ORDSYS.ORDImage.processCopy(image_data, the_Command, destination_data);
   ORDSYS.ORDImage.getProperties(destination_data, dest_attrib);
   UPDATE timg SET img = destination_data, attributes=dest_attrib WHERE N = 2;
   COMMIT;
EXCEPTION
   WHEN OTHERS THEN
   RAISE;
END;
/
```

10.6 Static Methods Unique to the ORDVideo Object Type Relational Interface

This section presents reference information on the Oracle *inter*Media static methods unique to the ORDVideo relational interface.

The relational interface adds *inter*Media support to video data stored in BLOBs and BFILEs rather than in the ORDVideo type. The following interface is defined in the ordvspec.sql file:

```
.
.
.
  -- Static Methods for the relational interface
  STATIC PROCEDURE export(ctx             IN OUT RAW,
                          local_data      IN BLOB,
                          source_type     IN VARCHAR2,
                          source_location IN VARCHAR2,
                          source_name     IN VARCHAR2),
  --
  STATIC PROCEDURE importFrom(ctx             IN OUT RAW,
                              local_data      IN OUT NOCOPY BLOB,
                              source_type     IN VARCHAR2,
                              source_location IN VARCHAR2,
                              source_name     IN VARCHAR2),
 --
  STATIC PROCEDURE importFrom(ctx             IN OUT RAW,
                              local_data      IN OUT NOCOPY BLOB,
                              source_type     IN VARCHAR2,
                              source_location IN VARCHAR2,
                              source_name     IN VARCHAR2,
                              format          OUT VARCHAR2,
                              mime_type       OUT VARCHAR2),
  --
  STATIC PROCEDURE getProperties(ctx            IN OUT RAW,
                                 videoBlob      IN BLOB,
                                 attributes     IN OUT NOCOPY CLOB,
                                 format         IN VARCHAR2),
  --
  STATIC PROCEDURE getProperties(ctx              IN OUT RAW,
                                 videoBlob        IN BLOB,
                                 attributes       IN OUT NOCOPY CLOB,
                                 mimeType         OUT VARCHAR2,
                                 format           IN OUT VARCHAR2,
                                 width            OUT INTEGER,
                                 height           OUT INTEGER,
                                 frameResolution  OUT INTEGER,
                                 frameRate        OUT INTEGER,
                                 videoDuration    OUT INTEGER,
                                 numberOfFrames   OUT INTEGER,
                                 compressionType  OUT VARCHAR2,
                                 numberOfColors   OUT INTEGER,
                                 bitRate          OUT INTEGER),

 --
```

```
  STATIC PROCEDURE getProperties(ctx            IN OUT RAW,
                                 videoBfile     IN OUT NOCOPY BFILE,
                                 attributes     IN OUT NOCOPY CLOB,
                                 format         IN VARCHAR2),
  --
  STATIC PROCEDURE getProperties(ctx              IN OUT RAW,
                                 videoBfile       IN OUT NOCOPY BFILE,
                                 attributes       IN OUT NOCOPY CLOB,
                                 mimeType         OUT VARCHAR2,
                                 format           IN OUT VARCHAR2,
                                 width            OUT INTEGER,
                                 height           OUT INTEGER,
                                 frameResolution  OUT INTEGER,
                                 frameRate        OUT INTEGER,
                                 videoDuration    OUT INTEGER,
                                 numberOfFrames   OUT INTEGER,
                                 compressionType  OUT VARCHAR2,
                                 numberOfColors   OUT INTEGER,
                                 bitRate          OUT INTEGER),
.
.
.
```

10.6.1 Example Table Definitions

The methods described in this section show examples based on a test video table TVID. Refer to the TVID table definition that follows when reading through the examples:

TVID Table Definition

```
CREATE TABLE tvid(n NUMBER,
                  vid BLOB,
                  attributes CLOB,
                  mimetype VARCHAR2(4000),
                  format VARCHAR2(31),
                  width INTEGER,
                  height INTEGER,
                  frameresolution INTEGER,
                  framerate INTEGER,
                  videoduration INTEGER,
                  numberofframes INTEGER,
                  compressiontype VARCHAR2(4000),
                  numberofcolors INTEGER,
                  bitrate INTEGER)
```

```
STORAGE (INITIAL 100K NEXT 100K PCTINCREASE 0);

INSERT INTO tvid VALUES(1, EMPTY_BLOB(), EMPTY_CLOB(), NULL, NULL,
NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL);
INSERT INTO tvid VALUES(2, EMPTY_BLOB(), EMPTY_CLOB(), NULL, NULL,
NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL);
COMMIT;
```

getProperties( ) for BLOBs

Format

```
getProperties(ctx          IN OUT RAW,
              videoBlob    IN BLOB,
              attributes   IN OUT NOCOPY CLOB,
              format       IN VARCHAR2);
```

Description

Reads the video BLOB data to get the values of the media attributes for supported formats and then stores them in the input CLOB. This method populates the CLOB with an extensive set of format and application properties in XML form.

Parameters

ctx
The format plug-in context information.

videoBlob
The video data represented as a BLOB.

attributes
The CLOB to hold the XML attribute information generated by the getProperties( ) method. This CLOB is populated with an extensive set of format and application properties of the video BLOB data in XML form.

format
The optional format of the video data. If this parameter is specified, then the format plug-in for this format type is invoked.

Usage Notes

None.

Pragmas

None.

Exceptions

VIDEO_PLUGIN_EXCEPTION

This exception is raised if you call the getProperties( ) method and the video plug-in raises an exception.

ORDSourceExceptions.EMPTY_SOURCE

This exception is raised when the source is local but the source is NULL.

Examples

Get the property information for known video attributes:

```
DECLARE
  vid_attrib CLOB;
  ctx RAW(4000) :=NULL;
  vid_data BLOB;
  vid_format VARCHAR2(31) := NULL;
BEGIN
  SELECT vid, attributes INTO vid_data, vid_attrib FROM tvid WHERE N = 1 FOR UPDATE;
  ORDSYS.ORDVideo.getProperties(ctx, vid_data, vid_attrib, vid_format);

  DBMS_OUTPUT.put_line('Size of XML Annotations ' ||
                  TO_CHAR(DBMS_LOB.GETLENGTH(vid_attrib)));
  UPDATE tvid SET vid=vid_data, attributes=vid_attrib WHERE N=1;
  COMMIT;
EXCEPTION
  WHEN OTHERS THEN
  RAISE;
END;
/
```

getProperties( ) (all attributes) for BLOBs

Format

```
getProperties(ctx                IN OUT RAW,
              videoBlob          IN BLOB,
              attributes         IN OUT NOCOPY CLOB,
              mimeType           OUT VARCHAR2,
              format             IN OUT VARCHAR2
              width              OUT INTEGER,
              height             OUT INTEGER,
              frameResolution    OUT INTEGER,
              frameRate          OUT INTEGER,
              videoDuration      OUT INTEGER,
              numberOfFrames     OUT INTEGER,
              compressionType    OUT VARCHAR2,
              numberOfColors     OUT INTEGER,
              bitRate            OUT INTEGER);
```

Description

Reads the video BLOB data to get the values of the media attributes for supported formats and then stores them in the input CLOB as explicit parameters. This method gets the properties for the following attributes of the video data: MIME type, format, frame size, frame resolution, frame rate, video duration, number of frames, compression type, number of colors, and bit rate. It populates the CLOB with an extensive set of format and application properties in XML form.

Parameters

ctx
The format plug-in context information.

videoBlob
The video data represented as a BLOB.

attributes
The CLOB to hold the XML attribute information generated by the getProperties( ) method. This CLOB is populated with an extensive set of format and application properties of the video BLOB data in XML form.

mimeType
The MIME type of the video data.

format
The optional format of the video data. If this parameter is specified, then the format plug-in for this format type is invoked. If specified as NULL, the format of the video data is returned.

width
The width of the frame in pixels of the video data.

height
The height of the frame in pixels of the video data.

frameResolution
The number of pixels per inch of frames in the video data.

frameRate
The number of frames per second at which the video data was recorded.

videoDuration
The total time required to play the video data.

numberOfFrames
The total number of frames in the video data.

compressionType
The compression type of the video data.

numberOfColors
The number of colors in the video data.

bitRate
The bit rate in the video data.

Usage Notes

If the property cannot be extracted from the media source, then the respective parameter is set to NULL.

Pragmas

None.

Exceptions

VIDEO_PLUGIN_EXCEPTION

This exception is raised if you call the getProperties( ) method and the video plug-in raises an exception.

ORDSourceExceptions.EMPTY_SOURCE

This exception is raised when the source is local but the source is NULL.

Examples

Get the property information for known video attributes:

```
DECLARE
  vid_attrib        CLOB;
  ctx               RAW(4000) :=NULL;
  vid_data          BLOB;
  mimeType          VARCHAR2(80);
  format            VARCHAR2(32);
  width             NUMBER;
  height            NUMBER;
  frameResolution   NUMBER;
  frameRate         NUMBER;
  videoDuration     NUMBER;
  numberOfFrames    NUMBER;
  compressionType   VARCHAR2(160);
  numberOfColors    NUMBER;
  bitRate           NUMBER;
BEGIN
  SELECT vid, attributes, mimetype, format, width, height, frameresolution, framerate,
videoduration, numberofframes, compressiontype, numberofcolors, bitrate INTO vid_data,
vid_attrib, mimeType, format, width, height, frameResolution, frameRate, videoDuration,
numberOfFrames, compressionType, numberOfColors, bitRate FROM tvid WHERE N = 1;

  ORDSYS.ORDVideo.getProperties(ctx, vid_data, vid_attrib, mimeType, format,
          width, height, frameResolution, frameRate,
          videoDuration, numberOfFrames, compressionType, numberOfColors, bitRate);
```

```
  DBMS_OUTPUT.put_line('Size of XML Annotations ' ||
                  TO_CHAR(DBMS_LOB.GETLENGTH(vid_attrib)));
  DBMS_OUTPUT.put_line('mimeType: ' || mimeType );
  DBMS_OUTPUT.put_line('format: ' || format );
  DBMS_OUTPUT.put_line('width: ' || width );
  DBMS_OUTPUT.put_line('height: ' || height );
  DBMS_OUTPUT.put_line('frameResolution: ' || frameResolution );
  DBMS_OUTPUT.put_line('frameRate: ' || frameRate );
  DBMS_OUTPUT.put_line('videoDuration: ' || videoDuration );
  DBMS_OUTPUT.put_line('numberOfFrames: ' || numberOfFrames );
  DBMS_OUTPUT.put_line('compressionType: ' || compressionType );
  DBMS_OUTPUT.put_line('numberOfColors: ' || numberOfColors );
  DBMS_OUTPUT.put_line('bitRate: ' || bitRate );
  UPDATE tvid SET
        vid=vid_data,
        attributes=vid_attrib,
        mimetype=mimeType,
        format=format,
        width=width,
        height=height,
        frameresolution=frameResolution,
        framerate=frameRate,
        videoduration=videoDuration,
        numberofframes=numberOfFrames,
        compressiontype=compressionType,
        numberofcolors=numberOfColors,
        bitrate=bitRate
     WHERE N=1;
  COMMIT;
EXCEPTION
  WHEN OTHERS THEN
  RAISE;
END;
/
```

getProperties( ) for BFILEs

Format

```
getProperties(ctx          IN OUT RAW,
              videoBfile   IN OUT NOCOPY BFILE,
              attributes   IN OUT NOCOPY CLOB,
              format       IN VARCHAR2);
```

Description

Reads the video BFILE data to get the values of the media attributes for supported formats and then stores them in the input CLOB. This method populates the CLOB with an extensive set of format and application properties in XML form.

Parameters

ctx
The format plug-in context information.

videoBfile
The video data represented as a BFILE.

attributes
The CLOB to hold the XML attribute information generated by the getProperties( ) method. This CLOB is populated with an extensive set of format and application properties of the video BFILE data in XML form.

format
The optional format of the video data. If this parameter is specified, then the format plug-in for this format type is invoked.

Usage Notes

None.

Pragmas

None.

Exceptions

VIDEO_PLUGIN_EXCEPTION

This exception is raised if you call the getProperties( ) method and the video plug-in raises an exception.

ORDSourceExceptions.EMPTY_SOURCE

This exception is raised when the source is local but the source is NULL.

Examples

Get the property information for known video attributes:

```
DECLARE
  vid_attrib CLOB;
  ctx RAW(4000) :=NULL;
  vid_data BFILE := BFILENAME('VIDEODIR','testvid.dat');
  vid_format VARCHAR2(160) := NULL;
BEGIN
  DBMS_LOB.CREATETEMPORARY(vid_attrib, FALSE, DBMS_LOB.CALL);
  ORDSYS.ORDVideo.getProperties(ctx, vid_data, vid_attrib, vid_format);

  DBMS_OUTPUT.put_line('Size of XML Annotations ' ||
                  TO_CHAR(DBMS_LOB.GETLENGTH(vid_attrib)));
EXCEPTION
  WHEN OTHERS THEN
  RAISE;
END;
/
```

getProperties( ) (all attributes) for BFILEs

Format

```
getProperties(ctx                IN OUT RAW,
              videoBfile         IN OUT NOCOPY BFILE,
              attributes         IN OUT NOCOPY CLOB,
              mimeType           OUT VARCHAR2,
              format             IN OUT VARCHAR2,
              width              OUT INRTEGER,
              height             OUT INTEGER,
              frameResolution    OUT INTEGER,
              frameRate          OUT INTEGER,
              videoDuration      OUT INTEGER,
              numberOfFrames     OUT INTEGER,
              compressionType    OUT VARCHAR2,
              numberOfColors     OUT INTEGER,
              bitRate            OUT INTEGER);
```

Description

Reads the video BFILE data to get the values of the media attributes for supported formats and then stores them in the input CLOB as explicit parameters. This method gets the properties for the following attributes of the video data: MIME type, format, frame size, frame resolution, frame rate, video duration, number of frames, compression type, number of colors, and bit rate. It populates the CLOB with an extensive set of format and application properties in XML form.

Parameters

ctx
The format plug-in context information.

videoBfile
The video data represented as a BFILE.

attributes
The CLOB to hold the XML attribute information generated by the getProperties( ) method. This CLOB is populated with an extensive set of format and application properties of the video BFILE data in XML form.

mimeType
The MIME type of the video data.

format
The optional format of the video data. If this parameter is specified, then the format plug-in for this format type is invoked. If specified as NULL, the format of the video data is returned.

width
The width of the frame in pixels of the video data.

height
The height of the frame in pixels of the video data.

frameResolution
The number of pixels per inch of frames in the video data.

frameRate
The number of frames per second at which the video data was recorded.

videoDuration
The total time required to play the video data.

numberOfFrames
The total number of frames in the video data.

compressionType
The compression type of the video data.

numberOfColors
The number of colors in the video data.

bitRate
The bit rate in the video data.

Usage Notes

If the property cannot be extracted from the media source, then the respective parameter is set to NULL.

Pragmas

None.

Exceptions

VIDEO_PLUGIN_EXCEPTION

This exception is raised if you call the getProperties( ) method and the video plug-in raises an exception.

ORDSourceExceptions.EMPTY_SOURCE

This exception is raised when the source is local but the source is NULL.

Examples

Get the property information for known video attributes:

```
DECLARE
  vid_attrib        CLOB;
  ctx               RAW(4000) :=NULL;
  vid_data          BFILE := BFILENAME('VIDEODIR','testvid.dat');
  mimeType          VARCHAR2(80);
  format            VARCHAR2(32);
  width             NUMBER;
  height            NUMBER;
  frameResolution   NUMBER;
  frameRate         NUMBER;
  videoDuration     NUMBER;
  numberOfFrames    NUMBER;
  compressionType   VARCHAR2(160);
  numberOfColors    NUMBER;
  bitRate           NUMBER;
BEGIN
  DBMS_LOB.CREATETEMPORARY(vid_attrib, FALSE, DBMS_LOB.CALL);

  ORDSYS.ORDVideo.getProperties(ctx, vid_data, vid_attrib, mimeType, format,
         width, height, frameResolution, frameRate,
         videoDuration, numberOfFrames, compressionType, numberOfColors, bitRate);

  DBMS_OUTPUT.put_line('Size of XML Annotations ' ||
                  TO_CHAR(DBMS_LOB.GETLENGTH(vid_attrib)));
  DBMS_OUTPUT.put_line('mimeType: ' || mimeType );
```

```
  DBMS_OUTPUT.put_line('format: ' || format );
  DBMS_OUTPUT.put_line('width: ' || width );
  DBMS_OUTPUT.put_line('height: ' || height );
  DBMS_OUTPUT.put_line('frameResolution: ' || frameResolution );
  DBMS_OUTPUT.put_line('frameRate: ' || frameRate );
  DBMS_OUTPUT.put_line('videoDuration: ' || videoDuration );
  DBMS_OUTPUT.put_line('numberOfFrames: ' || numberOfFrames );
  DBMS_OUTPUT.put_line('compressionType: ' || compressionType );
  DBMS_OUTPUT.put_line('numberOfColors: ' || numberOfColors );
  DBMS_OUTPUT.put_line('bitRate: ' || bitRate );
EXCEPTION
  WHEN OTHERS THEN
  RAISE;
END;
/
```

11

Tuning Tips for the DBA

This chapter provides tuning tips for the Oracle DBA who wants to achieve more efficient storage and management of multimedia data in the database when using Oracle *inter*Media.

The goals of your *inter*Media application determine the resource needs and how those resources should be allocated. Because application development and design decisions have the greatest effect on performance, standard tuning methods must be applied to the system planning, design, and development phases of the project to achieve optimal results for your *inter*Media application in a production environment.

Multimedia data consists of a variety of media types including character text, images, audio clips, video clips, line drawings, and so forth. All these media types are typically stored in LOBs, in either internal LOBs (stored in an internal database tablespace) or in BFILEs (external LOBs in operating system files outside of the database tablespaces). This chapter discusses only the management of audio, image, and video data.

Internal LOBs consist of: CLOBs, NCLOBs, and BLOBs and can be up to 4 gigabytes in size. BFILEs can be as large as the operating system will allow up to a maximum of 4 gigabytes.

Oracle *inter*Media manages a variety of LOB types. The following general topics will help you to better manage your *inter*Media LOB data:

- Setting database initialization parameters
- Issues to consider in creating tables with *inter*Media objects containing LOBs
- Improving multimedia data INSERT performance in *inter*Media objects containing LOBs
- Putting into practice user guidelines for best performance results

- Improving *inter*Media LOB data retrieval and update performance

For more information about LOB partitioning, LOB tuning, and LOB buffering, see *Oracle9i Application Developer's Guide - Large Objects (LOBs)*, *Oracle Call Interface Programmer's Guide*, *Oracle9i Database Concepts*, and *Oracle9i Database Performance Guide and Reference*.

For information on restrictions to consider when using LOBs, see *Oracle9i Application Developer's Guide - Large Objects (LOBs).*

For guidelines on using the DIRECTORY feature in Oracle9*i*, see *Oracle9i Application Developer's Guide - Large Objects (LOBs).* This feature enables a simple, flexible, nonintrusive, and secure mechanism for the DBA to manage access to large files in the server file system.

11.1 Setting Database Initialization Parameters

The information that follows is an excerpt from *Oracle9i Database Performance Guide and Reference* and *Oracle9i Database Reference*, and is presented as an overview of the topic. Refer to *Oracle9i Database Performance Guide and Reference* and *Oracle9i Database Reference* for more information.

Database tuning of the Oracle instance consists of tuning the system global area (SGA). The SGA is used to store data in memory for fast access. The SGA consumes a portion of your system's physical memory. The SGA must be sufficiently large to keep your data in memory but neither too small nor so large that performance begins to degrade. Degrading performance occurs when the operating system begins to page unused information to disk to make room for new information needed in memory, or begins to temporarily swap active processes to disk so other processes needing memory can use it. Excessive paging and swapping can bring a system to a standstill. The goal in sizing the SGA is to size it for the data that must be kept in main memory to keep performance optimal. With this in mind, you must size the SGA required for your *inter*Media application. This may mean increasing the physical memory of your system and monitoring your operating system behavior to ensure paging and swapping remains minimal.

The size of the SGA is determined by the values of the following database initialization parameters: DB_BLOCK_SIZE, DB_CACHE_SIZE, SHARED_POOL_SIZE, and LOG_BUFFER.

Beginning with Oracle9*i*, the SGA infrastructure is dynamic. This means that the following primary parameters used to size the SGA can be changed while the instance is running:

- Buffer cache (DB_CACHE_SIZE) -- the size in bytes of the cache of standard blocks
- Shared pool (SHARED _POOL_SIZE) -- the size in bytes of the area devoted to shared SQL and PL/SQL statements
- Large pool (LARGE_POOL_SIZE) (default is 0 bytes) -- the size in bytes of the large pool used in shared server systems for session memory, parallel execution for message buffers, and by backup and restore processes for disk I/O buffers

The LOG_BUFFER parameter is used when buffering redo entries to a redo log. It is a static parameter and represents a very small portion of the SGA and can be changed only by stopping and restarting the database to read the changed value for this parameter from the initialization parameter file (init.ora).

Note that even though you cannot change the MAX_SGA_SIZE parameter value dynamically, you do have the option of changing any of its three dependent primary parameters (DB_CACHE_SIZE, SHARED_POOL_SIZE, and LARGE_POOL_SIZE) to make memory tuning adjustments on the fly. To help you specify an optimal cache value, you can use the dynamic DB_CACHE_ADVICE parameter with statistics gathering enabled to predict behavior with different cache sizes through the V$DB_CACHE_ADVICE performance view. Use the ALTER SYSTEM...SET clause... statement to enable this parameter. See *Oracle9i Database Performance Guide and Reference* for more information about using this parameter.

Beginning with Oracle9*i*, there is a concept of creating tablespaces with multiple block sizes and specifying cache sizes corresponding with each block size. The SYSTEM tablespace uses a standard block size and additional tablespaces can use up to five non-standard block sizes.

The standard block size is specified by the DB_BLOCK_SIZE parameter. Its cache size is specified by the DB_CACHE_SIZE parameter. Non-standard block sizes are specified by the BLOCKSIZE clause of the CREATE TABLESPACE statement. The cache size for each corresponding non-standard block size is specified using the notation: DB_*n*K_CACHE_SIZE parameter, where the value *n* is 2, 4, 8, 16, or 32 K bytes.

The standard block size, known as the default block size, is usually set to the same size in bytes as the operating system block size, or a multiple of this size. The DB_CACHE_SIZE parameter, known as the DEFAULT cache size, specifies the size of the cache of standard block size (default is 48M bytes). The system tablespace uses the standard block size and the DEFAULT cache size.

Either the standard block size or any of the non-standard block sizes and their associated cache sizes can be used for any of your other tablespaces. If you intend to

use multiple block sizes in your database storage design, you must specify at least the DB_CACHE_SIZE and one DB_*n*K_CACHE_SIZE parameter value. You must specify all sub-caches for all the other non-standard block sizes that you intend to use. This block size/cache sizing scheme lets you use up to five different non-standard block sizes for your tablespaces and lets you specify respective cache sizes for each corresponding block size. For example, you can size your system tablespace to the normal 2K or 4K bytes standard block size with a default DB_CACHE_SIZE of 48M bytes or whatever size you want to specify. Then you can use the remaining non-standard block sizes of 2K or 4K, 8K, 16K, or the maximum 32K bytes for storing your *inter*Media LOB data in appropriate block-sized tablespaces and respective caches to achieve optimal LOB storage and retrieval performance.

Because the DB_BLOCK_SIZE parameter value can be changed only by re-creating the database, the value for this parameter must be chosen carefully and remain unchanged for the life of the database. See the next section "DB_BLOCK_SIZE" for more information about this parameter.

The following sections describe these and some related initialization parameters and their importance to *inter*Media performance.

DB_BLOCK_SIZE

The DB_BLOCK_SIZE parameter is the size in bytes of Oracle database blocks (2048-32768). Oracle manages the storage space in the data files of a database in units called data blocks. The data block is the smallest unit of I/O operation used by a database; this value should be a multiple of the operating system's block size within the maximum (port-specific) limit to avoid unnecessary I/O operations. This parameter value is set for each Oracle database from the DB_BLOCK_SIZE parameter value in the initialization parameter file when you create the database. This value cannot be changed unless you create the database again.

The size of a database block determines how many rows of data Oracle can store in a single database page. The size of an average row is one piece of data that a DBA can use to determine the correct database block size. *inter*Media objects with instantiated LOB locators range in size from 175 bytes for ORDImage to 260 bytes for ORDAudio and ORDVideo. This figure does *not* include the size of the media data. (The difference in row sizes between instantiated image and audio and video data is that audio and video data contain a Comments attribute that is about 85 bytes in size to hold the LOB locator.)

If LOB data is less than 4000 bytes, then it can be stored in line or on the same database page as the rest of the row data. LOB data can be stored in line only when the block size is large enough to accommodate it.

LOB data that is stored out of line, on database pages that are separate from the row data, is accessed (read and written) by Oracle in CHUNK size pieces where CHUNK is specified in the LOB storage clause (see Section 11.2 for more information about the CHUNK option). CHUNK must be an integer multiple of DB_BLOCK_SIZE and defaults to DB_BLOCK_SIZE if not specified. Generally, it is more efficient for Oracle to access LOB data in large chunks, up to 32 KB. However, when LOB data is updated, it may be versioned (for read consistency) and logged both to the rollback segments and the redo log in CHUNK size pieces. If updates to LOB data are frequent then it may be more efficient space wise to manipulate smaller chunks of LOB data, especially when the granularity of the update is much less than 32 KB.

The preceding discussion is meant to highlight the differences between the initialization parameter DB_BLOCK_SIZE and the LOB storage parameter CHUNK. Each parameter controls different aspects of the database design, and though related, they should not be automatically equated.

Tuning Memory Allocation

Allocating memory to database structures and proper sizing of these structures can greatly improve database performance when working with LOB data. See *Oracle9i Database Performance Guide and Reference* for a comprehensive, in-depth presentation of this subject, including understanding memory allocation issues as well as detecting and solving memory allocation problems. The following sections describe a few of the important initialization parameters specifically useful for optimizing LOB performance relative to tuning memory allocation.

DB_CACHE_SIZE

The DB_CACHE_SIZE parameter specifies the size of the DEFAULT buffer pool for buffers in bytes. This value is the database buffer value that is displayed when you issue a SQL SHOW SGA statement. Because you cannot change the value of the DB_BLOCK_SIZE parameter without re-creating the database, change the value of the DB_CACHE_SIZE parameter to control the size of the database buffer cache using the ALTER SYSTEM...SET clause... statement. The DB_CACHE_SIZE parameter is dynamic.

BUFFER_POOL_KEEP and BUFFER_POOL_RECYCLE - Tuning Multiple Buffer Pools Using the Standard Block Size

To greatly reduce I/O operations while reading and processing LOB data, tune the database instance by partitioning your buffer cache into multiple buffer pools for the tables containing the LOB columns.

> **Note:** Multiple buffer pools are available only for the standard block size. Non-standard block size caches have a single DEFAULT pool. Therefore, the information presented in this section applies to only the scenario in which you are using only the standard block size.

By default, all tables are assigned to the DEFAULT pool. Tune this main cache buffer using the DB_CACHE_SIZE initialization parameter and assign the appropriate tables to the keep pool using the DB_KEEP_CACHE_SIZE initialization parameter and to the recycle pool using the DB_RECYCLE_CACHE_SIZE initialization parameter.

The keep pool contains buffers that always stay in memory and is intended for frequently accessed tables that contain important data. The recycle pool contains buffers that can always be recycled and is intended for infrequently accessed tables that contain much less important data. The size of the main buffer cache (DEFAULT) is calculated from the value specified for the DB_CACHE_SIZE parameter minus the values specified for the DB_KEEP_CACHE_SIZE and DB_RECYCLE_CACHE_SIZE parameters. Tables are assigned to respective buffer pools (KEEP, RECYCLE, DEFAULT) using the STORAGE (buffer_pool) clause of the CREATE or ALTER TABLE statement. Determine what tables you want allocated to which of these memory buffers and the ideal size of each buffer when you implement your memory allocation design. These parameter values can be changed only in the initialization parameter file and take effect only after stopping and restarting the database.

When working with very large images, set the DB_CACHE_SIZE parameter to a large number for your Oracle instance. For example, to cache a 40MB image, set this parameter to a value of 48MB. Some general guidelines to consider when working with LOB data are:

- You should have enough buffers to hold the object, regardless of table LOB logging and cache settings. See Section 11.2 for more information.
- When using log files you should make the log files larger, otherwise, more time is spent waiting for log switches. See Section 11.2 for more information.
- If the same BLOB is to be accessed frequently, set the table LOB CACHE parameter to TRUE. See Section 11.2 for more information.
- Use a large page size (DB_BLOCK_SIZE) if the database is going to contain primarily large objects.

See *Oracle9i Database Performance Guide and Reference* for more information about tuning multiple buffer pools.

SHARED_POOL_SIZE

The SHARED_POOL_SIZE parameter specifies the size in bytes of the shared pool that contains the library cache of shared SQL requests, shared cursors, stored procedures, the dictionary cache, and control structures, Parallel Execution message buffers, and other cache structures specific to a particular instance configuration. This parameter value is dynamic. This parameter represents most of the variable size value that displays when you issue a SQL SHOW SGA statement. Specifying a large value improves performance in multi-user systems. A large value for example, accommodates the loading and execution of *inter*Media PL/SQL scripts and stored procedures; otherwise, execution plans are more likely to be swapped out. A large value can also accommodate many clients connecting to the server with each client connection using some shared pool space. However, when the shared pool is full, the server is unable to accept additional client connections.

SHARED_POOL_RESERVED_SIZE

The SHARED_POOL_RESERVED_SIZE parameter specifies the shared pool space that is reserved for large contiguous requests for shared pool memory. This static parameter should be set high enough to avoid performance degradation in the shared pool from situations where pool fragmentation forces Oracle to search for free chunks of unused pool to satisfy the current request.

Ideally, this parameter should be large enough to satisfy any request scanning for memory on the reserved list without flushing objects from the shared pool.

The default value is 5% of the shared pool size, while the maximum value is 50% of the shared pool size. For *inter*Media applications, a value at or close to the maximum can provide performance benefits.

LOG_BUFFER

The LOG_BUFFER parameter specifies the amount of memory, in bytes, used for buffering redo entries to the redo log file. Redo entries are written to the on disk log file when a transaction commits or when the LOG_BUFFER is full and space must be made available for new redo entries. Large values for LOG_BUFFER can reduce the number of redo log file I/O operations by allowing more data to be flushed per write. Large values can also eliminate the waits that occur when redo entries are flushed to make space in the log buffer pool. *inter*Media applications that have buffering enabled for the LOB data can generate large amounts of redo data when

media is inserted or updated. These applications would benefit from a larger LOG_BUFFER size. This is a static parameter.

11.2 Issues to Consider in Creating Tables with *inter*Media Column Objects Containing BLOBs

The following information provides some strategies to consider when you create tables with *inter*Media column objects containing BLOBs. You can explicitly indicate the tablespace and storage characteristics for each BLOB. These topics are discussed in more detail and with examples in *Oracle9i Application Developer's Guide - Large Objects (LOBs)*. The information that follows is excerpted from Chapter 2 and is briefly presented to give you an overview of the topic. Refer to *Oracle9i Application Developer's Guide - Large Objects (LOBs)* for more information.

11.2.1 Initializing Internal *inter*Media Column Objects Containing BLOBs to NULL or EMPTY

An *inter*Media column object containing a LOB value set to NULL has no locator. By contrast, an empty LOB stored in a table is a LOB of zero length that has a locator. So, if you select from an empty LOB column or attribute, you get back a locator, which you can use to fill the LOB with data using the OCI or DBMS_LOB routines or ORDxxx.import method.

Setting *inter*Media Column Objects Containing a BLOB to NULL

You may want to set the BLOB value to NULL upon inserting the row whenever you do not have the BLOB data at the time of the INSERT operation. In this case, you can issue a SELECT statement at some later time to obtain a count of the number of rows in which the value of the BLOB is NULL, and determine how many rows must be populated with BLOB data for that particular column object.

However, the drawback to this approach is that you must then issue a SQL UPDATE statement to reset the NULL BLOB column to EMPTY_BLOB( ). The point is that you cannot call the OCI or the PL/SQL DBMS_LOB functions on a BLOB that is NULL. These functions work only with a locator, and if the BLOB column is NULL, there is no locator in the row.

Setting an *inter*Media Column Object Containing a BLOB to EMPTY

If you do not want to set an *inter*Media column object containing a BLOB to NULL, another option is to set the BLOB value to EMPTY by using the EMPTY_BLOB( ) function in the INSERT statement. Even better, set the BLOB value to EMPTY by

using the EMPTY_BLOB( ) function in the INSERT statement, and use the RETURNING clause (thereby eliminating a round-trip that is necessary for the subsequent SELECT statement). Then, immediately call OCI, the import method, or the PL/SQL DBMS_LOB functions to fill the LOB with data. See *Oracle9i Application Developer's Guide - Large Objects (LOBs)* for an example.

11.2.2 Specifying Tablespace and Storage Characteristics for *inter*Media Column Objects Containing BLOBs

When you create tables and define *inter*Media column objects containing BLOBs, you can explicitly indicate the tablespace and storage characteristics for each BLOB. The following guidelines can help you fine-tune BLOB storage.

Tablespace

The best performance for *inter*Media column objects containing BLOBs can often be achieved by specifying storage for BLOBs in a tablespace that is different from the one used for the table that contains the *inter*Media object with a BLOB. See the ENABLE | DISABLE STORAGE IN ROW clause near the end of this section for further considerations on storing BLOB data inline or out of line. If many different LOBs are to be accessed frequently, it may also be useful to specify a separate tablespace for each BLOB or attribute in order to reduce device contention. Preallocate the tablespace to the required allocation size to avoid allocation when inserting BLOB data. See the *Oracle9i SQL Reference* manual for examples, specifically the CREATE TABLE statement and the LOB column example. See Example 11–1.

Example 11–1 assumes that you have already issued a CONNECT statement as a suitably privileged user. This example creates a separate tablespace, called MONTANA, that is used to store the *inter*Media column object containing BLOB data for the image column. Ideally, this tablespace would be located on its own high-speed storage device to reduce contention. Other image attributes and the imageID column are stored in the default tablespace. The initial allocation allows 100MB of storage space. The images to be inserted are about 20KB in size. To improve insert performance, NOCACHE and NOLOGGING options are specified along with a CHUNK size of 24KB.

Example 11–1 Create a Separate Tablespace to Store an interMedia Column Object Containing LOB Data

```
SVRMGR> CREATE TABLESPACE MONTANA DATAFILE 'montana.tbs' SIZE 400M;
Statement processed.
SVRMGR> CREATE TABLE images (imageID INTEGER ,image ORDSYS.ORDImage)
```

```
LOB (image.source.localData) STORE AS
       (
       TABLESPACE MONTANA
       STORAGE (
              INITIAL 100M
              NEXT 100M
              )
       CHUNK 24K
       NOCACHE NOLOGGING
       );
```

LOB Index and LOB_index_clause

The LOB index is an internal structure that is strongly associated with the LOB storage.

Note: The LOB_index_clause in the CREATE TABLE statement is deprecated beginning with release 8.1.5. Oracle generates an index for each LOB column and beginning with release 8.1.5, LOB indexes are system named and system managed. For information on how Oracle manages LOB indexes in tables migrated from earlier versions, see *Oracle9i Database Migration*.

PCTVERSION Option

When an *inter*Media column object containing a BLOB is modified, a new version of the BLOB page is made in order to support consistent reading of prior versions of the BLOB value.

PCTVERSION is the percent of all used LOB data space that can be occupied by old versions of LOB data pages. As soon as old versions of LOB data pages start to occupy more than the PCTVERSION amount of used LOB space, Oracle tries to reclaim the old versions and reuses them. In other words, PCTVERSION is the percentage of used LOB data blocks that is available for versions of old LOB data.

One way of approximating PCTVERSION is to set PCTVERSION = (% of LOBs updated at any given point in time) times (% of each LOB updated whenever a LOB is updated) times (% of LOBs being read at any given point in time). Allow for a percentage of LOB storage space to be used as old versions of LOB pages so users can get consistent read results of data that has been updated.

Setting PCTVERSION to twice the default allows more free pages to be used for old versions of data pages. Because large queries may require consistent reading of LOBs, it is useful to keep more old versions of LOB pages around. LOB storage may

increase if you increase the PCTVERSION value because Oracle will not be reusing free pages aggressively.

The more infrequent and smaller the LOB updates are, the less space that needs to be reserved for old versions of LOB data. If existing LOBs are known to be read-only, you could safely set PCTVERSION to 0% because there would never be any pages needed for old versions of data.

CACHE or NOCACHE Option

Use the CACHE option on *inter*Media column objects containing BLOBs if the same BLOB data is to be accessed frequently. The CACHE option puts the data into the database buffer and makes it accessible for subsequent read operations. If you specify CACHE, then LOGGING is used; you cannot have CACHE and NOLOGGING.

Use the NOCACHE option (the default) if BLOB data is to be read only once or infrequently, or if you have too much BLOB data to cache, or if you are reading lots of images but none more frequently than others.

See Example 11–1.

LOGGING or NOLOGGING Option

An example of when NOLOGGING is useful is with bulk loading or inserting of data. See Example 11–1. For instance, when loading data into the *inter*Media column objects containing BLOBs, if you do not care about redo logging and can just start the load over if it fails, set the BLOB data segment storage characteristics to NOCACHE NOLOGGING. This setting gives good performance for the initial loading of data. Once you have successfully completed loading the data, you can use the ALTER TABLE statement to modify the BLOB storage characteristics for the BLOB data segment to the desired storage characteristics for normal BLOB operations, such as CACHE or NOCACHE LOGGING.

CHUNK Option

Set the CHUNK option to the number of blocks of *inter*Media column objects containing BLOB data that are to be accessed at one time. That is, the number of blocks that are to be read or written using the object.readFromSource or object.writeToSource *inter*Media audio and video object methods or call, OCILobRead( ), OCILobWrite( ), DBMS_LOB.READ( ), or DBMS_LOB.WRITE( ) during one access of the BLOB value. Note that the default value for the CHUNK option is 1 Oracle block and does not vary across systems. If only 1 block of BLOB

data is accessed at a time, set the CHUNK option to the size of 1 block. For example, if the database block size is 2K, then set the CHUNK option to 2K.

Set the CHUNK option to the next largest integer multiple of database block size that is slightly larger than the audio, image, or video data size being inserted. Specifying a slightly larger CHUNK option allows for some variation in the actual sizes of the multimedia data and ensures that the benefit is realized. For large-sized media data, a general rule is to set the CHUNK option as large as possible. The maximum is 32K in Oracle9*i*. For example, if the database block size is 2K or 4K or 8K and the image data is mostly 21K in size, set the CHUNK option to 24K. See Example 11–1.

INITIAL and NEXT Parameters

If you explicitly specify the storage characteristics for the *inter*Media column object containing a BLOB, make sure that the INITIAL and NEXT parameters for the BLOB data segment storage are set to a size that is larger than the CHUNK size. For example, if the database block size is 2K and you specify a CHUNK value of 8K, make sure that the INITIAL and NEXT parameters are at least 8K, preferably higher (for example, at least 16K).

For LOB storage, Oracle automatically builds and maintains a LOB index that allows quick access to any chunk and thus any portion of a LOB. The LOB index gets the same storage extent parameter values as its LOBs. Consequently, to optimize LOB storage space, you should calculate the size of your LOB index size as well as the total storage space needed to store the media data including its overhead.

Assume that *N* files comprising of *M* total bytes of media data are to be stored and that the value *C* represents the size of the LOB chunk storage parameter. To calculate the total number of bytes Y needed to store the media data:

Y = M + (N*C)

The expression (N*C) accounts for the worst case in which the last chunk of each LOB contains a single byte. Therefore, an extra chunk is allowed for each file that is stored. On average, the last chunk will be half full.

To calculate the total number of bytes X to store the LOB index:

X = CEIL(M / C) * 32

The value 32 indicates that the LOB index requires roughly 32 bytes for each chunk that is stored.

The total storage space needed for the media data plus its LOB index is then X + Y.

The following two examples describe these calculations in detail.

Example 1: Assume you have 500 video clips comprising a total size of 250MB with an average size is 512K bytes. Assume a LOB chunk size of 32768 bytes. The total space needed for the media data is 250MB + (5000*32768) or 266MB. The overhead is 16MB or about 6.5% storage overhead. The total space needed to store the LOB index is CEIL(250MB / 32768) * 32 or 244KB. The total space needed to store the media data plus its LOB index is then about 266.6MB.

```
SQL> SELECT 250000000+(500*32768)+CEIL(250000000/32768)*32 FROM dual;

250000000+(500*32768)+CEIL(250000000/32768)*32
----------------------------------------------
                                     266628160
```

The following table definition could be used to store this amount of data:

```
CREATE TABLE video_items
(
  video_id          NUMBER,
  video_clip       ORDSYS.ORDVideo
)
-- storage parameters for table in general
TABLESPACE video1 STORAGE (INITIAL 1M NEXT 10M)
-- special storage parameters for the video content
LOB(video_clip.source.localdata) STORE AS
  (TABLESPACE video2 STORAGE (INITIAL 260K NEXT 270M)
   DISABLE STORAGE IN ROW NOCACHE NOLOGGING CHUNK 32768);
```

Example 2: Assume you have 5000 images comprising a total size of 274MB with an average size of 56K bytes. Because the average size of the images are smaller than the video clips in the preceding example, it is more space efficient to choose a smaller chunk size, for example 8192 bytes to store the data in the LOB. The total space needed for the media data is 274MB + (5000*8192) or 314MB. The overhead is about 40MB or about 15% storage overhead. The total space needed to store the LOB index is CEIL(274MB / 8192) * 32 or 1.05MB. The total space needed to store the media data plus its LOB index is then about 316MB.

```
SQL> SELECT 274000000+(5000*8192)+CEIL(274000000/8192)*32 FROM dual;

274000000+(5000*8192)+CEIL(274000000/8192)*32
---------------------------------------------
                                    316030336
```

The following table definition could be used to store this amount of data:

```
CREATE TABLE image_items
(
  image_id         NUMBER,
  image            ORDSYS.ORDImage
)
-- storage parameters for table in general
TABLESPACE image1 STORAGE (INITIAL 1M NEXT 10M)
-- special storage parameters for the image content
LOB(image.source.localdata) STORE AS
  (TABLESPACE image2 STORAGE (INITIAL 1200K NEXT 320M)
   DISABLE STORAGE IN ROW NOCACHE NOLOGGING CHUNK 8192);
```

When working with very large BLOBs on the order of 1 gigabyte in size, choose a proportionately large INITIAL and NEXT extent parameter size, for example an INITIAL value slightly larger than your calculated LOB index size and a NEXT value of 100 megabytes, to reduce the frequency of extent creation, or commit the transaction more often to reuse the space in the rollback segment; otherwise, if the number of extents is large, the rollback segment can become saturated.

PCTINCREASE Parameter

Set the PCTINCREASE parameter value to 0 to make the growth of new extent sizes more manageable. When working with very large BLOBs and the BLOB is being filled up piece by piece in a tablespace, numerous new extents are created in the process. If the extent sizes keep increasing by the default value of 50 percent each time one is created, extents will become unmanageably big and eventually will waste space in the tablespace.

MAXEXTENTS Parameter

Set the MAXEXTENTS parameter value to suit the projected size of the BLOB or set it to UNLIMITED for safety. That is, when MAXEXTENTS is set to UNLIMITED, extents will be allocated automatically as needed and this minimizes fragmentation.

ENABLE | DISABLE STORAGE IN ROW Clause

You use the ENABLE | DISABLE STORAGE IN ROW clause to indicate whether the *inter*Media column objects containing a BLOB should be stored inline (that is, in the row) or out of line. You may not alter this specification once you have made it: if you ENABLE STORAGE IN ROW, you cannot alter it to DISABLE STORAGE IN ROW or the reverse. The default is ENABLE STORAGE IN ROW.

The maximum amount of LOB data that will be stored in the row is the maximum VARCHAR size (4000). Note that this includes the control information as well as the

LOB value. If the user indicates that the LOB should be stored in the row, once the LOB value and control information are larger than 4000 bytes, the LOB value is automatically moved out of the row.

This suggests the following guideline: If the *inter*Media column object containing a BLOB is small (that is, less than 4000 bytes), then storing the BLOB data out of line will decrease performance. However, storing the BLOB in the row increases the size of the row. This has a detrimental impact on performance if you are doing a lot of base table processing, such as full table scans, multiple row accesses (range scans), or doing many UPDATE or SELECT statements to columns other than the *inter*Media column objects containing BLOBs. If you do not expect the BLOB data to be less than 4000 bytes, that is, if all BLOBs are big, then the default is the best choice because:

- The LOB data is automatically moved out of line once it gets bigger than 4000 bytes.
- Performance can be better if the BLOB data is small (less than 4000 bytes including control information) and is stored inline because the LOB locator and the BLOB data can be retrieved in the same buffer, thus reducing I/O operations.

11.2.3 Segment Attributes and Physical Attributes

The following physical attribute is important for optimum storage of BLOB data in the data block and consequently achieving optimum retrieval performance.

PCTFREE Parameter

The PCTFREE parameter specifies the percentage of space in each data block of the table or partition reserved for future updates to each row of the table. Setting this parameter to an appropriate value is useful for efficient inline storage of multimedia data. The default value is 10%.

Set this parameter to a high enough value to avoid row chaining or row migration. Because the INSERT statement for BLOBs requires an EMPTY_BLOB column object initialization followed by an UPDATE statement to load the BLOB data into the data block, you must set the PCTFREE parameter value to a proper value especially if the BLOB data will be stored inline. For example, row chaining can result after a row INSERT operation when insufficient space is reserved in the existing data block to store the entire row, including the inline BLOB data in the subsequent UPDATE operation. As a result, the row would be broken into multiple pieces and each piece stored in a separate data block. Consequently, more I/O operations would be needed to retrieve the entire row, including the BLOB data, resulting in poorer

performance. Row migration can also result if there is insufficient space in the data block to store the entire row during the initial INSERT operation, and thus the row is stored in another data block.

To make best use of the PCTFREE parameter, determine the average size of the BLOB data being stored inline in each row, and then determine the entire row size, including the inline BLOB data. Set the PCTFREE parameter value to allow for sufficient free space to store an entire row of data in the data block. For example, if you have a large number of thumbnail images that are about 3K bytes in size, and each row is about 3.8K bytes in size, and the database block size is 8K, set the value of PCTFREE to a value that ensures that two complete rows can be stored in each data block in the initial INSERT operation. This approach initially uses 1.6K bytes of space (0.8K bytes / row *2 rows) leaving 6.4K bytes of free space. Because two rows initially use 20% of the data block and 95% after an UPDATE operation and adding a third row would initially use 30% of the data block causing a chain to occur when the third row is updated, set the PCTRFEE parameter value to 75. This setting permits a maximum of two rows to be stored per data block and leaves sufficient space to update each row with its 3K image thumbnail leaving about 0.4K bytes free space minus overhead per data block.

11.2.4 Accommodating Temporary LOBs in the Buffer Cache

Temporary LOBs created when you have set the table LOB CACHE parameter to TRUE move through the buffer cache; otherwise, they are read directly from and written to disk if the CACHE parameter is set to FALSE.

Use durations for automatic cleanup to save time and effort. Let the database end a duration and free all temporary LOBs associated with a duration because this is more efficient than freeing each one explicitly.

Temporary LOBs create deep copies of themselves on assignments; that is, a new copy of the temporary LOB is created. Use the OCILobLocatorAssign( ) call to assign the source locator to the destination locator when assigning one LOB locator to another. If the source locator refers to a temporary LOB, specify the equals sign (=) in the assignment to ensure that the two LOB locator pointers refer to the same LOB locator; otherwise, the source temporary LOB is deep-copied and a destination locator is created to refer to the new deep copy of the temporary LOB.

You may also want to consider using pass-by reference semantics in PL / SQL or declare pointers to locators, because a pointer assignment does not cause a deep copy. Instead, it causes the pointer to point to the same thing. See the *PL/SQL User's Guide and Reference*, *Oracle9i Database Performance Guide and Reference*, and *Oracle Call Interface Programmer's Guide* for more information.

11.2.5 Using *inter*Media Column Objects Containing BLOBs in Table Partitions

Because you can partition tables containing *inter*Media column objects that have BLOBs, BLOB segments can be spread between several tablespaces to:

- Balance I/O load
- Make backup and recovery operations more manageable
- Make BLOB maintenance easier

*inter*Media column objects containing BLOB data can be partitioned to improve I/O problems and to better balance the I/O load across the data files of the tablespace containing the BLOB data. You can allocate data storage across devices to further improve performance in a practice known as striping. This permits multiple processes to access different portions of the table concurrently, without disk contention.

*inter*Media column objects containing BLOB data can be partitioned to tune database backup and recovery operations to make more efficient use of resources. For example, having two or more tablespaces that are partitioned lets you perform partial database backup and recovery operations on specific data files.

Similarly, tablespaces with *inter*Media column objects containing BLOBs can be partitioned for easy maintenance of the BLOB data. This is done by logically grouping BLOB data together into smaller partitions that are grouped by date, by subject, by category, and so forth. This makes it easier to add, merge, split, or delete partitions as needed, based on your application.

See *Oracle9i Application Developer's Guide - Large Objects (LOBs)* for examples and further discussion of each of these topics. See the *Oracle9i SQL Reference* manual for examples, specifically the CREATE TABLE statement and the Partitioned Table with LOB Columns example.

11.2.6 LOB Buffering for Client Applications

Use LOB buffering if you need to repeatedly read or write small pieces of *inter*Media column objects containing BLOB data to specific regions of the BLOB on the client. Typically, for releases of Oracle8*i* or higher, options, Web servers, and other applications may need to buffer the contents of one or more LOBs in the client address space. Using LOB buffering, you can use up to 512K bytes of buffered access. The advantages of LOB buffering include:

- Allowing deferred write operations to the server. Flushing several write operations in the LOB buffer to the server reduces the number of network

round-trips from the client application to the server, thereby improving overall LOB update performance.

- Reducing the overall number of *inter*Media column objects containing BLOB update operations on the server reduces the number of BLOB versions and amount of logging, which improves overall BLOB performance and disk space usage.

See *Oracle9i Application Developer's Guide - Large Objects (LOBs)* for further considerations and the use of LOB buffering.

11.3 Improving Multimedia Data INSERT Performance in *inter*Media Objects Containing LOBs

There are a number of bulk loading methods available for loading FILE data into *inter*Media objects containing BLOBs. These include:

- *inter*Media import( ) method in a PL/SQL stored procedure
- SQL*Loader (conventional path load and direct path load)
- OCILobLoadFromFile( ) relational function
- DBMS_LOB.LOADFROMFILE( ) procedure in the DBMS_LOB package
- Java loadDataFromFile( ) or loadDataFromInputStream( ) methods of *inter*Media Java Classes to load media data from a client file

Using *inter*Media Import( ) Method in a PL/SQL Stored Procedure

Example 11–2 shows the contents of the load1.bat file, which invokes SQL*Plus and runs the t1.sql procedure (Example 11–3). The db_block_size for this schema is 8K bytes.

Example 11–2 Show the Load1.bat File

```
sqlplus scott/tiger@intertcp @t1
```

Example 11–3 shows the contents of the t1.sql file. This procedure:

- Creates two tablespaces.
- Creates the image_items table and defines the physical properties of the table, specifically the physical attributes and LOB storage attributes.
- Partitions the table storage into each tablespace by range using the image_id value.

- Creates the load_image stored procedure that:
  - Declares a variable nxtseq defined as the ROWID data type.
  - Inserts a row into the image_items table and uses the INSERT RETURNING ROWID statement to return the ROWID value for fastest access to the row for loading the image BLOB data into the object columns of each row using the import method.
  - Sets the image attribute properties automatically (by means of the import operation) for each loaded image (note that thumbnail images are stored inline, and regular images are stored out of line).
  - Commits the update operation.

Example 11–3 Show the T1.SQL File

```
spool t1.log
set echo on
connect internal/internal

create tablespace Image_h default storage (initial 30m next 400m pctincrease 0)
   datafile 'h:\IMPB\Image_h.DBF'
   size 2501M reuse;

create tablespace Image_i default storage (initial 30m next 400m pctincrease 0)
   datafile 'i:\IMPB\Image_i.DBF'
   size 2501M reuse;

connect scott/tiger

drop table image_items;

create table image_items(
  image_id               number,-- constraint pl_rm primary key,
  image_title            varchar2(128),
  image_artist           varchar2(128),
  image_publisher        varchar2(128),
  image_description varchar2(1000),
  image_price            number(6,2),
  image_file_path varchar2(128),
  image_thumb_path varchar2(128),
  image_thumb            ordsys.ordimage,
  image_clip             ordsys.ordimage
)
--
```

```
-- physical properties of table
--
  -- physical attributes clause
  pctfree 35 storage (initial 30M next 400M pctincrease 0)

  -- LOB storage clause (applies to LOB column)
  LOB (image_clip.source.localdata)
      store as (disable storage in row nocache nologging chunk 32768)
--
-- table properties (applies to whole table)
--
Partition by range (image_id)
(
Partition Part1 values less than (110001)
Tablespace image_h,
Partition Part2 values less than (maxvalue)
Tablespace image_i
);

connect scott/tiger;

create or replace procedure load_image
(
  image_id           number,
  image_title        varchar2,
  image_artist       varchar2,
  image_publisher    varchar2,
  image_description  varchar2,
  image_price        number,
  image_file_path    varchar2,
  image_thumb_path   varchar2,
  thumb_dir          varchar2,
  content_dir        varchar2,
  file_name1         varchar2,
  file_name2         varchar2)
as
  ctx raw(4000) := NULL;
  obj1           ORDSYS.ORDIMAGE;
  obj2           ORDSYS.ORDIMAGE;
  nxtseq         rowid;

Begin
  Insert into image_items(
    image_id,
    image_title,
```

```
    image_artist,
    image_publisher,
    image_description,
    image_price,
    image_file_path,
    image_thumb_path ,
    image_thumb,
    image_clip)
 values (
    image_id,
    image_title,
    image_artist,
    image_publisher,
    image_description,
    image_price,
    image_file_path,
    image_thumb_path ,
    ORDSYS.ORDIMAGE.init('FILE',upper(thumb_dir),file_name1),
    ORDSYS.ORDIMAGE.init('FILE',upper(content_dir),file_name2))
    returning rowid into nxtseq;

-- load up the thumbnail image
  select t.image_thumb,
t.image_clip
  into obj1, obj2
  from image_items t
    where t.rowid = nxtseq for update;
  obj1.import(ctx);        -- import sets properties
  obj2.import(ctx);
  Update image_items I
  set I.image_thumb = obj1,
      I.image_clip  = obj2
    where i.rowid = nxtseq;

  Commit;
End;
/
spool off
set echo off
```

Example 11–4 shows the contents of the load1.sql file. The image load directories are created and specified for each tablespace and user scott is granted read privilege on each load directory. The stored procedure named load_image is then executed,

which loads values for each column row. By partitioning the data into different tablespaces, each partition can be loaded in a parallel data load operation.

Example 11–4 Show the Load1.sql File that Executes the load_image Stored Procedure

```
connect internal/internal
drop directory IMAGE_H;
drop directory IMAGE_I;
create directory IMAGE_H as 'h:\image_files';
create directory IMAGE_I as 'i:\image_files';
grant read on directory IMAGE_H to scott;
grant read on directory IMAGE_I to scott;
EXEC Load_image(100001,'T_100001',1916,'Publisher','Visit our WEB page'
,8.71,'image_I\T_100001.jpg','image_I\T_100001_thumb1.jpg','image_I','image_
I','T_100001_thumb1.jpg','T_100001.jpg');
EXEC Load_image(100002,'T_100002',2050,'Publisher','Visit our WEB page'
,9.61,'image_I\T_100002.jpg','image_I\T_100002_thumb10.jpg','image_I','image_
I','T_100002_thumb10.jpg','T_100002.jpg');
exit
```

Using SQL*Loader

SQL*Loader provides two methods for loading data:

- Conventional Path Load

  A conventional path load (the default) uses the SQL INSERT statement and a bind array buffer to load data into database tables. When SQL*Loader performs a conventional path load, it competes equally with all other processes for buffer resources. This can slow the load significantly. Extra overhead is added as SQL commands are generated, passed to Oracle, and executed. Oracle looks for partially filled blocks and attempts to fill them on each insert. Although appropriate during normal use, this can slow bulk loads dramatically. Use conventional path load if you encounter certain restrictions on direct path loads.

- Direct Path Load

  A direct path load eliminates much of the Oracle database overhead by formatting Oracle data blocks and writing the data blocks directly to the database files. A direct load does not compete with other users for database resources, so it can usually load data at near disk speed. In addition, if asynchronous I/O operations is available on your host platform, multiple buffers are used for the formatted data blocks to further increase load performance.

See *Oracle9i Database Utilities* for a complete list of restrictions for using either the conventional path load or direct path load method for loading data using SQL*Loader. See *Oracle9i Application Developer's Guide - Fundamentals* for more information on LOBs.

Using SQL*Loader to Load Multimedia Data into Oracle9*i* Using *inter*Media Column Objects

Example 11–5 shows the use of the control file to load one ORDVideo object per file into a table named JUKE that has three columns, with the last one being a column object. Each LOB file is the source of a single LOB and follows the column object name with the LOBFILE data type specifications. Two LOB files are loaded in this example.

Example 11–5 Show the Control File for Loading Video Data

```
LOAD DATA
INFILE *
INTO TABLE JUKE
REPLACE
FIELDS TERMINATED BY ',' 
( id integer external,
 file_name char(1000),
  mediacontent column object
   (
       source column object
       (
1)       localData_fname FILLER CHAR(128),
2)     localData LOBFILE (mediacontent.source.localData_fname) terminated by EOF
       )

    )
)

BEGINDATA
1,slynne,slynne.rm
2,Commodores,Commodores - Brick House.rm
```

Notes:

1. The filler field is mapped to the 128-byte long data field which is read using the SQL*Loader CHAR data type.
2. SQL*Loader gets the LOB file name from the localData_fname FILLER field. It then loads the data from the LOB file (using the BLOB data type) from its

beginning to the EOF character, whichever is reached first. Note that if no existing LOB file is specified, the localData field is initialized to empty.

Using the OCILobLoadFromFile( ) Relational Function

Oracle Call Interface (OCI) is an application programming interface (API) that allows you to manipulate data and schemas in an Oracle database using a host programming language, such as C.

The OCI relational function, OCILobLoadFromFile( ), loads or copies all or a portion of a file into an *inter*Media column object containing a specified BLOB. The data is copied from the source file to the destination *inter*Media column objects containing a BLOB. When binary data is loaded into an *inter*Media column object containing a BLOB, no character set conversions are performed. Therefore, the file data must already be in the same character set as the BLOB in the database. No error checking is performed to verify this.

See *Oracle Call Interface Programmer's Guide* for more information.

Using the DBMS_LOB.LOADFROMFILE( ) Procedure in the DBMS_LOB Package

The DBMS_LOB package provides subprograms to operate on BLOBs, CLOBs, NCLOBs, BFILEs, and temporary LOBs. You can use the DBMS_LOB package for access and manipulation of specific parts of an *inter*Media column object containing a BLOB, as well as complete BLOBs. DBMS_LOB can read as well as modify BLOBs, CLOBs, and NCLOBs, and provides read-only operations for BFILEs. The majority of the LOB operations are provided by this package.

The DBMS_LOB.LOADFROMFILE( ) procedure copies all, or part of, a source-external LOB (BFILE) to a destination internal LOB.

You can specify the offsets for both the source LOB (BFILE) and destination *inter*Media column object containing the BLOB and the number of bytes to copy from the source BFILE. The amount and src_offset, because they refer to the BFILE, are in terms of bytes, and the destination offset is either in bytes or characters for BLOBs and CLOBs respectively.

The input BFILE must have been opened prior to using this procedure. No character set conversions are performed implicitly when binary BFILE data is loaded into a CLOB. The BFILE data must already be in the same character set as the CLOB in the database. No error checking is performed to verify this. See *Oracle9i Supplied PL/SQL Packages Reference* for more information.

Using Java loadDataFrom...( ) Methods to Load Media Data from a Client File

From the Java client, you can use the Java loadDataFromByteArray( ), loadDataFromFile( ), or loadDataFromInputStream( ) methods of interMedia Java Classes to load media data from a given file into a server-side media object designated by the corresponding media locator parameters. You must specify the name of the file from which to load the data and the method returns true if loading is successful, false otherwise. See *Oracle interMedia Java Classes User's Guide and Reference* for more information.

11.4 Loading Multimedia Data Using the *inter*Media Clipboard

You can use the Oracle *inter*Media Clipboard (Release 2) to:

- Upload multimedia objects from files and URLs and store them in the database

See *Oracle interMedia Clipboard (Release2) Installation and Configuration Guide* for more information. See Section 1.13.5 for information on how obtain this software and documentation.

11.5 Loading Multimedia Data Using *inter*Media Annotator Utility

You can use the Oracle *inter*Media Annotator utility to upload media data and an associated annotation into an Oracle8*i* or higher database where Oracle *inter*Media is installed. Annotator does this using an Oracle PL/SQL upload template, which contains both PL/SQL calls and Annotator-specific keywords.

See *Oracle interMedia Annotator User's Guide* for more information.

11.6 Reading Data from an ORDVideo Object Using the *inter*Media readFromSource( ) Method in a PL/SQL Script

Example 11–6 shows the contents of the readvideo1.sql file. This procedure reads data from an ORDVideo object with the video stored in a BLOB in the database using the readFromSource method in a PL/SQL script until no more data is found. The procedure then returns a NO_DATA_FOUND exception when the read operation is complete and displays an "End of data" message.

Note: This example can be modified to work with the ORDAudio and ORDImage objects too.

Example 11–6 Read Data from an ORDVideo Column Object Using interMedia readFromSource( ) Method in a PL/SQL Stored Procedure

```
create or replace procedure readVideo1(i integer)  as

   obj ORDSYS.ORDVideo;
   buffer RAW (32767);
   numbytes BINARY_INTEGER := 32767;
   startpos integer := 1;
   read_cnt integer := 1;
   ctx RAW(4000) := NULL;

BEGIN

   Select  mediacontent into obj from juke where id = 100001;

   LOOP
           obj.readFromSource(ctx,startpos,numbytes,buffer);
           startpos := startpos + numBytes;
           read_cnt := read_cnt + 1;

   END LOOP;

EXCEPTION

   WHEN NO_DATA_FOUND THEN
   DBMS_OUTPUT.PUT_LINE('End of data ');
   DBMS_OUTPUT.PUT_LINE('doing read  '|| read_cnt);
   DBMS_OUTPUT.PUT_LINE('start position :'|| startpos);

END;

/
show errors
```

11.7 Reading Results of an *inter*Media Benchmark

The benchmark environment for the hardware and software for the interMedia BLOB read tests that were performed are described in this section.

Benchmark Environment

The server side consisted of a quad 200MHz Pentium Pro processor with 3GB of memory. The I/O disk subsystem consisted of a raid 0 stripe set supported by four Adaptec controllers. The system was running MS Windows NT V4.0 Service Pack 3.

The OCI experiments were conducted in a client/server environment where the client was also a quad 200MHz Pentium Pro processor linked to the server using a 100Mbits Ethernet connection.

The database was partitioned by range using a range ID such that each client reader or loader process used a dedicated database partition. Tests were conducted with a database block size set to 8K and 16K, a LOB chunk size set to 32K, and a read size (1 round-trip) set to 32K for the *inter*Media import( ) method in PL/SQL tests, and a LOB buffer size set to 32K to 64K for the OCI tests.

Test Description and Results

BLOB I/O tests were conducted in an MS Windows NT environment running Oracle *inter*Media. BLOB read tests were conducted with the *inter*Media readFromSource( ) method in a PL/SQL script to read BLOBs from the database, as well as making OCI calls without callbacks to perform BLOB read operations from C++. Parallel processes were submitted on the client system to read BLOBs residing on the server side making use of the 100 megabit network bandwidth. Database connections ranged from 6 to 16 for the BLOB read tests.

A benchmark was performed to measure the performance of an Oracle-based system in a setting modeling a real-life audio server application. The Oracle server serves multiple requests by clients to a set of CDs. CDs are stored in Oracle8*i* or higher using the Oracle *inter*Media. The CD access pattern is modeled by an exponential distribution to simulate that some CDs are more popular than others. A client has a tolerance on the response time of a request. Each request asks for a particular amount of audio data. The throughput of the server, defined by the amount of audio data provided per unit time, is measured, subjected to the following constraints:

- Number of users
- Maximum or average response time of requests
- Size of each request
- Access patterns

Throughput levels as high as 29 MB/second using a large cache of 1.7GB, a LOB chunk size set to 32K, and with OCI using buffered read operations to read BLOBs locally on the backend, memory-rich server. Using a less memory-rich server system with a 320MB cache buffer size, throughput decreased by one third to a low of 20MB/second level.

The performance-limiting factor was the 100 megabit bandwidth, which became saturated in the client/server tests. All tests with OCI had caching turned on. Using

the *inter*Media readFromSource( ) method in a PL/SQL procedure, and with no cache set, the throughput was limited to 18MB/second. The limiting factor for performance for reading BLOB data was the I/O subsystem in the absence of caching.

11.8 Getting the Best Performance Results

The following guidelines can be used to help you achieve the best performance when working with *inter*Media objects:

- Because *inter*Media objects are big, you can attain the best performance by reading and writing large chunks of an *inter*Media object value at a time. This helps in several respects:
  - If you are accessing the *inter*Media object from the client side and the client is on a different node than the server, large read/write operations reduce network overhead.
  - If you are using the NOCACHE option, each small read/write operation incurs an I/O impact. Reading and writing large quantities of data reduces the I/O impact.
  - Writing to the *inter*Media object creates a new version of the *inter*Media object chunk. Therefore, writing small amounts at a time will incur the cost of a new version for each small write operation. If logging is on, the chunk is also stored in the redo log.
- If you need to read or write small pieces of *inter*Media object data on the client, use LOB buffering (see OCILobEnableBuffering( ), OCILobDisableBuffering( ), OCILobFlushBuffer( ), OCILobWrite( ), OCILobRead( ) in *Oracle Call Interface Programmer's Guide* for more information.). Turn on LOB buffering before reading or writing small pieces of *inter*Media object data.
- Use *inter*Media methods (readFromSource( ) and writeToSource( )) for audio and video data or OCILobWrite( ) and OCILobRead( ) with a callback for image data so media data is streamed to and from the BLOB. Ensure that the length of the entire write operation is set in the *numBytes* parameter using *inter*Media methods or in the *amount* parameter using OCI calls on input. Whenever possible, read and write in multiples of the LOB chunk size.
- Use a checkout/checkin model for LOBs. LOBs are optimized for the following:
  - Updating *inter*Media object data: SQL UPDATE operations, which replaces the entire BLOB value.

- Copying the entire LOB data to the client, modifying the LOB data on the client side, and copying the entire LOB data back to the database. This can be done using OCILobRead( ) and OCILobWrite( ) with streaming.

See *Oracle9i Application Developer's Guide - Large Objects (LOBs)* for more information.

11.9 Improving Multimedia LOB Data Retrieval and Update Performance

Once the LOB data is stored in the database, a modified strategy must be used to improve the performance of retrieving and updating the LOB data compared to the insertion strategy described in Section 11.3. The following guidelines should be considered:

- Use the CACHE option on LOBs if the same LOB data is to be accessed frequently by other users.
- Increase the number of buffers if you are going to use the CACHE option.
- Have enough buffers to hold the object. Using a small number of buffers for large objects is not good. Set the DB_CACHE_SIZE parameter to a value that you know will hold the object.
- Ensure that your redo log files are much larger than they usually are; otherwise, you may be waiting for log switches, especially if you are making many updates to your LOB data.
- Ensure that you use a larger page size (DB_BLOCK_SIZE), especially if the majority of the data in the database is LOB data.

A

Audio File and Compression Formats

A.1 Supported Audio File and Compression Formats

The following tables describe the audio file and compression formats and other audio features supported by *inter*Media.

To use these tables, find the data format you are interested in, and then determine the supported formats. For example, Table A–1 shows that *inter*Media supports AIFF format for single channel, stereo, 8-bit and 16-bit samples, linear PCM encoding, and uncompressed format.

Table A–1 AIFF Data Format

Format	Audio Feature
AIFF Format ID 'AIFF' File Format: 'AIFF' File Ext: .aff MIME type: audio/x-aiff	Single channel Stereo 8-bit samples 16-bit samples Linear PCM encoding
Format	**Encoding/CompressionType**
Standard AIFF Uncompressed	TWOS

Table A–2 AIFF-C Data Format

Format	Audio Feature
AIFF-C Format ID 'AIFC' File Format: 'AIFC' File Ext: .afc MIME type: audio/x-aiff	Single channel Stereo 8-bit samples 16-bit samples
Format	**Encoding/CompressionType**
Choose one of these compression formats[1] Not compressed ACE 2-to-1 ACE 8-to-3 MACE 3-to-1 MACE 6-to-1	 Uncompressed (TWOS) ACE2 ACE8 MAC3 MAC6

[1] Other than "uncompressed (TWOS)", all other codes are the FourCC (uppercased) directly from the compressionType field of Common Chunk of the AIFC file. The table lists only the ones known.

Table A–3 AU Data Format

Format	Audio Feature
AU Format ID 'AUFF' File Format: 'AUFF' File Ext: .au MIME type: audio/basic	Single channel Stereo 8-bit samples 16-bit samples Mu-law encoding Linear PCM encoding

Table A–3 AU Data Format

Format	Encoding/CompressionType
Choose one of these compression formats:	
Unspecified format	UNSPECIFIED
8-bit mu-law samples	MULAW
8-bit linear samples	LINEAR
16-bit linear samples	LINEAR
24-bit linear samples	LINEAR
32-bit linear samples	LINEAR
Floating-point samples	FLOAT
Double-precision float samples	DOUBLE
Fragmented sample data	FRAGMENTED
Nested format	NESTED
DSP program	DSP_CORE
8-bit fixed-point samples	DSP_DATA
16-bit fixed-point samples	DSP_DATA
24-bit fixed-point samples	DSP_DATA
32-bit fixed-point samples	DSP_DATA
Unknown AU's format	UNKNOWN
Non-audio display data	DISPLAY
Squelch format	MULAW_SQUELCH
16-bit linear with emphasis	EMPHASIZED
16-bit linear with compression	COMPRESSED
16-bit linear with emphasis and compression	COMPRESSED_EMPHASIZED
Music Kit DSP commands	DSP_COMMANDS
DSP commands samples	DSP_COMMANDS_SAMPLES
adpcm G721	ADPCM_G721
adpcm G722	ADPCM_G722
adpcm G723_3	ADPCM_G723_3
adpcm G723_5	ADPCM_G723_5
8-bit a-law samples	ALAW

Table A–4 WAV Data Format

Format	Audio Feature
WAV Format ID 'WAVE' File Format: 'WAVE' File Ext: .wav MIME type: audio/x-wav	Single channel Stereo 8-bit samples 16-bit samples Linear PCM encoding
Format	**Encoding/CompressionType**

Table A–4 WAV Data Format

Choose one of these compression formats:	
Unknown Wave Format	UNKNOWN
Microsoft PCM Wave Format	MS_PCM
Microsoft ADPCM Wave Format	MS_ADPCM
IBM CVSD Wave Format	IBM_CVSD
Microsoft aLaw Wave Format	ALAW
Microsoft uLaw Wave Format	MULAW
OKI ADPCM Wave Format	OKI_ADPCM
Intel DVI/IMA ADPCM Wave Format	DVI_ADPCM
VideoLogic Media Space ADPCM Wave Format	MEDIASPACE_ADPCM
Sierra Semiconductor ADPCM Wave Format	SIERRA_ADPCM
Antex Electronics G723 ADPCM Wave Format	ANTEX_G723_ADPCM
DSP Solutions DIGISTD Wave Format	DIGISTD
DSP Solutions DIGIFIX Wave Format	DIGIFIX
Dialogic OKI ADPCM Wave Format	DIALOGIC_OKI_ADPCM
Yamaha ADPCM Wave Format	YAMAHA_ADPCM
Speech Compression Sonarc Wave Format	SONARC
DSP Group TrueSpeech Wave Format	DSPGROUP_TRUESPEECH
Echo Speech Wave Format	ECHOSC1
Audiofile AF36 Wave Format	AUDIOFILE_AF36
Audio Processing Technology Wave Format	APTX
Audiofile AF10 Wave Format	AUDIOFILE_AF10
Dolby AC-2 Wave Format	DOLBY_AC2
Microsoft GSM 610 Wave Format	MS_GSM610
Antex Electronics ADPCME Wave Format	ANTEX_ADPCME
Control Resources VQLPC Wave Format	CONTROL_RES_VQLPC
DSP Solutions DIGIREAL Wave Format	DIGIREAL
DSP Solutions DIGIADPCM Wave Format	DIGIADPCM
Control Resources CR10 Wave Format	CONTROL_RES_CR10
Natural Microsystems NMS VBXADPCM Wave Format	NMS_VBXADPCM
Crystal Semiconductor IMA ADPCM Wave Format	CS_IMAADPCM
Antex Electronics G721 ADPCM Wave Format	ANTEX_G721_ADPCM
MPEG-1 Audio Wave Format	MPEG
Creative Labs ADPCM Wave Format	CREATIVE_ADPCM
Creative Labs FastSpeech8 Wave Format	CREATIVE_FASTSPEECH8
Creative Labs FastSpeech10 Wave Format	CREATIVE_FASTSPEECH10
Fujitsu FM Towns Wave Format	FM_TOWNS_SND
Olivetti GSM Wave Format	OLIGSM
Olivetti ADPCM Wave Format	OLIADPCM
Olivetti CELP Wave Format	OLICELP
Olivetti SBC Wave Format	OLISBC
Olivetti OPR Wave Format	OLIOPR

Table A–5 Audio MPEG Data Format

Format	**Audio Feature**
MPEG Format ID 'MPEG' File Format: 'MPGA' File Ext: .mpg MIME type: audio/mpeg	Layer I Layer II Layer III
Format	**Encoding/CompressionType**
Choose one of these compression formats: MPEG Audio, Layer I MPEG Audio, Layer II MPEG Audio, Layer III	 LAYER1 LAYER2 LAYER3

B

Image File and Compression Formats

B.1 Supported Image File and Compression Formats

Descriptions of each of the supported image file formats and image compression formats are presented in Section B.1.1 and Section B.1.2.

B.1.1 Image File Formats

Image file formats are listed alphabetically.

BMPF

extension: .bmp

mime: image/bmp

BMPF is the Microsoft Windows bitmap format and is based on the internal data structures used by Windows to store bitmap data in memory. This format is used extensively by Microsoft Windows, and a variant of this format is used by the IBM OS/2 operating system. Because this format is supported directly by Windows, its use is very popular in that environment and has spread to other systems.

BMPF is a very flexible image format in that it can store a wide variety of image data types, but it does not offer powerful compression. The only compression available is a run-length encoding variant that is supported only by certain content formats. It is worth noting that BMPF is unusual in that the ordinary scanline order for this format is bottom-up, which Oracle *inter*Media calls INVERSE.

CALS

extension: .cal

mime: image/x-ora-cals

CALS is an image format developed by the Computer Aided Acquisition and Logistics Support office of the United States government for document interchange. There are actually two variants of the CALS image format; Oracle *inter*Media supports CALS Type I. Because the CALS format is monochrome-only, it is primarily useful for storing simple documents, scanned or otherwise.

Foreign Images

Foreign Images are images for which Oracle *inter*Media does not provide native recognition and support, but that can sometimes be read if the image data complies with the rules outlined in the Foreign Image Support section of the Raw Pixel appendix (see Section E.10).

FPIX

extension: .fpx

mime: image/x-fpx

FPIX, or FlashPix, is a format developed by Kodak, Microsoft Corporation, Hewlett-Packard Company, and Live Picture, Inc., for storing digital photography. FlashPix images are composed of a series of different resolutions of the same image, and each resolution is composed of individual tiles. These tiles can be uncompressed or compressed using JPEG. The multi-resolution capability of FlashPix images is intended to promote easy use in a wide variety of applications by allowing low resolution versions of the image to be used where high resolution versions are not necessary (such as browsing, viewing on screen), while high resolution versions are available when needed (printing or zooming in on an image detail).

Oracle *inter*Media includes a simple FlashPix decoder that always selects the largest resolution plane in a FlashPix image. Lower resolutions are not accessible. Oracle *inter*Media does not write FlashPix images.

GIFF

extension: .gif

mime: image/gif

GIFF is the Oracle *inter*Media name for the Graphics Interchange Format (GIF), which was developed by CompuServe to transfer images between users in their early network system. Because GIF (pronounced "jif") is an early format and was developed for use on limited hardware it does not support content formats which store more than 8 bits per pixel. This makes the format less suitable for storing

photographic or photo-realistic images than deeper formats such as PNG or JFIF, but it is a good choice for other applications. There are two specific variants of the GIF format, called 87a and 89a; Oracle *inter*Media reads both variants but writes the 87a variant.

Despite its pixel depth limitations, the GIF format remains a powerful and flexible image format, and includes support for limited transparency effects and simple animations by encoding a series of image frames and frame transition effects. Oracle *inter*Media can read GIF images that include these options but only the first frame of an animated GIF is made available, and there is no support for writing animated GIFs.

All GIF images are compressed using a GIF-specific LZW compression scheme which Oracle *inter*Media calls GIFLZW.

JFIF

extension: .jpg

mime: image/jpeg

JFIF is the JPEG File Interchange Format, developed by C-Cube Microsystems for storing JPEG encoded images. The JFIF format is actually just a JPEG data stream with an identifying header and a few enforced conventions. As such, it provides minimal support for anything but the actual image data. By definition, all JFIF files are JPEG compressed, making them less appropriate for some applications as explained in the description of the JPEG compression format.

Oracle *inter*Media identifies several distinct image formats as JFIF, including actual JFIF files, non-JFIF pure JPEG data streams, and EXIF files. The last is a JFIF variant produced by digital cameras.

PBMF, PGMF, PPMF and PNMF

extension: .pbm, .pgm, .ppm, .pnm

mime: image/x-portable-bitmap, image/x-portable-graymap, image/x-portable-pixmap, image/x-portable-anymap

These are a family of file formats derived from Jef Poskanzer's Portable Bitmap Utilities suite. These file formats are Portable Bitmap (PBM), Portable Graymap (PGM), Portable Pixmap (PPM) and Portable Anymap (PNM). Because of their wide support and the free availability of software to handle these formats, they are frequently used for uncompressed image interchange.

PBM files are monochrome only files (the term "bitmap" being used in the sense of a map of bits, that is, each pixel is either 0 or 1). PGM files are grayscale only, while PPM files are full color pixel maps.

PNM does not refer to a distinct file format, but instead refers to any of the other three types (PBM, PGM or PPM). Images written using the file format designation PNMF will be written as the most appropriate variant depending on the input data content format.

These formats do not include data compression, but have two encoding formats: ASCII or RAW.

PCXF

extension: .pcx

mime: image/pcx

PCX, or PCXF in Oracle *inter*Media notation, is an early and widely used image file format developed for ZSoft's PC Paintbrush, and later used in derivatives of that program. Despite its ancestry, it provides support for many pixel depths, from monochrome to 24-bit color. It supports a fast compression scheme designated PCXRLE by Oracle *inter*Media. Oracle *inter*Media reads but does not write PCX images.

PICT

extension: .pct

mime: image/pict

The Macintosh PICT format was developed by Apple Computer, Inc., as part of the QuickDraw toolkit built into the Macintosh ROM. It provides the ability to "record" and "playback" QuickDraw sequences, including both vector and raster graphics painting. Oracle *inter*Media supports only the raster elements of PICT files. Both Packbits and JPEG compressed PICT images are supported.

PNGF

extension: .png

mime: image/png

PNGF is the Oracle *inter*Media designation for the Portable Network Graphics (PNG) format (pronounced "ping"). PNG was developed by the PNG Development Group as a legally unencumbered and more capable replacement for some uses of the GIF and TIFF file formats. PNG includes support for deep images (up to 16 bits

per sample and up to 4 samples per pixel), full alpha support, rich metadata storage including metadata compression, built-in error and gamma correction, a powerful and free compression algorithm called DEFLATE, and much more. The main feature found in GIF that is absent in PNG is the ability to store animations.

PNG support for a broad variety of pixel depths (1 bit to 16 bits per sample) makes it suitable for a very wide variety of applications, spanning the separate domains previously filled by GIF and JPEG, and being very similar to the uses of the powerful TIFF format. Because the DEFLATE compression scheme is lossless, PNG is a good choice for storing deep images that must be edited often.

All PNG images are compressed using the DEFLATE scheme.

RPIX

extension: .rpx

mime: image/x-ora-rpix

RPIX, or Raw Pixel, is a format developed by Oracle Corporation for storing simple raw pixel data without compression, and using a simple well-described header structure. It was designed to be used by applications whose native image format is not supported by Oracle *inter*Media but for which an external translation might be available. It flexibly supports N-banded image data (8 bits per sample) where N < 256 bands, and can handle data that is encoded in a variety of channel orders (such as RGB, BGR, BRG, and so forth), a variety of pixel orders (left-to-right and right-to-left), a variety of scanline orders (top-down or bottom-up) and a variety of band orders (band interleaved by pixel, by scanline, and by plane). The flexibility of the format includes a data offset capability, which can allow an RPIX header to be prepended to other image data, thus allowing the RPIX decoder to read an otherwise compliant image format. See Appendix E for more information.

In addition to its support for 8 bits per sample data, RPIX supports single-band monochrome images compressed using the FAX3 and FAX4 compression schemes.

When an RPIX image is decoded, only 1 or 3 bands are read. Which bands are selected can be determined by the image header or by the InputChannels operator. Similarly, Oracle *inter*Media writes only 1 or 3 band RPIX images.

RASF

extension: .ras

mime: image/x-ora-rasf

The Sun Raster image format, called RASF by Oracle *inter*Media, was developed by Sun Microsystems for its UNIX operating systems and has a wide distribution in the UNIX community. It supports a variety of pixel depths and includes support for a format-specific, run-length encoding compression scheme called SUNRLE by Oracle *inter*Media.

TGAF

extension: .tga

mime: image/x-ora-tgaf

The Truevision Graphics Adapter format (TGA, or TGAF to Oracle *inter*Media) was developed by Truevision, Inc., for their line of Targa and related graphics adapters. This format includes support for color images with 8, 16, 24 and 32 bits per pixel and also includes support for a run-length encoding compression scheme called TARGARLE by Oracle *inter*Media.

TIFF

extension: .tif

mime: image/tiff

The Tag Image File Format (TIFF) was originally developed by the Aldus Corporation. The format has become something of a benchmark for image interchange and is extremely versatile, including support for a wide variety of compression and data formats, multiple image pages per file, and a wide variety of metadata. Because of its many options, TIFF is a good choice for many applications, including document storage, simple art, photographic and photo-realistic images, and others.

Oracle *inter*Media supports the "baseline TIFF" specification and also includes support for some TIFF "extensions," including tiled images and certain compression formats not included as part of the baseline TIFF specification. "Planar" TIFF images are not supported. It is important to note that the JPEG support in TIFF provided by Oracle *inter*Media is based on the revised JPEG in TIFF specification and not the original JPEG in TIFF specification. TIFF images in either big endian or little endian format can be read, but Oracle *inter*Media always writes big endian TIFFs.

Although the TIFF decoder in Oracle *inter*Media includes support for page selection using the "page" verb in the process( ) and processCopy( ) methods, the setProperties( ) method always returns the properties of the initial page in the file. It is important to note that this initial page is accessed by setting "page=0" in the

process command string. Oracle *inter*Media currently does not support writing multiple page TIFF files.

WBMP

extension: .wbmp

mime: image/vnd.wap.wbmp

The Wireless Bitmap format (WBMP) was developed for the Wireless Application Protocol as a means of transmitting bitmap (monochrome) images to WAP-compliant devices. An extremely minimalist format, it does not even include identifying markers or support for compression. It is most appropriate for very small images being transmitted over limited bandwidth networks.

The WBMP format is not related to the BMPF format.

B.1.2 Image Compression Formats

Image compression formats are listed alphabetically.

ASCII

Not an actual compression format by itself, ASCII is an encoding used by PBM, PGM, and PPM images to represent images in plain ASCII text form. Each pixel value is represented by an individual integer in an ASCII-encoded PBM (or PGM or PPM) file.

BMPRLE

BMPRLE is the description that Oracle *inter*Media gives to images that are compressed with the BMP run-length encoding compression scheme. This compression format is available only for 4-bit and 8-bit LUT data, and only for images that are stored in INVERSE scanline order (the default order for BMP files). For very complex images, this compression can occasionally actually increase the file size.

DEFLATE

DEFLATE is the compression scheme employed by the PNG image format, and has also been adapted to work in the TIFF image format. DEFLATE is based on the ZIP algorithm and is a very adaptable compression scheme that handles a wide variety of image data formats well. Besides being used to compress image data in PNG and TIFF files, DEFLATE is also used within PNG files to compress some metadata.

DEFLATE-ADAM7

DEFLATE-ADAM7 is the same compression format as DEFLATE, but refers to images whose scanlines are interlaced for progressive display as the image is decoded. The intention of this technique is to allow a user to observe the image being progressively decoded as it is downloaded through a low bandwidth link and abort the image before completion of the download. While the low bandwidth requirement is not typically relevant anymore, many existing images employ this encoding. Unlike JPEG-PROGRESSIVE and GIFLZW-INTERLACED, DEFLATE-ADAM7 interlaces images both horizontally and vertically.

Oracle *inter*Media provides read support for this encoding, but does not provide write support.

FAX3

FAX3 is the Oracle *inter*Media designation for CCITT Group 3 2D compression, which was developed by the CCITT (International Telegraph and Telephone Consultative Committee) as a protocol for transmitting monochrome images over telephone lines by facsimile and similar machines. The more official designation for this compression scheme is CCITT T.4.

Because this compression format supports only monochrome data, it cannot be used for color or grayscale images. This compression scheme uses a fixed dictionary that was developed using handwritten and typewritten documents and simple line graphics that were meant to be representative of documents being transmitted by facsimile. For this reason, although the compression can be used on images that have been dithered to monochrome, it may not produce as high a compression ratio as more adaptive schemes such as LZW or DEFLATE in those cases. It is most appropriate for scanned documents.

FAX4

FAX4 is the Oracle *inter*Media designation for CCITT Group 4 2D compression, which was developed by the CCITT (International Telegraph and Telephone Consultative Committee) as a protocol for transmitting monochrome images over telephone lines by facsimile and similar machines. The more official designation for this compression scheme is CCITT T.6.

Because this compression format supports only monochrome data, it cannot be used for color or grayscale images. This compression scheme uses a fixed dictionary that was developed using handwritten and typewritten documents and simple line graphics that were meant to be representative of documents being transmitted by facsimile. For this reason, although the compression can be used on images that have been dithered to monochrome, it may not produce as high a compression ratio

as more adaptive schemes such as LZW or DEFLATE in those cases. It is most appropriate for scanned documents.

GIFLZW

GIFLZW is the Oracle *inter*Media designation for the LZW compression system used within GIF format images, and is different from LZW compression as used by other file formats. GIFLZW is an adaptive compression scheme that provides good compression for a wide variety of image data, although it is least effective on very complex images, such as photographs.

GIFLZW-INTERLACED

GIFLZW-INTERLACED is the same compression format as GIFLZW, but refers to images whose scanlines are interlaced for progressive display as the image is decoded. The intention of this technique is to allow a user to observe the image being progressively decoded as it is downloaded through a low bandwidth link and abort the image before completion of the download. While the low bandwidth requirement is not typically relevant anymore, many existing images employ this encoding.

Oracle *inter*Media provides read support for this encoding, but does not provide write support.

HUFFMAN3

HUFFMAN3 is the Oracle *inter*Media designation for the Modified Huffman compression scheme used by the TIFF image format. This compression format is based on the CCITT Group 3 1D compression format, but is not an official CCITT standard compression format.

Because this compression format supports only monochrome data, it cannot be used for color or grayscale images. This compression scheme uses a fixed dictionary that was developed using handwritten and typewritten documents and simple line graphics that were meant to be representative of documents being transmitted by facsimile. For this reason, although the compression can be used on images that have been dithered to monochrome, it may not produce as high a compression ratio as more adaptive schemes such as LZW or DEFLATE in those cases. It is most appropriate for scanned documents.

JPEG

The JPEG compression format was developed by the Joint Photographic Experts Group for storing photographic and photo-realistic images. The JPEG compression

format is very complex, but most images belong to a class called "baseline JPEG" which is a much simpler subset. Oracle *inter*Media supports only baseline JPEG compression.

The JPEG compression scheme is a lossy compression format; that is, images compressed using JPEG can never be reconstructed exactly. JPEG works by eliminating spatial and chromatic details that the eye will probably not notice. While JPEG can compress most data quite well, the results may include serious cosmetic flaws for images that are not photographic, such as monochrome or simple art. Other compression schemes are more appropriate for those cases (FAX formats or PNG and GIF). Also, the lossy nature of this compression scheme makes it inappropriate for images that must be edited, but it is a good choice for finished images that must be compressed as tightly as possible for storage or transmission.

JPEG-PROGRESSIVE

This compression format is a variation of the JPEG compression format in which image scanlines are interlaced, or stored in several passes, all of which must be decoded to compute the complete image. This variant is intended to be used in low bandwidth environments where users can watch the image take form as intermediate passes are decoded and abort the image display if desired. While the low bandwidth requirement is not typically relevant anymore, this variant sometimes results in a smaller encoded image and is still popular. Oracle *inter*Media provides read, but not write, support for this encoding.

LZW

LZW is the Oracle *inter*Media designation for the LZW compression system used within TIFF format images, and is different from LZW compression as used by other file formats. TIFF LZW is an adaptive compression scheme that provides good compression for a wide variety of image data, although it is least effective on very complex images. TIFF LZW works best when applied to monochrome or 8-bit gray or LUT data; the TIFF method of applying LZW compression to other data formats results in much lower compression efficiency.

LZWHDIFF

LZWHDIFF is the description that Oracle *inter*Media gives to images employing the TIFF LZW compression system and also utilizing the TIFF horizontal differencing predictor. This scheme is a technique that can improve the compression ratios for 24-bit color and 8-bit grayscale images in some situations, without loss of data. It generally does not improve compression ratios for other image types.

NONE

This is the description that Oracle *inter*Media gives to image data that is not compressed.

PACKBITS

The Packbits compression scheme was developed by Apple Computer, Inc., as a simple byte-oriented, run-length encoding scheme for general use. This scheme is used by the PICT image format and has been adapted to work in TIFF images as well. Like other run-length encoding schemes, this compression can actually increase the data size for very complex images.

PCXRLE

PCXRLE is the description given by Oracle *inter*Media to images that are compressed using the PCX run-length encoding scheme. For very complex images, this compression can occasionally actually increase the file size.

RAW

Not an actual compression format by itself, RAW is encoding used by PBM, PGM, and PPM images to represent images in binary form (versus the plain text form employed by the ASCII encoding). The PBM family documentation refers to this format as RAWBITS.

SUNRLE

SUNRLE is the description used within Oracle *inter*Media for the run-length encoding scheme used in Sun Raster images. For very complex images, this compression can occasionally actually increase the file size.

TARGARLE

TARGARLE is the description given by Oracle *inter*Media to images compressed using the run-length encoding scheme supported by the TGAF file format. For very complex images, this compression can occasionally actually increase the file size.

B.1.3 Summary of Image File Format and Image Compression Format

Table B–1 summarizes read / write support for image file formats relative to content format characteristics, such as content format, pixel layout, interpretation, and color space. Table B–2 summarizes read / write support for image file formats relative to compression format and other format specific characteristics, such as channel order, pixel order, and scanline order.

Table B–1 Summary of Read/Write Access[1] for Supported Image File Formats -- Content Format Specific Characteristics

File Format	Content Format											Pixel Layout		
	1bitLUT (RGB& GRAY)	4bitLUT (RGB& GRAY)	8bitLUT (RGB& GRAY)	4bit direct GRAY	8bit direct GRAY	16bit direct RGB	24bit direct RGB	32bit direct RGB	48bit direct RGB	64bit direct RGB	Monochrome	BIP	BIL	BSQ
BMPF	RW	RW	RW			R	RW	R			RW	RW		
CALS											RW	RW		
FPIX[2]			R				R					R		
GIFF[3]	RW	RW	RW								RW	RW		
JFIF[4]					RW		RW					RW		
PBMF											RW	RW		
PCXF[5]	R	R	R				R				R	R		
PGMF					RW							RW		
PICT[6]	R	R	RW		RW	R	RW				RW	RW		
PNGF	RW	RW	RW	RW	RW	R	RW	R	R	R	RW	RW		
PNMF[7]					W		W				W	W		
PPMF							RW					RW		
RPIX[8]					RW		RW				RW	RW	RW	RW
RASF			RW		RW		RW				RW	RW		
TGAF			RW		RW	R	RW	R				RW		
TIFF[9]	RW	RW	RW	RW	RW	R	RW	R	R	R	RW	RW		
WBMP											RW	RW		

[1] R = Read access; W = Write access

[2] No write support.

[3] Animated GIFFs may not be encoded.

[4] Supports EXIF images.

[5] No write support.

[6] Vector and object graphics are not supported.

[7] PNMF format is supported as PBMF, PGMF, or PPMF; output will be PBMF, PGMF, or PPMF as appropriate.

[8] Can decode 1 or 3 bands from an n-band image; only 1 or 3 bands may be encoded.

[9] TIFF image file format also supports the following content formats as read or read / write as specified: Tiled data - Read, Photometric interpretation - Read / Write, MSB - Read / Write, and LSB - Read; Planar (BSQ) is not supported; both MSB and LSB ordered files may be decoded; output is MSB.

Table B–2 Summary of Read/Write Access[1] for Supported Image File Formats -- Compression Format and Other Format Specific Characteristics

File Format	Compression Format																			Channel Order		Pixel Order			Scan line Order		Other Options		
	NONE	JPEG[2]	JPEG-PROGRESSIVE	BMPRLE	PCXRLE	SUNRLE	TARGARLE	GIFLZW	GIFLZW-INTERLACED	LZW	LZWHDIF[3]	FAX3[4]	FAX4[5]	HUFFMAN[6]	PACKBITS	DEFLATE	DEFLATE-ADAM7	ASCII	RAW	Quality Specification	RGB	RBG, GRB, GBR, BRG, BGR	NORMAL	REVERSE	OS/2	NORMAL	INVERSE	Input Channels	Page Selection	Tiled Data/ Tiled Output
BMPF[7]	RW			RW																	RW		RW		R	RW	RW			
CALS													RW										RW			RW				
FPIX		R																			R		R			R				
GIFF								RW	R												RW		RW			RW				
JFIF[8]		RW	R																	W	RW		RW			RW				
PBMF																		RW	RW				RW			RW				
PCXF					R																R		R			R				
PGMF																		RW	RW				RW			RW				

Table B–2 Summary of Read/Write Access[1] for Supported Image File Formats -- Compression Format and Other Format Specific Characteristics (Cont.)

File Format	Compression Format																				Channel Order		Pixel Order			Scan line Order		Other Options		
	NONE	JPEG[2]	JPEG-PROGRESSIVE	BMPRLE	PCXRLE	SUNRLE	TARGARLE	GIFLZW	GIFLZW-INTERLACED	LZW	LZWHDIFF[3]	FAX34	FAX45	HUFFMAN36	PACKBITS	DEFLATE	DEFLATE-ADAM7	ASCII	RAW	Quality Specification	RGB	RBG, GRB, GBR, BRG, BGR	NORMAL	REVERSE	OS/2	NORMAL	INVERSE	Input Channels	Page Selection	Tiled Data/ Tiled Output
PICT		RW													RW						RW		RW			RW				
PNGF																RW	R				RW		RW			RW				
PNMF																		RW	RW		W		W			W				
PPMF																		RW	RW		RW		RW			RW				
RPIX	RW											RW	RW								RW	RW	RW	RW		RW	RW	R		
RASF	RW					RW															RW		RW			RW				
TGAF	RW						RW														RW		RW			RW				
TIFF	RW	RW								RW	RW	RW	RW	RW	RW	RW					RW		RW			RW			R	RW
WBMP	RW																						RW			RW				

[1] R = Read access; W = Write access

[2] Supports 8-bit gray and 24-bit RGB data only.

[3] Supports 8-bit and 24-bit data only.
[4] Supports MONOCHROME data only.
[5] Supports MONOCHROME data only.
[6] Supports MONOCHROME data only.
[7] Compression is supported only for scanlineOrder=INVERSE (inverse DIB), which is the default.
[8] Supports EXIF images.

C

Video File and Compression Formats

C.1 Supported Video File and Compression Formats

The following tables describe the video file and compression formats supported by *inter*Media.

To use these tables, find the data format you are interested in, and then determine the supported formats. For example, Table C–1 shows that *inter*Media supports Apple QuickTime 3.0 MOOV file format and a variety of compression formats from Cinepak to Sorenson Video.

Table C–1 Apple QuickTime 3.0 Data Format

Format	
Apple QuickTime 3.0 File Format: 'MOOV' File Ext: .mov MIME type: video/quicktime	
	Compression Format
Choose one of these compression formats[1]: Cinepak JPEG Uncompressed RGB Uncompressed YUV422 Graphics Animation: Run Length Encoded Apple Video Compression Kodak Photo CD QuickDraw GX MPEG Still Image Motion-JPEG (Format A) Motion-JPEG (Format B) Sorenson Video	 CVID JPEG RGB YUV2 SMC RLE RPZA KPCD QDGX MPEG MJPA MJPB SVQ1

[1] All codes are the FourCC (uppercased) directly obtained from the dataFormat field of the video sample description entry of 'stsd' Atom of the QuickTime file. The table lists only the ones known.

Table C–2 Microsoft Video for Windows (AVI) Data Format

Format	
Microsoft AVI File Format: 'AVI' File Ext: .avi MIME type: video/x-msvideo	
	Compression Format
Choose one of these compression formats[1]: Microsoft Video 1 Intel Indeo 3.1 Intel Indeo 3.2 Intel Indeo 4.0 Intel Indeo 4.1 Intel Indeo 5.0 Intel Indeo 5.1 Cinepak	 CRAM IV31 IV32 IV40 IV41 IV50 IV51 CVID

[1] All codes are the FourCC (uppercased) directly obtained from the compression field of 'strf' Chunk of the AVI file. The table lists only the ones known.

Table C–3 RealNetworks Real Video Data Format

Format
RealNetworks Real Video File Format: 'RMFF' File Ext: .rm MIME type: audio/x-pn-realaudio

D

Image process( ) and processCopy( ) Operators

This appendix describes the command options, or operators, used in the process( ) and processCopy( ) methods.

The available operators fall into three broad categories, each described in its own section:

- Section D.2, "Image Formatting Operators"
- Section D.3, "Image Processing Operators"
- Section D.4, "Format-Specific Operators"

Section D.1, "Common Concepts" describes the relative order of these operators.

D.1 Common Concepts

This section describes concepts common to all the image operators and the process( ) and processCopy( ) methods.

D.1.1 Source and Destination Images

The process( ) and processCopy( ) methods operate on one image, called the source image, and produce another image, called the destination image. In the case of the process( ) method, the destination image is written into the same storage space as the source image, replacing it permanently. For the processCopy( ) method, the storage for the destination image is distinct from the storage for the source image.

D.1.2 process( ) and processCopy( )

The process( ) and processCopy( ) methods are functionally identical except for the fact that process( ) writes its output into the same BLOB from which it takes its input while processCopy( ) writes its output into a different BLOB. Their command string options are identical and no distinction is drawn between them.

For the rest of this appendix, the names process( ) and processCopy( ) are used interchangeably, and the use of the name process( ) implies both process( ) and processCopy( ) unless explicitly noted otherwise.

D.1.3 Operator and Value

All of the process( ) operators appear in the command string in the form <operator> = <value>. No operator takes effect merely by being present in the command string. The right-hand side of the expression is called the **value** of the operator, and determines how the operator will be applied.

D.1.4 Combining Operators

In general, any number of operators can be combined in the command string passed into the process( ) method if the combination makes sense. However, certain operators are supported only if other operators are present or if other conditions are met. For example, the compressionQuality operator is supported only if the compression format of the destination image is JPEG. Other operators require that the source or destination image be a Raw Pixel or foreign image.

The flexibility in combining operators allows a single operation to change the format of an image, reduce or increase the number of colors, compress the data, and cut or scale the resulting image. This is highly preferable to making multiple calls to do each of these operations sequentially.

D.2 Image Formatting Operators

At the most abstract level, the image formatting operators are used to change the layout of the data within the image storage. They do not change the semantic content of the image, and unless the source image contains more information than the destination image can store, they do not change the visual appearance of the image at all. Examples of a source image with more information than the destination image can store are:

- Converting a 24-bit image to an 8-bit image (too many bits per pixel)

- Converting a color image to a grayscale or monochrome image (too many color planes)
- Converting an uncompressed image, or an image stored in a lossless compression format, to a lossy compression format (too much detail)

D.2.1 FileFormat

The **FileFormat** operator determines the image file type, or format, of the output image. The value of this operator is a 4-character code, which is a mnemonic for the new file format name. The list of allowable values for the file format operator is shown in Table 8–1. Appendix B contains basic information about each file format, including its mnemonic (file format), typical file extension, allowable compression and content formats, and other notable features.

The value given to the file format operator is the single most important detail when specifying the output for process( ). This value determines the range of allowable content and compression formats, whether or not compression quality will be useful, and whether or not the format-specific operators will be useful.

If the FileFormat operator is not used in the process( ) command string, *inter*Media will determine the file format of the source image and use that as the default file format value. If the file format of the source image does not support output, then an error will occur. If the source image is a foreign image, then the output image will be written as Raw Pixel.

D.2.2 ContentFormat

The **ContentFormat** operator determines the format of the image content. The content means the number of colors supported by the image and the manner in which they are supported. Depending on which file format is used to store the output image, some or most of the content formats may not be supported.

The supported values for the ContentFormat operator are described in Table 8–1.

The content formats that specify gray[scale] or grey[scale] support only shades of gray. The differences between these content formats is how many shades are allowed. The “4bit” formats support 16 shades while the formats with “8bit” support 256 shades of gray. There is no distinction between grayscale and greyscale.

The content formats that specify RGB store pixel data as Red, Green, Blue triplets. The number of bits specified will determine how many colors are supported. If 8 or fewer bits are specified, most formats will default to a LUT representation. Otherwise, DRCT (direct) will be used.

The content formats that specify LUT use a color lookup table to support various colors. The “1bitlut” format allows 2 distinct colors, “4bitlut” supports 16 unique colors, and “8bitlut” supports 256 colors.

The content formats that specify DRCT store the color values directly in the pixel data as a Red, Green, Blue triplet or gray value. The total number of bits of data is specified separately and individual formats allocate these bits to red, green, and blue in different ways. However, more bits of data allow for finer distinctions between different shades. Not all bits are used by some image formats. Note that currently most formats allow only 8-bit gray or 24-bit RGB.

The monochrome content format allows only black and white to be stored, with no gray shades in between.

If the ContentFormat operator is not passed to the process( ) method, then *inter*Media attempts to duplicate the content format of the source image if it is supported by the file format of the destination image. Otherwise, a default content format is chosen depending on the destination file format.

D.2.3 CompressionFormat

The **CompressionFormat** operator determines the compression algorithm used to compress the image data. The range of supported compression formats depends heavily upon the file format of the output image. Some file formats support but a single compression format, and some compression formats are supported only by one file format.

The supported values for the CompressionFormat operator are listed in Table 8–1.

All compression formats that include RLE in their mnemonic are run-length encoding compression schemes, and work well only for images that contain large areas of identical color. The PACKBITS compression type is a run-length encoding scheme that originates from the Macintosh system but is supported by other systems. It has limitations that are similar to other run-length encoding compression formats. Formats that contain LZW or HUFFMAN are more complex compression schemes that examine the image for redundant information and are more useful for a broader class of images. FAX3 and FAX4 are the CCITT Group 3 and Group 4 standards for compressing facsimile data and are useful only for monochrome images. All the compression formats mentioned in this paragraph are **lossless** compression schemes, which means that compressing the image does not discard data. An image compressed into a lossless format and then decompressed will look the same as the original image.

The JPEG compression format is a special case. Developed to compress photographic images, the JPEG format is a **lossy** format, which means that it compresses the image typically by discarding unimportant details. Because this format is optimized for compressing photographic and similarly noisy images, it often produces poor results for other image types, such as line art images and images with large areas of similar color. JPEG is the only lossy compression scheme currently supported by *inter*Media.

The DEFLATE compression type is ZIP Deflate and is used by PNG image file formats. The DEFLATE-ADAM7 compression format is interlaced ZIP Deflate and is used by PNG image file formats. The ASCII compression type is ASCII encoding and the RAW compression type is binary encoding and both are for PNM image file formats.

If the CompressionFormat operator is not specified, then *inter*Media will use the default compression format; often this default is "None" or "No Compression."

If the CompressionFormat operator is not specified and the file format of the destination image is different from that of the source image, then a default compression format will be selected depending on the destination image file format. This default compression is often "None" or "No Compression."

D.2.4 CompressionQuality

The **CompressionQuality** operator determines the relative quality of an image compressed with a lossy compression format. This operator has no meaning for lossless compression formats, and therefore is not currently supported for any compression format except JPEG.

The CompressionQuality operator accepts integer values between 1 (lowest quality) and 99 (highest quality) in addition to five values, ranging from most compressed image (lowest visual quality) to least compressed image (highest visual quality): MAXCOMPRATIO, HIGHCOMP, MEDCOMP, LOWCOMP, and MAXINTEGRITY. Using the MAXCOMPRATIO value allows the image to be stored in the smallest amount of space but may introduce visible aberrations into the image. Using the MAXINTEGRITY value keeps the resulting image more faithful to the original but will require more space to store.

If the CompressionQuality operator is not supplied and the destination compression format supports compression quality control, the quality will default to MEDCOMP for medium quality.

D.3 Image Processing Operators

The image processing operators supported by *inter*Media directly change the way the image looks on the display. The operators supported by *inter*Media represent only a fraction of all possible image processing operations, and are not intended for users performing intricate image analysis.

D.3.1 Cut

The **Cut** operator is used to create a subset of the original image. The values supplied to the cut operator are the origin coordinates (x,y) of the cut window in the source image, and the width and height of the cut window in pixels. This operator is applied before any scaling that is requested.

If the Cut operator is not supplied, the entire source image is used.

D.3.2 Scale

The **Scale** operator enlarges or reduces the image by the ratio given as the value for the operator. If the value is greater than 1.0, then the destination image will be scaled up (enlarged). If the value is less than 1.0, then the output will be scaled down (reduced). A scale value of 1.0 has no effect, and is not an error. No scaling is applied to the source image if the Scale operator is not passed to the process( ) method.

There are two scaling techniques used by *inter*Media. The first technique is "scaling by sampling" and is used only if the requested compression quality is MAXCOMPRATIO or HIGHCOMP, or if the image is being scaled up in both dimensions. This scaling technique works by selecting the source image pixel that is closest to the pixel being computed by the scaling algorithm and using the color of that pixel. This technique is faster, but results in a poorer quality image.

The second scaling technique is "scaling by averaging," and is used in all other cases. This technique works by selecting several pixels that are close to the pixel being computed by the scaling algorithm and computing the average color. This technique is slower, but results in a better quality image.

If the Scale operator is not used, the default scaling value is 1.0. This operator cannot be combined with other scaling operators.

D.3.3 XScale

The **XScale** operator is similar to the scale operator but only affects the width (x-dimension) of the image. The important difference between XScale and Scale is

that with XScale, scaling by sampling is used whenever the image quality is specified to be MAXCOMPRATIO or HIGHCOMP, and is not dependent on whether the image is being scaled up or down.

This operator may be combined with the YScale operator to scale each axis differently. It may not be combined with other scaling operators (Scale, FixedScale, MaxScale).

D.3.4 YScale

The **YScale** operator is similar to the scale operator but only affects the height (y-dimension) of the image. The important difference between YScale and Scale is that with YScale, scaling by sampling is used whenever the image quality is specified to be MAXCOMPRATIO or HIGHCOMP, and is not dependent on whether the image is being scaled up or down.

This operator may be combined with the XScale operator to scale each axis differently. It may not be combined with other scaling operators (Scale, FixedScale, MaxScale).

D.3.5 FixedScale

The **FixedScale** operator provides an alternate method for specifying scaling values. The Scale, XScale, and YScale operators all accept floating-point scaling ratios, while the FixedScale (and MaxScale) operators specify scaling values in **pixels**. This operator is intended to simplify the creation of images with a specific size, such as thumbnail images.

The two integer values supplied to the FixedScale operator are the desired dimensions (width and height) of the destination image. The supplied dimensions may be larger or smaller (or one larger and one smaller) than the dimensions of the source image.

The scaling method used by this operator will be the same as used by the Scale operator in all cases. This operator cannot be combined with other scaling operators.

D.3.6 MaxScale

The **MaxScale** operator is a variant of the FixedScale operator that preserves the aspect ratio (relative width and height) of the source image. MaxScale also accepts two integer dimensions, but these values represent the maximum value of the

appropriate dimension after scaling. The final dimension may actually be less than the supplied value.

Like the FixedScale operator, this operator is also intended to simplify the creation of images with a specific size. MaxScale is even better suited to thumbnail image creation than the FixedScale operator because thumbnail images created using MaxScale will have the correct aspect ratios.

The MaxScale operator scales the source image to fit within the dimensions specified while preserving the aspect ratio of the source image. Because the aspect ratio is preserved, only one dimension of the destination image may actually be equal to the values supplied to the operator. The other dimension may be smaller than or equal to the supplied value. Another way to think of this scaling method is that the source image is scaled by a single scale factor that is as large as possible with the constraint that the destination image fit entirely within the dimensions specified by the MaxScale operator.

If the Cut operator is used in conjunction with the MaxScale operator, then the aspect ratio of the cut window is preserved instead of the aspect ratio of the input image.

The scaling method used by this operator is the same as used by the Scale operator in all cases. This operator cannot be combined with other scaling operators.

D.4 Format-Specific Operators

The following operators are supported only when the destination image file format is Raw Pixel or BMPF (ScanlineOrder operator only), with the exception of the InputChannels operator, which is supported only when the source image is Raw Pixel or a foreign image. It does not matter if the destination image format is set to Raw Pixel or BMPF explicitly using the FileFormat operator, or if the Raw Pixel or BMPF format is selected by *inter*Media automatically because the source format is Raw Pixel, BMPF, or a foreign image.

D.4.1 ChannelOrder

The **ChannelOrder** operator determines the relative order of the red, green, and blue channels (bands) within the destination Raw Pixel image. The order of the characters R, G, and B within the mnemonic value passed to this operator determine the order of these channels within the output. The header of the Raw Pixel image will be written such that this order is not lost.

For more information about the Raw Pixel file format and the ordering of channels in that format, see Appendix E.

D.4.2 Interleaving

The **Interleaving** operator controls the layout of the red, green, and blue channels (bands) within the destination Raw Pixel image.The three mnemonic values supported by this operator: BIP, BIL, and BSQ force the output image to be “band interleaved by pixel,” “band interleaved by line,” and “band sequential” respectively.

For more information about the Raw Pixel file format, the interleaving of channels in that format, or the meaning of these interleaving values, see Appendix E.

Note: The interleaving operator is deprecated beginning with release 9.0.1 and its functionality has been moved into the contentFormat operator.

D.4.3 PixelOrder

The **PixelOrder** operator controls the direction of pixels within a scanline in a Raw Pixel Image. The value Normal indicates that the leftmost pixel of a scanline will appear first in the image data stream. The value Reverse causes the rightmost pixel of the scanline to appear first.

For more information about the Raw Pixel file format and pixel ordering, see Appendix E.

D.4.4 ScanlineOrder

The **ScanlineOrder** operator controls the order of scanlines within a Raw Pixel or BMPF image. The value Normal indicates that the top display scanline will appear first in the image data stream. The value Inverse causes the bottom scanline to appear first. For BMPF, ScanlineOrder = inverse is the default and ordinary value.

For more information about the Raw Pixel or BMPF file format and scanline ordering, see Appendix E.

D.4.5 InputChannels

As stated in Section D.4, the **InputChannels** operator is supported only when the source image is in Raw Pixel format or if the source is a foreign image.

The InputChannels operator assigns individual bands from a multiband image to be the red, green, and blue channels for later image processing. Any band within the source image can be assigned to any channel. If desired, only a single band may be specified and the selected band will be used as the gray channel, resulting in a grayscale output image. The first band in the image is number 1, and the band numbers passed to the Input Channels operator must be greater than or equal to one, and less than or equal to the total number of bands in the source image. Only the bands selected the by InputChannels operator are written to the output. Other bands are not transferred, even if the output image is in Raw Pixel format.

It should be noted that every Raw Pixel or foreign image has these input channel assignments written into its header block, but that this operator overrides those default assignments.

For more information about the Raw Pixel file format and input channels, see Appendix E.

D.4.6 Dither

The **Dither** operator controls how dithering is done. Dithering happens whenever needed to accomodate the user's request for a change in content format; the Dither operator can be used to change how the dithering happens.

Dithering is the process of approximating colors that do not actually exist in an image by mixing pixels of other colors in a ratio that fools the eye into seeing the approximated color. *inter*Media will dither images when required to accomodate a requested ContentFormat that can specify fewer colors than are present in the input image, or when a change in ContentFormat is required to store an image in the requested FileFormat. The Dither operator can be used to select by value which of the available dithering algorithms is used to perform this approximation.

The ORDEREDDITHER algorithm is a fast method for approximating colors based on a conversion color table. The resulting image often contains areas of alternating pixel values, which may be objectionable in some cases.

The ERRORDIFFUSION algorithm produces a more accurate destination image by diffusing the error caused by changing one pixel value into neighboring pixels. The resulting image is usually of higher quality than the ordereddither algorithm, but this operator is slower and may produce poor results with certain images."

D.4.7 Page

The **Page** operator selects a page from a multipage file and can only be used with TIFF file format images.

D.4.8 Tiled

The **Tiled** operator forces the output image to be tiled and can only be used with TIFF file format images.

E

Image Raw Pixel Format

This appendix describes the Oracle Raw Pixel image format and is intended for developers and advanced users who wish to use the Raw Pixel format to import unsupported image formats into *inter*Media, or as a means to directly access the pixel data in an image.

Much of this appendix is also applicable to foreign images.

E.1 Raw Pixel Introduction

*inter*Media supports many popular image formats suitable for storing artwork, photographs, and other images in an efficient, compressed way, and provides the ability to convert between these formats. However, most of these formats are proprietary to at least some degree, and the format of their content is often widely variable and not suited for easy access to the pixel data of the image.

The Raw Pixel format is useful for applications that need direct access to the pixel data without the burden of the complex computations required to determine the location of pixels within a compressed data stream. This simplifies reading the image for applications that are performing pixel-oriented image processing, such as filtering and edge detection. This format is even more useful to applications that need to write data back to the image. Because changing even a single pixel in a compressed image can have implications for the entire image stream, providing an uncompressed format enables applications to write pixel data directly, and later compress the image with a single process( ) command.

This format is also useful to users who have data in a format not directly supported by *inter*Media, but is in a simple, uncompressed format. These users can prepend a Raw Pixel identifier and header onto their data and import it into *inter*Media. For users who only need to read these images (such as for import or conversion), this

capability is built into *inter*Media as “Foreign Image Support”. How this capability is related to the Raw Pixel format is described in Section E.10.

In addition to supporting image types not already built into *inter*Media, the Raw Pixel format also permits the interpretation of N-band imagery, such as satellite images. Using Raw Pixel, one or three bands of an N-band image may be selected during conversion to another image format, allowing easy visualization within programs that do not otherwise support N-band images. Note that images written with the Raw Pixel format still may only have one or three bands.

The current version of the Raw Pixel format is “1.0”. This appendix is applicable to Raw Pixel images of this version only, as the particulars of the format may change with other versions.

E.2 Raw Pixel Image Structure

A Raw Pixel image consists of a 4-byte image identifier, followed by a 30-byte image header, followed by an arbitrary gap of zero or more bytes, followed by pixel data.

It is worth noting that Raw Pixel images are never color-mapped, and therefore do not contain color lookup tables.

The Raw Pixel header consists of the “Image Identifier” and the “Image Header”. The Image Header is actually composed of several fields.

Note that the first byte in the image is actually offset ‘0’. All integer fields are unsigned and stored in big endian byte order.

Name	Byte(s)	Description
Image Identifier	0:3	4-byte character array containing ASCII values for RPIX. This array identifies the image as a Raw Pixel image.
Image Header Length	4:7	Length of this header in bytes, excluding the identifier field. The value of this field may be increased to create a gap between the header fields and the pixel data in the image.
Major Version	8	Major version number of the Raw Pixel format used in the image.
Minor Version	9	Minor version number of the Raw Pixel format used in the image.
Image Width	10:13	Width of the image in pixels.

Image Height	14:17	Height of the image in pixels.
Compression Type	18	Compression type of the image: None, CCITT FAX Group 3, or CCITT FAX Group 4.
Pixel Order	19	Pixel order of the image: Normal or Reverse.
Scanline Order	20	Scanline order of the image: Normal or Inverse.
Interleave	21	Interleave type of the image: BIP, BIL, or BSQ.
Number of Bands	22	Number of bands in the image. Must be in the range 1 to 255.
Red Channel Number	23	The band number of the channel to use as a default for red. This field is the gray channel number if the image is grayscale.
Green Channel Number	24	The band number of the channel to use as a default for green. This field is zero if the image is grayscale.
Blue Channel Number	25	The band number of the channel to use as a default for blue. This field is zero if the image is grayscale.
Reserved Area	26:33	Not currently used. All bytes ***must*** be zero.

E.3 Raw Pixel Header Field Descriptions

This section describes the fields of the Raw Pixel Header in greater detail.

Image Identifier

Occupying the first 4 bytes of a Raw Pixel image, the identifier string must always be set to the ASCII values “RPIX” (hex 52 50 49 58). These characters identify the image as being encoded in RPIX format.

This string is currently independent of the Raw Pixel version.

Image Header Length

The Raw Pixel reader uses the value stored in this field to find the start of the pixel data section within a Raw Pixel image. To find the offset of the pixel data in the image, the reader adds the length of the image identifier (always ‘4’) to the value in the image header length field. Thus, for Raw Pixel 1.0 images with no post-header gap, the pixel data starts at offset 34.

For Raw Pixel version 1.0 images, this field normally contains the integer value '30', which is the length of the Raw Pixel image header (not including the image identifier). However, the Raw Pixel format allows this field to contain any value equal to or greater than 30. Any information in the space between the end of the header data and the start of the pixel data specified by this header length is ignored by the Raw Pixel reader. This is useful for users who wish to prepend a Raw Pixel header onto an existing image whose pixel data area is compatible with Raw Pixel. In this case, the header length would be set to 30 plus the length of the existing header. The maximum length of this header is 4,294,967,265 bytes (the maximum value that can be stored in the 4-byte unsigned field minus the 30-byte header required by Raw Pixel). This field is stored in big endian byte order.

Major Version

A single-byte integer containing the major version number of the Raw Pixel format version used to encode the image. The current Raw Pixel version is "1.0", therefore this field is '1'.

Minor Version

A single-byte integer containing the minor version number of the Raw Pixel format version used to encode the image. The current Raw Pixel version is "1.0", therefore this field is '0'.

Image Width

The width (x-dimension) of the image in pixels.

Although this field is capable of storing an image dimension in excess of 4 billion pixels, limitations within *inter*Media require that this field be in the range 1<= width <= 32767. This field is stored in big endian byte order.

Image Height

The height (y-dimension) of the image in pixels.

Although this field is capable of storing an image dimension in excess of 4 billion pixels, limitations within *inter*Media require that this field be in the range 1 <= height <= 32767. This field is stored in big endian byte order.

Compression Type

This field contains the compression type of the Raw Pixel image. As of version 1.0, this field may contain the following values:

Value	Name	Compression
1	NONE	No compression
2	FAX3	CCITT Group 3 compression
3	FAX4	CCITT Group 4 compression

For grayscale, RGB, and N-band images, the image is always uncompressed, and only a value of 0 is valid. If the compression type is value 1 or 2, then the image is presumed to be monochrome. In this case the image is presumed to contain only a single band, and must specify normal pixel order, normal scanline order, and BIP interleave.

Pixel Order

This field describes the pixel order within the Raw Pixel image. Normally, pixels in a scanline are ordered from left to right, along the traditional positive x-axis. However, some applications require that scanlines be ordered from right to left.

This field may contain the following values:

Value	Name	Pixel Order
1	NORMAL	Leftmost pixel first
2	REVERSE	Rightmost pixel first

This field cannot contain 0, as this indicates an unspecified pixel order; this would mean the image could not be interpreted. For images with CCITT G3 and G4 compression types, this field must contain the value ‘1’.

Scanline Order

This field describes the scanline order within the Raw Pixel image. Normally, scanlines in an image are ordered from top to bottom. However, some applications require that scanlines are ordered from bottom to top.

This field may contain the following values:

Value	Name	Scanline Order
1	NORMAL	Topmost scanline first
2	INVERSE	Bottommost scanline first

This field cannot contain 0, as this indicates an unspecified scanline order; this would mean the image could not be interpreted. For images with CCITT G3 and G4 compression types, this field must contain the value 1.

Interleave

This field describes the interleaving of the various bands within a Raw Pixel image. For more information on the meaning of the various interleave options, see Section E.5.3.

This field may contain the following values:

Value	Name	Interleave
1	BIP	Band Interleave by Pixel, or "chunky"
2	BIL	Band Interleave by Line
3	BSQ	Band SeQuential, or "planar"

This field cannot contain 0, as this indicates an unspecified interleave; this would mean the image could not be interpreted. For images with CCITT G3 and G4 compression types, this field must contain the value 1.

Number of Bands

This field contains the number of bands or planes in the image, and must be in the range 1 <= number of bands <= 255. This field may not contain the value 0.

For CCITT images, this field must contain the value 1.

Red Channel Number

This field contains the number of the band that is to be used as the red channel during image conversion operations. This may be used to change the interpretation of a normal RGB image, or to specify a default band to be used as red in an N-band image. This default may be overridden using the inputChannels operator in the process( ) or processCopy( ) methods.

If the image has only one band, or only one band from an N-band image should be selected for display, then the band number should be encoded as the red channel. In this case, the green and blue channels should be set to 0.

This field may not contain the value 0; only values in the range (1 <= red <= number of bands) may be specified.

Green Channel Number

This field contains the number of the band that is to be used as the green channel during image conversion operations. This may be used to change the interpretation of a normal RGB image, or to specify a default band to be used as green in an

N-band image. This default may be overridden using the inputChannels operator in the process( ) or processCopy( ) method.

If the image has only one band, or only one band from an N-band image should be selected for display, then the band number should be encoded as the red channel. In this case, the green and blue channels should be set to 0.

This field may contain values in the range 0 <= green <= number of bands.

Blue Channel Number

This field contains the number of the band that is to be used as the blue channel during image conversion operations. This may be used to change the interpretation of a normal RGB image, or to specify a default band to be used as blue in an N-band image. This default may be overridden using the inputChannels operator in the process( ) or processCopy( ) method.

If the image has only one band, or only one band from an N-band image should be selected for display, then the band number should be encoded as the red channel. In this case, the green and blue channels should be set to 0.

This field may contain values in the range 0 <= blue <= number of bands.

Reserved Area

The application of these 8 bytes titled Reserved Area is currently under development, but they are reserved even within Raw Pixel 1.0 images. These bytes must all be cleared to zero. Failure to do so will create undefined results.

E.4 Raw Pixel Post-Header Gap

Apart from the image identifier and the image header, Raw Pixel version 1.0 images contain an optional post-header gap, which precedes the actual pixel data. Unlike the reserved area of the image header, the bytes in this gap can contain any values you want. This is useful to store additional metadata about the image, which in some cases may be the actual image header from another file format.

However, because there is no standard for the information stored in this gap, care must be taken if metadata is stored in this area as other users may interpret this data differently. It is also worth noting that when a Raw Pixel image is processed, information stored in this gap is not copied to the destination image. In the case of the process( ) method, which writes its output to the same location as the input, the source information will be lost unless the transaction in which the processing took place is rolled back.

E.5 Raw Pixel Data Section and Pixel Data Format

The data section of a Raw Pixel image is where the actual pixel data of an image is stored; this area is sometimes called the *bitmap data*. This section describes the layout of the bitmap data.

For images using CCITT compression, the bitmap data area stores the raw CCITT stream with no additional header. The rest of this section applies only to uncompressed images.

Bitmap data in a Raw Pixel image is stored as 8-bit per plane, per pixel, direct color, packed data. There is no pixel, scanline, or band blocking or padding. Scanlines may be presented in the image as either topmost first, or bottommost first. Within a scanline, pixels may be ordered leftmost first, or rightmost first. All these options are affected by interleaving in a relatively straightforward way; see the sections that follow for examples.

E.5.1 Scanline Ordering

On the screen, an image may look like the following:

1111111111...
2222222222...
3333333333...
4444444444...

Each digit represents a single pixel; the value of the digit is the scanline that the pixel is on.

Generally the scanline that forms the upper or topmost row of pixels is stored in the image data stream before lower scanlines. The preceding image would appear as follows in the bitmap data stream:

...1111111111...2222222222...3333333333...4444444444...
Note that the first scanline appears earlier than the remaining scanlines. The Raw Pixel format refers to this scanline ordering as normal.

However, some applications prefer that the bottommost scanline appear in the data stream first:

...4444444444...3333333333...2222222222...1111111111...
The Raw Pixel format refers to this scanline ordering as inverse.

E.5.2 Pixel Ordering

On the screen, a scanline of an image may look like the following:

...123456789...

Each digit represents a single pixel; the value of the digit is the column that the pixel is on.

Generally the data that forms the leftmost pixels is stored in the image data stream before pixels toward the right. The preceding scanline would appear as follows in the bitmap data stream:

...123456789...
Note that the left pixel appears earlier than the remaining pixels. The Raw Pixel format refers to this pixel ordering as normal.

However, some applications prefer that the rightmost pixel appear in the data stream first:

...987654321...
The Raw Pixel format refers to this pixel ordering as reverse.

E.5.3 Band Interleaving

Band interleaving describes the relative location of different bands of pixel data within the image buffer.

Bands are ordered by their appearance in an image data stream, with 1 being the first band, *n* being the last band. Band 0 would indicate no band or no data.

Band Interleaved by Pixel (BIP), or *Chunky*

BIP, or *chunky*, images place the various bands or channels of pixel data sequentially by pixel, so that all data for one pixel is in one place. If the bands of the image are the red, green, and blue channels, then a BIP image might look like this:

```
scanline 1: RGBRGBRGBRGBRGBRGBRGB...
scanline 2: RGBRGBRGBRGBRGBRGBRGB...
scanline 3: RGBRGBRGBRGBRGBRGBRGB...
...
```

Band Interleaved by Line (BIL)

BIL images place the various bands of pixel data sequentially by scanline, so that data for one pixel is spread across multiple notional rows of the image. This reflects the data organization of a sensor that buffers data by scanline. If the bands of the image are the red, green, and blue channels, then a BIL image might look like this:

```
scanline 1: RRRRRRRRRRRRRRRRRRRRR...
            GGGGGGGGGGGGGGGGGGGGG...
```

```
            BBBBBBBBBBBBBBBBBBBB...
scanline 2: RRRRRRRRRRRRRRRRRRRR...
            GGGGGGGGGGGGGGGGGGGG...
            BBBBBBBBBBBBBBBBBBBB...
scanline 3: RRRRRRRRRRRRRRRRRRRR...
            GGGGGGGGGGGGGGGGGGGG...
            BBBBBBBBBBBBBBBBBBBB...
...
```

Band Sequential (BSQ), or Planar

Planar images place the various bands of pixel data sequentially by bit plane, so that data for one pixel is spread across multiple planes of the image. This reflects the data organization of some video buffer systems, which control the different electron guns of a display from different locations in memory. If the bands of the image are the red, green, and blue channels, then a planar image might look like this:

```
plane 1: RRRRRRRRRRRRRRRRRR...  (part of scanline 1)
         RRRRRRRRRRRRRRRRRR...  (part of scanline 2)
         RRRRRRRRRRRRRRRRRR...  (part of scanline 3)
...
plane 2: GGGGGGGGGGGGGGGGGG...  (part of scanline 1)
         GGGGGGGGGGGGGGGGGG...  (part of scanline 2)
         GGGGGGGGGGGGGGGGGG...  (part of scanline 3)
...
plane 3: BBBBBBBBBBBBBBBBBB...  (part of scanline 1)
         BBBBBBBBBBBBBBBBBB...  (part of scanline 2)
         BBBBBBBBBBBBBBBBBB...  (part of scanline 3)
...
```

E.5.4 N-Band Data

The Raw Pixel format supports up to 255 bands of data in an image. The relative location of these bands of data in the image is described in Section E.5.3, which gives examples of interleaving for 3 bands of data.

In the case of a single band of data, there is no interleaving; all three schemes are equivalent. Examples of interleaving other numbers of bands are given in the following table. All images have three scanlines and four columns. Each band of each pixel is represented by a single-digit band number. Normal text numbers in *italic* represent the second scanline of the image, and numbers in **boldface** represent the third scanline of the image.

Bands	BIP	BIL	BSQ
2	12121212 *12121212* **12121212**	11112222 *11112222* **11112222**	1111*1111***1111** 2222*2222***2222**
4	1234123412341234 *1234123412341234* 1**234123412341234**	1111222233334444 *1111222233334444* **1111222233334444**	1111*1111***1111** 2222*2222***2222** 3333*3333***3333** 4444*4444***4444**
5	12345123451234512345 *12345123451234512345* **12345123451234512345**	11112222333344445555 *11112222333344445555* **11112222333344445555**	1111*1111***1111** 2222*2222***2222** 3333*3333***3333** 4444*4444***4444** 5555*5555***5555**

E.6 Raw Pixel Header “C” Structure

The following C language structure describes the Raw Pixel header in a programmatic way. This structure is stored unaligned in the image file (that is, fields are aligned on 1 byte boundaries) and all integers are stored in big endian byte order.

```
struct RawPixelHeader
{
unsigned char identifier[4]; /* Always "RPIX" */

unsigned longhdrlength; /* Length of this header in bytes */
/* Including the hdrlength field */
/* Not including the identifier field */
/* &k.hdrlength + k.hdrlength = pixels */

unsigned char majorversion; /* Major revision # of RPIX format */
unsigned char minorversion; /* Minor revision # of RPIX format */

unsigned long width;  /* Image width in pixels */
unsigned long height; /* Image height in pixels */
unsigned char comptype;   /* Compression (none, FAXG3, FAXG4, ... ) */
unsigned char pixelorder; /* Pixel order */
unsigned char scnlorder;  /* Scanline order */
unsigned char interleave; /* Interleaving (BIP/BIL/Planar) */

unsigned char numbands; /* Number of bands in image (1-255) */
unsigned char rchannel; /* Default red channel assignment */
unsigned char gchannel; /* Default green channel assignment */
```

```
unsigned char bchannel; /* Default blue channel assignment */
/* Grayscale images are encoded in R */
/* The first band is '1', not '0' */
/* A value of '0' means "no band" */

unsigned char reserved[8]; /* For later use */
};
```

E.7 Raw Pixel Header “C” Constants

The following C language constants define the values used in the Raw Pixel header.

```
#define RPIX_IDENTIFIER "RPIX"

#define RPIX_HEADERLENGTH 30

#define RPIX_MAJOR_VERSION 1
#define RPIX_MINOR_VERSION 0

#define RPIX_COMPRESSION_UNDEFINED 0
#define RPIX_COMPRESSION_NONE 1
#define RPIX_COMPRESSION_CCITT_FAX_G3 2
#define RPIX_COMPRESSION_CCITT_FAX_G4 3
#define RPIX_COMPRESSION_DEFAULT RPIX_COMPRESSION_NONE

#define RPIX_PIXEL_ORDER_UNDEFINED 0
#define RPIX_PIXEL_ORDER_NORMAL 1
#define RPIX_PIXEL_ORDER_REVERSE 2
#define RPIX_PIXEL_ORDER_DEFAULT RPIX_PIXEL_ORDER_NORMAL

#define RPIX_SCANLINE_ORDER_UNDEFINED 0
#define RPIX_SCANLINE_ORDER_NORMAL 1
#define RPIX_SCANLINE_ORDER_INVERSE 2
#define RPIX_SCANLINE_ORDER_DEFAULT RPIX_SCANLINE_ORDER_NORMAL

#define RPIX_INTERLEAVING_UNDEFINED 0
#define RPIX_INTERLEAVING_BIP 1
#define RPIX_INTERLEAVING_BIL 2
#define RPIX_INTERLEAVING_BSQ 3
#define RPIX_INTERLEAVING_DEFAULT RPIX_INTERLEAVING_BIP

#define RPIX_CHANNEL_UNDEFINED 0
```

Note that the various macros for the UNDEFINED values are meant to be illustrative and not necessarily used, except for "RPIX_CHANNEL_UNDEFINED" which is used for the green and blue channels of single band images.

E.8 Raw Pixel PL/SQL Constants

The following PL/SQL constants define the values used in the raw pixel information. The constants represent the length of the RPIX image identifier plus the length of the RPIX header.

```
CREATE OR REPLACE PACKAGE ORDImageConstants AS
  RPIX_HEADER_LENGTH_1_0   CONSTANT INTEGER := 34;
END ORDImageConstants;
```

E.9 Raw Pixel Images Using CCITT Compression

Although the Raw Pixel format is generally aimed at uncompressed direct color images, provision is also made to store monochrome images using CCITT Fax Group 3 or Fax Group 4 compression. This is useful for storing scans of black and white pages, such as for document management applications. These images are generally impractical to store as even grayscale, as the unused data bits combined with the very high resolution used in these images would use excessive disk space.

Raw Pixels images using CCITT compression are treated as normal Raw Pixel images, with the following restrictions:

- The “compression type” field must contain the value 1 or 2 as outlined in Section E.3 (FAX3 or FAX4).
- The “pixel order” field must contain the value 1 (normal pixel order).
- The “scanline order” field must contain the value 1 (normal scanline order).
- The “interleave” field must contain the value 1 (BIP interleave).
- The “number of bands” field must contain the value 1 (one band).
- The “red channel number” field must contain the value 1.
- The “green channel number” and “blue channel number” fields must contain the value 0 (no band).

In addition to these restrictions, applications which attempt to access pixel data directly will need to understand how to read and write the CCITT formatted data.

E.10 Foreign Image Support and the Raw Pixel Format

*inter*Media provides support for reading certain foreign images that can be described in terms of a few simple parameters, and whose data is arranged in a certain straightforward way within the image file. There is no list of the supported formats because the list would be so large and continually changing. Instead, there are some simple guidelines to determine if an image can be read using the foreign image support in *inter*Media. These rules are summarized in the following sections.

Header

Foreign images may have any header (or no header), in any format, as long as its length does not exceed 4,294,967,265 bytes. As has been noted before, all information in this header will be ignored.

Image Width

Foreign images may be up to 32,767 pixels wide.

Image Height

Foreign images may be up to 32,767 pixels high.

Compression Type

Foreign images must be uncompressed or compressed using CCITT Fax Group 3 or Fax Group 4. Other compression schemes, such as run-length encoding, are not currently supported.

Pixel Order

Foreign images may store pixels from left-to-right or right-to-left. Other pixel ordering schemes, such as boustrophedonic ordering, are not currently supported.

Scanline Order

Foreign images may have top-first or bottom-first scanline orders. Scanlines that are adjacent in the image display must be adjacent in the image storage. Some image formats stagger their image scanlines so that, for example, scanlines 1,5,9, and so forth are adjacent, and then 2,6,10 are also adjacent. This is not currently supported.

Interleaving

Foreign images must use BIP, BIL, or BSQ interleaving. Other arrangements of data bands are not allowed, nor may bands have any pixel, scanline, or band-level blocking or padding.

Number of Bands

Foreign images may have up to 255 bands of data. If there are more bands of data, the first 255 can be accessed *if* the interleaving of the image is "band sequential." In this case, the additional bands of data lie past the accessible bands and do not affect the layout of the first 255 bands. Images with other interleaving types may not have more than 255 bands because the additional bands will change the layout of the bitmap data.

Trailer

Foreign images may have an image trailer following the bitmap data, and this trailer may be of arbitrary length. However, such data is completely ignored by *inter*Media, and there is no method (or need) to specify the presence or length of such a trailer.

If an image with such a trailer is modified with the process( ) or processCopy( ) methods, the resulting image will not contain this trailer. In the case of the processCopy( ) method, the source image will still be intact.

F

Sample Programs

Oracle *inter*Media includes a number of scripts and sample programs that you can use.

Sample Oracle *inter*Media scripts and programs are available in the following directories after you install this product:

$ORACLE_HOME/ord/aud/demo/
$ORACLE_HOME/ord/doc/demo/
$ORACLE_HOME/ord/img/demo/
$ORACLE_HOME/ord/vid/demo/

F.1 Sample Audio Scripts

The audio scripts consist of the following files:

- auddemo.sql - audio demonstration (demo) that shows features of the audio object including:
  - Checking *inter*Media objects
  - Creating a sample table with audio in it
  - Inserting NULL rows into the audio table
  - Checking the rows out
  - Checking all the audio attributes directly
  - Checking all the audio attributes by calling methods
  - Installing your own format plug-in using the two files, fplugins.sql and fpluginb.sql described in the next two list items and in Section 3.1.12 on how to extend *inter*Media audio services to support a new audio data format

- fplugins.sql - demo format plug-in specification that you can use as a guideline to write any format plug-in you want to support
- fpluginb.sql - demo format plug-in body that you can use as a guideline to write any format plug-in you want to support

See the README.txt file in the *$ORACLE_HOME*/ord/aud/demo directory for requirements and instructions on running this SQL demo.

See Section F.5 for a description of the Java demo that is provided to help you learn to use the audio client-side Java classes so you can build your own applications.

F.2 Sample Document Scripts

The document scripts consist of the following files:

- docdemo.sql - document demonstration (demo) that shows features of the document object and includes the testaud.dat, testdoc.dat, testimg.dat, and testvid.dat files.

See the README.txt files in the *$ORACLE_HOME*/ordim/demo/doccheck and *ORACLE_HOME*/ordim/demo/doccjpub directories for requirements and instructions on running this SQL demo.

See Section F.5 for a description of the Java demo that is provided to help you learn to use the document client-side Java classes so you can build your own applications.

F.3 Sample Program for Modifying Images or Testing the Image Installation

Once you have installed Oracle *inter*Media, you may choose to run the Oracle *inter*Media image services demonstration program. This program can also be used as a test to confirm successful installation.

This section contains the steps required to build and run the *inter*Media image services demo.

The *inter*Media image services demo files are located in <*ORACLE_HOME*>*/ord/img/*demo, where <*ORACLE_HOME*> is the ORACLE_HOME directory.

F.3.1 Demonstration (Demo) Installation Steps

For *inter*Media image services, see the README.txt file at <*ORACLE_HOME*>/ord/img/demo/README.txt (on UNIX), and <*ORACLE_HOME*>\ord\img\demo\README.txt (on Windows NT), where <*ORACLE_HOME*> is the ORACLE_HOME directory.

F.3.2 Running the Demo

The file imgdemo is a sample program that shows how Oracle *inter*Media image services can be used from within a program. The demo is written in C and uses OCI, Oracle Call Interface, to access the database and exercise Oracle *inter*Media image services.

The program operates on imgdemo.dat, which is a bitmap (BMP) image in the demo directory. Optionally, you can supply an image file name on the command line, provided the file resides in the same directory as the demo. In either case, once the image has been manipulated by Oracle *inter*Media image services, the resulting image is written to the file imgdemo.out and can then be viewed with common rendering tools that you supply.

When the demo is run, it deletes and re-creates a table named IMGDEMOTAB in the SCOTT/TIGER schema of the default database. This table is used to hold the demo data. Once the table is created, a reference to the image file is inserted into the table. The data is then loaded into the table and converted to JFIF using the processCopy( ) method of ORDImage.

The image properties are extracted within the database using the setProperties( ) method. An UPDATE command is issued after the setProperties( ) invocation. This is required because the setProperties( ) invocation has only updated a local copy of the type attributes.

Next, the Oracle *inter*Media image services process( ) method is used to cut and scale the image within the database. This is followed by an update that commits the change. The program cuts a portion of the image 100 pixels wide by 100 pixels high starting from pixel location (100,100). This subimage is scaled to twice its original size and the resulting image is written out to the file system in a file named imgdemo.out.

Upon completion, the demo program leaves the imgdemo.out file in the current directory. It also leaves the table IMGDEMOTAB in the SCOTT/TIGER schema of the database.

Execute the demo by typing imgdemo on the command line. Optionally, a different image can be used in the demo by first copying the file to the directory in which the

demo resides and then specifying its file name on the command line as an argument to imgdemo.

Use the command shown in Example F–1.

Example F–1 Execute the Demo from the Command Line

```
$ imgdemo <optional-image-filename>
```

The demo displays a number of messages describing its progress, along with any errors encountered in the event that something was not set up correctly. Expect to see the following messages:

```
Dropping table IMGDEMOTAB...
Creating and populating table IMGDEMOTAB...
Loading data into cartridge...
Modifying image characteristics...
Writing image to file imgdemo.out...
Disconnecting from database...
Logged off and detached from server.
Demo completed successfully.
```

If the program encounters any errors, it is likely that either Oracle *inter*Media image services software has not been installed correctly or the database has not been started. If the program completes successfully, the original image and the resultant image, which has undergone the cutting and scaling described earlier, can be viewed with common image rendering tools.

See Section F.5 for a description of the Java demo that is provided to help you learn to use the image client-side Java classes so you can build your own applications.

F.4 Sample Video Scripts

The video scripts consist of the following files:

- viddemo.sql - video demo that shows features of the video object including:
  - Checking *inter*Media objects
  - Creating a sample table with video in it
  - Inserting NULL rows into the video table
  - Checking the rows out
  - Checking all the video attributes directly

  - Checking all the video attributes by calling methods
  - Installing your own format plug-in using the two files, fplugins.sql and fpluginb.sql described in the next two list items and in Section 3.4.12 on how to extend *inter*Media video services to support a new video data format

- fplugins.sql - demo format plug-in specification that you can use as a guideline to write any format plug-in you want to support
- fpluginb.sql - demo format plug-in body that you can use as a guideline to write any format plug-in you want to support

See the README.txt file in the *$ORACLE_HOME*/ord/vid/demo directory for requirements and instructions on how to run this SQL demo.

See Section F.5 for a description of the Java demo that is provided to help you learn to use the video client-side Java classes so you can build your own applications.

F.5 Java Demo

A Java demo has been provided to help you learn to use both the audio, video, image, and document client-side Java classes so you can build your own applications. In these four demos, the audio, video, image, and document object is instantiated at the client side and a number of accessor methods are invoked. The audio Java demo files are located in the *ORACLE_HOME*/ord/aud/demo/java directory, the video Java demo files are located in the *$ORACLE_HOME*/ord/vid/demo/java directory, the image Java demo files are located in the *ORACLE_HOME*/ord/img/demo/java directory, and the document Java demo files are located in the *$ORACLE_HOME*/ord/doc/demo/java directory. See the README.txt file in each directory for requirements and instructions on how to run each respective Java demo.

G

Frequently Asked Questions

A text file containing a list of frequently asked questions is available on line after installing Oracle *inter*Media.

This text file can be found as follows:

$ORACLE_HOME/ord/im/admin/imfaq.txt

H

Exceptions and Error Messages

H.1 Exceptions

The following sections describe the exceptions and error messages of *inter*Media objects.

H.1.1 ORDAudioExceptions Exceptions

The following exceptions are associated with the ORDAudio object:

LOCAL_DATA_SOURCE_REQUIRED

Cause: This exception is raised if the data source is external.

Action: Set the source information to a local source.

DESCRIPTION_IS_NOT_SET

Cause: This exception is raised when calling the getDescription function and the description attribute is not set.

Action: Set the description attribute.

INVALID_DESCRIPTION

Cause: This exception is raised when you call the setDescription( ) method with a value that is not valid.

Action: Set the value of the user_description parameter to an acceptable value.

INVALID_MIME_TYPE

Cause: This exception is raised if the MIME parameter value of the setMimeType procedure is NULL.

Action: Set the MIME parameter value to a known value.

AUDIO_FORMAT_IS_NULL

Cause: This exception is raised when calling the getFormat function and the format is NULL.

Action: Set the format for the audio object to a known format.

AUDIO_ENCODING_IS_NULL

Cause: This exception is raised when calling the getEncoding function and the encoding is NULL.

Action: Set the encoding for the audio object to a known value.

AUDIO_NUM_CHANNELS_IS_NULL

Cause: This exception is raised when calling the getNumberOf Channels function and the number of channels is NULL.

Action: Set the number of channels for the audio object to a known value.

AUDIO_SAMPLING_RATE_IS_NULL

Cause: This exception is raised when calling the getSamplingRate function and the sampling rate is NULL.

Action: Set the sampling rate for the audio object to a known value.

AUDIO_SAMPLE_SIZE_IS_NULL

Cause: This exception is raised when calling the getSampleSize function and the sample size is NULL.

Action: Set the sample size for the audio object to a known value.

AUDIO_DURATION_IS_NULL

Cause: This exception is raised when calling the getAudioDuration function and the duration is NULL.

Action: Set the duration for the audio object to a known value.

NULL_INPUT_VALUE

Cause: This exception is raised if the knownFormat parameter value of the setFormat procedure is NULL.

Action: Set these parameters with known values.

METHOD_NOT_SUPPORTED

Cause: This exception is raised when the method called is not supported.

Action: Call a supported method.

AUDIO_PLUGIN_EXCEPTION

Cause: This exception is raised when the audio plug-in raises an exception.

Action: Refer to Section 6.4.1 for more information.

H.1.2 ORDDocExceptions Exceptions

The following exceptions are associated with the ORDDoc object:

DOC_PLUGIN_EXCEPTION

Cause: This exception is raised when the document plug-in raises an exception.

Action: Refer to Section 7.4.1 for more information.

INVALID_MIME_TYPE

Cause: This exception is raised if the MIME parameter value of the setMimeType procedure is NULL.

Action: Set the MIME parameter value to a known value.

INVALID_FORMAT_TYPE

Cause: This exception is raised if the FORMAT parameter value of the setFormat procedure is NULL.

Action: Set the FORMAT parameter value to a known value.

METHOD_NOT_SUPPORTED

Cause: This exception is raised when the method called is not supported.

Action: Call a supported method.

NULL_INPUT_VALUE

Cause: This exception is raised if the knownFormat parameter value of the setFormat procedure is NULL.

Action: Set these parameters with known values.

H.1.3 ORDImageExceptions Exceptions

The following exceptions are associated with the ORDImage object:

NULL_LOCAL_DATA

Cause: This exception is raised when source.localData is NULL.

Action: Initialize source.localData with an empty_blob( ).

NULL_PROPERTIES_DESCRIPTION

Cause: This exception is raised when the description parameter to setProperties is not set.

Action: Set the description attribute if you are using a foreign image. Otherwise, do not pass the description parameter.

NULL_DESTINATION

Cause: This exception is raised when the destination image is NULL.

Action: Pass an initialized destination image.

DATA_NOT_LOCAL

Cause: This exception is raised when the source information is not set to local.

Action: Reset the source attribute information to a local image source. Call the import( ) or importFrom( ) method to import the data into the local BLOB.

NULL_CONTENT

Cause: This exception is raised when the content attribute of an ORDImgB or ORDImgF image is NULL.

Action: Initialize the content attribute.

NULL_SOURCE

Cause: This exception is raised when the source image is NULL.

Action: Pass an initialized source image.

H.1.4 ORDVideoExceptions Exceptions

The following exceptions are associated with the ORDVideo object:

LOCAL_DATA_SOURCE_REQUIRED

Cause: This exception is raised if the data source is external.

Action: Set the source information to a local source.

DESCRIPTION_IS_NOT_SET

Cause: This exception is raised when calling the getDescription function and the description attribute is not set.

Action: Set the description attribute.

INVALID_MIME_TYPE

Cause: This exception is raised if the MIME parameter value of the setMimeType procedure is NULL.

Action: Set the MIME parameter value to a known value.

VIDEO_FORMAT_IS_NULL

Cause: This exception is raised when calling the getFormat function and the format is NULL.

Action: Set the format for the video object to a known format.

NULL_INPUT_VALUE

Cause: This exception is raised if either the knownWidth or knownHeight parameter values of the setFrameSize procedure is NULL.

Action: Set these parameters with known values.

METHOD_NOT_SUPPORTED

Cause: This exception is raised when the method called is not supported.

Action: Call a supported method.

VIDEO_PLUGIN_EXCEPTION

Cause: This exception is raised when the video plug-in raises an exception.

Action: Refer to Section 9.4.1 for more information.

H.1.5 ORDSourceExceptions Exceptions

The following exceptions are associated with the ORDSource object:

INCOMPLETE_SOURCE_INFORMATION

Cause: This exception is raised when the source information is incomplete or srcType is NULL and data is not stored locally in the BLOB.

Action: Check your source information and set srcType, srcLocation, or srcName attributes as needed.

INCOMPLETE_SOURCE_LOCATION

Cause: This exception is raised when the value of srcLocation is NULL.

Action: Check your source location and set the srcLocation attribute.

INCOMPLETE_SOURCE_NAME

Cause: This exception is raised when the value of srcName is NULL.

Action: Check your source name and set the srcName attribute.

EMPTY_SOURCE

Cause: This exception is raised when the source is local but the source is NULL.

Action: Pass an initialized source.

NULL_SOURCE

Cause: This exception is raised when the local source is NULL.

Action: Pass an initialized source.

INVALID_SOURCE_TYPE

Cause: This exception is raised when the getBFile method detects a source type other than 'file'.

Action: Ensure that the source type is 'file'.

METHOD_NOT_SUPPORTED

Cause: This exception is raised when the method called is not supported.

Action: Call a supported method.

SOURCE_PLUGIN_EXCEPTION

Cause: This exception is raised when the source plug-in raises an exception.

Action: Refer to Section I.3.1, Section I.3.2, and Section I.3.3 for more information.

H.2 ORDAudio Error Messages

AUD-00702 unable to initialize audio processing environment

Cause: The initalization of the audio processing external procedure failed.

Action: See the database administrator to make sure that enough memory has been allocated to JServer. If JServer does have enough memory, contact Oracle Customer Support Services.

AUD-00703 unable to read audio data

Cause: An error occurred while accessing the audio source.

Action: Make sure the audio source is valid. For external sources, make sure all access priviliges are granted.

AUD-00704 invalid input format

Cause: The audio data in the source was not in the format specified by the format field of the audio object. In some unusual case, the audio data is actually corrupted.

Action: Provide a correct value in the format field. If the correct value is unknown, put NULL in the format field to invoke the DEFAULT format plug-in.

AUD-00705 unsupported input format

Cause: The file format of the audio data was not supported. This error can only occur in the DEFAULT format plug-in package.

Action: Refer to *Oracle interMedia User's Guide and Reference* for supported formats.

AUD-00706 unsupported or corrupted input format

Cause: The audio data was either corrupted or the file format was not supported.

Action: Refer to *Oracle interMedia User's Guide and Reference* for supported formats. If the audio data is not corrupted and is in a supported file format, contact Oracle Customer Support Services.

AUD-00713 internal error while parsing audio data

Cause: An internal error occurred during parsing.

Action: Contact Oracle Customer Support Services.

AUD-00714 internal error

Cause: An internal error occurred.

Action: Contact Oracle Customer Support Services.

H.3 ORDImage Error Messages

IMG-00001, "unable to initialize Oracle interMedia environment"

Cause: The image processing external procedure initialization process failed.

Action: Contact Oracle Customer Support Services.

IMG-0002,"unrecoverable error"

Cause: This is an internal error.

Action: Contact Oracle Customer Support Services.

IMG-00502, "invalid scale value"

Cause: An invalid scale value was found while parsing the parameters for the image process function.

Action: Correct the statement by using a valid scale value. Refer to *Oracle interMedia User's Guide and Reference* documentation for a description of the correct usage and syntax for the image processing command string.

IMG-00505, "missing value in CUT rectangle"

Cause: An incorrect number of values was used to specify a rectangle.

Action: Use exactly four integer values for the lower-left and upper-right vertices.

IMG-00506, "extra value in CUT rectangle"

Cause: An incorrect number of values was used to specify a rectangle.

Action: Use exactly four integer values for the lower-left and upper-right vertices.

IMG-00510, application-specific-message

Cause: A syntax error was found while parsing the parameters for the image process function.

Action: Correct the statement by using valid parameter values. Refer to *Oracle interMedia User's Guide and Reference* documentation for a description of the correct usage and syntax for the image processing command string.

IMG-00511, application-specific-message

Cause: An error was found while accessing image data.

Action: Contact Oracle Customer Support Services.

IMG-00512, "multiple incompatible scaling parameters found"

Cause: Multiple incompatible scaling parameters were found in the image process command string. With the exception of XSCALE and YSCALE, which can be used together in a process command string, scaling functions are mutually exclusive and cannot be combined.

Action: Remove scaling functions until only one remains (or two, if they are XSCALE and YSCALE).

IMG-00513, "missing value in scaling operation"

Cause: An incorrect number of values was used to specify image dimensions. fixedScale and maxScale require exactly two integer values for the X and Y dimensions of the desired image.

Action: Use two values for fixedScale and maxScale.

IMG-00514, "extra value in scaling operation"

Cause: An incorrect number of values was used to specify image dimensions. fixedScale and maxScale require exactly two integer values for the X and Y dimensions of the desired image.

Action: Use two values for fixedScale and maxScale.

IMG-00515, "incorrect number of input channels"

Cause: An incorrect number of values was used to specify input channels. InputChannels requires either one or three channel numbers for the gray or red, green, and blue channel assignments.

Action: Use either one or three values to specify the input channels.

IMG-00516, "default channel out of range"

Cause: An incorrect value was used to specify the default channel selection.

Action: Use a channel number that is less than or equal to the number of bands and greater than zero.

IMG-00517, "height or width not present in parameter string"

Cause: Height and/or width were not specified in the setProperties parameter string.

Action: Specify both the height and width.

IMG-00518, "invalid value for height or width"

Cause: Height and width were not positive integers.

Action: Specify both the height and width as positive integers.

IMG-00519, "illegal combination of parameters"

Cause: Other than height, width, dataOffset, and userString, no other parameters may be specified in the setProperties parameter string when CCITTG3 or CCITTG4 is used as the compressionFormat.

Action: Supply only the height and width when compressionFormat is either CCITTG3 or CCITTG4. The dataOffset and userString may optionally be supplied as well.

IMG-00520, "invalid value for numberOfBands"

Cause: NumberOfBands was not a positive integer.

Action: Specify numberOfBands as a positive integer.

IMG-00521, "invalid value for dataOffset"

Cause: dataOffset was not a positive integer.

Action: Specify dataOffset as a positive integer.

IMG-00522, "invalid format for parameter value"

Cause: A floating-point value was specified where an integer is required, or a character value was specified where a numeric value is required.

Action: Specify the correct type of values for process parameters.

IMG-00523, "invalid process verb"

Cause: A process verb was specified that is not understood by Oracle *inter*Media.

Action: Refer to the Oracle *inter*Media documentation for a description of valid process verbs.

IMG-00524, "mismatched quotes"

Cause: Quotation marks used within a process command string were not matched.

Action: Ensure that quotation marks occur in pairs.

IMG-00525,"locale error"

Cause: This is an internal error.

Action: Contact Oracle Customer Support Services.

IMG-00526,"error parsing foreign image description"

Cause: An internal error occurred while processing a foreign image.

Action: Use setProperties to correct the foreign image description. Contact Oracle Customer Support Services.

IMG-00530, "internal error while parsing command"

Cause: An internal error occurred while parsing the command passed to the image processing function or the foreign image setProperties function.

Action: Check the command passed to the function. Refer to *Oracle interMedia User's Guide and Reference* for a description of the correct usage and syntax for the image processing command string or the foreign image setProperties function. If you are certain that your command is correct, then contact Oracle Customer Support Services.

IMG-00531, "empty or null image processing command"

Cause: An empty or null image processing command was passed to the image process function.

Action: Refer to *Oracle interMedia User's Guide and Reference* for a description of the correct usage and syntax for the image processing command string.

IMG-00540,"contentFormat and interleave conflict"

Cause: Interleave values were specified using both the contentFormat and interleave verbs.

Action: Specify interleave values using either contentFormat or interleave, but not both.

IMG-00541,"invalid contentFormat specified"

Cause: The specified contentFormat was not valid.

Action: Refer to *Oracle interMedia User's Guide and Reference* for a description of valid contentFormat specifications.

IMG-00542,"contentFormat includes invalid extra information"

Cause: The specified contentFormat included invalid characters at the end of the parameter string.

Action: Refer to *Oracle interMedia User's Guide and Reference* for a description of valid contentFormat specifications.

IMG-00543,"invalid compressionFormat specified"

Cause: The specified compressionFormat was not valid.

Action: Refer to *Oracle interMedia User's Guide and Reference* for a description of valid compressionFormat specifications.

IMG-00544,"invalid compressionQuality specified"

Cause: The specified compressionQuality was not valid.

Action: Refer to *Oracle interMedia User's Guide and Reference* for a description of valid compressionQuality specifications.

IMG-00545,"invalid cut values specified"

Cause: An invalid value was found while parsing the parameters for the cut operation.

Action: Correct the statement by using valid values for the cut operation that are not negative. Refer to *Oracle interMedia User's Guide and Reference* for a description of the correct usage and syntax for the image processing command string.

IMG-00546,"invalid page number specified"

Cause: An invalid page number was specified.

Action: Specify page numbers that are not negative.

IMG-00547,"invalid channelOrder specified"

Cause: The specified channel order was not valid.

Action: Refer to *Oracle interMedia User's Guide and Reference* for a description of valid channelOrder specifications.

IMG-00548,"invalid interleave specified"

Cause: The specified interleave was not valid.

Action: Refer to *Oracle interMedia User's Guide and Reference* for a description of valid interleave specifications.

IMG-00549,"invalid pixelOrder specified"

Cause: The specified pixel order was not valid.

Action: Refer to *Oracle interMedia User's Guide and Reference* for a description of valid pixelOrder specifications.

IMG-00550,"invalid scanlineOrder specified"

Cause: The specified scanline order was not valid.

Action: Refer to *Oracle interMedia User's Guide and Reference* for a description of valid scanlineOrder specifications.

IMG-00551,"invalid dither type specified"

Cause: The specified dither type was not valid.

Action: Refer to *Oracle interMedia User's Guide and Reference* for a description of valid dither specifications.

IMG-00552,"invalid inputChannels specified"

Cause: An invalid value was specified for the inputChannels verb.

Action: Specify non-negative values for inputChannels. Refer to *Oracle interMedia User's Guide and Reference* for a description of the correct usage and syntax for the image processing command string.

IMG-00560,"input format does not support page selection"

Cause: The page verb was specified for an input format that does not support selecting pages.

Action: Remove the page selection verb. Refer to *Oracle interMedia User's Guide and Reference* for a description of which image formats support page selection.

IMG-00561,"input format does not support channel selection"

Cause: The inputChannels verb was specified for an input format that does not support selecting channels.

Action: Remove the inputChannels verb. Refer to *Oracle interMedia User's Guide and Reference* for a description of which image formats support input channel selection.

IMG-00580,"specified format does not support output"

Cause: The output format specified by fileFormat does support output.

Action: Change the specified fileFormat to one that supports output. Refer to *Oracle interMedia User's Guide and Reference* for a description of which formats support output.

IMG-00581,"output format does not support the specified contentFormat"

Cause: The specified contentFormat is not supported by the explicitly or implicitly specified output format.

Action: Refer to *Oracle interMedia User's Guide and Reference* for a description of which contentFormat values are supported for each output format.

IMG-00582,"output format does not support the specified interleave"

Cause: The specified interleave is not supported by the explicitly or implicitly specified output format.

Action: Refer to *Oracle interMedia User's Guide and Reference* for a description of which interleave values are supported for each output format.

IMG-00583,"output format does not support the specified compressionFormat"

Cause: The specified compressionFormat is not supported by the explicitly or implicitly specified output format.

Action: Refer to *Oracle interMedia User's Guide and Reference* for a description of which compressionFormat values are supported for each output format.

IMG-00584,"output format does not support the specified compressionQuality"

Cause: The specified compressionQuality is not supported by the explicitly or implicitly specified output format.

Action: Refer to *Oracle interMedia User's Guide and Reference* for a description of which compressionQuality values are supported for each output format.

IMG-00585,"output format does not support the specified channelOrder"

Cause: The specified channelOrder is not supported by the explicitly or implicitly specified output format.

Action: Refer to *Oracle interMedia User's Guide and Reference* for a description of which channelOrder values are supported for each output format.

IMG-00586,"output format does not support the specified pixelOrder"

Cause: The specified pixelOrder is not supported by the explicitly or implicitly specified output format.

Action: Refer to *Oracle interMedia User's Guide and Reference* for a description of which pixelOrder values are supported for each output format.

IMG-00587,"output format does not support the specified scanlineOrder"

Cause: The specified scanlineOrder is not supported by the explicitly or implicitly specified output format.

Action: Refer to *Oracle interMedia User's Guide and Reference* for a description of which scanlineOrder values are supported for each output format.

IMG-00599, "internal error"

Cause: An internal error has occurred.

Action: Contact Oracle Customer Support Services.

IMG-00601, "out of memory while copying image"

Cause: Operating system process memory has been exhausted while copying the image.

Action: See the database administrator or operating system administrator to increase process memory quota.

IMG-00602, "unable to access image data"

Cause: An error occurred while reading or writing image data.

Action: Contact your system administrator.

IMG-00603, "unable to access source image data"

Cause: The source image SOURCE attribute is invalid.

Action: Ensure that the SOURCE attribute of the source image is populated with image data.

IMG-00604, "unable to access destination image data"

Cause: The destination image SOURCE attribute is invalid.

Action: Ensure that the SOURCE attribute of the destination image is populated with image data.

IMG-00606, "unable to access image data"

Cause: An attempt was made to access an invalid image.

Action: Ensure that the SOURCE attribute of the image is populated with image data.

IMG-00607, "unable to write to destination image"

Cause: The destination image SOURCE attribute is invalid.

Action: Ensure that the SOURCE attribute of the destination image is initialized correctly and that you have sufficient tablespace.

IMG-00609, "unable to read image stored in a BFILE"

Cause: The image stored in a BFILE cannot be opened for reading.

Action: Ensure that the access privileges of the image file and the image file's directory allow read access.

IMG-00701, "unable to set the properties of an empty image"

Cause: There is no data in the image object.

Action: Refer to *Oracle interMedia User's Guide and Reference* for information on how to put image data into the image object.

IMG-00702, "unable to initialize image processing environment"

Cause: The image processing external procedure initialization process failed.

Action: Contact Oracle Customer Support Services.

IMG-00703, "unable to read image data"

Cause: There is no image data in the image object.

Action: Refer to *Oracle interMedia User's Guide and Reference* for information on how to populate image data into the image object.

IMG-00704, "unable to read image data"

Cause: There is no image data in the image object.

Action: Refer to *Oracle interMedia User's Guide and Reference* for information on how to populate image data into the image object.

IMG-00705, "unsupported or corrupted input format"

Cause: This is an internal error.

Action: Contact Oracle Customer Support Services.

IMG-00706, "unsupported or corrupted output format"

Cause: This is an internal error.

Action: Contact Oracle Customer Support Services.

IMG-00707, "unable to access image data"

Cause: An error occurred while reading or writing image data.

Action: Contact your system administrator.

IMG-00710, "unable write to destination image"

Cause: The destination image is invalid.

Action: Ensure that the SOURCE attribute of the destination image is initialized and that you have sufficient tablespace.

IMG-00711, "unable to set properties of destination image"

Cause: This is an internal error.

Action: Contact Oracle Customer Support Services.

IMG-00712, "unable to write to destination image"

Cause: The destination image is invalid.

Action: Ensure that the SOURCE attribute of the destination image is initialized and that you have sufficient tablespace. Ensure the row containing the destination image has been locked (this does not apply to temporary BLOBs).

IMG-00713, "unsupported destination image format"

Cause: A request was made to convert an image to a format that is not supported.

Action: Refer to *Oracle interMedia User's Guide and Reference* for supported formats.

IMG-00714, "internal error"

Cause: This is an internal error.

Action: Contact Oracle Customer Support Services.

IMG-00715, "Unable to open image stored in a BFILE"

Cause: The image stored in a BFILE could not be opened for reading.

Action: Ensure that the access privileges of the image file and the image file's directory allow read access.

IMG-00716, "source image format does not support process options"

Cause: A request was made to apply a processing option not supported by the source image format.

Action: Refer to *Oracle interMedia User's Guide and Reference* for a discussion of supported processing options.

IMG-00717, "destination image format does not support process options"

Cause: A request was made to apply a processing option not supported by the destination image format.

Action: Refer to *Oracle interMedia User's Guide and Reference* for a discussion of supported processing options.

IMG-00718, "the same Temporary LOB cannot be used as both source and destination"

Cause: A call was made to processCopy with the same temporary LOB being specified as both the source and destination.

Action: Specify a different LOB for parameter "dest".

IMG-00719,"image processing internal error"

Cause: This is an internal error.

Action: Contact Oracle Customer Support Services.

IMG-00720,"image processing internal error"

Cause: This is an internal error.

Action: Contact Oracle Customer Support Services.

IMG-00730,"unable to process empty image"

Cause: There is no data in the input image object.

Action: Refer to *Oracle interMedia User's Guide and Reference* for information on how to put image data into the image object.

IMG-00731,"specified page could not be found in input image"

Cause: The specified page does not exist in the input image.

Action: Restrict the value of the page parameter to values specifying pages that exist within the input image object.

IMG-00732,"specified inputChannels could not be found in input image"

Cause: The specified input channel does not exist in the input image.

Action: Restrict the value of the inputChannels parameter to values specifying channels that exist within the input image object.

IMG-00800, "internal error while parsing attribute string"

Cause: An internal error occurred while parsing the attribute string containing the weights of the attributes.

Action: Check the command passed to the function. Refer to *Oracle interMedia User's Guide and Reference* for a description of the correct usage and syntax for the attributes string for image matching. If you are certain that your command is correct, then contact Oracle Customer Support Services.

IMG-00801, "cannot extract height and width"

Cause: Height and width are not set in the image object.

Action: Set the properties of the image object by calling setProperties and then generate the signature.

IMG-00802, "empty or null attributes string"

Cause: An empty or null attributes string was passed to the image matching operators.

Action: Refer to *Oracle interMedia User's Guide and Reference* for a description of the correct usage and syntax of the attributes string.

IMG-00803, "invalid attributes value"

Cause: An invalid value was found while parsing the attributes string for the image matching operators.

Action: Refer to *Oracle interMedia User's Guide and Reference* for a description of the correct usage and syntax for the attributes string. The weight values should be between 0.0 and 1.0.

IMG-00804," Syntax error in attributes string"

Cause: A syntax error was found while parsing the attributes string for the image matching operators.

Action: Refer to *Oracle interMedia User's Guide and Reference* for a description of the correct usage and syntax of the attributes string.

IMG-00805, "SIGNATURE data has been corrupted or is invalid"

Cause: The data in the signature is not a valid signature.

Action: Re-create the signature using the generateSignature method.

IMG-00806, "invalid input image"

Cause: The image data is either corrupt or is in an unsupported format.

Action: Repopulate the image object, set properties of the image, and generate the signature.

IMG-00807, "no weights specified in weight string"

Cause: All weights passed were zero. At least one attribute must be weighted.

Action: Specify a non-zero weight for at least one attribute.

IMG-00808, "unable to read an empty image"

Cause: There is no data in the image object.

Action: Refer to *Oracle interMedia User's Guide and Reference* for information on how to populate the image object with image data.

IMG-00809, "usage of IMGSimilar is incorrect"

Cause: Syntax error while using IMGSimilar.

Action: Refer to *Oracle interMedia User's Guide and Reference* for information on how to use the IMGSimilar operator. Check if the value returned by IMGSimilar is compared to the value 1.

IMG-00810, "Boundary queue initialization failed"

Cause: Operating system process memory has been exhausted while initializing the boundary queue.

Action: See the database administrator or operating system administrator to increase process memory quota.

IMG-00811, "Fail queue initialization failed"

Cause: Operating system process memory has been exhausted while initializing the fail queue.

Action: See the database administrator or operating system administrator to increase process memory quota.

IMG-00812, "Merged area queue initialization failed"

Cause: Operating system process memory has been exhausted while initializing the merged area queue.

Action: See the database administrator or operating system administrator to increase process memory quota.

IMG-00813, "Boundary queue free failed"

Cause: An internal error has occurred.

Action: Contact Oracle Customer Support Services with the error number.

IMG-00814, "Fail queue free failed"

Cause: An internal error has occurred.

Action: Contact Oracle Customer Support Services with the error number.

IMG-00815, "Merged area queue free failed"

Cause: An internal error has occurred.

Action: Contact Oracle Customer Support Services with the error number.

IMG-00820, "Area 0 queue clear failed"

Cause: An internal error has occurred.

Action: Contact Oracle Customer Support Services with the error number.

IMG-00821, "Area N queue clear failed"

Cause: An internal error has occurred.

Action: Contact Oracle Customer Support Services with the error number.

IMG-00822, "Area queue reset failed"

Cause: An internal error has occurred.

Action: Contact Oracle Customer Support Services with the error number.

IMG-00823, "Boundary queue pop failed"

Cause: An internal error has occurred.

Action: Contact Oracle Customer Support Services with the error number.

IMG-00824, "Fail queue pop failed"

Cause: An internal error has occurred.

Action: Contact Oracle Customer Support Services with the error number.

IMG-00825, "Merged area queue pop failed"

Cause: An internal error has occurred.

Action: Contact Oracle Customer Support Services with the error number.

IMG-00830, "Boundary queue is full"

Cause: An internal error has occurred.

Action: Contact Oracle Customer Support Services with the error number and the image causing this problem.

IMG-00831, "Boundary queue size exceeds expected size"

Cause: An internal error has occurred.

Action: Contact Oracle Customer Support Services with the error number and the image causing this problem.

IMG-00832, "Fail queue is full"

Cause: An internal error has occurred.

Action: Contact Oracle Customer Support Services with the error number and the image causing this problem.

IMG-00833, "Boundary queue size exceeds expected size"

Cause: An internal error has occurred.

Action: Contact Oracle Customer Support Services with the error number and the image causing this problem.

IMG-00834, "Merged area queue is full"

Cause: An internal error has occurred.

Action: Contact Oracle Customer Support Services with the error number and the image causing this problem.

IMG-00835, "Merged area queue size exceeds expected size"

Cause: An internal error has occurred.

Action: Contact Oracle Customer Support Services with the error number and the image causing this problem.

IMG-00836, "Area queue merge failed"

Cause: An internal error has occurred.

Action: Contact Oracle Customer Support Services with the error number and the image causing this problem.

IMG-00840, "Image structure allocation failed"

Cause: Operating system process memory has been exhausted while initializing the image structure.

Action: See the database administrator or operating system administrator to increase process memory quota.

IMG-00841, "Image data allocation failed"

Cause: Operating system process memory has been exhausted while initializing the image data.

Action: See the database administrator or operating system administrator to increase process memory quota.

IMG-00842, "Image index allocation failed"

Cause: Operating system process memory has been exhausted while initializing the image index.

Action: See the database administrator or operating system administrator to increase process memory quota.

IMG-00843, "Internal image structure allocation failed"

Cause: Operating system process memory has been exhausted while initializing the internal image structure.

Action: See the database administrator or operating system administrator to increase process memory quota.

IMG-00844, "Internal image data allocation failed"

Cause: Operating system process memory has been exhausted while initializing the internal image data.

Action: See the database administrator or operating system administrator to increase process memory quota.

IMG-00845, "Internal image index allocation failed"

Cause: Operating system process memory has been exhausted while initializing the internal image index.

Action: See the database administrator or operating system administrator to increase process memory quota.

IMG-00846, "Adjacency matrix allocation failed"

Cause: Operating system process memory has been exhausted while initializing the adjacency matrix.

Action: See the database administrator or operating system administrator to increase process memory quota.

IMG-00847, "Area list allocation failed"

Cause: Operating system process memory has been exhausted while initializing the area list.

Action: See the database administrator or operating system administrator to increase process memory quota.

IMG-00850, "Image structure free failed"

Cause: An internal error has occurred.

Action: Contact Oracle Customer Support Services with the error number.

IMG-00851, "Image data free failed"

Cause: An internal error has occurred.

Action: Contact Oracle Customer Support Services with the error number.

IMG-00852, "Image index free failed"

Cause: An internal error has occurred.

Action: Contact Oracle Customer Support Services with the error number.

IMG-00853, "Internal image structure free failed"

Cause: An internal error has occurred.

Action: Contact Oracle Customer Support Services with the error number.

IMG-00854, "Internal image data free failed"

Cause: An internal error has occurred.

Action: Contact Oracle Customer Support Services with the error number.

IMG-00855, "Internal image index free failed"

Cause: An internal error has occurred.

Action: Contact Oracle Customer Support Services with the error number.

IMG-00856, "Adjacency matrix free failed"

Cause: An internal error has occurred.

Action: Contact Oracle Customer Support Services with the error number.

IMG-00857, "Area list free failed"

Cause: An internal error has occurred.

Action: Contact Oracle Customer Support Services with the error number.

IMG-00860, "Assert failure, number of regions exceeds allocated"

Cause: An internal error has occurred.

Action: Contact Oracle Customer Support Services with the error number and the image causing this problem.

IMG-00861, "Assert failure, inconsistency in area merge operation"

Cause: An internal error has occurred.

Action: Contact Oracle Customer Support Services with the error number and the image causing this problem.

IMG-00862, "Assert failure, inconsistency in merged area labels"

Cause: An internal error has occurred.

Action: Contact Oracle Customer Support Services with the error number and the image causing this problem.

IMG-00870, "Unsupported aspect ratio or image size"

Cause: An internal error has occurred.

Action: Contact Oracle Customer Support Services with the error number and the image causing this problem.

IMG-00871, "Unexpected number of seeds"

Cause: An internal error has occurred.

Action: Contact Oracle Customer Support Services with the error number and the image causing this problem.

IMG-00872, "Unsupported image model"

Cause: An internal error has occurred.

Action: Contact Oracle Customer Support Services with the error number and the image causing this problem.

IMG-00899, "Signature cannot be generated"

Cause: generateSignature could not generate the signature.

Action: Verify that the input image is a format supported by *inter*Media.

H.4 ORDVideo Error Messages

VID-00702 unable to initialize video processing environment

Cause: The initialization of the video processing procedure failed.

Action: See the database administrator to make sure that enough memory has been allocated to JServer. If JServer does have enough memory, contact Oracle Customer Support Services.

VID-00703 unable to read video data

Cause: An error occurred while accessing the video source.

Action: Make sure the video source is valid. For external sources, make sure all access priviliges are granted.

VID-00704 invalid input format

Cause: The video data in the source was not in the format specified by the format field of the video object. In some unusual case, the video data is actually corrupted.

Action: Provide a correct value in the format field. If the correct value is unknown, put NULL in the format field to invoke the DEFAULT format plug-in.

VID-00705 unsupported input format

Cause: The file format of the video data was not supported. This error can only occur in the DEFAULT format plug-in package.

Action: Refer to *Oracle interMedia User's Guide and Reference* for supported formats.

VID-00706 unsupported or corrupted input format

Cause: The video data was either corrupted or the file format was not supported.

Action: Refer to *Oracle interMedia User's Guide and Reference* for supported formats. If the video data is not corrupted and is in a supported file format, contact Oracle Customer Support Services.

VID-00713 internal error while parsing video data

Cause: An internal error occurred during parsing.

Action: Contact Oracle Customer Support Services.

VID-00714 internal error

Cause: An internal error occurred.

Action: Contact Oracle Customer Support Services.

ORDSource Reference Information

Oracle *inter*Media contains the following information about the ORDSource type:

- Object type -- see Section I.1.
- Methods -- see Section I.2.
- Packages or PL/SQL plug-ins -- see Section I.3.

This object is used only by other Oracle *inter*Media objects. The information in this chapter is included for reference only. Oracle Corporation does not recommend that you use this type.

Methods invoked at the ORDSource level that are handed off to the source plug-in for processing have ctx (RAW(4000)) as the first argument. Before calling any of these methods for the first time, the client must allocate the ctx structure, initialize it to NULL, and invoke the open( ) method. At this point, the source plug-in can initialize context for this client. When processing is complete, the client should invoke the close( ) method.

Methods invoked from a source plug-in call have the first argument as obj (ORDSource) and the second argument as ctx (RAW(4000)).

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

The ORDSource object does not attempt to maintain consistency, for example, with local and upDateTime attributes. It is up to you to maintain consistency. ORDAudio, ORDDoc, ORDImage, and ORDVideo objects all maintain consistency of their included ORDSource object.

I.1 Object Types

Oracle *inter*Media provides the ORDSource object type, which supports access to a variety of sources of multimedia data.

ORDSource Object Type

The ORDSource object type supports access to data sources locally in a BLOB within an Oracle database, externally from a BFILE on a local file system, externally from a URL on an HTTP server (within the firewall), or externally from a user-defined source on another server. This object type is defined as follows:

```
CREATE OR REPLACE TYPE ORDsource
AS OBJECT
(
  -- ATTRIBUTES
localData           BLOB,
srcType             VARCHAR2(4000),
srcLocation         VARCHAR2(4000),
srcName             VARCHAR2(4000),
updateTime          DATE,
local               NUMBER,
  -- METHODS
-- Methods associated with the local attribute
MEMBER PROCEDURE setLocal,
MEMBER PROCEDURE clearLocal,
MEMBER FUNCTION isLocal RETURN BOOLEAN,
PRAGMA RESTRICT_REFERENCES(isLocal, WNDS, WNPS, RNDS, RNPS),
-- Methods associated with the updateTime attribute
MEMBER FUNCTION getUpdateTime RETURN DATE,
PRAGMA RESTRICT_REFERENCES(getUpdateTime, WNDS, WNPS, RNDS, RNPS),
MEMBER PROCEDURE setUpdateTime(current_time DATE),
-- Methods associated with the source information
MEMBER PROCEDURE setSourceInformation(
                                      source_type     IN VARCHAR2,
                                      source_location IN VARCHAR2,
                                      source_name     IN VARCHAR2),
MEMBER FUNCTION getSourceInformation RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getSourceInformation, WNDS, WNPS, RNDS, RNPS),

MEMBER FUNCTION getSourceType RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getSourceType, WNDS, WNPS, RNDS, RNPS),

MEMBER FUNCTION getSourceLocation RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getSourceLocation, WNDS, WNPS, RNDS, RNPS),

MEMBER FUNCTION getSourceName RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getSourceName, WNDS, WNPS, RNDS, RNPS),
```

```
MEMBER FUNCTION getBFile RETURN BFILE,
PRAGMA RESTRICT_REFERENCES(getBFile, WNDS, WNPS, RNDS, RNPS),

-- Methods associated with source import/export operations
MEMBER PROCEDURE import(
                      ctx      IN OUT RAW,
                      mimetype OUT VARCHAR2,
                      format   OUT VARCHAR2),
MEMBER PROCEDURE importFrom(
                           ctx             IN OUT RAW,
                           mimetype        OUT VARCHAR2,
                           format          OUT VARCHAR2,
                           source_type     IN VARCHAR2,
                           source_location IN VARCHAR2,
                           source_name     IN VARCHAR2),
MEMBER PROCEDURE export(
                      ctx             IN OUT RAW,
                      source_type     IN VARCHAR2,
                      source_location IN VARCHAR2,
                      source_name     IN VARCHAR2),
-- Methods associated with source content-related operations
MEMBER FUNCTION getContentLength(ctx IN OUT RAW) RETURN INTEGER,
PRAGMA RESTRICT_REFERENCES(getContentLength, WNDS, WNPS, RNDS, RNPS),

MEMBER FUNCTION getSourceAddress(ctx IN OUT RAW,
                                 userData IN VARCHAR2)
               RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getSourceAddress, WNDS, WNPS, RNDS, RNPS),

MEMBER FUNCTION getLocalContent RETURN BLOB,
PRAGMA RESTRICT_REFERENCES(geLocalContent, WNDS, WNPS, RNDS, RNPS),

MEMBER PROCEDURE getContentInTempLob(
                                    ctx      IN OUT RAW,
                                    tempLob  IN OUT NOCOPY BLOB,
                                    mimetype OUT VARCHAR2,
                                    format   OUT VARCHAR2,
                                    duration IN PLS_INTEGER := 10,
                                    cache    IN BOOLEAN := TRUE),
MEMBER PROCEDURE deleteLocalContent,

-- Methods associated with source access methods
MEMBER FUNCTION open(userArg IN RAW, ctx OUT RAW) RETURN INTEGER,
MEMBER FUNCTION close(ctx IN OUT RAW) RETURN INTEGER,
```

```
MEMBER FUNCTION trim(ctx     IN OUT RAW,
                     newlen  IN INTEGER) RETURN INTEGER,

-- Methods associated with content read/write operations
MEMBER PROCEDURE read(
                     ctx      IN OUT RAW,
                     startPos IN INTEGER,
                     numBytes IN OUT INTEGER,
                     buffer   OUT RAW),
MEMBER PROCEDURE write(
                      ctx      IN OUT RAW,
                      startPos IN INTEGER,
                      numBytes IN OUT INTEGER,
                      buffer   IN RAW),
-- Methods associated with any commands to be sent to the external source
MEMBER FUNCTION processCommand(
                              ctx     IN OUT RAW,
                              command IN VARCHAR2,
                              arglist IN VARCHAR2,
                              result  OUT RAW)
               RETURN RAW
);
```

where:

- localData: contains the locally stored multimedia data stored as a BLOB within the object. Up to 4 gigabytes of data can be stored as a BLOB within an Oracle database and is protected by the Oracle security and transaction environment.
- srcType: identifies the data source type. Supported values for srcType are:

srcType	Source Type
"file"	A BFILE on a local file system
"HTTP"	An HTTP server
"<name>"	User-defined

Note: The keyword file for the plug-in is a reserved word for the BFILE source provided by Oracle Corporation. To implement for your own file plug-in, select a different name, for example, MYFILE.

- srcLocation: identifies the place where data can be found based on the srcType value. Valid srcLocation values for corresponding srcType values are:

srcType	Location Value
"file"	<DIR> or name of the directory object
"HTTP"	<SourceBase> or URL needed to find the base directory
"<name>"	<iden> or identifier string required to access a user-defined source

- srcName: identifies the data object name. Valid srcName values for corresponding srcType values are:

srcType	Name Value
"file"	<file> or name of the file
"HTTP"	<Source> or name of the object
"<name>"	<object name> or name of the object

- updateTime: the time at which the data was last updated.
- local: a flag to determine whether or not the data is local:

  1 means the data is in the BLOB.

  0 means the data is in external sources.

  NULL, which may be a default state when you first insert an empty row, is assumed to mean data is local.

I.2 Methods

This section presents ORDSource reference information on the Oracle *inter*Media methods provided for source data manipulation. These methods are described in the following groupings:

ORDSource Methods Associated with the local Attribute

- setLocal: sets the flag value for the local attribute to "1", meaning that the source of the data is local.
- clearLocal: resets the flag value for the local attribute to "0", meaning that the source of the data is external.
- isLocal: returns TRUE to indicate that the source of the data is local or in the BLOB, or FALSE, meaning the data is in an external source. The value of the local attribute is used to determine the return value.

ORDSource Methods Associated with the updateTime Attribute

- getUpdateTime: returns the value of the updateTime attribute.
- setUpdateTime: sets the value of the updateTime attribute to the specified time provided in the argument.

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

- setSourceInformation( ): sets or alters information about the source of the data.
- getSourceInformation: returns a formatted string containing complete information about the data source formatted as a URL.
- getSourceType: returns the external source type of the data.
- getSourceLocation: returns the external source location of the data.
- getSourceName: returns the external source name of the data.
- getBFile: returns the external content as a BFILE, if srcType is of type file.

ORDSource Methods Associated with import and export Operations

- import( ): transfers data from an external data source (specified by calling setSourceInformation( )) to the local source (localData) within an Oracle database.
- importFrom( ): transfers data from the specified external data source (source, location, name) to the local source (localData) within an Oracle database.
- export( ): copies data from a local source (localData) within an Oracle database to the specified external data source.

Note: The export( ) method natively supports only sources of source type file. User-defined sources may support the export( ) method.

ORDSource Methods Associated with the localData Attribute

- getContentLength( ): returns the length of the data source (as number of bytes).
- getSourceAddress( ): returns the address of the data source.
- getLocalContent: returns the handle to the BLOB used to store contents locally.
- getContentInTempLob( ): returns content into a temporary LOB.
- deleteLocalContent: deletes the content of the local BLOB.

ORDSource Methods Associated with Source Input/Output Operations

- open( ): opens a data source.
- close( ): closes a data source.
- trim( ): trims a data source.
- read( ): reads a buffer of n bytes from a source beginning at a start position.
- write( ): writes a buffer of n bytes to a source beginning at a start position.

ORDSource Methods Associated with Processing Commands to the External Source

- processCommand( ): process as any command to the external source. This method is supported only for user-defined sources.

For more information on object types and methods, see *Oracle9i Database Concepts*.

clearLocal

Format

clearLocal;

Description

Resets the flag value from local, meaning the source of the data is stored locally in a BLOB in Oracle, to nonlocal meaning the source of the data is stored externally.

Parameters

None.

Usage Notes

This method sets the local attribute to a 0, meaning the data is stored externally or outside of Oracle.

Pragmas

None.

Exceptions

None.

Examples

None.

close( )

Format

close(ctx IN OUT RAW) RETURN INTEGER;

Description

Closes a data source.

Parameters

ctx
The source plug-in context information.

Usage Notes

Calling this method uses the ORDPLUGINS.ORDX_<srcType>_SOURCE plug-in package.

The return INTEGER is 0 (zero) for success and >0 (for example, 1) for failure. The exact number and the meaning for that number is plug-in defined. For example, for the file plug-in, 1 might mean "File not found," 2 might mean "No such directory," and so forth.

Pragmas

None.

Exceptions

INCOMPLETE_SOURCE_INFORMATION

This exception is raised if you call the close( ) method and the value for srcType is NULL and data is not local.

METHOD_NOT_SUPPORTED

This exception is raised if you call the close( ) method and this method is not supported by the source plug-in being used.

SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the close( ) method within a source plug-in when any other exception is raised.

Examples

None.

deleteLocalContent

Format

deleteLocalContent;

Description

Deletes the local data from the current local source (localData).

Parameters

None.

Usage Notes

This method can be called after you export the data from the local source to an external data source and you no longer need this data in the local source.

Pragmas

None.

Exceptions

None.

Examples

None.

export( )

Format

```
export(
    ctx                 IN OUT RAW,
    source_type         IN VARCHAR2,
    source_location IN VARCHAR2,
    source_name         IN VARCHAR2);
```

Description

Copies data from a local source (localData) within an Oracle database to an external data source.

> **Note:** The export( ) method natively supports only sources of source type file. User-defined sources may support the export( ) method.

Parameters

ctx
The source plug-in context information.

source_type
The source type of the location to where data is to be exported.

source_location
The location where the data is to be exported.

source_name
The name of the object to where the data is to be exported.

Usage Notes

This method exports data out of the localData to another source.

Calling this method uses the ORDPLUGINS.ORDX_<srcType>_SOURCE plug-in package.

After exporting data, the srcType, srcLocation, and srcName attributes are updated with input parameter values. After calling the export( ) method, call the clearLocal( ) method to indicate the data is stored outside the database and call the deleteLocalContent method if you want to delete the content of the local data.

This method is also available for user-defined sources that can support the export method.

The only server-side native support for the export method is for the srcType file.

The export( ) method for a source type of file is similar to a file copy operation in that the original data stored in the BLOB is not touched other than for reading purposes.

The export( ) method is not an exact mirror operation to the import( ) method in that the clearLocal( ) method is not automatically called to indicate the data is stored outside the database, whereas the import( ) method automatically calls the setLocal( ) method.

Call the deleteLocalContent method after calling the export( ) method to delete the content from the database if you no longer intend to manage the multimedia data within the database.

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

For example, the following grants the user, MEDIAUSER, the permission to write to the file named filename.dat:

```
CALL DBMS_JAVA.GRANT_PERMISSION(
        'MEDIAUSER',
        'java.io.FilePermission',
        '/actual/server/directory/path/filename.dat',
        'write');
```

See the security and performance section in *Oracle9i Java Developer's Guide* for more information.

Invoking this method implicitly calls the setUpdateTime( ) method.

Pragmas

None.

Exceptions

INCOMPLETE_SOURCE_INFORMATION

This exception is raised if you call the export( ) method and the value of srcType is NULL.

METHOD_NOT_SUPPORTED

This exception is raised if you call the export( ) method and this method is not supported by the source plug-in being used.

SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the export( ) method within a source plug-in when any other exception is raised.

Examples

None.

getBFile

Format

getBFile RETURN BFILE;

Description

Returns a BFILE handle, if the srcType is file.

Parameters

None.

Usage Notes

This method can only be used for a srcType of file or BFILE sources.

Pragmas

PRAGMA RESTRICT_REFERENCES(getBFile, WNDS, WNPS, RNDS, RNPS)

Exceptions

INCOMPLETE_SOURCE_INFORMATION

This exception is raised if you call the getBFILE method and the value of srcType is NULL.

INVALID_SOURCE_TYPE

This exception is raised if you call the getBFile method and the value of srcType is other than file.

Examples

None.

getContentInTempLob( )

Format

```
getContentInTempLob(
                    ctx           IN OUT RAW,
                    tempLob   IN OUT NOCOPY BLOB,
                    mimetype OUT VARCHAR2,
                    format      OUT VARCHAR2,
                    duration    IN PLS_INTEGER := 10,
                    cache       IN BOOLEAN := TRUE);
```

Description

Transfers data from the current data source into a temporary LOB, which will be allocated and initialized as a part of this call.

Parameters

ctx
The source plug-in context information.

tempLob
Uninitialized BLOB locator, which will be allocated in this call.

mimetype
Out parameter to receive the MIME type of the data, for example, 'audio/basic'.

format
Out parameter to receive the format of the data, for example, 'AUFF'.

duration
The life of the temporary LOB to be allocated. The life of the temporary LOB can be for the duration of the call, the transaction, or for the session. The default is DBMS_LOB.SESSION. Valid values for each duration state are as follows:

DBMS_LOB.CALL

DBMS_LOB.TRANSACTION

DBMS_LOB.SESSION

cache
Whether or not you want to keep the data cached. The value is either TRUE or FALSE. The default is TRUE.

Usage Notes

None.

Pragmas

None.

Exceptions

NO_DATA_FOUND

This exception is raised if you call the getContentInLob( ) method when working with temporary LOBs for looping read operations that reach the end of the LOB, and there are no more bytes to be read from the LOB.

SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the getContentInLob( ) method within a source plug-in when any other exception is raised.

Examples

None.

getContentLength( )

Format

getContentLength(ctx IN OUT RAW) RETURN INTEGER;

Description

Returns the length of the data content stored in the source. For a file source and for data in a local BLOB data source, the length is returned as a number of bytes. The unit type of the returned value is defined by the plug-in that implements this method.

Parameters

ctx
The source plug-in context information.

Usage Notes

This method is not supported for all source types. For example, HTTP type sources do not support this method. If you want to implement this call for HTTP type sources, you must define your own modified HTTP source plug-in and implement this method.

Calling this method uses the ORDPLUGINS.ORDX_<srcType>_SOURCE plug-in package.

Pragmas

PRAGMA RESTRICT_REFERENCES(getContentLength, WNDS, WNPS, RNDS, RNPS)

Exceptions

INCOMPLETE_SOURCE_INFORMATION

This exception is raised if you call the getContentLength( ) method and the value of srcType is NULL and data is not stored locally in the BLOB.

SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the getContentLength( ) method within a source plug-in when any other exception is raised.

Examples

None.

getLocalContent

Format

getLocalContent RETURN BLOB;

Description

Returns the content or BLOB handle of the local data.

Parameters

None.

Usage Notes

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getLocalContent, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Examples

None.

getSourceAddress( )

Format

```
getSourceAddress(ctx IN OUT RAW,
                 userData IN VARCHAR2)  RETURN VARCHAR2;
```

Description

Returns the source address for data located in an external data source. This method is only implemented for user-defined sources.

Parameters

ctx
The source plug-in context information.

userData
Information input by the user needed by some sources to obtain the desired source address.

Usage Notes

Use this method to return the address of an external data source when the source needs to format this information in some unique way. For example, call the getSourceAddress( )method to obtain the address for RealNetworks server sources or URLs containing data sources located on Oracle Application Server.

Calling this method uses the ORDPLUGINS.ORDX_<srcType>_SOURCE plug-in package.

Pragmas

PRAGMA RESTRICT_REFERENCES(getSourceAddress, WNDS, WNPS, RNDS, RNPS)

Exceptions

INCOMPLETE_SOURCE_INFORMATION

This exception is raised if you call the getSourceAddress( ) method and the value of srcType is NULL.

SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the getSource Address( ) method within a source plug-in when any other exception is raised.

Examples

None.

getSourceInformation

Format

getSourceInformation RETURN VARCHAR2;

Description

Returns a URL formatted string containing complete information about the external data source.

Parameters

None.

Usage Notes

This method returns a VARCHAR2 string formatted as: <srcType>://<srcLocation>/<srcName>, where srcType, srcLocation, and srcName are the ORDSource attribute values.

Pragmas

PRAGMA RESTRICT_REFERENCES(getSourceInformation, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Examples

None.

getSourceLocation

Format

getSourceLocation RETURN VARCHAR2;

Description

Returns the external data source location.

Parameters

None.

Usage Notes

This method returns the current value of the srcLocation attribute, for example BFILEDIR.

Pragmas

PRAGMA RESTRICT_REFERENCES(getSourceLocation, WNDS, WNPS, RNDS, RNPS)

Exceptions

INCOMPLETE_SOURCE_LOCATION

This exception is raised if you call the setSourceLocation( ) method and the value of srcLocation is NULL.

Examples

None.

getSourceName

Format

getSourceName RETURN VARCHAR2;

Description

Returns the external data source name.

Parameters

None.

Usage Notes

This method returns the current value of the srcName attribute, for example testaud.dat.

Pragmas

PRAGMA RESTRICT_REFERENCES(getSourceName, WNDS, WNPS, RNDS, RNPS)

Exceptions

INCOMPLETE_SOURCE_NAME

This exception is raised if you call the setSourceName( ) method and the value of srcName is NULL.

Examples

None.

getSourceType

Format

getSourceType RETURN VARCHAR2;

Description

Returns the external data source type.

Parameters

None.

Usage Notes

This method returns the current value of the srcType attribute, for example file.

Pragmas

PRAGMA RESTRICT_REFERENCES(getSourceType, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Examples

None.

getUpdateTime

Format

```
getUpdateTime RETURN DATE;
```

Description

Returns the value of the updateTime attribute for the ORDSource object. This is the timestamp when the object was last changed, or what the user explicitly set by calling the setUpdateTime( ) method.

Parameters

None.

Usage Notes

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getUpdateTime, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Examples

None.

import( )

Format

```
import(
    ctx          IN OUT RAW,
    mimetype OUT VARCHAR2,
    format       OUT VARCHAR2);
```

Description

Transfers data from an external data source (specified by first calling setSourceInformation( )) to a local source within an Oracle database.

Parameters

ctx
The source plug-in context information.This information is passed along uninterpreted to the source plug-in handling the import( ) call.

mimetype
Out parameter to receive the MIME type of the data, if any, for example, 'audio/basic'.

format
Out parameter to receive the format of the data, if any, for example, 'AUFF'.

Usage Notes

Call setSourceInformation( ) to set the srcType, srcLocation, and srcName attribute information to describe where the data source is located prior to calling the import( ) method.

You must ensure that the directory exists or is created before you use this method.

Calling this method uses the ORDPLUGINS.ORDX_<srcType>_SOURCE plug-in package.

Pragmas

None.

Exceptions

INCOMPLETE_SOURCE_INFORMATION

This exception is raised if you call the import( ) method and the value of srcType is NULL.

NULL_SOURCE

This exception is raised if you call the import( ) method and the value of dlob is NULL.

METHOD_NOT_SUPPORTED

This exception is raised if you call the import( ) method and this method is not supported by the source plug-in being used.

SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the import( ) method within a source plug-in when any other exception is raised, raises a exception.

Examples

None.

importFrom( )

Format

```
importFrom(
        ctx              IN OUT RAW,
        mimetype         OUT VARCHAR2,
        format           OUT VARCHAR2
        source_type      IN VARCHAR2,
        source_location IN VARCHAR2,
        source_name      IN VARCHAR2);
```

Description

Transfers data from the specified external data source (type, location, name) to a local source within an Oracle database, and resets the source attributes and the timestamp.

Parameters

ctx
The source plug-in context information.This information is passed along uninterpreted to the source plug-in handling the importFrom( ) call.

mimetype
Out parameter to receive the MIME type of the data, if any, for example, 'audio/basic'.

format
Out parameter to receive the format of the data, if any, for example, 'AUFF'.

source_type
Source type from where the data is to be imported. This also sets the srcType attribute.

source_location
Source location from where the data is to be imported. This also sets the srcLocation attribute.

source_name
Name of the source to be imported. This also sets the srcName attribute.

Usage Notes

This method describes where the data source is located by specifying values for the type, location, and name parameters, which set the srcType, srcLocation, and srcName attribute values, respectively, after the importFrom operation succeeds.

You must ensure that the directory exists or is created before you use this method.

This method is a combination of a setSourceInformation( ) call followed by an import( ) call.

Calling this method uses the ORDPLUGINS.ORDX_<srcType>_SOURCE plug-in package.

Pragmas

None.

Exceptions

NULL_SOURCE

This exception is raised if you call the importFrom( ) method and the value of dlob is NULL.

METHOD_NOT_SUPPORTED

This exception is raised if you call the importFrom( ) method and this method is not supported by the source plug-in being used.

SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the importFrom( ) method within a source plug-in when any other exception is raised.

Examples

None.

isLocal

Format

isLocal RETURN BOOLEAN;

Description

Returns TRUE if the data is stored locally in a BLOB in Oracle9*i* or FALSE if the data is stored externally.

Parameters

None.

Usage Notes

If the local attribute is set to1 or NULL, this method returns TRUE, otherwise this method returns FALSE.

Pragmas

PRAGMA RESTRICT_REFERENCES(isLocal, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Examples

None.

open( )

Format

open(userArg IN RAW, ctx OUT RAW) RETURN INTEGER;

Description

Opens a data source. It is recommended that this method be called before invoking any other methods that accept the ctx parameter.

Parameters

userArg
The user argument.

ctx
The source plug-in context information.

Usage Notes

Calling this method uses the ORDPLUGINS.ORDX_<srcType>_SOURCE plug-in package.

The return INTEGER is 0 (zero) for success and >0 (for example, 1) for failure. The exact number and the meaning for that number is plug-in defined. For example, for the file plug-in, 1 might mean "File not found," 2 might mean "No such directory," and so forth.

Pragmas

None.

Exceptions

INCOMPLETE_SOURCE_INFORMATION

This exception is raised if you call the open( ) method and the value for srcType is NULL and data is not local.

METHOD_NOT_SUPPORTED

This exception is raised if you call the open( ) method and this method is not supported by the source plug-in being used.

SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the open( ) method within a source plug-in when any other exception is raised.

Examples

None.

processCommand( )

Format

```
processCommand(
                    ctx          IN OUT RAW,
                    command IN VARCHAR2,
                    arglist      IN VARCHAR2,
                    result       OUT RAW)
RETURN RAW;
```

Description

Allows you to send commands and related arguments to the source plug-in. This method is supported only for user-defined sources.

Parameters

ctx
The source plug-in context information.

command
Any command recognized by the source plug-in.

arglist
The arguments for the command.

result
The result of calling this method returned by the plug-in.

Usage Notes

Use this method to send any commands and their respective arguments to the plug-in. Commands are not interpreted; they are taken and passed through to be processed.

Calling this method uses the ORDPLUGINS.ORDX_<srcType>_SOURCE plug-in package.

Pragmas

None.

Exceptions

INCOMPLETE_SOURCE_INFORMATION

This exception is raised if you call the processCommand( ) method and the value of srcType is NULL.

METHOD_NOT_SUPPORTED

This exception is raised if you call the processCommand( ) method and this method is not supported by the source plug-in being used.

SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the processCommand( ) method within a source plug-in when any other exception is raised.

Examples

None.

read( )

Format

```
read(
    ctx          IN OUT RAW,
    startPos   IN INTEGER,
    numBytes IN OUT INTEGER,
    buffer       OUT RAW);
```

Description

Allows you to read a buffer of numBytes from a source beginning at a start position (startPos).

Parameters

ctx
The source plug-in context information.

startPos
The start position in the data source.

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

buffer
The buffer to where the data will be read.

Usage Notes

This method is not supported for HTTP sources.

To successfully read HTTP source types, the entire URL source must be requested to be read. If you want to implement a read method for an HTTP source type, you must provide your own implementation for this method in the modified source plug-in for the HTTP source type.

Calling this method uses the ORDPLUGINS.ORDX_<srcType>_SOURCE plug-in package.

Pragmas

None.

Exceptions

NULL_SOURCE

This exception is raised if you call the read( ) method and the data is stored locally and localData is NULL.

INCOMPLETE_SOURCE_INFORMATION

This exception is raised if you call the read( ) method and the value of srcType is NULL and data is not local.

METHOD_NOT_SUPPORTED

This exception is raised if you call the read( ) method and this method is not supported by the source plug-in being used.

SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the read( ) method within a source plug-in when any other exception is raised.

Examples

None.

setLocal

Format

setLocal;

Description

Sets the local attribute to indicate that the data is stored in a BLOB within Oracle9*i*.

Parameters

None.

Usage Notes

This method sets the local attribute to 1, meaning the data is stored locally in the localData attribute.

Pragmas

None.

Exceptions

None.

Examples

None.

setSourceInformation( )

Format

```
setSourceInformation(
                    source_type      IN VARCHAR2,
                    source_location IN VARCHAR2,
                    source_name      IN VARCHAR2);
```

Description

Sets the provided subcomponent information for the srcType, srcLocation, and srcName that describes the external data source.

Parameters

source_type
The source type of the external data. See the "ORDSource Object Type" definition in this chapter for more information.

source_location
The source location of the external data. See the "ORDSource Object Type" definition in this chapter for more information.

source_name
The source name of the external data. See the "ORDSource Object Type" definition in this chapter for more information.

Usage Notes

Before you call the import( ) method, you must call the setSourceInformation( ) method to set the srcType, srcLocation, and srcName attribute information to describe where the data source is located. If you call the importFrom( ) or the export( ) method, then these attributes are set after the importFrom( ) or export( ) call succeeds.

You must ensure that the directory exists or is created before you use this method.

Pragmas

None.

Exceptions

INCOMPLETE_SOURCE_INFORMATION

This exception is raised if you call the setSourceInformation( ) method and the value for source_type is NULL.

Examples

None.

setUpdateTime( )

Format

setUpdateTime(current_time DATE);

Description

Sets the value of the updateTime attribute to the time you specify.

Parameters

current_time
The update time.

Usage Notes

If current_time is NULL, updateTime is set to SYSDATE (the current time).

Pragmas

None.

Exceptions

None.

Examples

None.

trim( )

Format

```
trim(ctx IN OUT RAW,
     newlen IN INTEGER) RETURN INTEGER;
```

Description

Trims a data source.

Parameters

ctx
The source plug-in context information.

newlen
The trimmed new length.

Usage Notes

Calling this method uses the ORDPLUGINS.ORDX_<srcType>_SOURCE plug-in package.

The return INTEGER is 0 (zero) for success and >0 (for example, 1) for failure. The exact number and the meaning for that number is plug-in defined. For example, for the file plug-in, 1 might mean "File not found," 2 might mean "No such directory," and so forth.

Pragmas

None.

Exceptions

INCOMPLETE_SOURCE_INFORMATION

This exception is raised if you call the trim( ) method and the value for srcType is NULL and data is not local.

METHOD_NOT_SUPPORTED

This exception is raised if you call the trim( ) method and this method is not supported by the source plug-in being used.

SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the trim( ) method within a source plug-in when any other exception is raised.

Examples

None.

write( )

Format

```
write(
    ctx          IN OUT RAW,
    startPos   IN INTEGER,
    numBytes IN OUT INTEGER,
    buffer       IN RAW);
```

Description

Allows you to write a buffer of numBytes to a source beginning at a start position (startPos).

Parameters

ctx
The source plug-in context information.

startPos
The start position in the source to where the buffer should be copied.

numBytes
The number of bytes to be written to the source.

buffer
The buffer of data to be written.

Usage Notes

This method assumes that the writable source allows you to write numBytes at a random byte location. For example, the file and HTTP source types are not writable sources and do not support this method.

Calling this method uses the ORDPLUGINS.ORDX_<srcType>_SOURCE plug-in package.

Pragmas

None.

Exceptions

NULL_SOURCE

This exception is raised if you call the write( ) method and local is 1 or NULL and localData is NULL.

INCOMPLETE_SOURCE_INFORMATION

This exception is raised if you call the write( ) method and the value of srcType is NULL and data is not local.

METHOD_NOT_SUPPORTED

This exception is raised if you call the write( ) method and this method is not supported by the source plug-in being used.

SOURCE_PLUGIN_EXCEPTION

This exception is raised if you call the write( ) method within a source plug-in when any other exception is raised.

Examples

None.

I.3 Packages or PL/SQL Plug-ins

This section presents reference information on the packages or PL/SQL plug-ins provided.

Any method invoked from a source plug-in call has the first argument as obj (ORDSource) and the second argument as ctx (RAW).

Plug-ins must be named as ORDX_<name>_<module_name> where the <module_name> is SOURCE for ORDSource. For example, the file plug-in described in Section I.3.1, is named ORDX_FILE_SOURCE and <name> is the source type.

Exceptions must be raised from and recorded in a package named as ORD_<module_name>Exceptions. For example, ORDSource exceptions are raised and recorded in a package named ORDSourceExceptions (see Appendix H).

I.3.1 ORDPLUGINS.ORDX_FILE_SOURCE Package

The ORDPLUGINS.ORDX_FILE_SOURCE package or PL/SQL plug-in is provided.

```
CREATE OR REPLACE PACKAGE ORDX_FILE_SOURCE AS
  -- functions/procedures
  FUNCTION processCommand(obj     IN OUT NOCOPY ORDSYS.ORDSource,
                          ctx     IN OUT RAW,
                          cmd     IN VARCHAR2,
                          arglist IN VARCHAR2,
                          result  OUT RAW)
           RETURN RAW;
  PROCEDURE import(obj      IN OUT NOCOPY ORDSYS.ORDSource,
                   ctx      IN OUT RAW,
                   mimetype OUT VARCHAR2,
                   format   OUT VARCHAR2);
  PROCEDURE import(obj      IN OUT NOCOPY ORDSYS.ORDSource,
                   ctx      IN OUT RAW,
                   dlob     IN OUT NOCOPY BLOB,
                   mimetype OUT VARCHAR2,
                   format   OUT VARCHAR2);
  PROCEDURE importFrom(obj      IN OUT NOCOPY ORDSYS.ORDSource,
                       ctx      IN OUT RAW,
                       mimetype OUT VARCHAR2,
                       format   OUT VARCHAR2,
                       loc      IN VARCHAR2,
                       name     IN VARCHAR2);
  PROCEDURE importFrom(obj      IN OUT NOCOPY ORDSYS.ORDSource,
                       ctx      IN OUT RAW,
                       dlob     IN OUT NOCOPY BLOB,
                       mimetype OUT VARCHAR2,
                       format   OUT VARCHAR2,
                       loc      IN VARCHAR2,
                       name     IN VARCHAR2);
  PROCEDURE export(obj  IN OUT NOCOPY ORDSYS.ORDSource,
                   ctx  IN OUT RAW,
                   slob IN OUT NOCOPY BLOB,
                   loc  IN VARCHAR2,
                   name IN VARCHAR2);
  FUNCTION  getContentLength(obj  IN ORDSYS.ORDSource,
                             ctx  IN OUT RAW),
            RETURN INTEGER;
  PRAGMA RESTRICT_REFERENCES(getContentLength, WNDS, WNPS, RNDS, RNPS);
  FUNCTION  getSourceAddress(obj  IN ORDSYS.ORDSource,
                             ctx  IN OUT RAW,
                             userData IN VARCHAR2)
```

```
            RETURN VARCHAR2;
  PRAGMA RESTRICT_REFERENCES(getSourceAddress, WNDS, WNPS, RNDS, RNPS);

  FUNCTION open(obj IN OUT NOCOPY ORDSYS.ORDSource,
           userArg IN RAW,
           ctx OUT RAW) RETURN INTEGER;
  FUNCTION close(obj IN OUT NOCOPY ORDSYS.ORDSource, ctx IN OUT RAW)
         RETURN INTEGER;
  FUNCTION trim(obj IN OUT NOCOPY ORDSYS.ORDSource,
                ctx IN OUT RAW,
                newlen IN INTEGER) RETURN INTEGER;
PROCEDURE read(obj     IN OUT NOCOPY ORDSYS.ORDSource,
               ctx      IN OUT RAW,
               startPos IN INTEGER,
               numBytes IN OUT INTEGER,
               buffer   OUT RAW);
PROCEDURE write(obj      IN OUT NOCOPY ORDSYS.ORDSource,
                ctx      IN OUT RAW,
                startPos IN INTEGER,
                numBytes IN OUT INTEGER,
                buffer   OUT RAW);
END ORDX_FILE_SOURCE;
/
```

Table I–1 shows the methods supported in the ORDX_FILE_SOURCE package and the exceptions raised if you call a method that is not supported.

Table I–1 Methods Supported in the ORDPLUGINS.ORDX_FILE_SOURCE Package

Name of Method	Level of Support
processCommand	Not supported - raises exception: METHOD_NOT_SUPPORTED
import	Supported
import	Supported
importFrom	Supported
importFrom	Supported
export	Supported
getContentLength	Supported
getSourceAddress	Supported
open	Supported

Table I–1 Methods Supported in the ORDPLUGINS.ORDX_FILE_SOURCE Package (Cont.)

Name of Method	Level of Support
close	Supported
trim	Not supported - raises exception: METHOD_NOT_SUPPORTED
read	Supported
write	Not supported - raises exception: METHOD_NOT_SUPPORTED

I.3.2 ORDPLUGINS.ORDX_HTTP_SOURCE Package

The ORDPLUGINS.ORDX_HTTP_SOURCE package or PL/SQL plug-in is provided.

```
CREATE OR REPLACE PACKAGE ORDX_HTTP_SOURCE AS
  -- functions/procedures
  FUNCTION processCommand(obj     IN OUT NOCOPY ORDSYS.ORDSource,
                          ctx     IN OUT RAW,
                          cmd     IN VARCHAR2,
                          arglist IN VARCHAR2,
                          result  OUT RAW)
           RETURN RAW;
  PROCEDURE import(obj      IN OUT NOCOPY ORDSYS.ORDSource,
                   ctx      IN OUT RAW,
                   mimetype OUT VARCHAR2,
                   format   OUT VARCHAR2);
  PROCEDURE import(obj      IN OUT NOCOPY ORDSYS.ORDSource,
                   ctx      IN OUT RAW,
                   dlob     IN OUT NOCOPY BLOB,
                   mimetype OUT VARCHAR2,
                   format   OUT VARCHAR2);
  PROCEDURE importFrom(obj      IN OUT NOCOPY ORDSYS.ORDSource,
                       ctx      IN OUT RAW,
                       mimetype OUT VARCHAR2,
                       format   OUT VARCHAR2,
                       loc      IN VARCHAR2,
                       name     IN VARCHAR2);
  PROCEDURE importFrom(obj      IN OUT NOCOPY ORDSYS.ORDSource,
                       ctx      IN OUT RAW,
                       dlob     IN OUT NOCOPY BLOB,
                       mimetype OUT VARCHAR2,
                       format   OUT VARCHAR2,
                       loc      IN VARCHAR2,
                       name     IN VARCHAR2);
```

```
  PROCEDURE export(obj  IN OUT NOCOPY ORDSYS.ORDSource,
                   ctx  IN OUT RAW,
                   dlob IN OUT NOCOPY BLOB,
                   loc  IN VARCHAR2,
                   name IN VARCHAR2);
  FUNCTION  getContentLength(obj  IN ORDSYS.ORDSource,
                             ctx  IN OUT RAW)
            RETURN INTEGER;
  PRAGMA RESTRICT_REFERENCES(getContentLength, WNDS, WNPS, RNDS, RNPS);
  FUNCTION  getSourceAddress(obj  IN ORDSYS.ORDSource,
                             ctx  IN OUT RAW,
                             userData IN VARCHAR2)
            RETURN VARCHAR2;
  PRAGMA RESTRICT_REFERENCES(getSourceAddress, WNDS, WNPS, RNDS, RNPS);
  FUNCTION open(obj IN OUT NOCOPY ORDSYS.ORDSource, userArg IN RAW,
           ctx OUT RAW) RETURN INTEGER;
  FUNCTION close(obj IN OUT NOCOPY ORDSYS.ORDSource, ctx IN OUT RAW)
           RETURN INTEGER;
  FUNCTION trim(obj IN OUT NOCOPY ORDSYS.ORDSource,
           ctx IN OUT RAW,
           newlen IN INTEGER) RETURN INTEGER;
  PROCEDURE read(obj      IN OUT NOCOPY ORDSYS.ORDSource,
                 ctx      IN OUT RAW,
                 startPos IN INTEGER,
                 numBytes IN OUT INTEGER,
                 buffer   OUT RAW);
  PROCEDURE write(obj      IN OUT NOCOPY ORDSYS.ORDSource,
                  ctx      IN OUT RAW,
                  startPos IN INTEGER,
                  numBytes IN OUT INTEGER,
                  buffer   OUT RAW);
END ORDX_HTTP_SOURCE;
/
```

Table I–2 shows the methods supported in the ORDX_HTTP_SOURCE package and the exceptions raised if you call a method that is not supported.

Table I–2 Methods Supported in the ORDPLUGINS.ORDX_HTTP_SOURCE Package

Name of Method	Level of Support
processCommand	Not supported - raises exception: METHOD_NOT_SUPPORTED
import	Supported

Table I–2 Methods Supported in the ORDPLUGINS.ORDX_HTTP_SOURCE Package (Cont.)

Name of Method	Level of Support
import	Supported
importFrom	Supported
importFrom	Supported
export	Not supported - raises exception: METHOD_NOT_SUPPORTED
getContentLength	Supported
getSourceAddress	Supported
open	Supported
close	Supported
trim	Not supported - raises exception: METHOD_NOT_SUPPORTED
read	Not supported - raises exception: METHOD_NOT_SUPPORTED
write	Not supported - raises exception: METHOD_NOT_SUPPORTED

I.3.3 ORDPLUGINS.ORDX_<srcType>_SOURCE Package

Use the ORDPLUGINS.ORDX_<srcType>_SOURCE package or PL/SQL plug-in as a template to create your own source type. Use the ORDPLUGINS.ORDX_FILE_SOURCE and ORDPLUGINS.ORDX_HTTP_SOURCE packages as a guide in developing your new source type package.

I.3.4 Extending *inter*Media to Support a New Data Source

Extending *inter*Media to support a new data source consists of four steps:

1. Design your new data source.
2. Implement your new data source and name it, for example, ORDX_MY_SOURCE.SQL.
3. Install your new ORDX_MY_SOURCE.SQL plug-in in the ORDPLUGINS schema.
4. Grant EXECUTE privileges on your new plug-in, for example, ORDX_MY_SOURCE.SQL plug-in to PUBLIC.

Section 3.5 briefly describes how to extend *inter*Media to support a new data source for audio and video data and describe the interfaces. A package body listing is provided in Example I–1 to assist you in this operation. Add your variables to the

places that say "--Your variables go here" and add your code to the places that say "--Your code goes here".

Example I–1 Show the Package Body for Extending Support to a New Data Source

```
CREATE OR REPLACE PACKAGE BODY ORDX_MY_SOURCE
AS
  -- functions/procedures
  FUNCTION processCommand(
                   obj  IN OUT NOCOPY ORDSYS.ORDSource,
                   ctx  IN OUT RAW,
                   cmd  IN VARCHAR2,
                   arglist IN VARCHAR2,
                   result OUT RAW)
  RETURN RAW
  IS
  --Your variables go here
  BEGIN
  --Your code goes here
  END processCommand;
  PROCEDURE import( obj  IN OUT NOCOPY ORDSYS.ORDSource,
                   ctx  IN OUT RAW,
                   mimetype OUT VARCHAR2,
                   format   OUT VARCHAR2)
  IS
  --Your variables go here
  BEGIN
  --Your code goes here
  END import;
  PROCEDURE import( obj  IN OUT NOCOPY ORDSYS.ORDSource,
                   ctx  IN OUT RAW,
                   dlob IN OUT NOCOPY BLOB,
                   mimetype OUT VARCHAR2,
                   format   OUT VARCHAR2)
  IS
  --Your variables go here
  BEGIN
  --Your code goes here
  END import;
  PROCEDURE importFrom( obj      IN OUT NOCOPY ORDSYS.ORDSource,
                        ctx      IN OUT RAW,
                        mimetype OUT VARCHAR2,
                        format   OUT VARCHAR2,
                        loc      IN VARCHAR2,
                        name     IN VARCHAR2)
```

```
IS
--Your variables go here
BEGIN
--Your code goes here
END importFrom;
PROCEDURE importFrom( obj      IN OUT NOCOPY ORDSYS.ORDSource,
                      ctx      IN OUT RAW,
                      dlob     IN OUT NOCOPY BLOB,
                      mimetype OUT VARCHAR2,
                      format   OUT VARCHAR2,
                      loc      IN VARCHAR2,
                      name     IN VARCHAR2)
IS
--Your variables go here
BEGIN
--Your code goes here
END importFrom;
PROCEDURE export( obj  IN OUT NOCOPY ORDSYS.ORDSource,
                  ctx  IN OUT RAW,
                  dlob IN OUT NOCOPY BLOB,
                  loc  IN VARCHAR2,
                  name IN VARCHAR2)
IS
--Your variables go here
BEGIN
--Your code goes here
END export;

FUNCTION  getContentLength( obj  IN ORDSYS.ORDSource,
                            ctx  IN OUT RAW)
RETURN INTEGER
IS
--Your variables go here
BEGIN
--Your code goes here
END getContentLength;
FUNCTION  getSourceAddress(obj  IN ORDSYS.ORDSource,
                           ctx  IN OUT RAW,
                           userData IN VARCHAR2)
RETURN VARCHAR2
IS
--Your variables go here
BEGIN
--Your code goes here
END getSourceAddress;
```

```
  FUNCTION open(obj IN OUT NOCOPY ORDSYS.ORDSource, userArg IN RAW, ctx OUT RAW)
  RETURN INTEGER
  IS
  --Your variables go here
  BEGIN
  --Your code goes here
  END open;
  FUNCTION close(obj IN OUT NOCOPY ORDSYS.ORDSource, ctx IN OUT RAW)
  RETURN INTEGER
  IS
  --Your variables go here
  BEGIN
  --Your code goes here
  END close;
  FUNCTION trim(obj    IN OUT NOCOPY ORDSYS.ORDSource,
                       ctx    IN OUT RAW,
                       newlen IN INTEGER)
  RETURN INTEGER
  IS
  --Your variables go here
  BEGIN
  --Your code goes here
  END trim;
  PROCEDURE read(obj      IN OUT NOCOPY ORDSYS.ORDSource,
                 ctx      IN OUT RAW,
                 startPos IN INTEGER,
                 numBytes IN OUT INTEGER,
                 buffer   OUT RAW)
  IS
  --Your variables go here
  BEGIN
  --Your code goes here
  END read;
  PROCEDURE write(obj      IN OUT NOCOPY ORDSYS.ORDSource,
                  ctx      IN OUT RAW,
                  startPos IN INTEGER,
                  numBytes IN OUT INTEGER,
                  buffer   OUT RAW)
  IS
  --Your variables go here
  BEGIN
  --Your code goes here
  END write;
END ORDX_MY_SOURCE;
/
```

```
show errors;
```

J

Deprecated Methods

J.1 Deprecated Audio and Video Methods

The following ORDAudio and ORDVideo *get* methods that accept a ctx parameter were deprecated in release 8.1.6:

ORDAudio

```
getFormat(ctx IN OUT RAW) RETURN VARCHAR2
getEncoding(ctx IN OUT RAW) RETURN VARCHAR2
getNumberOfChannels(ctx IN OUT RAW) RETURN INTEGER
getSamplingRate(ctx IN OUT RAW) RETURN INTEGER
getSampleSize(ctx IN OUT RAW) RETURN INTEGER
getCompressionType(ctx IN OUT RAW) RETURN VARCHAR2
getAudioDuration(ctx IN OUT RAW) RETURN INTEGER
```

ORDVideo

```
getFormat(ctx IN OUT RAW) RETURN VARCHAR2
getFrameSize(SELF IN OUT NOCOPY ORDVideo,
                            ctx IN OUT RAW,
                            retWidth OUT INTEGER,
                            retHeight OUT INTEGER)
getFrameResolution(ctx IN OUT RAW) RETURN INTEGER
getFrameRate(ctx IN OUT RAW) RETURN INTEGER
getVideoDuration(ctx IN OUT RAW) RETURN INTEGER
getNumberOfFrames(ctx IN OUT RAW) RETURN INTEGER
getCompressionType(ctx IN OUT RAW) RETURN VARCHAR2
getNumberOfColors(ctx IN OUT RAW) RETURN INTEGER
getBitRate(ctx IN OUT RAW) RETURN INTEGER
```

The following ORDAudio and ORDVideo comments methods were deprecated in release 9.0.1:

ORDAudio

```
-- Methods associated with the comments attribute
MEMBER PROCEDURE appendToComments(amount IN BINARY_INTEGER,
                                  buffer IN VARCHAR2),
MEMBER PROCEDURE writeToComments(offset IN INTEGER,
                                 amount IN BINARY_INTEGER,
                                 buffer IN VARCHAR2),
MEMBER FUNCTION readFromComments(offset IN INTEGER,
                                 amount IN BINARY_INTEGER := 32767)
                RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(readFromComments, WNDS, WNPS, RNDS, RNPS),

MEMBER FUNCTION locateInComments(pattern    IN VARCHAR2,
                                 offset     IN INTEGER := 1,
                                 occurrence IN INTEGER := 1)
                 RETURN INTEGER,
MEMBER PROCEDURE trimComments(newlen IN INTEGER),
MEMBER PROCEDURE eraseFromComments(amount IN OUT NOCOPY INTEGER,
                                   offset IN INTEGER := 1),
MEMBER PROCEDURE deleteComments,
MEMBER PROCEDURE loadCommentsFromFile(fileobj  IN BFILE,
                                      amount   IN INTEGER,
                                      from_loc IN INTEGER := 1,
                                      to_loc   IN INTEGER := 1),
MEMBER PROCEDURE copyCommentsOut(dest     IN OUT NOCOPY CLOB,
                                 amount   IN INTEGER,
                                 from_loc IN INTEGER := 1,
                                 to_loc   IN INTEGER := 1),
MEMBER FUNCTION compareComments(
                   compare_with_lob        IN CLOB,
                   amount                  IN INTEGER := 4294967295,
                   starting_pos_in_comment IN INTEGER := 1,
                   starting_pos_in_compare IN INTEGER := 1)
                RETURN INTEGER,
PRAGMA RESTRICT_REFERENCES(compareComments, WNDS, WNPS, RNDS, RNPS),

MEMBER FUNCTION getCommentLength RETURN INTEGER,
PRAGMA RESTRICT_REFERENCES(getCommentLength, WNDS, WNPS, RNDS, RNPS),
```

ORDVideo

```
-- Methods associated with the comments attribute
MEMBER PROCEDURE appendToComments(amount IN BINARY_INTEGER,
                                  buffer IN VARCHAR2),
```

```
MEMBER PROCEDURE writeToComments(offset IN INTEGER,
                                 amount IN BINARY_INTEGER,
                                 buffer IN VARCHAR2),
MEMBER FUNCTION readFromComments(offset IN INTEGER,
                                 amount IN BINARY_INTEGER := 32767)
                RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(readFromComments, WNDS, WNPS, RNDS, RNPS),

MEMBER FUNCTION locateInComments(pattern    IN VARCHAR2,
                                 offset     IN INTEGER := 1,
                                 occurrence IN INTEGER := 1)
                 RETURN INTEGER,
MEMBER PROCEDURE trimComments(newlen IN INTEGER),
MEMBER PROCEDURE eraseFromComments(amount IN OUT NOCOPY INTEGER,
                                   offset IN INTEGER := 1),
MEMBER PROCEDURE deleteComments,
MEMBER PROCEDURE loadCommentsFromFile(fileobj  IN BFILE,
                                      amount   IN INTEGER,
                                      from_loc IN INTEGER :=1,
                                      to_loc   IN INTEGER :=1),
MEMBER PROCEDURE copyCommentsOut(dest     IN OUT NOCOPY CLOB,
                                 amount   IN INTEGER,
                                 from_loc IN INTEGER :=1,
                                 to_loc   IN INTEGER :=1),
MEMBER FUNCTION compareComments(
                   compare_with_lob        IN CLOB,
                   amount                  IN INTEGER := 4294967295,
                   starting_pos_in_comment IN INTEGER := 1,
                   starting_pos_in_compare IN INTEGER := 1)
                RETURN INTEGER,
PRAGMA RESTRICT_REFERENCES(compareComments, WNDS, WNPS, RNDS, RNPS),

MEMBER FUNCTION getCommentLength RETURN INTEGER,
PRAGMA RESTRICT_REFERENCES(getCommentLength, WNDS, WNPS, RNDS, RNPS),
```

The following ORDAudio and ORDVideo accessor methods were deprecated in release 9.0.1:

ORDAudio

```
MEMBER PROCEDURE setProperties(ctx IN OUT RAW),
```

ORDVideo

```
MEMBER PROCEDURE setProperties(ctx IN OUT RAW),
```

Index

A

adding images, 3-42
advantages of using
  LOB buffering, 11-17
AIFF data format, A-1
AIFF-C data format, A-2
Apple QuickTime data format, C-2
AU data format, A-2
AVI data format, C-3

B

BFILE, 3-45, 3-46
BLOBs in table partitions
  using *inter*Media column objects, 11-17
BUFFER_POOL_KEEP parameter, 11-5
BUFFER_POOL_RECYCLE parameter, 11-5
bulk data loading methods, 11-18

C

CACHE option, 11-11
checkProperties( ) method, 6-17, 8-15, 9-18
CHUNK option, 11-11
clearLocal( ) method, 5-5, I-9
close( ) method, I-10
closeSource( ) method, 5-6
codecs (compression and decompression
    schemes), 1-4
color visual attribute, 2-4
  location visual attribute, 2-5
  specified with location, 2-5
compatibility, 4-1
compatibilityInit( ) method, 4-3
compression
  formats, A-1, B-1, C-1
content-based retrieval
  benefits, 2-1
  example, 3-50
  overview, 2-1
converting
  images, 3-54
copy( ) method, 8-16
copying
  images, 3-53

D

data
  loading multimedia, 1-15
data format, 1-8
database initialization parameter
  BUFFER_POOL_KEEP, 11-5
  BUFFER_POOL_RECYCLE, 11-5
  DB_BLOCK_SIZE, 11-2, 11-4, 11-29
  DB_CACHE_SIZE, 11-3, 11-5, 11-29
  LARGE_POOL_SIZE, 11-3
  LOG_BUFFER, 11-7
  setting, 11-2
  SHARED_POOL_RESERVED_SIZE, 11-7
  SHARED_POOL_SIZE, 11-3, 11-7
DB_BLOCK_SIZE parameter, 11-2, 11-4, 11-29
DB_CACHE_SIZE parameter, 11-3, 11-5, 11-29
DBA tuning tips, 11-1
DBMS_LOB package
  loading data, 11-24
deleteContent( ) method, 5-8

deleteLocalContent method, I-12
distance, 2-8
domain index, 2-12

E

ensuring future compatibility
  with evolving *inter*Media object types, 4-1
evaluateScore( ) method, 8-46
evolving *inter*Media object types
  ensuring future compatibility, 4-1
examples
  retrieving an image (simple read), 3-48
  retrieving images similar to an image (content-based), 3-50
  retrieving video data (simple read), 3-76
exceptions and error messages, H-1
export( ) method, 5-9, 10-5, I-13
extending *inter*Media
  audio default format, 6-56
  document default format, 7-29
  new audio format, 3-9, 3-27, 6-59
  new audio object type, 3-9, 3-28
  new data source, 3-86, I-52
  new document format, 7-30
  new image object type, 3-55
  new video format, 3-76, 9-68
  new video object type, 3-77
  video default format, 9-65
extensible index, 2-12

F

file format, A-1, B-1, C-1
formats
  compression, A-1, B-1, C-1
  file, A-1, B-1, C-1
frequently asked questions (FAQ), G-1

G

generateSignature( ) method, 8-48
getAllAttributes( ) method, 6-19, 9-20
getAttribute( ) method, 6-21, 9-22
getAudioDuration( ) method, 6-23
getBFILE( ) method, 5-13
getBFile( ) method, I-16
getBitRate method, 9-24
getCompressionFormat( ) method, 8-18
getCompressionType( ) method, 6-25, 9-25
getContent( ) method, 5-15
getContentFormat( ) method, 8-19
getContentInLob( ) method, 6-26, 7-14, 9-26
getContentInTempLob( ) method, I-17
getContentLength( ) method, 6-24, 7-16, 8-20, 9-28, I-19
getDescription( ) method, 6-28, 9-29
getEncoding( ) method, 6-29
getFileFormat( ) method, 8-21
getFormat( ) method, 6-30, 7-17, 9-30
getFrameRate method, 9-32
getFrameResolution method, 9-33
getFrameSize( ) method, 9-34
getHeight( ) method, 8-22
getLocalContent method, I-21
getMimeType( ) method, 5-17
getNumberOfChannels( ) method, 6-31
getNumberOfColors method, 9-36
getNumberOfFrames method, 9-37
getProperties( ) method (all attributes) for BFILEs, 10-24, 10-36, 10-48, 10-68
getProperties( ) method (all attributes) for BLOBs, 10-18, 10-31, 10-43, 10-62
getProperties( ) method for BFILEs, 10-22, 10-34, 10-46, 10-66
getProperties( ) method for BLOBs, 10-16, 10-29, 10-41, 10-60
getSampleSize( ) method, 6-32
getSamplingRate( ) method, 6-33
getSource( ) method, 5-19
getSourceAddress( ) method, I-22
getSourceInformation method, I-24
getSourceLocation( ) method, 5-21, I-25
getSourceName( ) method, 5-22, I-26
getSourceType( ) method, 5-23, I-27
getUpdateTime( ) method, 5-25, I-28
getVideoDuration method, 9-38
getWidth( ) method, 8-23

I

image
  attributes, 2-2
import( ) method, 6-34, 7-18, 8-24, 9-39, I-29
importFrom( ) method, 6-36, 7-21, 8-26, 9-41, 10-8, I-31
importFrom( ) method (all attributes), 10-11
indexing signatures, 2-12
init( ) for ORDImage method, 8-7
init( ) for ORDImageSignature method, 8-44
init( ) method, 6-8, 7-6, 9-9
init(srcType,srcLocation,srcName) for ORDImage method, 8-9
init(srcType,srcLocation,srcName) method, 6-10, 7-8, 9-11
INITIAL and NEXT parameters, 11-12
initializing *inter*Media column objects, 11-8
inserting images, 3-43
interchange format, 1-8
*inter*Media
  guidelines for best performance results, 11-28
  improving multimedia LOB data retrieval and update performance, 11-29
  initializing column objects, 11-8
  media data storage model, 1-3
  objects types, 1-3
  reading data from an object, 11-25
  relational functional interface, 10-1
  setting column object to empty, 11-8
  setting column objects to NULL, 11-8
  strategies with column objects, 11-8
*inter*Media Clipboard
  loading data, 11-25
*inter*Media column objects
  tablespace, 11-9
*inter*Media object types evolution
  ensuring future compatibility, 4-1
isLocal method, I-33
isLocal( ) method, 5-26
isSimilar( ) method, 8-49

L

LARGE_POOL_SIZE parameter, 11-3
loading data
  bulk methods, 11-18
  multimedia, 1-15
  using DBMS_LOB package, 11-24
  using *inter*Media Clipboard, 1-15, 11-25
  using OCI, 11-24
  using PL/SQL, 1-15, 11-18
  using SQL*Loader, 1-15
loading FILE data into *inter*Media objects, 11-18
LOB buffering
  advantages of using, 11-17
LOB index
  using with *inter*Media column objects, 11-10
location visual attribute, 2-5
  specified with color, 2-5
LOG_BUFFER parameter, 11-7
LOGGING option, 11-11
lossless compression, 1-8
lossy compression, 1-8

M

matching
  preparing or selecting images for, 2-13
MAXEXTENTS parameter, 11-14
memory allocation
  tuning, 11-5
messages, error, exceptions, H-1
methods, 8-45, I-6
  checkProperties( ), 6-17, 8-15, 9-18
  clearLocal( ), 5-5, I-9
  close( ), I-10
  closeSource( ), 5-6
  common, 5-1
  compatibilityInit( ), 4-3
  copy( ), 8-16
  deleteContent( ), 5-8
  deleteLocalContent, I-12
  evaluateScore( ), 8-46
  export( ), 5-9, 10-5, I-13
  for ORDDoc, 7-10
  generateSignature( ), 8-48
  getAllAttributes( ), 6-19, 9-20
  getAttribute( ), 6-21, 9-22
  getAudioDuration( ), 6-23

getBFILE( ), 5-13
getBFile( ), I-16
getBitRate, 9-24
getCompressionFormat( ), 8-18
getCompressionType( ), 6-25, 9-25
getContent( ), 5-15
getContentFormat( ), 8-19
getContentInLob( ), 6-26, 7-14, 9-26
getContentInTempLob( ), I-17
getContentLength( ), 6-24, 7-16, 8-20, 9-28, I-19
getDescription( ), 6-28, 9-29
getEncoding( ), 6-29
getFileFormat( ), 8-21
getFormat( ), 6-30, 7-17, 9-30
getFrameRate, 9-32
getFrameResolution, 9-33
getFrameSize( ), 9-34
getHeight( ), 8-22
getLocalContent, I-21
getMimeType( ), 5-17
getNumberOfChannels( ), 6-31
getNumberOfColors, 9-36
getNumberOfFrames, 9-37
getProperties( ) (all attributes) for BFILEs, 10-24, 10-36, 10-48, 10-68
getProperties( ) (all attributes) for BLOBs, 10-18, 10-31, 10-43, 10-62
getProperties( ) for BFILEs, 10-22, 10-34, 10-46, 10-66
getProperties( ) for BLOBs, 10-16, 10-29, 10-41, 10-60
getSampleSize( ), 6-32
getSamplingRate( ), 6-33
getSource( ), 5-19
getSourceAddress( ), I-22
getSourceInformation, I-24
getSourceLocation( ), 5-21, I-25
getSourceName( ), 5-22, I-26
getSourceType( ), 5-23, I-27
getUpdateTime( ), 5-25, I-28
getVideoDuration, 9-38
getWidth( ), 8-23
import( ), 6-34, 7-18, 8-24, 9-39, I-29
importFrom( ), 6-36, 7-21, 8-26, 9-41, 10-8, I-31
importFrom( ) (all attributes), 10-11
init( ), 6-8, 7-6, 9-9
init( ) for ORDImage, 8-7
init( ) for ORDImageSignature, 8-44
init(srcType,srcLocation,srcName), 6-10, 7-8, 9-11
init(srcType,srcLocation,srcName) for ORDImage, 8-9
isLocal, I-33
isLocal( ), 5-26
isSimilar( ), 8-49
open( ), I-34
openSource( ), 5-27
ORDAudio, 6-12
ORDDoc, 7-10
ORDimage, 8-10
ORDImageSignature, 8-45
ORDSource, I-6
ORDVideo, 9-13
process( ), 8-29, 10-51
processAudioCommand( ), 6-39
processCommand( ), I-36
processCopy( ), 8-34
processCopy( ) for BFILEs, 10-55
processCopy( ) for BLOBs, 10-53
processSourceCommand( ), 5-29
processVideoCommand( ), 9-44
read( ), I-38
readFromSource( ), 5-32
relational interface, 10-2
setAudioDuration( ), 6-42
setBitRate( ), 9-47
setCompressionType( ), 6-43, 9-48
setDescription( ), 6-44, 9-49
setEncoding( ), 6-46
setFormat( ), 6-47, 7-24, 9-51
setFrameRate( ), 9-53
setFrameResolution( ), 9-54
setFrameSize( ), 9-55
setKnownAttributes( ), 6-49, 9-57
setLocal, I-40
setLocal( ), 5-34
setMimeType( ), 5-35
setNumberOfChannels, 6-51
setNumberOfColors( ), 9-60
setNumberOfFrames( ), 9-61

setProperties, 8-36
setProperties( ), 6-52, 9-62
setProperties( ) (XML), 6-52, 7-26
setProperties( ) for foreign images, 8-38
setSampleSize( ), 6-55
setSamplingRate( ), 6-54
setSource( ), 5-37
setSourceInformation( ), I-41
setUpdateTime( ), 5-39, I-43
setVideoDuration( ), 9-64
trim( ), I-44
trimSource( ), 5-40
write( ), I-46
writeToSource( ), 5-42
multimedia LOB data retrieval and update performance
improving, 11-29

O

object relational technology, 1-1
object types
ORDAudio, 6-3
ORDDoc, 7-3
ORDImage, 8-3
ORDImageSignature, 8-42
ORDSource, I-3
ORDVideo, 9-3
object views, 3-10, 3-28, 3-57, 3-77
OCI
loading data, 11-24
open( ) method, I-34
openSource( ) method, 5-27
ORDAudio object type
reference information, 6-3
ORDDoc object type
reference information, 7-3
ORDImage object type
reference information, 8-3
ORDImageSignature object type
reference information, 8-42
ORDPLUGINS.ORDX_<srcType>_SOURCE package, I-52
ORDPLUGINS.ORDX_DEFAULT_VIDEO package, 9-65
ORDPLUGINS.ORDX_FILE_SOURCE package, I-48
ORDPLUGINS.ORDX_HTTP_SOURCE package, I-50
ORDSource object type
reference information, I-3
ORDVideo object type
reference information, 9-3
ORDX_DEFAULT_AUDIO package, 6-56
ORDX_DEFAULT_DOC package, 7-29

P

packages
ORDPLUGINS.ORDX_<srcType>_SOURCE, I-52
ORDPLUGINS.ORDX_DEFAULT_VIDEO, 9-65
ORDPLUGINS.ORDX_FILE_SOURCE, I-48
ORDPLUGINS.ORDX_HTTP_SOURCE, I-50
ORDX_DEFAULT_AUDIO, 6-56
ORDX_DEFAULT_DOC, 7-29
packages or PL/SQL plug-ins, 6-56, 7-29, 9-65, I-47
PCTFREE parameter, 11-15
PCTINCREASE parameter, 11-14
PCTVERSION option, 11-10
performance results
guidelines for using *inter*Media objects, 11-28
PL/SQL
loading data, 1-15
example, 11-18
populating rows, 3-44
preparing
images for matching, 2-13
process( ) method, 8-29, 10-51
processAudioCommand( ) method, 6-39
processCommand( ) method, I-36
processCopy( ) method, 8-34
processCopy( ) method for BFILEs, 10-55
processCopy( ) method for BLOBs, 10-53
processSourceCommand( ) method, 5-29
processVideoCommand( ) method, 9-44
protocol, 1-9

Q

querying rows, 3-46

R

read( ) method, I-38
readFromSource( ) method, 5-32
reading data from an *inter*Media object, 11-25
reading *inter*Media data
  example, 11-26
reference information
  ORDAudio, 6-1
  ORDDoc, 7-1
  ORDImage, 8-1
  ORDImageSignature, 8-40
  ORDSource, I-1
  ORDVideo, 9-1
related documents, xxvii
relational functional interface reference
    information, 10-1
retrieval, content-based
  benefits, 2-1
  overview, 2-1
retrieving
  images from tables, 3-48
  images similar to an image
      (content-based), 3-50
  video data from table, 3-76
RMFF data format, C-3
roll back, 3-55

S

sample program, F-1, J-1
segment and physical attributes
  PCTFREE parameter, 11-15
selecting
  images for matching, 2-13
setAudioDuration( ) method, 6-42
setBitRate( ) method, 9-47
setCompressionType( ) method, 6-43, 9-48
setDescription( ) method, 6-44, 9-49
setEncoding( ) method, 6-46
setFormat( ) method, 6-47, 7-24, 9-51
setFrameRate( ) method, 9-53
setFrameResolution( ) method, 9-54
setFrameSize( ) method, 9-55
setKnownAttributes( ) method, 6-49, 9-57
setLocal method, I-40
setLocal( ) method, 5-34
setMimeType( ) method, 5-35
setNumberOfChannels method, 6-51
setNumberOfColors( ) method, 9-60
setNumberOfFrames( ) method, 9-61
setProperties method, 8-36
setProperties( ) method, 6-52, 9-62
setProperties( ) method (XML), 6-52, 7-26
setProperties( ) method for foreign images, 8-38
setSampleSize( ) method, 6-55
setSamplingRate( ) method, 6-54
setSource( ) method, 5-37
setSourceInformation( ) method, I-41
setting
  column object to empty, 11-8
  column objects to NULL, 11-8
setting database initialization parameters, 11-2
setUpdateTime( ) method, 5-39, I-43
setVideoDuration( ) method, 9-64
SGA, 11-2
  database initialization parameters, 11-2
  sizing, 11-2
  sizing using DB_BLOCK_SIZE parameter, 11-2
  sizing using DB_CACHE_SIZE parameter, 11-3
  sizing using LARGE_POOL_SIZE
      parameter, 11-3
  sizing using SHARED_POOL_SIZE
      parameter, 11-3
shape (visual attribute), 2-4
SHARED_POOL_RESERVED_SIZE
    parameter, 11-7
SHARED_POOL_SIZE parameter, 11-3, 11-7
signature, 2-2
  indexing, 2-12
similarity calculation, 2-9
SQL*Loader
  example loading multimedia data, 11-23
  loading data, 1-15
static methods
  ORDAudio relational functional interface, 10-4,
      10-13

ORDDoc relational functional interface, 10-26
ORDImage relational functional interface, 10-38
ORDVideo relational functional interface, 10-56
storage characteristics
CACHE option, 11-11
CHUNK option, 11-11
DB_BLOCK_SIZE parameter, 11-4
INITIAL and NEXT parameters, 11-12
LOGGING option, 11-11
MAXEXTENTS parameter, 11-14
PCTINCREASE parameter, 11-14
PCTVERSION option, 11-10
STORAGE IN ROW clause, 11-14
STORAGE IN ROW clause, 11-14
strategies for column objects, 11-8
system global area
See SGA

T

table partitions
using *inter*Media column objects containing BLOBs, 11-17
tablespace characteristics
LOB index, 11-10
tablespace, 11-9
temporary conversions, 3-55
texture (visual attribute), 2-4
threshold, 2-11
thumbnail images, 8-33, 10-52
trim( ) method, I-44
trimSource( ) method, 5-40
tuning
memory allocation, 11-5

V

visual attributes, 2-2

W

WAV data format, A-3, A-5
weight, 2-8
write methods
write( ), I-46
writeToSource( ) method, 5-42