

Oracle® Multimedia

DICOM Developer's Guide

11g Release 2 (11.2)

E10778-03

August 2010

Oracle Multimedia DICOM enables Oracle Database to store, manage, and retrieve DICOM content such as single-frame and multiframe images, waveforms, slices of 3-D volumes, video segments, and structured reports in an integrated fashion with other enterprise information. Oracle Multimedia DICOM extends Oracle Database reliability, availability, and data management to media objects in medical applications.

Oracle Multimedia DICOM supports Digital Imaging and Communications in Medicine, the standard for medical images.

Oracle Multimedia DICOM Developer's Guide, 11g Release 2 (11.2)

E10778-03

Copyright © 2007, 2010, Oracle and/or its affiliates. All rights reserved.

Primary Author: Sue Pelski

Contributors: Rob Abbott, Melliya Annamalai, Janet Blowney, Fengting Chen, Dongbai Guo, Dong Lin, Susan Mavis, Valarie Moore, David Noblet, James Steiner, Yingmei Sun, Manjari Yalavarthy, Jie Zhang

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this software or related documentation is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

This software and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Contents

Preface	xv
Audience	xv
Documentation Accessibility	xvi
Related Documents	xvi
Conventions	xvii
Syntax Descriptions.....	xvii
What's New in Oracle Multimedia DICOM?	xix
New Features for Release 11.2.....	xix
Part I DICOM Common Usage and Reference	
1 Introduction to Oracle Multimedia DICOM	
1.1 Medical Imaging and Communication	1-1
1.1.1 History of the DICOM Standard	1-2
1.1.2 Overview of DICOM Content.....	1-2
1.2 Oracle Multimedia and DICOM.....	1-3
1.2.1 Oracle Multimedia DICOM Format Support	1-3
1.2.2 ORDDicom Object Type	1-4
1.2.3 DICOM Metadata Extraction	1-4
1.2.4 DICOM Conformance Validation	1-4
1.2.5 DICOM Image Processing.....	1-5
1.2.6 Making Confidential Data in DICOM Content Anonymous.....	1-5
1.2.7 Creating ORDDicom Objects from Images or Video and Metadata	1-5
1.2.8 Run-Time, Updatable DICOM Data Model.....	1-6
2 Oracle Multimedia DICOM Concepts	
2.1 Oracle Multimedia DICOM Architecture.....	2-1
2.2 Oracle Multimedia DICOM Storage	2-3
2.3 Model-Driven Design.....	2-5
2.4 DICOM Data Model Repository	2-6
2.4.1 Configuration Documents in the Repository.....	2-6
2.4.2 Administrator and User Sessions in the Repository.....	2-8
2.5 Extraction of Metadata from DICOM Content.....	2-10

2.5.1	Overview of the Metadata Extraction and XML Mapping Process	2-11
2.5.2	Sample XML Documents Used in the Extraction and Mapping Process	2-13
2.6	Validation of DICOM Content	2-14
2.7	Image Conversion and Creation of New DICOM Content	2-15
2.8	Making DICOM Content Anonymous	2-16
2.9	Extraction of DICOM Metadata for Partitioning.....	2-18

3 Overview of DICOM Development

3.1	The DICOM Development Environment	3-1
3.1.1	APIs for Use With Oracle Multimedia DICOM	3-1
3.1.2	Views in the DICOM Repository	3-2
3.2	DICOM Developer and Administrator Tasks.....	3-3
3.2.1	Task 1: Load the Repository	3-3
3.2.2	Task 2: Load the DICOM Content	3-4
3.2.3	Task 3: Extract the DICOM Metadata	3-4
3.2.3.1	Extracting Metadata: Administrator Tasks.....	3-5
3.2.3.2	Extracting Metadata: Developer Tasks.....	3-5
3.2.4	Task 4: Search and Retrieve DICOM Attributes	3-5
3.2.5	Task 5: Write and Edit DICOM Metadata	3-6
3.2.6	Task 6: Process, Convert, and Compress DICOM Data	3-7
3.2.7	Task 7: Create DICOM Content from Secondary Capture Images and Video	3-7
3.2.8	Task 8: Validate Conformance with DICOM Constraints	3-8
3.2.8.1	Validating Conformance: Administrator Tasks	3-9
3.2.8.2	Validating Conformance: Developer Tasks	3-10
3.2.9	Task 9: Protect Confidential Patient Data	3-10
3.2.9.1	Protecting Privacy: Administrator Tasks	3-11
3.2.9.2	Protecting Privacy: Developer Tasks	3-11
3.2.10	Task 10: Improve Storage When Extracting DICOM Attributes	3-12

4 DICOM Data Model Utility Reference

4.1	Directory Definition and Setup for ORD_DICOM Examples	4-1
	DICOM Data Model Utility Functions.....	4-2
	getDictionaryTag() Function	4-3
	getMappingXPath() Function	4-5
	DICOM Data Model Utility Procedures	4-8
	setDataModel() Procedure.....	4-9
	DICOM Repository Public Views.....	4-10
	orddcm_conformance_vld_msgs	4-11
	orddcm_constraint_names	4-12
	orddcm_documents.....	4-13
	orddcm_document_types.....	4-14

Part II DICOM Development Usage and Reference

5 DICOM Application Development

5.1	Setting Up Your Environment	5-1
5.2	Creating a Table with an ORDDicom Column.....	5-2
5.3	Loading DICOM Content Using the SQL*Loader Utility	5-3
5.4	Developing DICOM Applications Using the PL/SQL API.....	5-7
5.4.1	Selecting DICOM Attributes	5-8
5.4.2	Creating Thumbnail Images and Changing Image Formats.....	5-9
5.4.3	Making Anonymous Copies of ORDDicom Objects	5-10
5.4.4	Checking the Conformance of ORDDicom Objects.....	5-11
5.4.5	Handling Oracle Multimedia DICOM Exceptions in PL/SQL.....	5-13
5.5	Developing DICOM Applications Using the DICOM Java API	5-13
5.5.1	Setting Up Your Environment Variables.....	5-14
5.5.2	Importing Oracle Java Classes into Your Application	5-14
5.5.3	Handling Oracle Multimedia DICOM Exceptions in Java	5-14

6 DICOM Sample Application

6.1	Overview of the DICOM Sample Application	6-1
6.2	Description of the DICOM Sample Application	6-3
6.2.1	Logging In to the DICOM Image Archive Interface.....	6-5
6.2.2	Searching for Specific DICOM Images	6-7
6.2.2.1	Attribute Searches	6-8
6.2.2.2	Keyword Searches	6-10
6.2.2.3	Semantic Searches.....	6-11
6.2.3	Importing DICOM Images as a Clinician.....	6-13
6.2.4	Processing DICOM Images as a Researcher	6-15
6.2.5	Logging In to the DICOM Image Archive Administration Interface.....	6-17
6.2.6	Inserting Configuration Documents	6-18

7 ORDDicom Object Type Reference

7.1	ORDDicom Object Examples	7-1
7.1.1	Directory Definition and Setup for ORDDicom Object Examples	7-2
7.1.2	MEDICAL_IMAGE_OBJ Table Definition.....	7-2
	ORDDicom Object Type.....	7-3
	ORDDicom Constructors.....	7-4
	ORDDicom() for BLOBs.....	7-5
	ORDDicom() for ORDImage.....	7-6
	ORDDicom() for Other Sources.....	7-7
	ORDDicom Methods	7-9
	export()	7-10
	extractMetadata()	7-12
	getAttributeByName()	7-14
	getAttributeByTag().....	7-15
	getContent().....	7-16
	getContentLength()	7-17

getSeriesInstanceUID().....	7-18
getSOPClassUID().....	7-19
getSOPInstanceUID()	7-20
getSourceInformation().....	7-21
getSourceLocation().....	7-22
getSourceName()	7-23
getSourceType()	7-24
getStudyInstanceUID()	7-25
import()	7-26
isAnonymous().....	7-28
isConformanceValid()	7-29
isLocal()	7-31
makeAnonymous().....	7-32
processCopy() to BLOBs.....	7-34
processCopy() to ORDDicom	7-35
processCopy() to ORDImage	7-37
setProperties().....	7-39
writeMetadata().....	7-40

8 DICOM Relational Interface Reference

8.1	Examples for DICOM Relational Functions and Procedures.....	8-1
8.1.1	Directory Definition and Setup for DICOM Relational Examples	8-2
8.1.2	MEDICAL_IMAGE_REL Table Definition	8-2
	DICOM Relational Functions.....	8-3
	extractMetadata() for BFILES	8-4
	extractMetadata() for BLOBs.....	8-6
	extractMetadata() for ORDImage.....	8-8
	isAnonymous() for BFILES	8-10
	isAnonymous() for BLOBs	8-11
	isAnonymous() for ORDImage.....	8-12
	isConformanceValid() for BFILES	8-13
	isConformanceValid() for BLOBs.....	8-15
	isConformanceValid() for ORDImage	8-17
	DICOM Relational Procedures	8-19
	createDicomImage() for BFILES.....	8-20
	createDicomImage() for BLOBs	8-22
	createDicomImage() for ORDImage	8-24
	export()	8-26
	importFrom().....	8-28
	makeAnonymous() for BFILES	8-30
	makeAnonymous() for BLOBs.....	8-32
	makeAnonymous() for ORDImage	8-34

processCopy() for BFILES	8-36
processCopy() for BFILES with SOP Instance UID	8-37
processCopy() for BLOBs	8-39
processCopy() for BLOBs with SOP Instance UID	8-40
processCopy() for ORDImage.....	8-42
processCopy() for ORDImage with SOP Instance UID.....	8-44
writeMetadata() for BFILES.....	8-46
writeMetadata() for BLOBs	8-48
writeMetadata() for ORDImage.....	8-50

Part III DICOM Administration Usage and Reference

9 Overview of DICOM Administration

9.1	Assigning Administrator Roles and Privileges	9-2
9.2	Managing XML Schemas	9-2
9.2.1	Registering XML Schemas.....	9-3
9.2.2	Finding User-Defined XML Schemas	9-3
9.3	Loading the Data Model Repository	9-4
9.4	Browsing the Repository with Views.....	9-4
9.5	Exporting Documents from the Repository	9-5
9.6	Inserting Documents into the Repository	9-5
9.6.1	Inserting Anonymity, Mapping, and Constraint Documents.....	9-6
9.6.2	Inserting Dictionary Documents	9-6
9.6.3	Inserting Preference and UID Definition Documents	9-6
9.6.4	Inserting Stored Tag List Documents	9-6
9.7	Updating Documents in the Repository	9-7
9.7.1	Updating Anonymity, Mapping, and Constraint Documents.....	9-7
9.7.2	Updating Dictionary Documents	9-7
9.7.3	Updating Preference and UID Definition Documents	9-8
9.8	Deleting Documents from the Repository.....	9-8
9.8.1	Deleting Anonymity, Mapping, and Constraint Documents.....	9-9
9.8.2	Deleting Dictionary Documents	9-9
9.8.3	Deleting Preference and UID Definition Documents	9-9
9.9	Oracle Data Pump Utilities Support for the Data Model Repository	9-9
9.9.1	Roles for Export and Import Operations.....	9-10
9.9.2	Modes for Export and Import Operations	9-10
9.9.2.1	Exporting in Schema Mode.....	9-10
9.9.2.2	Exporting in Full Mode.....	9-11
9.9.2.3	Importing in Schema Mode.....	9-11
9.9.2.4	Importing in Full Mode	9-12

10 Creating DICOM Configuration Documents

10.1	Characteristics of Configuration Documents.....	10-2
10.1.1	Characteristics of Anonymity Documents.....	10-2
10.1.2	Characteristics of Constraint Documents.....	10-2

10.1.3	Characteristics of Mapping Documents	10-3
10.1.4	Characteristics of Standard Dictionary Documents	10-3
10.1.5	Characteristics of Private Dictionary Documents	10-4
10.1.6	Characteristics of Preference Documents	10-4
10.1.7	Characteristics of UID Definition Documents	10-4
10.1.8	Characteristics of Stored Tag List Documents	10-4
10.2	Writing Configuration Documents	10-5
10.2.1	Creating Anonymity Documents	10-5
10.2.1.1	Making Standard Attributes Anonymous	10-7
10.2.1.2	Making Undefined Standard Attributes Anonymous	10-8
10.2.1.3	Making Selected Private Attributes Anonymous	10-8
10.2.1.4	Making All Private Attributes Anonymous	10-9
10.2.1.5	Using DICOM Value Locators in Anonymity Documents	10-10
10.2.2	Creating Constraint Documents	10-11
10.2.2.1	Defining a Simple Constraint Rule	10-12
10.2.2.2	Defining Constraint Rules by Importing Other Constraint Rules	10-15
10.2.2.3	Defining and Referencing Constraint Macros	10-15
10.2.2.4	Defining Recursive Constraint Macros	10-18
10.2.2.5	Using DICOM Value Locators in Constraint Documents	10-22
10.2.3	Creating Mapping Documents and Metadata XML Schemas	10-25
10.2.3.1	Structure of a Mapping Document	10-25
10.2.3.2	Structure of a Metadata XML Schema	10-26
10.2.3.3	Creating a Mapping Document for Metadata with No Schema Constraints .	10-27
10.2.3.4	Creating a Mapping Document for Metadata with Schema Constraints and a Mapped Section Only	10-29
10.2.3.5	Creating a Mapping Document for Metadata with Schema Constraints	10-33
10.2.3.6	Using DICOM Value Locators in Mapping Documents	10-38
10.2.4	Creating Standard Dictionary Documents	10-40
10.2.4.1	Defining Standard Attributes	10-40
10.2.4.2	Retiring Standard Attributes	10-41
10.2.5	Creating Private Dictionary Documents	10-42
10.2.5.1	Defining Private Attributes	10-42
10.2.5.2	Defining Attribute Definers	10-44
10.2.5.3	Retiring Private Attributes	10-45
10.2.6	Creating Preference Documents	10-45
10.2.6.1	Defining the BINARY_SKIP_INVALID_ATTR Preference Parameter	10-46
10.2.6.2	Defining the EXP_IF_NULL_ATTR_IN_CONSTRAINT Preference Parameter	10-47
10.2.6.3	Defining the MANDATE_ATTR_TAGS_IN_STL Preference Parameter	10-47
10.2.6.4	Defining the MAX_RECURSION_DEPTH Preference Parameter	10-48
10.2.6.5	Defining the SPECIFIC_CHARACTER_SET Preference Parameter	10-49
10.2.6.6	Defining the SQ_WRITE_LEN Preference Parameter	10-50
10.2.6.7	Defining the VALIDATE_METADATA Preference Parameter	10-50
10.2.6.8	Defining the XML_SKIP_ATTR Preference Parameter	10-51
10.2.7	Creating UID Definition Documents	10-51
10.2.7.1	Defining UID Definitions	10-52
10.2.7.2	Retiring UID Definitions	10-52
10.2.8	Creating Stored Tag List Documents	10-53

11 Administering the DICOM Repository

11.1	Sample Session 1: Inserting Two Documents	11-1
11.2	Sample Session 2: Updating a Mapping Document	11-3
11.3	Sample Session 3: Deleting a Constraint Document.....	11-4
11.4	Sample Session 4: Inserting a Stored Tag List Document.....	11-5
11.4.1	Inserting a Stored Tag List Document with a Known Set of Tags.....	11-6
11.4.2	Generating and Inserting a Stored Tag List Document	11-7

12 ORD_DICOM_ADMIN Package Reference

12.1	Directory Definition and Setup for ORD_DICOM_ADMIN Examples.....	12-1
12.2	Important Notes for DICOM Repository Administrators	12-2
	DICOM Data Model Repository Administrator Functions	12-3
	generateTagListDocument() Function.....	12-4
	getDocumentContent() Function.....	12-5
	DICOM Data Model Repository Administrator Procedures.....	12-6
	deleteDocument() Procedure	12-7
	editDataModel() Procedure.....	12-8
	exportDocument() Procedure	12-9
	insertDocument() Procedure.....	12-11
	publishDataModel() Procedure	12-13
	rollbackDataModel() Procedure	12-14
	DICOM Repository Administrator Views.....	12-15
	orddcm_document_refs.....	12-16
	General Format for DICOM Value Locators.....	12-17

Part IV DICOM Appendixes

A DICOM Configuration Documents

B DICOM XML Schemas

B.1	Anonymity Document Schema.....	B-2
B.2	Constraint Document Schema.....	B-4
B.3	Data Type Definition Schema	B-12
B.4	Default DICOM Metadata Schema.....	B-27
B.5	Manifest Document Schema.....	B-27
B.6	Mapping Document Schema	B-29
B.7	Metadata Data Type Definition Schema.....	B-32
B.8	Preference Document Schema.....	B-47
B.9	Private Dictionary Document Schema.....	B-52
B.10	Standard Dictionary Document Schema	B-53
B.11	Stored Tag List Document Schema	B-55
B.12	UID Definition Document Schema.....	B-56

C DICOM Encoding Rules

C.1	Transfer Syntax for Medical Imaging	C-1
C.2	Definitions for Transfer Syntax Abbreviations.....	C-2

D DICOM Processing and Supported Formats

D.1	DICOM Image Content and Compression Formats	D-1
D.1.1	DEFLATE Compression Format.....	D-2
D.1.2	MPEG Compression Format	D-3
D.1.3	RLE Compression Format	D-3
D.2	The frame Image Processing Operator	D-3
D.3	Other Image Processing Operators	D-3
D.4	Multiframe Image Processing and Creation	D-4
D.5	Multiframe DICOM Content Processing to AVI Format	D-4
D.6	Order of Precedence with processCopy() Method Arguments.....	D-5

E DICOM Sample Applications

E.1	Oracle Multimedia DICOM Tutorial.....	E-1
E.2	Oracle Multimedia DICOM Image Archive Demonstration	E-2

F Migrating from Release 10.2 DICOM Support

F.1	Using the DICOM Relational Interface to Migrate Applications.....	F-1
F.2	Copying Data and Rewriting Applications for DICOM	F-2
F.3	Choosing a Migration Option	F-3

Glossary

Index

List of Examples

2-1	Sample XML Mapping Document.....	2-13
2-2	Sample XML Metadata Document.....	2-13
2-3	Sample Constraint Document.....	2-14
2-4	Sample Anonymity Document.....	2-17
3-1	Constraint Rule for the Patient Module.....	3-9
5-1	Creating a Table for DICOM Content.....	5-2
5-2	Loading DICOM Content.....	5-3
5-3	Finish Loading and Initializing the DICOM Table.....	5-5
5-4	Selecting Metadata from the DICOM Content.....	5-8
5-5	Generating and Processing the New ORDImage Object.....	5-9
5-6	Populating the Column and Generating an Anonymous ORDDicom Object.....	5-11
5-7	Checking DICOM Conformance.....	5-12
6-1	Script to Create the Main Archive Table.....	6-2
9-1	Finding User-Registered XML Schemas.....	9-3
9-2	Schema Mode Export.....	9-10
9-3	Schema Mode Import.....	9-11
10-1	Making a Standard Attribute Anonymous.....	10-7
10-2	Removing All Undefined Standard Attributes.....	10-8
10-3	Making a Private Attribute Anonymous.....	10-8
10-4	Removing All Private Attributes.....	10-9
10-5	Removing All Undefined Private Attributes.....	10-9
10-6	Including Defined Private Attributes and Removing Undefined Private Attributes....	10-9
10-7	Removing Defined Private Attributes and Including Undefined Private Attributes..	10-10
10-8	Simple DICOM Value Locator in an Anonymity Document.....	10-10
10-9	DICOM Value Locator with a Wildcard Character in an Anonymity Document.....	10-10
10-10	Predicate for One Condition on SOP Class UID.....	10-13
10-11	Predicate for One Condition on SOP Instance UID.....	10-13
10-12	Predicate for Two Conditions.....	10-13
10-13	Global Rule for Two Boolean Functions.....	10-13
10-14	Global Rule for Two Conditions on Patient's Sex.....	10-14
10-15	Global Rule for a Logical Relation.....	10-15
10-16	External Rule for Three Constraint Rules.....	10-15
10-17	Global Macro for a Condition.....	10-16
10-18	Global Rule with a Constraint Macro.....	10-17
10-19	One Constraint Macro with Recursion.....	10-20
10-20	Two Constraint Macros with Recursion.....	10-21
10-21	DICOM Value Locator with a Wildcard Character in a Constraint Macro.....	10-23
10-22	DICOM Value Locator with a Special Wildcard Tag in a Global Rule.....	10-24
10-23	Sample Mapping Document for Metadata with No Schema Constraints.....	10-27
10-24	Resulting XML Document for Example 10-23	10-28
10-25	Sample Mapping Document for Metadata with Schema Constraints and a Mapped Section Only 10-29	
10-26	Sample XML Schema for Example 10-25	10-30
10-27	Resulting XML Document for Example 10-25	10-32
10-28	Sample Mapping Document for Metadata with Schema Constraints.....	10-33
10-29	Sample XML Schema for Example 10-28	10-34
10-30	Resulting XML Document for Example 10-28 (MAPPED).....	10-35
10-31	Resulting XML Document for Example 10-28 (ALL).....	10-36
10-32	DICOM Value Locator for an Attribute with Unibyte Encoding in a Mapping Document ... 10-38	
10-33	DICOM Value Locator for an Embedded Sequence Attribute in a Mapping Document 10-39	
10-34	DICOM Value Locator for a Single Attribute Value in a Mapping Document.....	10-39
10-35	Definition of a Standard Attribute with a Simple Attribute Tag.....	10-41

10-36	Definition of a Standard Attribute with a Wildcard Attribute Tag.....	10-41
10-37	Definition of a Standard Attribute As Retired.....	10-41
10-38	Definition of a Private Attribute with a Simple Attribute Tag.....	10-43
10-39	Definition of a Private Attribute with a Wildcard Attribute Tag	10-43
10-40	Definition of a Private Attribute with a Range Attribute Tag.....	10-44
10-41	Definition of an Attribute Definer	10-45
10-42	Definition of a Private Attribute as Retired	10-45
10-43	Definition of the BINARY_SKIP_INVALID_ATTR Preference Parameter	10-46
10-44	Definition of the EXP_IF_NULL_ATTR_IN_CONSTRAINT Preference Parameter ...	10-47
10-45	Definition of the MANDATE_ATTR_TAGS_IN_STL Preference Parameter	10-48
10-46	Definition of the MAX_RECURSION_DEPTH Preference Parameter.....	10-48
10-47	Definition of the SPECIFIC_CHARACTER_SET Preference Parameter	10-49
10-48	Definition of the SQ_WRITE_LEN Preference Parameter	10-50
10-49	Definition of the VALIDATE_METADATA Preference Parameter	10-50
10-50	Definition of the XML_SKIP_ATTR Preference Parameter	10-51
10-51	Definition of a Storage Class UID Definition.....	10-52
10-52	Definition of a UID Definition as Retired.....	10-52
11-1	Inserting a Stored Tag List Document with Known Attribute Tags	11-6
11-2	Sample Preference Document for the Mandate Attribute Tags Rule.....	11-7
11-3	Generating and Exporting a Stored Tag List Document to a File.....	11-8
B-1	Anonymity Document Schema	B-2
B-2	Constraint Document Schema.....	B-4
B-3	Data Type Definition Schema	B-12
B-4	Default DICOM Metadata Schema.....	B-27
B-5	Manifest Document Schema.....	B-27
B-6	Mapping Document Schema	B-29
B-7	Data Type Definition Schema	B-32
B-8	Preference Document Schema.....	B-47
B-9	Private Dictionary Document Schema.....	B-52
B-10	Standard Dictionary Document Schema	B-53
B-11	Stored Tag List Document Schema	B-55
B-12	UID Definition Document Schema.....	B-56

List of Figures

2-1	Oracle Multimedia DICOM Architecture.....	2-3
2-2	ORDDicom Object.....	2-4
2-3	Table in a Medical Image Database.....	2-5
2-4	DICOM Model-Based Parsing Details	2-6
2-5	DICOM Data Model Repository	2-10
2-6	DICOM Metadata Extraction and XML Mapping	2-11
2-7	Image Conversion Process.....	2-16
6-1	Login Page for Clinicians and Researchers	6-5
6-2	Search Area on the DICOM Image Archive Home Page	6-5
6-3	Thumbnails Area on the DICOM Image Archive Home Page.....	6-6
6-4	Tasks Area on the DICOM Image Archive Home Page.....	6-6
6-5	Image Summary Area on the DICOM Image Archive Home Page.....	6-7
6-6	Thumbnail Image Result for an Attribute Search	6-9
6-7	Thumbnail Image Result for a Keyword Search	6-10
6-8	Image Summary Result for a Keyword Search.....	6-11
6-9	Thumbnail Image Results for a Semantic Search	6-12
6-10	Image Summary Result for a Semantic Search	6-12
6-11	Import DICOM Images Page.....	6-14
6-12	Importing an Image as a Clinician	6-14
6-13	Process DICOM Images Page.....	6-16
6-14	Processing an Image as a Researcher	6-17
6-15	Login Page for Administrators	6-17
6-16	Administrator Home Page.....	6-18
6-17	Add Repository Documents Page	6-19
6-18	Administrator Home Page with a New Constraint Document.....	6-20

List of Tables

2-1	Configuration Documents and Their XML Schemas.....	2-7
3-1	Application Programming Interfaces for DICOM	3-2
3-2	DICOM Repository Views	3-2
5-1	Sample Contents of an ORDDicom Object in a Database Table	5-7
6-1	DICOM Image Archive Interface: Tasks and Designated Users.....	6-3
6-2	DICOM Image Archive Administration Interface: Tasks	6-4
6-3	DICOM Sample Application: Search Capabilities	6-4
10-1	action Attribute Values	10-5
10-2	Global Action Elements.....	10-6
C-1	Encoding Rules for Transfer Syntax.....	C-1
D-1	DICOM Content Photometric Interpretations	D-1
D-2	DICOM Content Compression Formats	D-2
E-1	Oracle Multimedia DICOM Sample Programs	E-1

Preface

This guide describes how to use the Digital Imaging and Communications in Medicine (DICOM) feature of Oracle Multimedia, which ships with Oracle Database.

In Oracle Database 11g Release 1 (11.1), the name Oracle *interMedia* was changed to Oracle Multimedia. The feature remains the same, only the name has changed. References to Oracle *interMedia* were replaced with Oracle Multimedia although, some references to Oracle *interMedia* or *interMedia* might still appear in graphical user interfaces, code examples, and related documents in the Documentation Library for Oracle Database 11g Release 2 (11.2).

This information in this guide is organized as follows:

Title	Contents
Part I, "DICOM Common Usage and Reference"	Introductory, conceptual, user, and reference information, which is common to administrators and developers of DICOM applications
Part II, "DICOM Development Usage and Reference"	User and reference information for developers of DICOM applications
Part III, "DICOM Administration Usage and Reference"	User and reference information for administrators of the DICOM data model repository
Part IV, "DICOM Appendixes"	Supplementary information about Oracle Multimedia DICOM

The sample code in this guide might not match the code shipped with Oracle Database Examples media. To run examples that are shipped with Oracle Database Examples media on your system, use the files provided with Oracle Database Examples media. Do not attempt to compile and run the code in this guide.

For information about Oracle Database and the features and options that are available to you, see *Oracle Database New Features Guide*.

Audience

This guide is for application developers and administrators who are interested in storing, retrieving, and manipulating DICOM format medical images and other objects in a database.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. 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. Accessibility 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 more information, visit the Oracle Accessibility Program Web site at <http://www.oracle.com/accessibility/>.

Accessibility of Code Examples in Documentation

Screen readers 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, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

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

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/support/contact.html> or visit <http://www.oracle.com/accessibility/support.html> if you are hearing impaired.

Related Documents

Note: For information added after the release of this guide, see the online README.txt file under your <ORACLE_HOME> directory. Depending on your operating system, this file may be in

<ORACLE_HOME>/ord/im/admin/README.txt

See your operating system-specific installation guide for more information.

For more information about using Oracle Multimedia in a development environment, see the following documents in the Oracle Database Online Documentation Library:

- *Oracle Multimedia Reference*
- *Oracle Multimedia User's Guide*
- *Oracle Call Interface Programmer's Guide*
- *Oracle Database Advanced Application Developer's Guide*
- *Oracle Database SecureFiles and Large Objects Developer's Guide*
- *Oracle Database Concepts*
- *Oracle Database PL/SQL Language Reference*

- *Oracle Database SQL Language Reference*
- *Oracle Database Error Messages*
- *Oracle Database Java Developer's Guide*
- *Oracle Database JDBC Developer's Guide and Reference*

For more information about using XML, see *Oracle XML DB Developer's Guide*.

For more information about medical imaging standards, see the documentation provided by the National Electrical Manufacturers Association (NEMA).

For reference information about Oracle Multimedia Java classes in Javadoc format, see the following Oracle API documentation (also known as Javadoc) in the Oracle Database Online Documentation Library:

- *Oracle Multimedia DICOM Java API Reference*
- *Oracle Multimedia Java API Reference*
- *Oracle Multimedia Servlets and JSP Java API Reference*
- *Oracle Multimedia Mid-Tier Java API Reference*

For more information about Java, including information about Java Advanced Imaging (JAI), see the API documentation provided by Oracle.

Many of the examples in this guide are based on the database user PM and the tables MEDICAL_IMAGE_OBJ and MEDICAL_IMAGE_REL, which are created in the Product Media (PM) sample schema. See *Oracle Database Sample Schemas* for information about how these schemas are installed and how you can use them.

Conventions

The following text conventions are used in this guide:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Syntax Descriptions

Syntax descriptions are provided in this guide for various SQL, PL/SQL, or other command-line constructs in graphic form or Backus Naur Form (BNF). See *Oracle Database SQL Reference* for information about how to interpret these descriptions.

What's New in Oracle Multimedia DICOM?

This document summarizes the new features, enhancements, APIs, and Oracle Database support introduced with Oracle Multimedia DICOM in the current release.

New Features for Release 11.2

Oracle Database 11g Release 2 (11.2) adds the following DICOM enhancements, which are described in this guide.

Recursive Calls within Constraint Definitions

Oracle Multimedia DICOM now supports recursive calls within constraint definitions. See [Section 10.2.2.4](#) for more information and examples about how to use this enhancement.

Limiting the List of DICOM Attributes for Extraction

Oracle Multimedia DICOM now provides the ability to limit the list of DICOM attributes extracted by the `setProperties()` method, thus improving performance and reducing storage requirements. See [Section 3.2.10](#) for more information about this enhancement.

Oracle Multimedia Mid-Tier Java API

Oracle Multimedia DICOM now includes the Oracle Multimedia Mid-Tier Java API, which enables the extraction of DICOM metadata outside of Oracle Database by a client tool or in the middle tier. This enhancement provides these benefits:

- Enables users to write Java applications for extracting DICOM metadata outside of Oracle Database.
- Facilitates the extraction of DICOM metadata attributes on which data is partitioned before the data is loaded into the database.

See [Section 2.9](#) for more information about this enhancement.

See Also:

Oracle Multimedia Mid-Tier Java API Reference for reference information

DICOM Value Locator Type Syntax

Oracle Multimedia DICOM now provides DICOM value locator type syntax to enable users to reference DICOM metadata tags used in the DICOM configuration documents and that refer to private attribute definers. See [General Format for DICOM Value Locators](#) for complete reference information. See [Section 10.2.1.5](#), [Section 10.2.2.5](#), and

[Section 10.2.3.6](#), for more information about using DICOM value locators in anonymity, constraint, and mapping documents, respectively.

See Also:

Oracle Multimedia Mid-Tier Java API Reference for reference information about DICOM value locators in the mid-tier Java API

Wildcard Character Syntax

Oracle Multimedia DICOM now provides wildcard character syntax to enable iteration through all the components in a single predicate. See [General Format for DICOM Value Locators](#) for reference information about the wildcard character. See [Section 10.2.2.5](#) for more information about this enhancement.

New Preference Parameters to Define Run-Time Behavior

These new preference parameters can be used in a preference document to define the run-time behavior of Oracle Multimedia DICOM:

- **BINARY_SKIP_INVALID_ATTR**
Oracle Multimedia DICOM now supports the ability to specify whether to include or skips invalid attributes and their values in the binary output of DICOM content when making a copy of the DICOM Part 10 file. See [Section 10.2.6.1](#) for more information about defining this preference parameter in a preference document.
- **MANDATE_ATTR_TAGS_IN_STL**
Oracle Multimedia DICOM now supports the ability to specify whether to enforce the rule that all tags used by the constraint and mapping documents must be listed in the stored tag list document. See [Section 10.2.6.3](#) for more information about defining this preference parameter in a preference document.
- **MAX_RECURSION_DEPTH**
Oracle Multimedia DICOM now supports the ability to specify the maximum level of recursion to use when evaluating recursive constraint macros during conformance validation. See [Section 10.2.6.4](#) for more information about defining this preference parameter in a preference document. See [Section 10.2.2.4](#) for samples of recursive constraint macros.
- **SPECIFIC_CHARACTER_SET**
Oracle Multimedia DICOM now supports the ability to specify a character set to override the default (ISO-IR 6 or ASCII) when the Specific Character Set attribute (0008,0005) is missing. See [Section 10.2.6.5](#) for more information about defining this preference parameter in a preference document.
- **SQ_WRITE_LEN**
Oracle Multimedia DICOM now supports the ability to specify how the method `writeMetadata()` encodes DICOM sequence (SQ) types. See [Section 10.2.6.6](#) for more information about defining this preference parameter in a preference document.

See [Section 10.1.6](#) and [Section 10.2.6](#) for more information about preference documents. See [Section B.8](#) for a listing of the preference document schema that constrains preference documents.

DICOM to AVI Conversion

Oracle Multimedia DICOM now supports the processing of multiframe DICOM content such as MRIs, CTs, and Ultrasound videos to the data format Microsoft Video for Windows Audio Video Interleave (AVI). See [Section D.5](#) for more information about this enhancement.

See Also:

Oracle Multimedia Reference for more information about the AVI data format

DEFLATE Compression Format

Oracle Multimedia DICOM now supports the DEFLATE compression format. See [Section D.1.1](#) for more information about this enhancement.

RLE Compression Format

Oracle Multimedia DICOM now supports the writing of DICOM images with the RLE compression format. See [Section D.1.3](#) and [Table D-2](#) for more information about this enhancement.

YBR Photometric Interpretation

Oracle Multimedia DICOM now supports YBR photometric interpretation. See [Table D-1](#) for more information about this enhancement.

Encoding of Multibit Monochrome Raw Content

Oracle Multimedia DICOM now supports the encoding of multibit monochrome raw content. See [Table D-2](#) for more information about this enhancement.

Oracle Data Pump Utilities Support for the DICOM Data Model Repository

Oracle Data Pump Utilities support is now available for the DICOM data model repository. See [Section 9.9](#) for guidelines and examples that use the Oracle Data Pump commands `expdp` and `impdp` to perform export and import operations on the DICOM data model repository.

See Also:

Oracle Database Utilities for more information about the Oracle Data Pump Export and Import utilities

Part I

DICOM Common Usage and Reference

This part contains introductory and conceptual information. It also contains user and reference information that is common to administrators and developers of DICOM applications.

Part I includes these chapters:

- [Chapter 1, "Introduction to Oracle Multimedia DICOM"](#)
- [Chapter 2, "Oracle Multimedia DICOM Concepts"](#)
- [Chapter 3, "Overview of DICOM Development"](#)
- [Chapter 4, "DICOM Data Model Utility Reference"](#)

Introduction to Oracle Multimedia DICOM

This chapter contains background information about medical imaging, and provides a general introduction to the Oracle Multimedia DICOM (formerly Oracle *interMedia* DICOM) feature.

This chapter includes these sections:

- [Medical Imaging and Communication](#) on page 1-1
- [Oracle Multimedia and DICOM](#) on page 1-3

1.1 Medical Imaging and Communication

Digital Imaging and Communications in Medicine (DICOM) is a medical imaging standard that was initiated by the American College of Radiology (ACR) and the National Electrical Manufacturers Association (NEMA) to enhance the connectivity of radiological devices. Before the **DICOM standard** became widely adopted, each manufacturer had its own proprietary image format and communication protocol. This proliferation of formats and protocols made it almost impossible to produce third-party software to manage or study medical content, or to connect hardware devices from different manufacturers.

For the most part, the DICOM standard is developed by volunteers. Working groups formed by domain experts propose additions and changes to the existing standards, and the changes are approved by a balloting process. Typically, the National Electrical Manufacturers Association publishes a new version of the standard each year, and makes it available worldwide on its Web site. In addition, the Integrating the Healthcare Enterprise (IHE) initiative provides information about issues related to DICOM content and communication, and makes this information available on its Web site.

The following subsections provide introductory information about DICOM:

- [History of the DICOM Standard](#)
- [Overview of DICOM Content](#)

See Also:

- <http://medical.nema.org/> for more information about the DICOM standard
- <http://www.ihe.net/> for more information about DICOM content and communication
- <http://www.iso.org/> for more information about the OSI Model and other communication protocols

1.1.1 History of the DICOM Standard

In 1985, the American College of Radiology and the National Electrical Manufacturers Association jointly published the ACR-NEMA medical imaging and communication standard to address the proliferation problem. In 1993, the ACR-NEMA standard was revised and renamed as the DICOM standard (Version 3.0). Since the release of Version 3.0, the DICOM standard has become the dominant standard for radiology imaging and communication. All major manufacturers now conform to the DICOM standard.

Today, any software component can take **DICOM content** from any manufacturer and manage that content with a uniform interface. The term DICOM content refers to standalone DICOM Information Objects that are encoded according to the data structure and encoding definitions of PS 3.10 of the DICOM standard (commonly referred to as **DICOM Part 10 files**). (In the DICOM standard, the phrase *DICOM objects* refers to DICOM content.)

The DICOM standard has two major areas of focus: the file format and the communication protocol. Content such as images and waveforms captured by medical devices is represented as information objects. Services such as get, find, and store operations can be defined on these information objects. The combination of services and information objects is a **service object pair (SOP)**.

The DICOM standard (Part 5 and Part 6) defines different types of **transfer syntax** (or binary encoding rules) for sending objects across the network or encoding objects in files. Transfer syntax specifies the mapping of a DICOM object hierarchy into a binary stream.

The binary data can be stored on physical media such as tapes, CDs, or disks, and organized in accordance with the DICOM file hierarchy. It can also be exchanged over a network with the DICOM communication protocol. The DICOM communication protocol covers the upper three layers (Application, Presentation, and Session) of the Open Systems Interconnection (OSI) Seven Layer Model. The DICOM protocol is typically implemented on top of TCP/IP.

Note: The Oracle Multimedia DICOM feature deals with the storage, management, and manipulation of DICOM format medical images and other objects encoded into files. Oracle Multimedia does not support the DICOM communication protocol.

Messages exchanged between a DICOM server and a DICOM object involve operations such as these:

- Radiology workflow
- Grayscale image rendering
- Image printing
- Storage and retrieval

Recently, the DICOM standard introduced Web access to DICOM objects (WADO). WADO deals primarily with HTTP access to DICOM objects.

1.1.2 Overview of DICOM Content

DICOM content can include many different types of data, such as the following:

- Patient administration information

- Waveforms
- Images
- Slices of 3-D volumes
- Video segments
- Diagnostic reports
- Graphics
- Text annotations

DICOM content also contains standard attributes and private attributes. Standard attributes are defined and published in the DICOM standard. Private attributes are defined by and specific to private organizations, such as manufacturers and other enterprises. The DICOM data dictionary provides the definitions for DICOM standard and private attributes.

1.2 Oracle Multimedia and DICOM

The Digital Imaging and Communications in Medicine (DICOM) feature was first introduced to Oracle Multimedia in Oracle Database 10g Release 2 (10.2). For that release, Oracle Multimedia DICOM enhanced the previous behavior of the Oracle Multimedia `ORDImage` object type by enabling Oracle Multimedia to recognize DICOM content and extract a subset of embedded DICOM attributes relating to patient, study, and series.

Oracle Database 11g Release 1 (11.1) continued to provide the same DICOM support in `ORDImage`, while introducing more complete DICOM support in a new **ORDDicom** object type. See [Appendix F](#) for information about migrating from DICOM support (in `ORDImage` objects) to the DICOM support that was introduced in Oracle Database 11g Release 1 (11.1).

The following subsections briefly introduce the support provided by the DICOM feature in Oracle Database 11g Release 2 (11.2):

- [Oracle Multimedia DICOM Format Support](#)
- [ORDDicom Object Type](#)
- [DICOM Metadata Extraction](#)
- [DICOM Conformance Validation](#)
- [DICOM Image Processing](#)
- [Making Confidential Data in DICOM Content Anonymous](#)
- [Creating ORDDicom Objects from Images or Video and Metadata](#)
- [Run-Time, Updatable DICOM Data Model](#)

See Also:

Oracle Multimedia User's Guide for information about DICOM support for the `ORDImage` object type

1.2.1 Oracle Multimedia DICOM Format Support

With Oracle Database 11g Release 1 (11.1), Oracle Multimedia provided full support for the DICOM file format, which is universally recognized as the standard for

medical imaging. Applications can now use Oracle Multimedia DICOM Java and PL/SQL APIs to store, manage, and manipulate DICOM content.

Customers can build large archives of medical content that are managed and secured using Oracle Database. Complete DICOM metadata support enables customers to index and search the archived DICOM content for research purposes. Central storage of DICOM content makes telemedicine practical. Incorporating DICOM content in a database enables customers to build electronic healthcare records applications, using application development tools from Oracle or others.

1.2.2 ORDDicom Object Type

A new Oracle Multimedia object type, ORDDicom, natively supports DICOM content produced by medical devices. This object type holds the DICOM content and extracted metadata, and implements the methods to manipulate the DICOM content. A new Java proxy class, OrdDicom, provides access to the ORDDicom database object through JDBC in a Java application. For applications that store DICOM content directly in BLOBs or BFILEs, a relational interface is provided as a PL/SQL package (ORD_DICOM).

By presenting DICOM content stored in a database as objects, Oracle enables both rapid application development and easy, secure management of large archives of DICOM content.

1.2.3 DICOM Metadata Extraction

Oracle Database 10g Release 2 (10.2) provided support for extracting the most important metadata as DICOM attribute tags into an XML document, and then indexing and searching these tags to find DICOM content that matched certain conditions. Oracle Database 11g Release 1 (11.1) extended that capability by supporting complete and extensible metadata extraction. Customers can extract metadata according to an Oracle-specified XML schema, or create and use their own schema definition to extract subsets of the standard DICOM attribute tags or private tags. The extracted metadata can then be stored in a table to facilitate DICOM content searching based on standard or private DICOM attributes.

This enhanced metadata extraction capability enables customers to build large archives of DICOM content. By customizing extracted XML metadata documents, customers can create highly specialized indexes to DICOM content based on standard and private DICOM attribute tags.

1.2.4 DICOM Conformance Validation

Oracle Database 11g Release 1 (11.1) provided support for **conformance validation**, enabling customers to verify that DICOM content adheres to a set of user-specified constraint rules.

DICOM content is generated by many devices. While most conform to the DICOM standard, some do not. It is important to be able to identify DICOM content that does not conform to the standard, or to the constraint rules for a particular organization or enterprise. Validating DICOM content for conformance can ensure the consistency of a DICOM archive. It enables a database to accept DICOM content from multiple sources and verify the integrity of the content.

1.2.5 DICOM Image Processing

Oracle Database 11g Release 1 (11.1) added methods and functions to copy and process DICOM content into DICOM content or other formats (for example: JPEG, GIF, PNG, and TIFF), and to copy and generate scaled versions and thumbnail images. In addition, that release provided a set of optional methods and functions to copy and process (for example: compress, scale, rotate, and crop) image content, during the conversion process.

Oracle Database 11g Release 2 (11.2) adds functions to copy and process DICOM content into video formats. Oracle Multimedia supports the processing of multiframe DICOM content to the Microsoft Video for Windows Audio Video Interleave (AVI) format. You can generate output in AVI format from multiframe DICOM content such as MRIs, CTs, and Ultrasound videos (see [Section D.5](#)).

To view medical images stored in the DICOM format in Web applications, you must create a copy of the images in formats that are compatible with the browsers that are currently used in the industry. Oracle Database 11g Release 1 (11.1) enabled customers to automatically copy, reformat, and deliver DICOM images to applications that require popular industry-standard image formats such as JPEG.

1.2.6 Making Confidential Data in DICOM Content Anonymous

Oracle Database 11g Release 1 (11.1) added a method that makes new ORDDicom objects with DICOM content and extracted XML attributes anonymous, in accordance with the rules specified by an anonymity document. The anonymity document defines both the set of attributes to be made anonymous and the actions to make them anonymous.

This method can be used to generate new, anonymous ORDDicom objects, ensuring that users of a DICOM medical archive see only the DICOM content and metadata that they are authorized to see. For example, clinicians require full access to DICOM content and metadata for each patient they are treating. They must be able to view all the DICOM metadata included in DICOM content. Researchers, however, require only partial access to the same DICOM metadata for patients participating in a study. Patient privacy regulations require that this class of users not be permitted to view attributes and metadata included in ORDDicom objects that contain personally identifying information.

By providing anonymity services in the database, Oracle Database enables appropriate access for different classes of users of a DICOM medical archive.

1.2.7 Creating ORDDicom Objects from Images or Video and Metadata

Oracle Database 11g Release 1 (11.1) included the ability to generate new ORDDicom objects by combining digital images of various formats (for example: DICOM, JPEG, RAW, TIFF, and GIF) with an XML representation of the associated DICOM metadata. Oracle Database 11g Release 2 (11.2) supports the ability to generate new ORDDicom objects by combining digital video of MPEG format with an XML representation of the associated DICOM metadata. This operation results in well-formed and validated ORDDicom objects, which can be stored in a table in the database or delivered to a DICOM viewer. This feature is particularly useful for generating DICOM secondary capture images and video (see [Section 3.2.7](#)).

Storing and retrieving film-based medical images is expensive and prone to loss. Replacing film-based medical images with DICOM images reduces storage and retrieval costs, and reduces the risk of loss. Storing scanned images and digital video

with their metadata in DICOM format can make non-DICOM images and video more useful.

New DICOM content can also be generated to correct metadata errors in the original DICOM content.

1.2.8 Run-Time, Updatable DICOM Data Model

A key feature of DICOM support provided in Oracle Database 11g Release 1 (11.1) is that its run-time behavior is determined by a set of user-configurable documents. This set of documents is collectively managed by the data model repository. Administrators can update this data model repository to configure Oracle Multimedia DICOM for a particular database instance.

Hospitals must always be up and running. They cannot shut down the system for any of the following reasons:

- To update to a new version of the DICOM standard
- To incorporate private DICOM attribute tags for a new piece of equipment
- To change their DICOM conformance rules
- To modify the set of DICOM attribute tags they extract from each ORDDicom object or to change the XML encoding of the extracted attributes
- To modify their DICOM anonymity rules

This design enables customers to upgrade Oracle Multimedia DICOM at any time, without interfering with a running DICOM archive.

Oracle Multimedia DICOM Concepts

This chapter describes Oracle Multimedia DICOM at a conceptual level.

This chapter includes these sections:

- [Oracle Multimedia DICOM Architecture](#) on page 2-1
- [Oracle Multimedia DICOM Storage](#) on page 2-3
- [Model-Driven Design](#) on page 2-5
- [DICOM Data Model Repository](#) on page 2-6
- [Extraction of Metadata from DICOM Content](#) on page 2-10
- [Validation of DICOM Content](#) on page 2-14
- [Image Conversion and Creation of New DICOM Content](#) on page 2-15
- [Making DICOM Content Anonymous](#) on page 2-16
- [Extraction of DICOM Metadata for Partitioning](#) on page 2-18

2.1 Oracle Multimedia DICOM Architecture

Oracle Multimedia DICOM enables Oracle Database to store, manage, and retrieve DICOM content such as single-frame and multiframe images, waveforms, slices of 3-D volumes, video segments, and structured reports.

The Oracle Multimedia DICOM architecture defines the framework (see [Figure 2-1](#)) through which DICOM content is supported in the database. This DICOM content 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 database that supports thousands of users.

[Figure 2-1](#) shows the Oracle Multimedia DICOM architecture from a two-tier perspective:

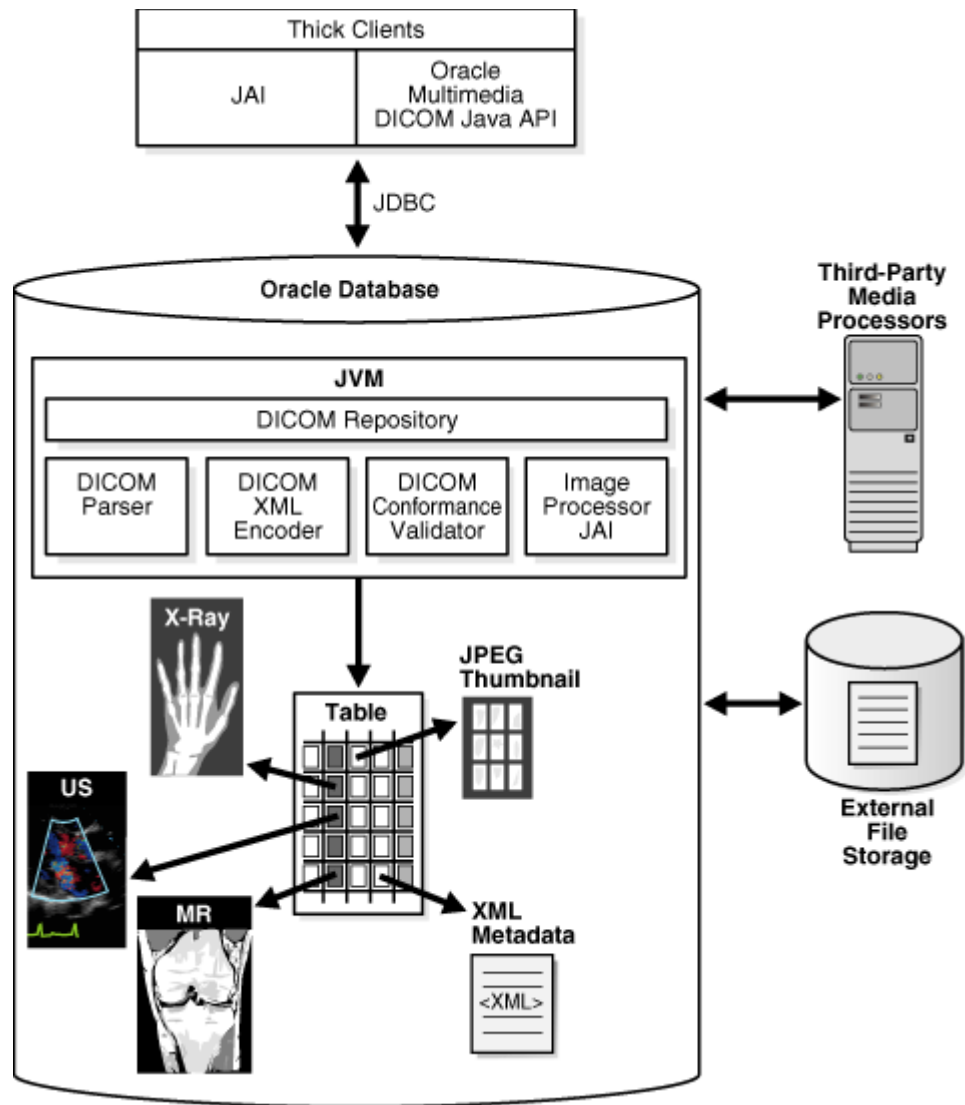
1. Database tier (Oracle Database)
2. Client tier (Thick Clients)

In the database tier, using Oracle Multimedia DICOM, Oracle Database holds DICOM content in tables. As illustrated, DICOM content stored in a column of a table can include DICOM data such as X-rays, ultrasound images, and magnetic resonance images. In the table, a separate column stores a JPEG thumbnail image of the DICOM image. Another column stores the XML metadata documents associated with each image. Within a Java Virtual Machine (JVM), there is a server-side DICOM data model repository and a [DICOM parser](#), a [DICOM XML encoder](#), a [DICOM conformance](#)

validator, and an **image processor**. The DICOM parser extracts metadata from DICOM content. The DICOM XML encoder maps the extracted DICOM attributes into an XML document, in accordance with the mapping rules defined in the **data model repository**. The DICOM conformance validator checks the syntactical and semantic consistency of DICOM content in accordance with the constraint rules specified in the data model repository. The image processor includes Java Advanced Imaging (JAI), and provides image processing for operations such as producing thumbnail-size images and converting between DICOM and other supported image formats. Using Oracle Multimedia DICOM methods enables import and export operations between the database and external file storage systems. The double-sided arrow connecting Oracle Database with External File Storage in [Figure 2-1](#) shows this type of data communication. Through JDBC calls, thick clients can access the content stored in an Oracle database and perform procedures such as image processing outside of the database. Other means of interacting with Oracle Database, such as the OCI call interface, can also be used to access DICOM content stored in an Oracle database. These types of data access can also be used to integrate Oracle Database with third-party media processors.

In the client tier, the ability to access ORDDicom objects in the database is supported through Oracle Multimedia DICOM Java API. Oracle Multimedia DICOM Java API supplies direct access to ORDDicom objects from Java applications.

Figure 2-1 Oracle Multimedia DICOM Architecture

**See Also:**

- *Oracle Multimedia User's Guide* for a view and description of the complete architecture for Oracle Multimedia
- *Oracle Multimedia DICOM Java API Reference* for information about using Oracle Multimedia DICOM with the Java programming language
- <http://java.sun.com/> for more information about Java Advanced Imaging (JAI)

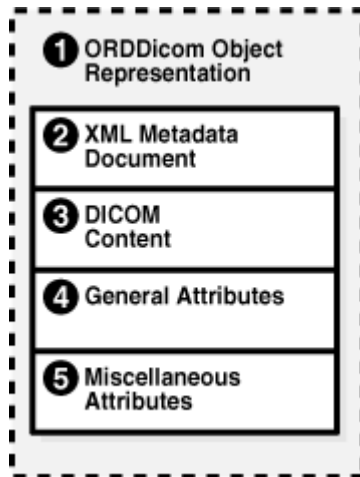
2.2 Oracle Multimedia DICOM Storage

When using the object interface, you must create an ORDDicom object in a table before you can perform Oracle Multimedia DICOM operations on DICOM content. Oracle Multimedia defines the ORDDicom object type, which is similar to a Java or C++ class, to contain DICOM content.

Figure 2–2 shows an ORDDicom object at a very high level. Items in Figure 2–2 are numbered to help identify the items in this description. An instance of an ORDDicom object type (Item 1) consists of methods and attributes. Methods are functions or procedures that can be performed on the ORDDicom object, such as `makeAnonymous()` and `setProperties()`. The attributes include the following:

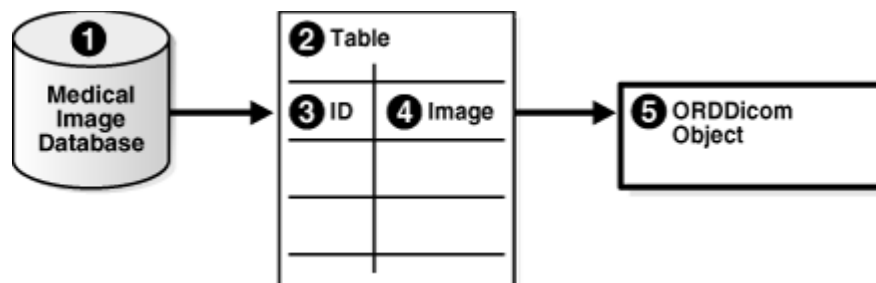
- Extracted DICOM attributes represented as an XML metadata document (Item 2)
- The DICOM content (Item 3), which is the original DICOM content in unmodified form stored within the database, under transaction control as a BLOB (recommended), or stored in an operating system-specific file in a local file system with pointers stored in the database
- Certain frequently accessed general attributes (Item 4), such as SOP Class UID, which are extracted and stored for ease of access and indexing
- Miscellaneous attributes (Item 5) that are meant for Oracle internal use

Figure 2–2 ORDDicom Object



Similar to the NUMBER or BLOB data types, you can use the ORDDicom data type as the data type of a table column.

Figure 2–3 shows the structure of a simple table in a medical database that contains an ORDDicom object. Items in Figure 2–3 are numbered to help identify the items in this description. Item 1 represents the medical image database. Item 2 represents a simple medical image table managed by the database. This table contains two columns: ID (Item 3) and Image (Item 4). Item 3 represents the identifier for a specified DICOM image in the database. Item 4 represents the DICOM content in the database, which can be stored as an ORDDicom object (Item 5). Thus, the column type for the Image column is ORDDicom.

Figure 2-3 Table in a Medical Image Database

2.3 Model-Driven Design

Oracle Multimedia DICOM is designed with a model-driven software architecture. Thus, the run-time behavior of Oracle Multimedia DICOM is controlled by a domain-specific data model. The DICOM data model is a collection of XML documents that are managed in the data model repository. DICOM administrators can manage and modify the DICOM data model. The XML documents that comprise the data model can be inserted and deleted at run time when multiple user sessions are accessing the data model. Changes to the data model are protected with database transaction semantics, and each user session can refresh to the latest data model when necessary or desired.

Figure 2-4 illustrates the principles of model-driven software architecture using the DICOM metadata extraction feature. The items above the dotted line show the portions of the data model that are related to the metadata extraction feature. The items below the dotted line show the software run-time components of the extract metadata feature that access the data model. A line that connects an item of the data model and an item of the run-time component shows the run-time access to the corresponding item that is managed by the data model repository.

At design time, DICOM administrators can change the DICOM data model. Publishing these changes affects the run-time behavior of the extract metadata feature, and other DICOM operations.

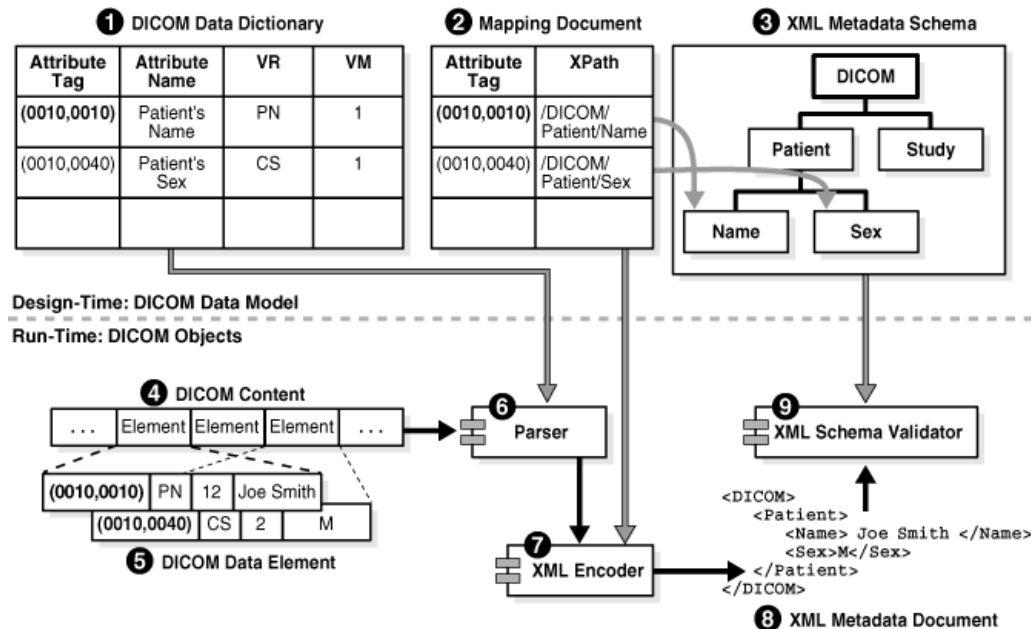
Figure 2-4 shows three components of the data model. Items in Figure 2-4 are numbered to help identify the items in this description. Item 1 represents the DICOM data dictionary, which provides the definitions for DICOM standard and private attributes. Item 2 represents a mapping document, which describes how an attribute is to be mapped into an XML document. Item 3 represents a sample DICOM XML metadata schema, which defines the structure and data type of an XML document that is used to store DICOM attributes. The lines connecting elements of Item 2 and Item 3 show the mapping between a DICOM attribute stored in DICOM content and an XML element stored in an XML document that conforms to the XML schema.

The process of converting DICOM content (Item 4) into an XML metadata document (Item 8) is shown in the bottom half of Figure 2-4. Solid lines connecting items show the flow of data between run-time components. Two of the attributes of the sample DICOM content are shown in Item 5. The first attribute contains the attribute tag (0010, 0010), the data type PN, the length in bytes 12, and the value Joe Smith. This attribute is encoded in the DICOM content, although its data type might not be encoded in the DICOM content. The parser (Item 6) can find an attribute definition by looking it up in the DICOM data dictionary (Item 1) using the attribute tag (0010, 0010). The attribute definition determines the interpretation of the DICOM content. The result is passed to an XML encoder (Item 7). Similarly, the XML encoder looks up the data model (Item 2) to find the XML encoding guidelines for the attribute,

and produces an XML document accordingly. Finally, an XML schema validator (Item 9) can validate the generated document against the XML metadata schema (Item 3).

In [Figure 2-4](#), everything that controls the run-time behavior is part of the data model, which can be configured by a DICOM administrator.

Figure 2-4 DICOM Model-Based Parsing Details



2.4 DICOM Data Model Repository

A key feature of Oracle Multimedia DICOM is that its run-time behavior is determined by a set of user-configurable documents (a data model). This set of documents is managed collectively in the data model repository. Administrators can update the data model repository to configure Oracle Multimedia DICOM for a particular database instance. With this design, customers can perform tasks such as upgrading Oracle Multimedia DICOM to a new version of the DICOM standard or adding new conformance validation rules at any time, without interfering with a running DICOM archive. Each database has its own set of configuration documents. Each organization or enterprise can customize the installed configuration documents according to its needs.

The following subsections describe these DICOM data model repository concepts:

- [Configuration Documents in the Repository](#)
- [Administrator and User Sessions in the Repository](#)

2.4.1 Configuration Documents in the Repository

The set of configuration documents that comprises the data model repository includes anonymity documents, constraint documents, mapping documents, preference documents, private and standard dictionary documents, stored tag list documents, and UID definition documents. Each **configuration document** comes with an XML schema definition. Other documents can be added to the repository as needed.

Oracle ships a set of default configuration documents with each software release. All schemas corresponding to the default documents are registered during installation. All schemas are fixed and must not be modified for a database installation.

[Table 2–1](#) lists the document type, the default XML document name, and the XML schema definition name for each type of document in the data model repository.

Table 2–1 Configuration Documents and Their XML Schemas

Document Type	Default XML Document	XML Schema Definition
Anonymity	ordcman.xml	ordcman.xsd (See Example B–1)
Constraint	ordcmct.xml	ordcmct.xsd (See Example B–2)
	ordcmcmd.xml	
	ordcmcmc.xml	
Mapping	ordcmap.xml	ordcmap.xsd (See Example B–6)
Preference	ordcmpf.xml	ordcmpf.xsd (See Example B–8)
Private Dictionary	ordcmpv.xml	ordcmpv.xsd (See Example B–9)
Standard Dictionary	ordcmsd.xml	ordcmsd.xsd (See Example B–10)
Stored Tag List	None	ordcmstl.xsd (See Example B–11)
UID Definition	ordcmui.xml	ordcmui.xsd (See Example B–12)

An **anonymity document** is an XML document that specifies the set of attributes to be made anonymous, and defines the actions required to make those attributes anonymous. The default anonymity document, `ordcman.xml`, lists a subset of the attributes defined in the Basic Application Level Confidentiality Profile in Part 15 of the DICOM standard.

A **constraint document** is an XML document that defines a collection of rules that check the conformance of DICOM content, according to the DICOM standard. The constraint document specifies attribute relationships and semantic constraints that cannot be expressed by the DICOM metadata schema. The default constraint documents, `ordcmct.xml`, `ordcmcmd.xml`, and `ordcmcmc.xml`, show a sample set of validation rules defined in accordance with a subset of Part 3 of the DICOM standard.

A **mapping document** is an XML document that defines how each attribute maps to a particular element in a DICOM XML metadata document. This document determines the structure of the extracted XML representation of the DICOM metadata. The default mapping document, `ordcmap.xml`, defines the mapping from DICOM content to XML, which is represented as a flat list where all XML encoded DICOM attributes are included under the root element `<DICOM_OBJECT>`.

A **preference document** is an XML document that defines run-time parameters, such as the size limits for DICOM attributes to omit when encoding to XML or whether to validate XML documents used in DICOM functions and procedures. The default preference document is `ordcmpf.xml`.

A **private dictionary document** is an XML document that enables users to extend the standard dictionary document definitions by adding manufacturer-specific or enterprise-specific attributes to DICOM content. The default private dictionary document, `ordcmpv.xml`, defines Oracle private attributes.

A **standard dictionary document** is an XML document that lists the standard attributes defined in Part 6 of the DICOM standard, and that can be used to reflect

updates to the DICOM standard. The default standard dictionary document is `ordcmsd.xml`.

A **stored tag list document** is an optional XML document that specifies the DICOM attributes to be extracted from the embedded DICOM content and stored in the XML metadata attribute of the ORDDicom object when the `setProperties()` method is called. Generally, stored tag list documents contain the attribute tags used in mapping and constraint documents.

A **UID definition document** is an XML document that lists unique identifiers (UIDs) defined by the DICOM standard or a private organization. The UID is based on an ISO object identifier (OID). Rather than defining the DICOM content UIDs, the UID definition document contains a registry of standard UIDs that classify the DICOM content and express standard semantics. The default UID definition document, `ordcmui.xml`, lists UIDs defined by Part 6 of the DICOM standard.

See [Appendix A](#) and [Appendix B](#), respectively, for more information about the installed configuration documents and their related XML schema definitions.

2.4.2 Administrator and User Sessions in the Repository

Administrators can manage configuration documents using the data model repository interface.

The data model repository must be loaded before any Oracle Multimedia DICOM methods, procedures, or functions are invoked. Loading the repository is accomplished through a DICOM package interface, using the `setDataModel()` procedure.

[Figure 2–5](#) uses a Unified Modeling Language (UML) sequence diagram to show the state of the DICOM data model repository in its installed state and in various states after being updated. Also shown are two administrator sessions and two user sessions working with one data model repository. The numbered items in [Figure 2–5](#) represent various components of, or operations on, the data model repository. The numbered items in the following list correspond to the numbered items in [Figure 2–5](#), respectively.

Data Model States:

In Item 10, all the boxes in this column represent the data model repository in the following states:

- **State 0:** the installed version.
- **State 1:** the version that includes updates from the **Admin Session 1** editing session **XG1**.
- **State 2:** the version that includes updates from the **Admin Session 1** editing session **XG1** and the **Admin Session 2** editing session **XG2**.
- **State 3:** the version that includes updates from the **Admin Session 1** editing sessions **XG1** and **XG3** and updates from the **Admin Session 2** editing session **XG2**.

Administrator Sessions:

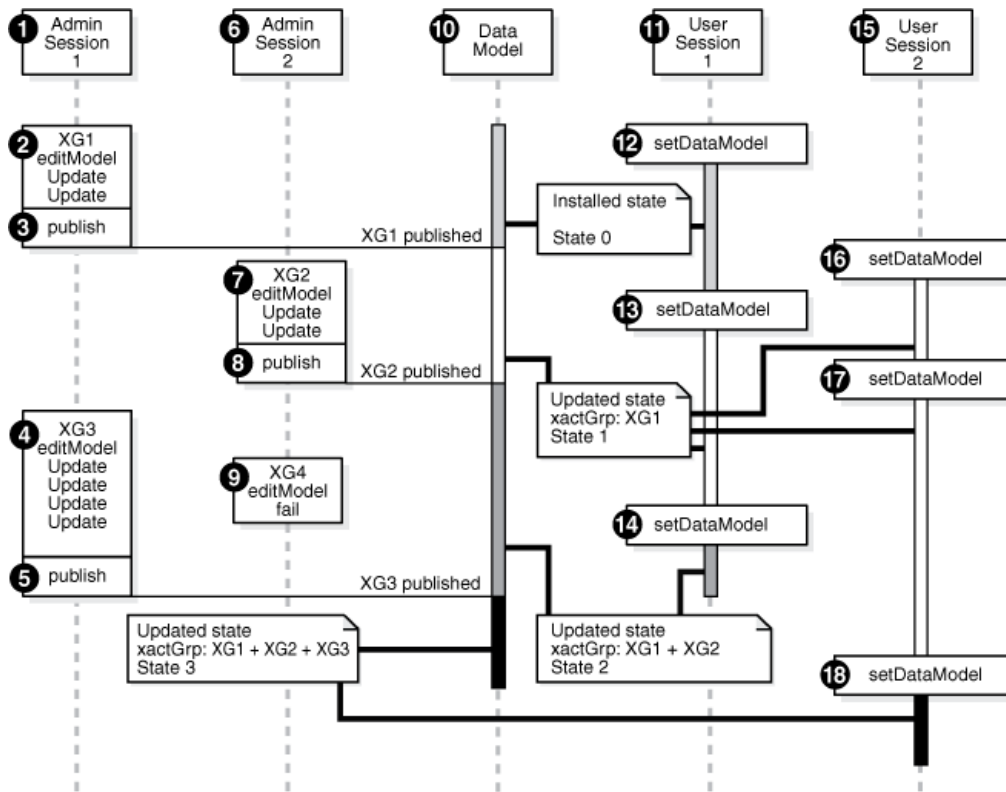
- Item 1: All boxes in this column represent tasks performed by **Admin Session 1**.
- Item 6: All boxes in this column represent tasks performed by **Admin Session 2**.
- Item 2: **Admin Session 1** calls the `editDataModel()` procedure to begin editing session **XG1**. This locks the installed version of the data model (**State 0**) and

prevents other administrators from editing the data model. During this time, users can view only the installed version of the data model (**State 0**). **Admin Session 1** edits the data model.

- Item 3: **Admin Session 1** completes editing session **XG1** and calls the `publishDataModel()` procedure to publish the changes to the data model. The data model is updated to **State 1** and the lock is released. Other administrators can now lock the data model for editing. And, other users can now view the updated data model by calling the `setDataModel()` procedure.
- Item 7: **Admin Session 2** calls the `editDataModel()` procedure to begin editing session **XG2**. This locks the data model (**State 1**) and prevents other administrators from editing the data model. During this time, users can view **State 1** of the data model. **Admin Session 2** edits the data model.
- Item 8: **Admin Session 2** completes editing session **XG2** and calls the `publishDataModel()` procedure to publish the changes to the data model. The data model is updated to **State 2** and the lock is released.
- Item 4: **Admin Session 1** calls the `editDataModel()` procedure to begin editing session **XG3**. This locks the data model (**State 2**) and prevents other administrators from editing the data model. During this time, users can view **State 2** of the data model. **Admin Session 1** edits the data model.
- Item 9: **Admin Session 2** calls the `editDataModel()` procedure to begin editing session **XG4**. Because **Admin Session 1** has locked the data model, **Admin Session 2** cannot obtain the lock, and the call to the `editDataModel()` procedure fails.
- Item 5: **Admin Session 1** completes editing session **XG3** and calls the `publishDataModel()` procedure to publish the changes to the data model. The data model is updated to **State 3** and the lock is released.

User Sessions:

- Item 11: All boxes in this column represent tasks performed by **User Session 1**.
- Item 15: All boxes in this column represent tasks performed by **User Session 2**.
- Item 12: **User Session 1** calls the `setDataModel()` procedure to load the data model. The data model is still at the installed version (**State 0**) because **Admin Session 1** has not yet published the changes for editing session **XG1**.
- Item 16: **User Session 2** calls the `setDataModel()` procedure to load the data model. The data model is at **State 1**, which reflects the published changes from editing session **XG1**.
- Item 13: **User Session 1** calls the `setDataModel()` procedure again. The data model is now at **State 1**, which reflects the published changes from editing session **XG1**.
- Item 17: **User Session 2** calls the `setDataModel()` procedure again. The data model is still at **State 1** because **Admin Session 2** has not yet published the changes for editing session **XG2**.
- Item 14: **User Session 1** calls the `setDataModel()` procedure again. The data model is now at **State 2**, which reflects the published changes from editing sessions **XG1** and **XG2**.
- Item 18: **User Session 2** calls the `setDataModel()` procedure again. The data model is now at **State 3**, which reflects the published changes from editing sessions **XG1**, **XG2**, and **XG3**.

Figure 2–5 DICOM Data Model Repository

As shown in [Figure 2–5](#), the `setDataModel()` procedure is invoked during user sessions. Applications must call this procedure at the beginning of each database session to load the repository from the database into memory structures. This procedure can also be called whenever the application requires the new data model changes. This procedure is available to users through the DICOM data model utility in the `ORD_DICOM` package interface.

See [Chapter 12](#) for more information about the data model repository administration interface. See [Chapter 4](#) for more information about the DICOM data model utility in the `ORD_DICOM` package interface.

2.5 Extraction of Metadata from DICOM Content

Extracting metadata from DICOM content involves several operations using an XML metadata schema and a mapping document.

A **DICOM metadata document** is an XML document that contains the metadata extracted from DICOM content. Optionally, each metadata document can be constrained by an XML schema. Each XML metadata schema has a matching XML mapping document.

The mapping of DICOM content to the DICOM metadata document is defined by a mapping document. The mapping document defines how attributes from the DICOM content are to be mapped into an XML document that conforms to the schema. Like other configuration documents, mapping documents are managed by the DICOM data model repository (see [Section 2.4](#)).

Oracle provides a default XML metadata schema (`ordcmmd.xsd`) and a matching XML mapping document (`ordcmmmp.xml`). Application designers who create their own metadata schemas must ensure that their schema definition and mapping

documents are compatible. They must also ensure that their data type definitions are compatible with the Oracle data type definitions (`ordcmddt.xsd`).

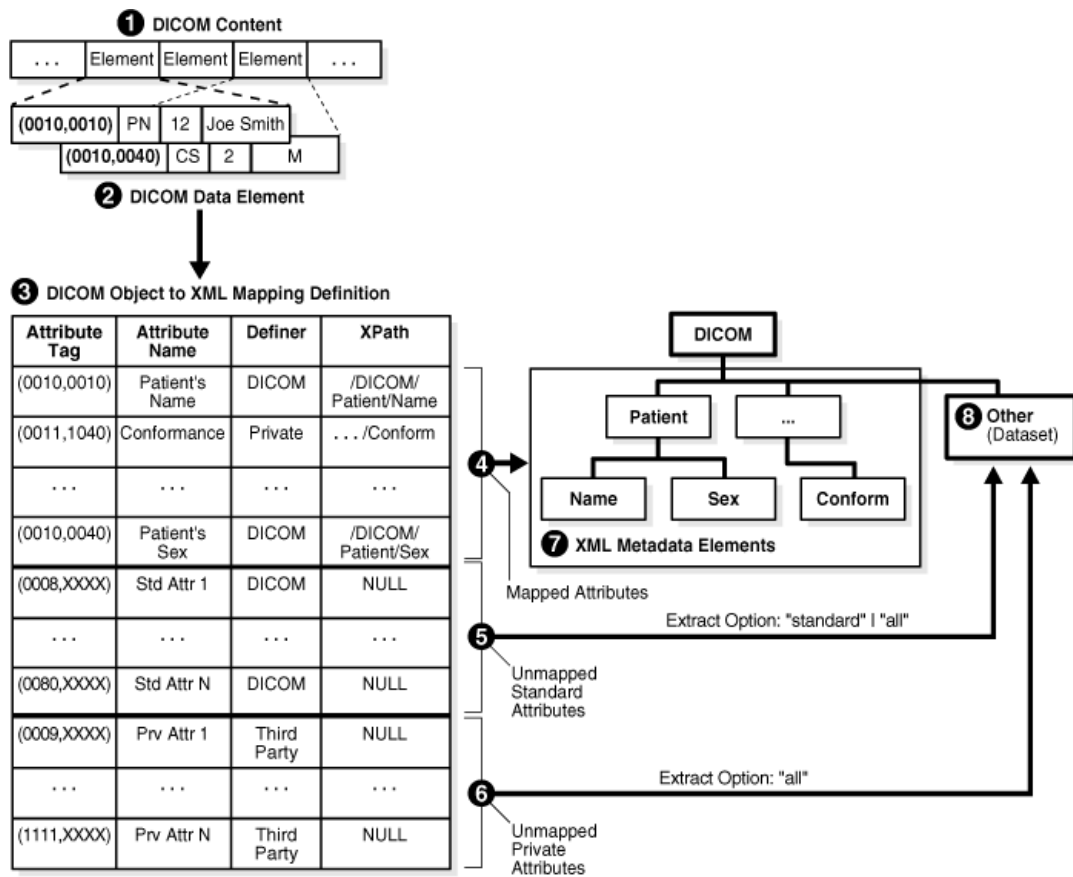
The following subsections describe these key extraction and mapping concepts:

- [Overview of the Metadata Extraction and XML Mapping Process](#)
- [Sample XML Documents Used in the Extraction and Mapping Process](#)

2.5.1 Overview of the Metadata Extraction and XML Mapping Process

Figure 2–6 shows the components involved in the metadata extraction and XML mapping process. Each numbered item in Figure 2–6 represents a component in this process.

Figure 2–6 DICOM Metadata Extraction and XML Mapping



The input is DICOM content (Item 1) in binary format, which can be stored in an ORDDicom object, or directly in a BLOB or a BFILE. The output is an XML metadata document (Items 7 and 8). The layout of the metadata document is specified by the mapping document (Item 3). Metadata extraction uses a mapping option parameter that specifies which group of attributes to include in the output metadata document. The solid lines connecting items (Items 4, 5, and 6) show the flow of data from the DICOM content to the XML metadata document. Item 3 can also be interpreted as a processing engine that performs metadata extraction according to the specifications of the mapping document.

The DICOM content (Item 1) encodes a DICOM data element (Item 2) in binary code. At run time, the parser reads the binary stream of DICOM content, and builds a

representation of the DICOM data element (Item 2) in memory. To map the in-memory representation of each DICOM attribute into an XML element, the XML encoder looks up the definition of the attribute in the mapping document (Item 3) that is stored in the data model repository.

For example, the first entry of this mapping document maps the DICOM attribute `Patient's Name (0010,0010)` to the XML path `/DICOM/Patient/Name`, where `Name` is a subelement of the `Patient` element, and `Patient` is the child element of the document root element `DICOM`.

Attributes that are part of a DICOM data element and whose XML paths are explicitly defined in a mapping document are called **mapped attributes**. Attributes that are part of a DICOM data element but whose XML paths are *not* explicitly defined in a mapping document are called **unmapped attributes**.

Based on the mapping document, each DICOM metadata document can contain two sections: a mapped section and an optional, unmapped section. In the mapped section (Item 7), attributes are organized according to a predefined hierarchy. Attributes in the mapped section can be addressed with a fixed XPath query. In the unmapped section (Item 8), attributes are sorted by attribute tag and listed by **value representation**. Attributes in the unmapped section can be addressed by an XPath query of the element tag in the following form:

```
/DICOM/Other/VR_TYPE (tag== 'HHHHHHHH' )
```

In this query, `HHHHHHHH` is the hexadecimal attribute tag, and `/DICOM/Other` is the specified path for the unmapped section. See [Section 10.2.3](#) for information about how to create a your own mapping document.

The mapping option of the extract metadata function specifies which group of attributes to include in the output XML metadata document. The three mapping options are mapped, standard, or all.

If the extract option is `mapped` (Item 4), then only mapped attributes are included in the XML metadata document. This option is useful when the application using the metadata document has a fixed set of required attributes. The resulting metadata document (Item 7) has a well-defined tree structure.

If the extract option is `standard` (Item 5), then all mapped attributes and all unmapped attributes that are defined by the DICOM standard are extracted into the XML metadata document. Private attributes whose mappings are not defined are excluded from the output. This option is useful when an application such as a full-text search can use all standard attributes that are included in the DICOM content.

If the extract option is `all` (Item 6), then all attributes that are included in the DICOM content are extracted and encoded into the XML metadata document. This option provides lossless mapping of DICOM attributes from binary to XML.

Note: Attributes whose binary length exceeds the user-specified limit are not included in the XML metadata document. See [Section 10.2.6](#) for more information about specifying these limits within a preference document.

If the mapping option is `all` or `standard`, then unmapped attributes of the DICOM data element are stored under the XML element `Other` (Item 8). The resulting XML document can be stored in a database table, indexed, and queried using keywords or XPath query statements. To define alternative mapping structures and element names for mapped and unmapped sections, see [Section 10.2.3](#).

2.5.2 Sample XML Documents Used in the Extraction and Mapping Process

Using an XML mapping document to extract mapped metadata, you can generate an XML metadata document. [Example 2–1](#) shows a sample XML mapping document (`sample_map.xml`).

Example 2–1 Sample XML Mapping Document

```
<?xml version="1.0" encoding="UTF-8"?>
<XML_MAPPING_DOCUMENT xmlns="http://xmlns.oracle.com/ord/dicom/mapping_1_0"
  xmlns:dt="http://xmlns.oracle.com/ord/dicom/datatype_1_0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.oracle.com/ord/dicom/mapping_1_0
  http://xmlns.oracle.com/ord/dicom/mapping_1_0">
  <DOCUMENT_HEADER>
    <dt:DOCUMENT_CHANGE_LOG>
      <dt:DOCUMENT_MODIFIER>Dongbai Guo</dt:DOCUMENT_MODIFIER>
      <dt:DOCUMENT_MODIFICATION_DATE>2006-01-13</dt:DOCUMENT_MODIFICATION_DATE>
      <dt:DOCUMENT_VERSION>0.0</dt:DOCUMENT_VERSION>
      <dt:MODIFICATION_COMMENT>Sample mapping document for metadata schema definition
1</dt:MODIFICATION_COMMENT>
    </dt:DOCUMENT_CHANGE_LOG>
  </DOCUMENT_HEADER>

  <NAMESPACE>http://xmlns.oracle.com/ord/dicom/metatest1</NAMESPACE>
  <ROOT_ELEM_TAG>DICOM_OBJECT</ROOT_ELEM_TAG>
  <UNMAPPED_ELEM>OTHER_ATTRIBUTES</UNMAPPED_ELEM>
  <MAPPED_ELEM>KEY_ATTRIBUTES</MAPPED_ELEM>

  <MAPPED_PATH occurs="true" notEmpty="true" writeTag="true" writeDefiner="true"
    writeName="true" writeRawValue="true">
    <ATTRIBUTE_TAG>00020002</ATTRIBUTE_TAG>
    <PATH>MEDIA_STORAGE_SOP_CLASS_UID</PATH>
  </MAPPED_PATH>

  <MAPPED_PATH occurs="true" notEmpty="true">
    <ATTRIBUTE_TAG>00020003</ATTRIBUTE_TAG>
    <PATH>MEDIA_STORAGE_SOP_INSTANCE_UID</PATH>
  </MAPPED_PATH>

  <MAPPED_PATH writeTag="true" writeDefiner="true" writeName="true" writeRawValue="true">
    <ATTRIBUTE_TAG>00100010</ATTRIBUTE_TAG>
    <PATH>PATIENT_NAME</PATH>
  </MAPPED_PATH>
</XML_MAPPING_DOCUMENT>
```

In [Example 2–1](#), the DICOM standard attribute `SOP_CLASS_UID` that has the DICOM attribute tag (0002, 0002) maps to the XML metadata element `MEDIA_STORAGE_SOP_CLASS_UID` at the XML tree location `/DICOM_OBJECT/KEY_ATTRIBUTES/`. Similarly, the DICOM standard attribute `SOP_INSTANCE_UID` (0002, 0003) maps to the XML metadata element `MEDIA_STORAGE_SOP_INSTANCE_UID`. The DICOM standard attribute `study date` (0008, 0020) has not been listed by the XML mapping document. Thus, if it exists in the DICOM content, it appears in the unmapped section of the DICOM metadata document under the XML tree location `(/DICOM_OBJECT/OTHER_ATTRIBUTES)`.

[Example 2–2](#) shows a sample XML metadata document that can be generated by extracting mapped metadata using the XML mapping document shown in [Example 2–1](#).

Example 2–2 Sample XML Metadata Document

```
<?xml version="1.0" encoding="DEC-MCS"?>
```

```

<DICOM_OBJECT xmlns="http://xmlns.oracle.com/ord/dicom/metatest1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.oracle.com/ord/dicom/metatest1
http://xmlns.oracle.com/ord/dicom/metatest1">

<KEY_ATTRIBUTES>

  <MEDIA_STORAGE_SOP_CLASS_UID definer="DICOM" tag="00020002"
  name="Media Storage SOP Class UID">1.2.840.10008.5.1.4.1.1.1</MEDIA_STORAGE_SOP_CLASS_UID>

  <MEDIA_STORAGE_SOP_INSTANCE_UID tag="00020003" definer="DICOM"
  name="media storage SOP instance UID">1.3.6.1.4.1.5962.1.1.10.1.2.20040119072730.12322
  </MEDIA_STORAGE_SOP_INSTANCE_UID>

  <PATIENT_NAME definer="DICOM" tag="00100010" name="Patient's Name">

    <NAME type="unibyte">

      <FAMILY>CompressedSamples</FAMILY>

      <GIVEN>RG2</GIVEN>

    </NAME>

    <VALUE>CompressedSamples^RG2</VALUE>

  </PATIENT_NAME>

</KEY_ATTRIBUTES>

</DICOM_OBJECT>

```

2.6 Validation of DICOM Content

Validating DICOM content involves verifying that the data conforms to a specified set of constraint rules.

There are several advantages of implementing **conformance validation** in the database instead of in the middle tier or the application tier. First, validating DICOM content can ensure the integrity and consistency of archived DICOM content by enabling a database to accept DICOM content from all sources and to check the integrity of that content. With this feature, the database can act as the centralized data store, connecting a variety of DICOM content sources while enforcing enterprise data constraint rules. Large organizations or government branches can establish their own sets of constraint rules that are more or less restrictive than the rules in the DICOM standard, and then enforce conformance with those constraint rules. In new areas, such as life sciences, where the DICOM standard is still being developed, constraint rules can serve as transitional tools to enforce a conventional representation of the DICOM content, thereby simplifying future transition to the DICOM standard. Finally, database systemwide conformance can greatly simplify enterprise (application) integration and data mining.

DICOM constraint documents define one or more constraint rules to check the conformance of DICOM content according to the DICOM standard or the guidelines for a particular organization or enterprise. [Example 2–3](#) shows a sample constraint document (`sample_ct.xml`).

Example 2–3 Sample Constraint Document

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- Copyright (c) 2007, Oracle. All rights reserved.
NAME

```

```

sample_ct.xml - Oracle Multimedia DICOM sample constraint document
-->

<CONFORMANCE_CONSTRAINT_DEFINITION xmlns="http://xmlns.oracle.com/ord/dicom/constraint_1_0"
xmlns:dt="http://xmlns.oracle.com/ord/dicom/datatype_1_0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.oracle.com/ord/dicom/constraint_1_0
http://xmlns.oracle.com/ord/dicom/constraint_1_0">

  <GLOBAL_RULE name="findJoeSmith">
    <PREDICATE>
      <DESCRIPTION>An example to find an object that has (patientName="Joe Smith"
        AND patientSex=="M")</DESCRIPTION>
      <LOGICAL operator="and">
        <PREDICATE>
          <RELATIONAL operator="eq">
            <ATTRIBUTE_TAG>00100010</ATTRIBUTE_TAG>
            <XML_VALUE>
              <dt:PERSON_NAME>
                <dt:NAME>
                  <dt:FAMILY>Smith</dt:FAMILY>
                  <dt:GIVEN>Joe</dt:GIVEN>
                </dt:NAME>
              </dt:PERSON_NAME>
            </XML_VALUE>
          </RELATIONAL>
        </PREDICATE>
        <PREDICATE>
          <RELATIONAL operator="eq">
            <ATTRIBUTE_TAG>00100040</ATTRIBUTE_TAG>
            <XML_VALUE>
              <dt:CODE_STRING>M</dt:CODE_STRING>
            </XML_VALUE>
          </RELATIONAL>
        </PREDICATE>
      </LOGICAL>
    </PREDICATE>
    <ACTION action="log" when="true">Found Joe Smith</ACTION>
  </GLOBAL_RULE>
</CONFORMANCE_CONSTRAINT_DEFINITION>

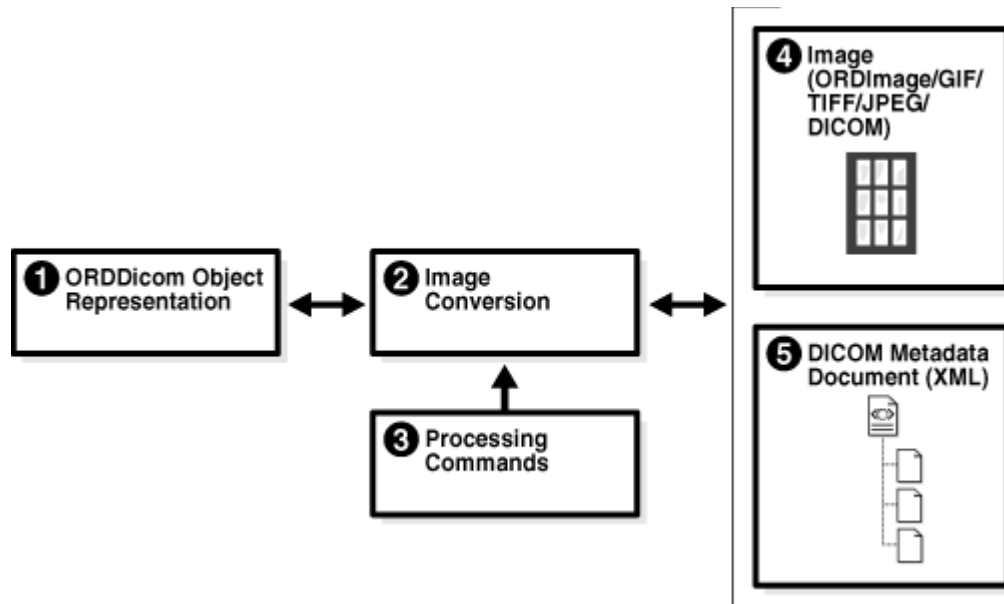
```

After a constraint document has been inserted into the repository, users can validate DICOM content against the global constraint rules defined in the constraint document. (Global constraint rules are defined with the `<GLOBAL_RULE>` tag.) For example, users can check whether the DICOM content conforms to the global constraint rule named `findJoeSmith`, in [Example 2-3](#).

2.7 Image Conversion and Creation of New DICOM Content

ORDDicom objects can be processed and converted to other image formats. In addition, new ORDDicom objects can be created from existing ORDDicom objects. [Figure 2-7](#) shows these operations.

Figure 2–7 Image Conversion Process



The following text describes [Figure 2–7](#) and discusses each of the components. The numbered items in the following text correspond to the numbered items in [Figure 2–7](#). The lines connecting the items in [Figure 2–7](#) show the direction for the flow of data.

The image converter (Item 2) can take DICOM content, specifically a DICOM image (Item 1), and processing commands (Item 3), and convert the DICOM content into an image (Item 4) of another format that is supported by Oracle Multimedia (for example: JPEG or GIF formats) for display in a Web browser or an application.

This process can also be reversed. Using an image such as an ORDImage object storing a JPEG file, TIFF file, or other supported image file (Item 4) and a DICOM metadata document (Item 5), the image converter can merge the two items and produce DICOM content that can be used to create a new ORDDicom object (Item 1). Similarly, the image converter (Item 2) can copy and convert the DICOM content (Item 4) and an XML metadata document (Item 5) into a new ORDDicom object (Item 1). Common uses of this type of process can include lossless compression on the converted image to save disk space, translation into a different type of transfer syntax to enable cross-platform image exchanging, or metadata updating.

When processing, the embedded image content can contain one or more frames. Depending on the processing command for frames, the image converter can read the pixel content of one or all the frames in an image. After the embedded DICOM image is written to an ORDImage object or a BLOB in a format such as GIF or JPEG, you can use existing Oracle Multimedia features to display the converted image on the Web with Oracle Multimedia JavaServer Pages (JSP) tag libraries or other tools.

In addition, multiframe DICOM images, such as MRIs, CTs, and Ultrasound videos, can be processed and converted to the Microsoft Video for Windows Audio Video Interleave (AVI) format. See [Chapter 3](#) and [Appendix D](#) for more information about these and other DICOM processing operations.

2.8 Making DICOM Content Anonymous

Government regulations, such as the Health Insurance Portability and Accountability Act (HIPAA) in the U. S., mandate the protection of confidential data about patients.

Sharing DICOM content with external resources often requires making confidential patient data anonymous. Making DICOM content anonymous in the database avoids exposing confidential patient data outside of the database, simplifying the protection of that information.

The process of making DICOM content anonymous can be customized using the data model repository. Users can create different anonymity documents in XML. Each anonymity document lists a set of attributes to be made anonymous, and the type of actions to be taken to make the attributes anonymous. Supported actions are remove and replace. The remove action is the default action that deletes an attribute or sets it to zero length in the DICOM content and in the ORDDicom object attributes. The replace action replaces an attribute with a string, which can either be empty or contain a user-defined string in the DICOM content and in the ORDDicom object attributes. [Example 2-4](#) shows a sample anonymity document.

Example 2-4 Sample Anonymity Document

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Copyright (c) 2006, 2007, Oracle. All rights reserved. -->
<ANONYMITY_RULE_DOCUMENT xmlns="http://xmlns.oracle.com/ord/dicom/anonymity_1_0"
  xmlns:dt="http://xmlns.oracle.com/ord/dicom/datatype_1_0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.oracle.com/ord/dicom/anonymity_1_0
  http://xmlns.oracle.com/ord/dicom/anonymity_1_0">
  <DOCUMENT_HEADER>
    <dt:DOCUMENT_CHANGE_LOG>
      <dt:DOCUMENT_MODIFIER>Dongbai Guo</dt:DOCUMENT_MODIFIER>
      <dt:DOCUMENT_MODIFICATION_DATE>2006-02-06</dt:DOCUMENT_MODIFICATION_DATE>
      <dt:DOCUMENT_VERSION>0.1</dt:DOCUMENT_VERSION>
      <dt:MODIFICATION_COMMENT>Sample anonymity document</dt:MODIFICATION_COMMENT>
      <dt:BASE_DOCUMENT>Test Document</dt:BASE_DOCUMENT>
      <dt:BASE_DOCUMENT_RELEASE_DATE>2004-01-01</dt:BASE_DOCUMENT_RELEASE_DATE>
      <dt:BASE_DOCUMENT_DESCRIPTION>Same as ordcman.xml from label 070321</dt:BASE_DOCUMENT_DESCRIPTION>
    </dt:DOCUMENT_CHANGE_LOG>
  </DOCUMENT_HEADER>
  <PRIVATE_ATTRIBUTES action="remove"></PRIVATE_ATTRIBUTES>
  <UNDEFINED_STANDARD_ATTRIBUTES action="remove"></UNDEFINED_STANDARD_ATTRIBUTES>
  <UNDEFINED_PRIVATE_ATTRIBUTES action="remove"></UNDEFINED_PRIVATE_ATTRIBUTES>
  <INDIVIDUAL_ATTRIBUTE>
    <ATTRIBUTE_TAG>00100010</ATTRIBUTE_TAG>
    <DESCRIPTION>Patient Name</DESCRIPTION>
    <ANONYMITY_ACTION action="replace">Smith^Joe</ANONYMITY_ACTION>
  </INDIVIDUAL_ATTRIBUTE>
  <INDIVIDUAL_ATTRIBUTE>
    <ATTRIBUTE_TAG>00100020</ATTRIBUTE_TAG>
    <DESCRIPTION>Patient ID</DESCRIPTION>
    <ANONYMITY_ACTION action="replace">madeAnonymous</ANONYMITY_ACTION>
  </INDIVIDUAL_ATTRIBUTE>
  <INDIVIDUAL_ATTRIBUTE>
    <ATTRIBUTE_TAG>00100030</ATTRIBUTE_TAG>
    <DESCRIPTION>Patient Birth Date</DESCRIPTION>
    <ANONYMITY_ACTION action="remove"></ANONYMITY_ACTION>
  </INDIVIDUAL_ATTRIBUTE>
</ANONYMITY_RULE_DOCUMENT>
```

See Also:

<http://www.hhs.gov/ocr/privacy/index.html> for more information about the HIPAA regulations in the U. S.

2.9 Extraction of DICOM Metadata for Partitioning

Oracle Multimedia enables the extraction of DICOM metadata outside the database by a client tool or in the middle tier. This feature enables the extraction of DICOM metadata before the data is loaded into the database, facilitating metadata-based partitioning of DICOM data in the database.

Partitioning supports very large tables and indexes by enabling you to reorganize them into smaller, and thus more usable, chunks called partitions. Partitioning can make your data more available, easier to manage, and faster to query.

You can include DICOM metadata attributes in the partition key, a set of one or more columns that determines the partition in which each row in a partitioned table is stored. To ensure that data is stored in the correct partition, you must provide values for the partition key when the data is loaded.

Oracle Multimedia DICOM includes the Oracle Multimedia Mid-Tier Java API. This client application programming interface for the middle tier enables developers to write Java applications to extract DICOM metadata outside of Oracle Database. This API enables the extraction of DICOM attributes that constitute the partition key before the data is loaded into the database.

See Also:

- *Oracle Multimedia Mid-Tier Java API Reference* for reference information
- *Oracle Database VLDB and Partitioning Guide* for more information about partitioning

Overview of DICOM Development

This chapter briefly describes developer and administrator tasks that are related to developing applications using Oracle Multimedia DICOM.

This chapter includes these sections:

- [The DICOM Development Environment](#) on page 3-1
- [DICOM Developer and Administrator Tasks](#) on page 3-3

3.1 The DICOM Development Environment

Oracle Multimedia DICOM provides capabilities for several operations related to DICOM content. For example, administrators can review the Oracle-defined configuration documents in the DICOM data model repository before determining whether to add user-defined documents for their environment. Administrators can use views or invoke data model utility functions to obtain attributes and other detailed information about these configuration documents. Developers can also work directly with the DICOM content, metadata attributes, and other objects to perform various operations.

After installation, each database includes a set of default configuration documents in the Oracle Multimedia DICOM data model repository. After installation, administrators can add configuration documents that are specific to a particular organization. See [Section 2.4.1](#) for general information about configuration documents. See [Part III](#) for detailed information about managing these documents in the DICOM repository.

The following subsections describe the DICOM development environment:

- [APIs for Use With Oracle Multimedia DICOM](#)
- [Views in the DICOM Repository](#)

3.1.1 APIs for Use With Oracle Multimedia DICOM

Because Oracle Multimedia DICOM is fully functional after installing Oracle Multimedia, developers can begin writing applications immediately. Oracle Multimedia DICOM provides several application programming interfaces (APIs) for developers, which are also available to administrators.

In addition, Oracle Multimedia DICOM provides the `ORD_DICOM_ADMIN` data model repository API for administrators only. Using this API, administrators can assist developers by inserting or deleting configuration documents from the data model repository.

Table 3–1 lists the application programming interfaces that are available for Oracle Multimedia DICOM, indicates who can use them, and points to additional information.

Table 3–1 Application Programming Interfaces for DICOM

Name of API	Type of User	More Information
DICOM data model utility API	Administrators and developers	DICOM Data Model Utility Reference
DICOM Java API	Administrators and developers	<i>Oracle Multimedia DICOM Java API Reference</i>
DICOM relational API	Administrators and developers	DICOM Relational Interface Reference
Mid-Tier Java API	Administrators and developers	<i>Oracle Multimedia Mid-Tier Java API Reference</i>
ORDDicom object API	Administrators and developers	ORDDicom Object Type Reference
ORD_DICOM_ADMIN data model repository API	Administrators only	ORD_DICOM_ADMIN Package Reference

3.1.2 Views in the DICOM Repository

Oracle Multimedia DICOM provides views to enable access to specific information about documents in the DICOM repository. Public views are available to users and administrators of the repository. These views provide details such as the names of documents, types of documents, names of constraints, and constraint validation messages. An administrator-only view provides details about document references within the repository.

Table 3–2 lists the views, indicates who can access them, and points to additional information.

Table 3–2 DICOM Repository Views

Name	Access	More Information
orddcm_conformance_vld_msgs	Public	DICOM Repository Public Views
orddcm_constraint_names	Public	DICOM Repository Public Views
orddcm_document_refs	Administrators only	DICOM Repository Administrator Views
orddcm_document_types	Public	DICOM Repository Public Views
orddcm_documents	Public	DICOM Repository Public Views

orddcm_conformance_vld_msgs

The [orddcm_conformance_vld_msgs](#) view lists the constraint messages that are generated for a set of constraints during a validation operation.

Note: This view lists the constraint messages for the current user's schema *only*.

orddcm_constraint_names

The [orddcm_constraint_names](#) view lists the names of the constraints installed in the repository.

orddcm_document_refs

For administrators only, the [orddcm_document_refs](#) view lists the documents in the repository that are referenced by other documents in the repository.

orddcm_document_types

The [orddcm_document_types](#) view identifies the supported Oracle Multimedia DICOM document types with a list of codes.

orddcm_documents

The [orddcm_documents](#) view lists details of the documents that are stored in the repository.

3.2 DICOM Developer and Administrator Tasks

The following tasks, which are described in these subsections, present a recommended approach for developing applications using Oracle Multimedia DICOM:

- [Task 1: Load the Repository](#)
- [Task 2: Load the DICOM Content](#)
- [Task 3: Extract the DICOM Metadata](#)
- [Task 4: Search and Retrieve DICOM Attributes](#)
- [Task 5: Write and Edit DICOM Metadata](#)
- [Task 6: Process, Convert, and Compress DICOM Data](#)
- [Task 7: Create DICOM Content from Secondary Capture Images and Video](#)
- [Task 8: Validate Conformance with DICOM Constraints](#)
- [Task 9: Protect Confidential Patient Data](#)
- [Task 10: Improve Storage When Extracting DICOM Attributes](#)

In addition, each of these tasks requires administrators and developers to perform a separate, specific set of steps:

- Extracting the DICOM metadata
- Validating conformance with DICOM constraints
- Protecting confidential patient data

3.2.1 Task 1: Load the Repository

At the start of every database session, users and administrators must load the data model repository from the database into memory structures. Users load the data model by calling the `setDataModel()` procedure. Administrators load the data model by calling either the `setDataModel()` procedure or the `editDataModel()` procedure.

After loading the repository into memory, users and administrators can call the `setDataModel()` procedure whenever the application requires the new data model changes.

Note: Users and administrators must call the `setDataModel()` procedure before calling any other DICOM methods, functions, or procedures.

Using the DICOM data model utility in the ORD_DICOM package, users and administrators call the `setDataModel()` procedure as follows:

```
exec ord_dicom.setdatamodel;
```

See the [setDataModel\(\) Procedure](#) and the [editDataModel\(\) Procedure](#) for more information.

3.2.2 Task 2: Load the DICOM Content

You can use the SQL*Loader utility to load DICOM content into existing tables in Oracle Database. First, you must create a table with the appropriate columns and initialize the columns (see [Section 5.2](#)). Then, you can call the SQL*Loader utility and load the DICOM content from your data files into the table columns as SecureFile LOBs (see [Section 5.3](#)). Before performing any operations on the DICOM content, you must call the `setDataModel()` procedure to load the DICOM data model (see the [setDataModel\(\) Procedure](#)).

After the DICOM content is loaded, you can perform other operations, such as extracting DICOM metadata, searching and retrieving DICOM attributes, writing and editing DICOM metadata, creating thumbnail images, validating the conformance of your DICOM content, or making confidential DICOM content anonymous.

See [Chapter 5](#) for examples that show how to use the SQL*Loader utility to load DICOM content and then make specific metadata attributes in the DICOM content anonymous.

See Also:

Oracle Database Utilities for more information about using the SQL*Loader utility to load objects and LOBs into Oracle Database

3.2.3 Task 3: Extract the DICOM Metadata

Oracle Multimedia DICOM provides support for extracting metadata from DICOM content. By searching the extracted metadata, applications can find the specific DICOM content that contains the associated metadata.

To extract all DICOM attributes into an XML document that conforms to the default metadata XML schema, first call the `setProperties()` method to extract and store the metadata XML document into the metadata attribute of the ORDDicom object. The default metadata XML schema defines a complete and generalized data model for storing DICOM attributes. A customized metadata schema and corresponding mapping document that is customized for your specific application may yield better performance for indexing and searching than the generalized, default metadata schema. Your custom schemas and mapping documents can be used to define frequently searched DICOM attributes within a hierarchical structure that is optimized for searching.

To extract the DICOM attributes into an application-specific XML document in XMLType, call the `extractMetadata()` method and specify the application-specific mapping document. The resulting application-specific metadata XML document can be stored in table columns. This metadata can be bound to the application-specific XML schema.

See the [setProperties\(\)](#) and [extractMetadata\(\)](#) methods for more information.

The following subsections list the steps for administrators and developers to perform when extracting metadata:

- [Extracting Metadata: Administrator Tasks](#)
- [Extracting Metadata: Developer Tasks](#)

3.2.3.1 Extracting Metadata: Administrator Tasks

As an administrator, perform the following tasks to initiate the process of extracting DICOM metadata:

1. Design the XML schema for the extracted metadata. Generally, the most frequently searched DICOM attributes are included in the mapped section of the XML schema.

See [Appendix B](#) for more information about XML schemas.

2. Register the schema as a global XML schema with Oracle XML DB.

See Also:

Oracle XML DB Developer's Guide for more information about registering XML schemas

3. Create the mapping document for the metadata XML schema.

See [Section 10.2.3](#) for more information.

4. Load the mapping document into the data model repository.

See [Section 9.6](#) for more information.

3.2.3.2 Extracting Metadata: Developer Tasks

As a developer, perform the following tasks to complete the process of extracting DICOM metadata:

1. Query the `orddcm_documents` view to ensure that the mapping document is loaded and available.

```
select * from orddcm_documents;
```

See the [orddcm_documents](#) view for more information.

2. Call the `extractMetadata()` method to extract metadata into an XML metadata document by specifying the name of the mapping document and the appropriate parameters.

See the [extractMetadata\(\)](#) method for more information.

3. Store the returned XML metadata document in a column in the database that is bound to the application-specific XML schema for later searching.

See Also:

Oracle XML DB Developer's Guide for more information about this task

3.2.4 Task 4: Search and Retrieve DICOM Attributes

Oracle Multimedia DICOM provides support for searching and retrieving DICOM attributes.

To extract the SOP instance UID, SOP class UID, study instance UID, and series instance UID attributes into ORDDicom object attributes, first call the `setProperties()` method to populate the attributes of the ORDDicom object. These DICOM attributes can be easily retrieved from within the ORDDicom object. To make searching faster,

these attributes can also be indexed by creating indexes on the corresponding object attributes. (See [Chapter 7](#) for reference information.)

To search and retrieve attributes within the metadata XML document, call the SQL XML functions `XMLCast()`, `XMLExists()`, `XMLQuery()`, and `XMLTable()` by specifying the XPath expression for the attributes. To make searching faster, the attributes in the metadata XML document can also be indexed.

See Also:

- *Oracle Database SQL Language Reference* for reference information
- *Oracle XML DB Developer's Guide* for more information about indexing XMLType data

To retrieve a single DICOM attribute, you can call either the `getAttributeByTag()` or the `getAttributeByName()` method. This process is not recommended for large tables, or for retrievals of multiple attributes. However, if you do use this process, Oracle recommends building function-based indexes on these methods to make searching faster. (See [Chapter 7](#) for reference information.)

Note: Before you can retrieve DICOM attributes, you must call the `setProperties()` method to populate the attributes of the `ORDDicom` object (see the [setProperties\(\)](#) method).

3.2.5 Task 5: Write and Edit DICOM Metadata

Oracle Multimedia DICOM provides support for creating a new `ORDDicom` object with overwritten or supplemented metadata from an existing `ORDDicom` object. The `writeMetadata()` method creates a new copy of the `ORDDicom` object from the original `ORDDicom` object and an XML document containing any additional and modified metadata. The original `ORDDicom` object is preserved.

Perform the following tasks to write and edit DICOM metadata:

1. Extract a copy of the original DICOM metadata from the original DICOM content by calling the `extractMetadata()` method, using the Oracle default mapping document.
See the [extractMetadata\(\)](#) method for more information.
2. Add or modify DICOM attributes in the copy of the extracted metadata XML document.
See [Chapter 9](#) for more information about working with DICOM metadata in XML documents.
3. Create an empty `ORDDicom` object using an empty `ORDDicom` constructor as the placeholder for the new DICOM content.
See [ORDDicom Constructors](#) for reference information.
4. Call the `writeMetadata()` method to write the modified copy of the metadata XML document and the DICOM content from the original `ORDDicom` object into the newly created `ORDDicom` object. The metadata XML document is used to overwrite existing attributes and add new attributes in the new `ORDDicom` object.
See the [writeMetadata\(\)](#) method for more information.

3.2.6 Task 6: Process, Convert, and Compress DICOM Data

Oracle Multimedia DICOM provides the `processCopy()` method to copy and process the DICOM content and save it as DICOM format or another media format, in accordance with the specified command parameters. The `processCopy()` method creates new content and preserves the original DICOM content.

The following list summarizes the processing, converting, and compressing operations supported by Oracle Multimedia DICOM, and includes examples of the command parameter of the `processCopy()` method that corresponds to each operation.

- Create a JPEG thumbnail image. For example:
`'fileFormat=jpeg, maxScale=100 100'`
- Create a JPEG image of the same size as the original DICOM image. For example:
`'fileFormat=jpeg'`
- Compress the image content within the DICOM content. For example:
`'fileFormat=dicom, compressionFormat=jpeg'`
- Retrieve a specified frame from DICOM multiframe content. For example:
`'frame=10'`
- Cut a specified region of a DICOM image. For example:
`'cut=20 20 100 100'`
- Process multiframe DICOM content into AVI output. For example:
`'fileFormat=avi, scale=0.5'`
- Write DICOM content with RLE compression. For example:
`'fileFormat=dicom, compressionFormat=dicomrle'`
- Extract MPEG content from DICOM format and process it into MPEG format. For example:
`'fileFormat=mpeg'`
- Write DICOM content with DEFLATE compression. For example:
`'fileformat=dicom, compressionFormat=deflate, deflateLevel=defaultcompression'`

See [ORDDicom Methods](#) for reference information about `processCopy()` methods and supported command parameters. See [Appendix D](#) for more information about DICOM processing.

3.2.7 Task 7: Create DICOM Content from Secondary Capture Images and Video

Oracle Multimedia DICOM provides the relational procedure `createDicomImage()` to create DICOM format images and video from secondary capture images and video, including content in formats such as JPEG, TIFF, or MPEG and DICOM metadata in data type XMLType.

Perform the following tasks to create DICOM format content from secondary capture images or video and DICOM metadata:

1. Load the provided image or video into a BLOB, or define it as a BFILE data type.

See Also:

Oracle Multimedia User's Guide for more information about these tasks

2. Create an XMLType object from the DICOM metadata of the corresponding image or video.
3. Create an empty BLOB object as the placeholder for the new DICOM content.
4. Create the DICOM format image or video using the createDicomImage() procedure.

See [DICOM Relational Procedures](#) for reference information about createDicomImage() procedures.

3.2.8 Task 8: Validate Conformance with DICOM Constraints

Oracle Multimedia DICOM provides support to validate the conformance of your DICOM content, according to DICOM specified constraints, with the isConformanceValid() method. Use this method to check if the embedded DICOM content conforms to a specific set of constraints.

Conformance constraints are a collection of rules for validating the conformance of DICOM content with the DICOM standard and other organization-wide guidelines. These rules are specified in an XML document called a **constraint document**, which is stored in the repository. The default constraint document shipped with Oracle Multimedia DICOM defines rules that enforce conformance with parts of the DICOM standard.

DICOM content can come from many sources, and the DICOM content may or may not have been created in accordance with the DICOM standard. Before you decide to store DICOM content in your repository, you can check the DICOM content for specific attributes. As an example, you can check the DICOM content for the value of the attribute for a patient's sex. As defined in the Patient Module Attributes of the DICOM standard, this attribute can have these values: M (male), F (female), or O (other). You can define constraint rules to check for correct values for specific attributes. Or, you can define a custom set of constraint rules and then ensure that all DICOM content in your repository conforms to those rules.

After installation, you can define constraint documents to include user-defined constraint rules that are specific to your organization. To see a list of constraint names, query the view orddcm_constraint_names.

During conformance validation, if a constraint rule was defined to generate messages for a specified predicate condition and the condition was met, these messages are generated to indicate the predicate conditions. To see a list of these constraint messages, query the view orddcm_conformance_vld_msgs.

See the [orddcm_constraint_names](#) and [orddcm_conformance_vld_msgs](#) views for more information.

Validating the conformance of DICOM content requires a combination of administrator and developer tasks. The following subsections describe these tasks.

Note: Perform the administrator tasks described in this section first.

The following subsections list the steps for administrators and developers to perform when during conformance validation:

- [Validating Conformance: Administrator Tasks](#)
- [Validating Conformance: Developer Tasks](#)

3.2.8.1 Validating Conformance: Administrator Tasks

As an administrator, perform the following tasks to initiate the process of validating the conformance of DICOM content:

1. Create a constraint document for your organization. Within this document, you can define constraint rules to generate messages for specified predicate conditions.

[Example 3-1](#) shows the constraint rule for the Patient Module of the DICOM standard, which is defined in the Oracle-installed constraint document `ordcmcmd.xml`.

Example 3-1 Constraint Rule for the Patient Module

```
<GLOBAL_RULE name="PatientModule">
  <DESCRIPTION>
    A subset of Patient Module defined in DICOM standard,
    PS 3.3-2007, Table C.7-1
  </DESCRIPTION>
  <PREDICATE>
    <BOOLEAN_FUNC operator="notEmpty">
      <ATTRIBUTE_TAG>00100040</ATTRIBUTE_TAG>
    </BOOLEAN_FUNC>
  </PREDICATE>
  <PREDICATE>
    <DESCRIPTION>Patient's Sex</DESCRIPTION>
    <RELATIONAL operator="in">
      <ATTRIBUTE_TAG>00100040</ATTRIBUTE_TAG>
      <STRING_VALUE>M</STRING_VALUE>
      <STRING_VALUE>F</STRING_VALUE>
      <STRING_VALUE>O</STRING_VALUE>
    </RELATIONAL>
  </PREDICATE>
  <PREDICATE>
    <DESCRIPTION>Referenced patient sequence constraint</DESCRIPTION>
    <LOGICAL operator="derive">
      <PREDICATE>
        <BOOLEAN_FUNC operator="occurs">
          <ATTRIBUTE_TAG>00081120</ATTRIBUTE_TAG>
        </BOOLEAN_FUNC>
      </PREDICATE>
      <PREDICATE>
        <LOGICAL operator="and">
          <PREDICATE>
            <BOOLEAN_FUNC operator="notEmpty">
              <ATTRIBUTE_TAG>00081120.00081150</ATTRIBUTE_TAG>
            </BOOLEAN_FUNC>
          </PREDICATE>
          <PREDICATE>
            <BOOLEAN_FUNC operator="notEmpty">
              <ATTRIBUTE_TAG>00081120.00081155</ATTRIBUTE_TAG>
            </BOOLEAN_FUNC>
          </PREDICATE>
        </LOGICAL>
      </PREDICATE>
    </LOGICAL>
  </PREDICATE>
  <ACTION action="log" when="false">Validation error:
    missing mandatory attribute for patient module</ACTION>
  <ACTION action="warning" when="false">Warning: validation failure</ACTION>
</GLOBAL_RULE>
```

See [Section 10.2.2](#) for more information.

2. Load the constraint document into the data model repository.

See [Section 9.6](#) for more information.

3.2.8.2 Validating Conformance: Developer Tasks

As a developer, perform the following tasks to complete the process of validating the conformance of DICOM content against your constraint rules:

1. Query the `orddcm_constraint_names` view to see the list of available constraint names for your organization as follows:

```
select * from orddcm_constraint_names;
```

See the [orddcm_constraint_names](#) view for more information.

2. Call the `isConformanceValid()` method to check the conformance of your DICOM content as follows:

```
declare
  cursor dicom_src_cur is
  select dicom_src from medical_image_obj order by id;
begin
  for rec in dicom_src_cur loop
    dbms_output.put_line('isConformanceValid(PatientModule): ' ||
      rec.dicom_src.isConformanceValid('PatientModule'));
  end loop;
end;
/
```

See the [isConformanceValid\(\)](#) method for more information.

3. Query the `orddcm_conformance_vld_msgs` view to see the list of constraint messages generated during constraint validation for your data as follows:

```
select * from orddcm_conformance_vld_msgs;
```

See the [orddcm_conformance_vld_msgs](#) view for more information.

If your DICOM content does not conform to the constraint rules defined for your organization, you can write another ORDDicom object with corrected DICOM metadata (see [Section 3.2.5](#)).

3.2.9 Task 9: Protect Confidential Patient Data

Oracle Multimedia DICOM provides support to protect the privacy of patients by ensuring the confidentiality of patient data. In general, call the `isAnonymous()` method before calling the `makeAnonymous()` method to check if the confidential patient data for a specified ORDDicom object has been removed or replaced, according to a specified anonymity document. Call the `makeAnonymous()` method to remove or replace confidential patient data. This method creates a new ORDDicom object, and preserves the original ORDDicom object.

Both of these methods use anonymity documents, XML documents that are stored in the repository, to determine the patient identifying information that must be made anonymous. In addition to specifying the set of attributes to be made anonymous, anonymity documents specify the actions to be taken to make those attributes anonymous.

After installation, you can use the default anonymity document shipped with Oracle Multimedia DICOM. Or, you can add customized anonymity documents to overwrite or remove confidential patient data, as necessary.

The following subsections list the steps for administrators and developers to perform when making confidential data anonymous:

- [Protecting Privacy: Administrator Tasks](#)
- [Protecting Privacy: Developer Tasks](#)

3.2.9.1 Protecting Privacy: Administrator Tasks

As an administrator, perform the following tasks to initiate the process of making confidential patient data anonymous:

1. Create an anonymity document that defines the DICOM attributes to be removed or replaced.
See [Section 10.2.1](#) for more information.
2. Load the anonymity document into the data model repository.
See [Section 9.6](#) for more information.

3.2.9.2 Protecting Privacy: Developer Tasks

As a developer, perform the following tasks to complete the process of making confidential patient data anonymous:

1. Query the `orddcm_documents` view to see the list of anonymity documents that have been loaded into the data model repository as follows:

```
select * from orddcm_documents;
```

See the [orddcm_documents](#) view for more information.

2. Create an empty ORDDicom object using the empty ORDDicom constructor if you plan to call the `makeAnonymous()` method. The empty ORDDicom object is used as a placeholder for the new ORDDicom object. Create the empty object as follows:

```
insert into medical_image_obj (id, dicom_src)
values (3, ORDDicom());
```

See [ORDDicom Constructors](#) for more information.

3. Call the `isAnonymous()` method to check if the original ORDDicom object is anonymous, in accordance with the specified anonymity document.

```
select t.dicom_src.isAnonymous('ordcman.xml') from medical_image_obj t;
```

See the [isAnonymous\(\)](#) method for more information.

4. Call the `makeAnonymous()` method to copy the original ORDDicom object, make it anonymous, and then write the new DICOM content into the new, empty ORDDicom object. Call this method as follows:

```
declare
  obj_src orddicom;
  obj_dest orddicom;
  dest_sop_instance_uid varchar2(128) := '<unique-UID>';
begin
  select dicom_src, dicom_dest into obj_src, obj_dest
  from medical_image_obj where id = 1 for update;
  obj_src.makeAnonymous(dest_sop_instance_uid, obj_dest, 'ordcman.xml');
```

```
        update medical_image_obj set dicom_dest = obj_dest where id = 1;
    end;
/
```

See the [makeAnonymous\(\)](#) method for more information.

3.2.10 Task 10: Improve Storage When Extracting DICOM Attributes

Oracle Database 11g Release 2 (11.2) provides the ability to limit the list of DICOM attributes extracted by the `setProperties()` method. Extracting only a selected set of attributes instead of all the attributes improves the performance of the `setProperties()` method and reduces storage requirements. Oracle Multimedia DICOM provides this support with a stored tag list configuration document in the repository. A stored tag list document specifies the DICOM attributes to be extracted from the embedded DICOM content and stored in the XML metadata attribute of the ORDDicom object when the `setProperties()` method is called.

See [Section 10.2.8](#) for more information about creating stored tag list documents.

See [Section 11.4](#) for sample sessions that show how to insert a stored tag list document into the repository.

See the [generateTagListDocument\(\) Function](#) for reference information.

See also [Section B.11](#) for a listing of the stored tag list document schema (`ordcmstl.xsd`).

DICOM Data Model Utility Reference

Oracle Multimedia provides the DICOM data model utility in the ORD_DICOM package. Oracle Multimedia DICOM also defines public views for the DICOM repository. The functions and procedures in the DICOM data model utility interface and the public views can be used by DICOM developers and administrators of the DICOM data model repository.

For administrative operations related to the Oracle Multimedia DICOM data model repository, the `setDataModel()` procedure must be called at the beginning of each database session (see the [setDataModel\(\) Procedure](#)). Developers and administrators must call the `setDataModel()` procedure before calling any other DICOM methods, functions, or procedures.

The ORD_DICOM package is defined in the `ordcpksp.sql` file. After installation, this file is available in the Oracle home directory at:

```
<ORACLE_HOME>/ord/im/admin (on Linux and UNIX)
```

```
<ORACLE_HOME>\ord\im\admin (on Windows)
```

This chapter describes the functions, procedures, and views in the DICOM data model utility interface, which operate on the DICOM data model repository. See [Table 3-1](#) for information about other DICOM application programming interfaces (APIs).

This chapter contains these sections:

- [Directory Definition and Setup for ORD_DICOM Examples](#) on page 4-1
- [DICOM Data Model Utility Functions](#) on page 4-2
- [DICOM Data Model Utility Procedures](#) on page 4-8
- [DICOM Repository Public Views](#) on page 4-10

4.1 Directory Definition and Setup for ORD_DICOM Examples

See the examples for each function or procedure in this chapter for specific directory definitions for DICOM data files and other details specific to that function or procedure.

DICOM Data Model Utility Functions

The ORD_DICOM package defines these DICOM data model utility functions:

- [getDictionaryTag\(\) Function](#) on page 4-3
- [getMappingXPath\(\) Function](#) on page 4-5

getDictionaryTag() Function

Format

```
getDictionaryTag(attributeName IN VARCHAR2,
                 definerName IN VARCHAR2 DEFAULT 'DICOM'
                 ) RETURN VARCHAR2
```

Description

Looks in the standard or private data dictionaries for the specified attribute name and definer name and returns the value of the attribute name as a hexadecimal DICOM attribute tag. This function can be used to get the hexadecimal tag value that is needed in the `getMappingXpath()` function.

Parameters

attributeName

The name of specified attribute in the standard or private data dictionary (for example: `Patient's Name`). The maximum length of this parameter is 128 characters.

definerName

The definer name of the specified attribute in the standard or private data dictionary. The default name, `DICOM`, refers to the DICOM standard. The maximum length of this parameter is 64 characters.

Pragmas

None.

Exceptions

None.

Usage Notes

Before calling this function, call the `setDataModel()` procedure.

Call the `setDataModel()` procedure at these times:

- At the beginning of each database session
- Whenever the application requires the new data model changes

See the [setDataModel\(\) Procedure](#) for more information.

Examples

Get the specified DICOM attribute name and return its value as a hexadecimal tag:

```
exec ord_dicom.setDataModel();
select ord_dicom.getDictionaryTag('Patient's Name', 'DICOM') as Patient_Name
from dual;
```

```
PATIENT_NAME
```

```
-----
00100010
```

```
select ord_dicom.getDictionaryTag('Audio Type', 'DICOM') as Audio_Type from dual;
```

AUDIO_TYPE

50XX2000

getMappingXPath() Function

Format

```
getMappingXPath(tag IN VARCHAR2,
                docName IN VARCHAR2 DEFAULT 'ordcmmmp.xml',
                definerName IN VARCHAR2 DEFAULT 'DICOM'
                ) RETURN VARCHAR2
```

Description

Returns the absolute XPath expression associated with the specified DICOM attribute tag and definer name from the specified mapping document. The XPath expression that is returned can be used to obtain values from an extracted XML metadata document.

Parameters

tag

A DICOM attribute tag from the specified mapping document, represented as an 8-character hexadecimal string (for example: 00100010).

docName

The name of a mapping document. The default name is `ordcmmmp.xml`.

definerName

The definer name of the DICOM attribute tag in the specified mapping document. The default name is `DICOM`, which refers to the DICOM standard.

Pragmas

None.

Exceptions

None.

Usage Notes

Before calling this function, call the `setDataModel()` procedure.

Call the `setDataModel()` procedure at these times:

- At the beginning of each database session
- Whenever the application requires the new data model changes

See the [setDataModel\(\) Procedure](#) for more information.

To see a list of all the mapping documents in the data model repository, query the public view [orddcm_documents](#).

Examples

Example 1:

Return the XPATH expression for DICOM attributes other than SEQUENCE type attributes:

```
col map_xpath new_value xpath
exec ord_dicom.setDataModel();
```

```

select ord_dicom.getMappingXPath('00100010') map_xpath from dual;

MAP_XPATH
-----
/DICOM_OBJECT/PERSON_NAME[@tag="00100010" and @definer="DICOM"]/VALUE/text()

1 row selected.

----- extract attribute from a document
select '&xpath' as map_xpath from dual;
select xmlquery(
  'declare default element namespace
   "http://xmlns.oracle.com/ord/dicom/metadata_1_0"; &xpath'
  passing t.metadata returning content) as
  Patient_Name from metadata_tab t where id=1;

PATIENT_NAME
-----
Anonymous

1 row selected.

```

Example 2:

Return the XPATH expression for DICOM SEQUENCE type attributes only:

```

col map_xpath new_value xpath
exec ord_dicom.setDataModel();
select ord_dicom.getMappingXPath('00082218') map_xpath from dual;

MAP_XPATH
-----
/DICOM_OBJECT/SEQUENCE[@tag="00082218" and @definer="DICOM"]

1 row selected.

----- extract attribute from a document
set long 1000
select '&xpath' as map_xpath from dual;
select xmlquery(
  'declare default element namespace
   "http://xmlns.oracle.com/ord/dicom/metadata_1_0"; &xpath'
  passing t.metadata returning content) as
  Anatomic_Region from metadata_tab t where id=2;

ANATOMIC_REGION
-----
<SEQUENCE xmlns="http://xmlns.oracle.com/ord/dicom/metadata_1_0"
tag="00082218"
definer="DICOM" name="Anatomic Region Sequence" offset="590"
length="52"><ITEM number="1"><SHORT_STRING tag="00080100"
definer="DICOM" name="Code Value" offset=
"606" length="8">T-11170</SHORT_STRING><SHORT_STRING tag="00080102"
definer="DICOM" name="Coding Scheme Designator" offset="622"
length="4">SNM3</SHORT_STRING><LONG_STRING tag="00080104" definer="DICOM"
name="Code Meaning" offset="634"
length="8">Maxilla</LONG_STRING></ITEM></SEQUENCE>

```

1 row selected.

where:

- `metadata_tab`: a table containing metadata that has been extracted from the DICOM content.

DICOM Data Model Utility Procedures

The ORD_DICOM package defines this DICOM data model utility procedure:

- [setDataModel\(\) Procedure](#) on page 4-9

setDataModel() Procedure

Format

```
setDataModel(modelName IN VARCHAR2 DEFAULT 'DEFAULT')
```

Description

Loads the data model repository from the database into the memory structures. This procedure must be called at the beginning of each database session. It can be called again whenever the application requires the new data model changes.

Parameters

modelName

The model name of the data model repository. The default name, `DEFAULT`, is the only value supported in Oracle Database 11g.

Pragmas

None.

Exceptions

None.

Usage Notes

To maximize performance, you may want to call this procedure only once during a database session.

If the data model has changed since you made the original call, and your application requires these data model changes, you can call this procedure again. Keep in mind, however, that subsequent calls to this procedure may result in changed behavior.

To use the same DICOM data model throughout a session, Oracle recommends following the call to this procedure with a `COMMIT` statement. Because data that is loaded by the `setDataModel()` procedure is subject to transaction semantics, the database session's copy of the data model is deleted during a rollback operation if the call to the `setDataModel()` procedure is made within the transaction that you are rolling back.

If you roll back the transaction in which you call the `setDataModel()` procedure, you may get an error message indicating that the data model is not loaded when you use the DICOM feature in the same session following the rollback operation. Call the `setDataModel()` procedure to reload the data model.

See the [rollbackDataModel\(\) Procedure](#) for more information.

Examples

Load the repository from the database into memory:

```
exec ord_dicom.setdatamodel;  
select * from orddcm_documents;
```

DICOM Repository Public Views

This section describes these Oracle Multimedia DICOM repository public views:

- [orddcm_conformance_vld_msgs](#) on page 4-11
- [orddcm_constraint_names](#) on page 4-12
- [orddcm_documents](#) on page 4-13
- [orddcm_document_types](#) on page 4-14

See [DICOM Repository Administrator Views](#) for information about the Oracle Multimedia DICOM repository view for administrators.

orddcm_conformance_vld_msgs

Format

Column Name	Data Type	Description
SOP_INSTANCE_UID	VARCHAR2(128 CHAR)	SOP_INSTANCE_UID of DICOM content
RULE_NAME	VARCHAR2(64 CHAR)	Constraint rule name
MESSAGE	VARCHAR2(1999 CHAR)	Message
MSG_TYPE	VARCHAR2(20 CHAR)	Message type, valid values are: log, warning, or error
MSG_TIME	TIMESTAMP	Message generation time

Description

This view lists the constraint messages generated during constraint validation. The public READ and DELETE access privileges are granted for this view.

Usage Notes

This view shows the constraint validation messages that are generated for a specified user schema only.

Examples

Show the list of constraint validation messages that were generated for the predicate conditions defined in the specified constraint document. The conformance validation rule shown in this example is PatientModule, as defined in the DICOM standard.

SOP_INSTANCE_UID	RULE_NAME	MESSAGE	MSG_TYPE	MSG_TIME
1.2.840.114346. 3384726461.899958945. 2180235641.3197827030	PatientModule	Validation error: missing mandatory attribute for patient module	log	01-MAR-07 01.40.21.158337 PM
1.2.840.114346. 3384726461.899958945. 2180235641.3197827030	PatientModule	Warning: validation failure	warning	01-MAR-07 01.40.21.168322 PM

2 rows selected.

orddcm_constraint_names

Format

Column Name	Data Type	Description
NAME	VARCHAR2(100)	Constraint name

Description

This read-only view lists the constraint names. The public READ access privilege is granted for this view.

Usage Notes

Before querying this view, call the `setDataModel()` procedure at least once during the database session. Call it again whenever the application requires the new data model changes.

See the [setDataModel\(\) Procedure](#) for more information.

Examples

Show a list of the constraint names that are available after installation:

```
-----  
NAME  
-----
```

```
PatientModule  
GeneralStudyModule  
GeneralSeriesModule  
SOPCommonModule  
ImagePixelMacro  
OracleOrdDicomImage  
OracleOrdObject
```

```
7 rows selected.
```


orddcm_documents

Format

Column Name	Data Type	Description
DOC_NAME	VARCHAR2(100)	Document name
DOC_TYPE	VARCHAR2(100)	Document type
CREATE_DATE	DATE	Document creation date
ISINSTALLED_BY_ORACLE	NUMBER(1)	A value of 1 indicates that the document was installed by Oracle. A value of 0 indicates that the document was loaded by an administrator.

Description

This read-only view lists details of the documents stored in the repository. The public READ access privilege is granted for this view.

Usage Notes

Before querying this view, call the `setDataModel()` procedure at least once during the database session. Call it again whenever the application requires the new data model changes.

See the [setDataModel\(\) Procedure](#) for more information.

Examples

Show a list of the configuration documents in the repository, by name, type, and date of creation and indicate whether the configuration document is Oracle-defined or user-defined. This example shows details about the default Oracle-defined configuration documents that are available upon installation.

```
-----
DOC_NAME          DOC_TYPE          CREATE_DA        INSTALLED_BY_ORACLE
-----
ordcmpv.xml       PRIVATE_DICTIONARY  04-FEB-09       1
ordcmmmp.xml      MAPPING            04-FEB-09       1
ordcman.xml       ANONYMITY          04-FEB-09       1
ordcmpf.xml       PREFERENCE          04-FEB-09       1
ordcmui.xml       UID_DEFINITION      04-FEB-09       1
ordcmcmc.xml      CONSTRAINT          04-FEB-09       1
ordcmcmd.xml      CONSTRAINT          04-FEB-09       1
ordcmct.xml       CONSTRAINT          04-FEB-09       1
ordcmsd.xml       STANDARD_DICTIONARY 04-FEB-09       1
-----
```

9 rows selected.

orddcm_document_types

Format

Column Name	Data Type	Description
DOC_TYPE	VARCHAR2(100)	Document type code of the document type This column contains these values: STANDARD_DICTIONARY PRIVATE_DICTIONARY CONSTRAINT MAPPING ANONYMITY PREFERENCE UID_DEFINITION STORED_TAG_LIST
SCHEMA_URL	VARCHAR2(700)	The URL of the XML schema registered with Oracle XML DB that is associated with this document type This column contains these values, which are listed respective to the order of the values in the DOC_TYPE column: http://xmlns.oracle.com/ord/dicom/standardDictionary_1_0 http://xmlns.oracle.com/ord/dicom/privateDictionary_1_0 http://xmlns.oracle.com/ord/dicom/constraint_1_0 http://xmlns.oracle.com/ord/dicom/mapping_1_0 http://xmlns.oracle.com/ord/dicom/anonymity_1_0 http://xmlns.oracle.com/ord/dicom/preference_1_0 http://xmlns.oracle.com/ord/dicom/UIDdefinition_1_0 http://xmlns.oracle.com/ord/dicom/attributeTag_1_0
DOC_TYPE_DESC	VARCHAR2(4000)	Document type description This column contains these values, which are listed respective to the order of the values in the DOC_TYPE column: DICOM standard data dictionary Private data dictionary Constraint document Mapping document Anonymity document Preference document DICOM UID definition document Stored tag list document

Description

This read-only view identifies the supported Oracle Multimedia DICOM document types. Use this view to find the list of codes for document types when inserting a new document into the Oracle Multimedia DICOM repository. The public READ access privilege is granted for this view.

Usage Notes

None.

Examples

Show the document type, schema URL, and document type description for the Oracle-installed configuration documents:

DOC_TYPE	SCHEMA_URL	DOC_TYPE_DSC
STANDARD_DICTIONARY	http://xmlns.oracle.com/ord/dicom/standardDictionary_1_0	DICOM Standard Data Dictionary
PRIVATE_DICTIONARY	http://xmlns.oracle.com/ord/dicom/privateDictionary_1_0	Private Data Dictionary
MAPPING	http://xmlns.oracle.com/ord/dicom/mapping_1_0	Mapping document
ANONYMITY	http://xmlns.oracle.com/ord/dicom/anonymity_1_0	Anonymity document
PREFERENCE	http://xmlns.oracle.com/ord/dicom/preference_1_0	Preference document
UID_DEFINITION	http://xmlns.oracle.com/ord/dicom/UIDdefinition_1_0	DICOM UID definition document
CONSTRAINT	http://xmlns.oracle.com/ord/dicom/constraint_1_0	Constraint document
STORED_TAG_LIST	http://xmlns.oracle.com/ord/dicom/attributeTag_1_0	Stored tag list document

8 rows selected.

Part II

DICOM Development Usage and Reference

This part contains user and reference information for developers of DICOM applications.

Part II includes these chapters:

- [Chapter 5, "DICOM Application Development"](#)
- [Chapter 6, "DICOM Sample Application"](#)
- [Chapter 7, "ORDDicom Object Type Reference"](#)
- [Chapter 8, "DICOM Relational Interface Reference"](#)

DICOM Application Development

This chapter describes how to develop applications using Oracle Multimedia DICOM. Oracle Multimedia DICOM provides support for Oracle Database with these application programming interfaces (APIs):

- ORDDicom object API
- DICOM data model utility API
- DICOM relational API
- DICOM Java API

Using these interfaces, you can quickly develop applications to upload to the database, retrieve from it, and manipulate DICOM content.

This chapter includes examples of how to import DICOM content into the database, write SQL queries based on DICOM metadata, perform basic image processing, make anonymous copies of ORDDicom objects, and check DICOM content for conformance to user-defined constraint rules. Some examples were extracted from the tutorial *Managing DICOM Format Data in Oracle Database 11g*, and adapted for this manual. See [Appendix E](#) for the location of this tutorial.

This chapter includes these sections:

- [Setting Up Your Environment](#) on page 5-1
- [Creating a Table with an ORDDicom Column](#) on page 5-2
- [Loading DICOM Content Using the SQL*Loader Utility](#) on page 5-3
- [Developing DICOM Applications Using the PL/SQL API](#) on page 5-7
- [Developing DICOM Applications Using the DICOM Java API](#) on page 5-13

For additional examples, articles, and other information about Oracle Multimedia, see the Oracle Multimedia Software section of the Oracle Technology Network Web site at

<http://www.oracle.com/technology/products/multimedia/>

5.1 Setting Up Your Environment

The examples in this chapter use the table `medical_image_table` with these four columns:

- `id` - an integer identifier
- `dicom` - an ORDSYS.ORDDicom object
- `imageThumb` - an ORDSYS.ORDImage object

- `anonDicom` - another ORDSYS.ORDDicom object

Issue the following statements before executing the examples in this chapter, where `c:\mydir\work` is the directory where the user `scott` can find the DICOM files:

```
CONNECT sys as sysdba
Enter password: password
CREATE OR REPLACE DIRECTORY FILE_DIR as 'c:\mydir\work';
GRANT READ ON DIRECTORY FILE_DIR TO scott;
```

To use a different directory for your DICOM files, replace `c:\mydir\work` with the directory specification where your files are located.

Note: All Oracle Multimedia objects and procedures provided by Oracle are defined in the schema ORDSYS.

5.2 Creating a Table with an ORDDicom Column

This section shows how to create a table with an ORDDicom column to store DICOM content.

The code segment shown in [Example 5-1](#) creates the table `medical_image_table`, with the four columns `id`, `dicom`, `imageThumb`, and `anonDicom`.

Example 5-1 Creating a Table for DICOM Content

```
CONNECT scott

Enter password: password

create table medical_image_table
  (id          integer primary key,
   dicom       ordsys.orddicom,
   imageThumb  ordsys.ordimage,
   anonDicom   ordsys.orddicom)
--
-- metadata extraction expands the ORDDicom object, allow room
pctfree 60
--
-- Use SecureFile LOBS for binary content
--
lob(dicom.source.localdata) store as SecureFile
  (nocache filesystem_like_logging),
lob(imageThumb.source.localdata) store as SecureFile
  (nocache filesystem_like_logging),
lob(anonDicom.source.localdata) store as SecureFile
  (nocache filesystem_like_logging),
--
-- disable in row storage for the extension
-- so that it does not consume page space
-- it is usually < 4k in size
--
lob(dicom.extension) store as SecureFile
  ( nocache disable storage in row ),
lob(anonDicom.extension) store as SecureFile
  ( nocache disable storage in row ),
--
-- store the metadata as a CLOB,
-- disable storage in row
```



```
--
xmltype dicom.metadata store as SecureFile clob
  ( nocache disable storage in row )
xmltype anonDicom.metadata store as SecureFile clob
  ( nocache disable storage in row )
;
```

[Example 5-1](#) uses SecureFile LOB storage for the media content. Oracle SecureFiles is a re-engineered binary large object (BLOB) that improves performance and strengthens the content management capabilities of Oracle Database.

See Also:

- *Oracle Database SecureFiles and Large Objects Developer's Guide* for more information about SecureFile LOBs
- *Oracle Multimedia User's Guide* for tuning tips with SecureFile LOBs
- *Oracle Database Security Guide* for more information about creating secure passwords

5.3 Loading DICOM Content Using the SQL*Loader Utility

This section shows how to use the SQL*Loader utility to load DICOM content into an existing table in Oracle Database. SQL*Loader is a high-performance utility for loading data from external files into tables in an Oracle database. The external data can be loaded across a network from a client system that differs from the system that is running the server for Oracle Database. The data can also be loaded locally on the same system as the database server.

A typical SQL*Loader session accepts a control file and one or more data files as input. The control file defines how to load the data into the database. The output of the SQL*Loader session is an Oracle database (where the data is loaded), a log file, and potentially, a discard file.

[Example 5-2](#) shows a control file for loading DICOM data into the table `medical_image_table`, which you created in [Example 5-1](#). The control file contains directives that map the input data, which is specified at the end of the control file as `sample1.dcm` and `sample2.dcm`, to the columns of the table `medical_image_table`. Only the `id` and `dicom` columns are loaded with externally supplied data. The `imageThumb` and `anonDicom` columns are initialized using constant and default values that are supplied in the control file.

Example 5-2 Loading DICOM Content

```
-- This file is a SQL*LDR control file to load DICOM data
-- into the table MEDICAL_IMAGE_TABLE. The control file contains directives
-- to load DICOM data into the DICOM column. It also contains directives
-- to initialize the IMAGETHUMB and ANONDICOM columns. The data to be loaded
-- is specified in this file after the BEGINDATA delimiter.
--
-- The following command invokes the SQL*Loader utility and then prompts you
-- to enter the password for the specified userid.
--
--   sqlldr userid=USER control=load_dicom.ctl
--
--
load data
--
```

```

-- The input data is contained in this file after the BEGINDATA delimiter.
--
infile *
into table medical_image_table
--
-- This example truncates the table. Change the following to "append"
-- if you want to add to an existing table.
--
truncate
fields terminated by whitespace optionally enclosed by '''
(
  --
  -- The primary key column.
  --
  id          integer external,
  --
  -- A filler field that holds the file path of the DICOM data.
  --
  dicomFilename  filler char,
  --
  -- Load the dicom column object
  --   The LOB attribute source.localData is loaded with the DICOM data.
  --   The srcType attribute is initialized to "local".
  --   The updateTime attribute is initialized to "SYSDATE".
  --   The LOB attribute extension is initialized with an empty LOB.
  --
  dicom          column object (
    source        column object (
      localData    lobfile(dicomFilename) terminated by EOF,
      srcType      constant 'local',
      updateTime   expression "SYSDATE"
    ),
    extension     lobfile(dicomFilename) terminated by EOF
                  defaultif dicom.source.srcType='local'
  ),
  --
  -- Initialize the imageThumb column object
  --   The LOB attribute source.localData is initialized with an empty LOB.
  --   This LOB will hold the content for the thumbnail image.
  --   The local attribute is initialized to "1".
  --
  imageThumb     column object (
    source        column object (
      localData    lobfile(dicomFilename) terminated by EOF
                  defaultif imageThumb.source.local=X'1',
      local        constant 1
    )
  ),
  --
  -- Initialize the anonDicom column object
  --   The LOB attributes source.localData and extension are initialized.
  --   with empty LOBs.
  --   The localData LOB will hold the content for the DICOM data to be
  --   made anonymous.
  --   The extension LOB is an internal field used by ORDDICOM.
  --   The srcType attribute is initialized to "local".
  --
  anonDicom      column object (
    source        column object (
      localData    lobfile(dicomFilename) terminated by EOF

```

```

                                defaultif anonDicom.source.srcType='local',
srcType                          constant 'local'
),
extension                          lobfile(dicomFilename) terminated by EOF
                                defaultif dicom.source.srcType='local'
)
)
--
-- Input data begins here
--
-- ID  DICOMFILENAME
BEGINDATA
  1   sample1.dcm
  2   sample2.dcm

```

Before invoking the SQL*Loader utility, you can temporarily disable logging for the LOB data to be loaded into the `dicom` column. When logging is disabled, the data is written to the database table only, and not to the redo log. Disabling logging can reduce the amount of time needed to load the DICOM data by cutting in half the amount of I/O to be performed.

To disable logging for the DICOM content in the `dicom` column, use the following SQL command:

```
alter table medical_image_table modify lob(dicom.source.localData) (nocache nologging);
```

To invoke the SQL*Loader utility, use the following command, then enter the password when prompted:

```
sqlldr userid=USER control=load_dicom.ctl
```

After the DICOM data is loaded, use the following SQL command to re-enable logging for the DICOM content in the `dicom` column:

```
alter table medical_image_table modify lob(dicom.source.localData) (nocache logging);
```

After the DICOM data is loaded into the table from the external files, another program is required to complete the initialization of the `dicom` column and to generate the data to populate the `imageThumb` and `anonDicom` columns. These tasks can be performed using methods of the `ORDDicom` object, as shown in [Example 5-3](#) (which includes a PL/SQL block). In this example, the `dicom` column is initialized using the `setProperties()` method. The `imageThumb` column object is created using the `processCopy()` method. And, the `anonDicom` column object is created using the `makeAnonymous()` method, which requires a unique identifier for one of its input arguments.

Note: [Example 5-3](#) is a complete code sample that includes a call to the `setDataModel()` procedure. Keep in mind that this procedure call might not be required in all situations.

See the [setDataModel\(\) Procedure](#) for more information about when to call this procedure.

Example 5-3 Finish Loading and Initializing the DICOM Table

```

--
-- The ORDDicom method makeAnonymous() takes a unique UID as an input parameter.
-- This utility function generates a simple UID to use in this example.
-- Replace the string value of UID_ROOT with the DICOM UID for your organization.
--

```

```

create or replace function genUID(in_id varchar2)
return varchar2
is
  -- Declare the DICOM UID root value for your organization
  -- You must replace this value.
  UID_ROOT varchar2(128) := '<unique-UID root>';
begin
  return UID_ROOT || '.' || in_id;
end;
/
show errors;

--
-- This PL/SQL block loops over all the rows in the MEDICAL_IMAGE_TABLE and:
-- 1. Calls the ORDDicom method setProperties() to initialize the dicom column
-- 2. Calls the ORDDicom method processCopy() to create a JPEG thumbnail image
--    that is stored in the imageThumb column.
-- 3. Calls the ORDDicom method makeAnonymous() to create an anonymous version
--    of the dicom column. The new version is stored in the column anonDicom.
--
declare
  dcm ordsys.orddicom;
begin
  -- load the DICOM data model
  ord_dicom.setDatamodel;

  -- loop over all rows in the medical image table
  for rec in (select * from medical_image_table for update) loop

    -- initialize the dicom column
    rec.dicom.setProperties();

    -- create a JPEG thumbnail
    rec.dicom.processCopy('fileFormat=jpeg fixedScale=75,100', rec.imageThumb);

    -- make a new anonymous version of the ORDDicom object
    rec.dicom.makeAnonymous(genUID(rec.id), rec.anonDicom);
    -- write the objects back to the row
    update medical_image_table
    set dicom = rec.dicom,
        imageThumb = rec.imageThumb,
        anonDicom = rec.anonDicom
    where id = rec.id;

  end loop;
  commit;
end;
/

```

Example 5-3 defines a function `genUID()` to generate a unique identifier (UID) by concatenating the value of the `id` column with a DICOM UID root value that you must define. You can replace this function with another function that generates unique UIDs, in accordance with the standards for your organization.

The PL/SQL block in **Example 5-3** loops once over all the rows in the table `medical_image_table`. Then, it reads and accesses each DICOM image in three passes. The first pass sets the properties of the `dicom` column. The second pass creates a JPEG thumbnail image. And, the third pass creates an anonymous DICOM image to store in the `anonDicom` column. Because of these repeated read operations, you may want to

alter the LOB storage property of the `dicom` column to enable caching of the DICOM content.

To enable caching for the DICOM content in the `dicom` column, use the following SQL command:

```
alter table medical_image_table modify lob(dicom.source.localData) (cache);
```

After the initialization is complete, use the following SQL command to disable caching for the DICOM content in the `dicom` column:

```
alter table medical_image_table modify lob(dicom.source.localData) (nocache logging);
```

[Section 5.4](#) describes this and other PL/SQL programs in more detail.

See Also:

- *Oracle Database Utilities* for more information about using the SQL*Loader utility to load objects and LOBs into Oracle Database
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for more information about LOBs and logging

5.4 Developing DICOM Applications Using the PL/SQL API

This section builds on the code examples in [Section 5.2](#) and [Section 5.3](#). It shows PL/SQL code examples that store and manipulate DICOM content inside a database using Oracle Multimedia DICOM.

Oracle Multimedia DICOM enables you to store DICOM content in database tables with columns of type `ORDDicom`. [Table 5–1](#) shows some attributes that are contained within an `ORDDicom` object in a database table.

Table 5–1 Sample Contents of an ORDDicom Object in a Database Table

ORDDicom
SOP_INSTANCE_UID
SOP_CLASS_UID
STUDY_INSTANCE_UID
SERIES_INSTANCE_UID
Source (ORDDataSource)
Metadata (SYS.XMLType)
ContentLength (integer)
Internal attributes

The following subsections describe sample tasks you can perform in PL/SQL using the code examples in this section:

- [Selecting DICOM Attributes](#)
- [Creating Thumbnail Images and Changing Image Formats](#)
- [Making Anonymous Copies of ORDDicom Objects](#)
- [Checking the Conformance of ORDDicom Objects](#)
- [Handling Oracle Multimedia DICOM Exceptions in PL/SQL](#)

5.4.1 Selecting DICOM Attributes

This section shows how to access DICOM attributes from the DICOM content that you loaded in [Section 5.3](#).

After the table `medical_image_table` is populated and metadata has been extracted, you can access metadata using SQL queries. [Example 5-4](#) demonstrates how to select extracted DICOM metadata from the DICOM content.

Example 5-4 Selecting Metadata from the DICOM Content

1. SOP_INSTANCE_UID.
2. SOP_CLASS_UID
3. STUDY_INSTANCE_UID
4. SERIES_INSTANCE_UID.
5. Content length (number of bytes of DICOM content)
6. Patient Name, Patient ID, and Modality from DICOM metadata

```

select id,
       t.dicom.getSOPInstanceUID() as SOP_Instance_UID
from medical_image_table t;

select id,
       t.dicom.getSOPClassUID() as SOP_Class_UID
from medical_image_table t;

select id,
       t.dicom.getStudyInstanceUID() as Study_Instance_UID
from medical_image_table t;

select id,
       t.dicom.getSeriesInstanceUID() as Series_Instance_UID
from medical_image_table t;

select id,
       t.dicom.getcontentlength() as content_Length
from medical_image_table t;

select m.id, t.PATIENT_NAME, t.PATIENT_ID, t.MODALITY
from medical_image_table m,
     xmltable
     (xmlnsnamespaces
      (default 'http://xmlns.oracle.com/ord/dicom/metadata_1_0'),
      '/DICOM_OBJECT'
      passing m.dicom.metadata
      columns
        patient_name varchar2(100)
          path '.*[@name="Patient''''s Name']/VALUE',
        patient_id   varchar2(100)
          path '.*[@name="Patient ID"]',
        modality     varchar2(100)
          path '.*[@name="Modality"]'
     ) t ;

```

Running [Example 5-4](#) generates the following output:

```

ID SOP_INSTANCE_UID
-----
1 1.2.392.200036.9116.2.2.2.1762676206.1077529882.102147

```

```

ID SOP_CLASS_UID
-----
1 1.2.840.10008.5.1.4.1.1.2

ID STUDY_INSTANCE_UID
-----
1 1.2.392.200036.9116.2.2.2.1762929498.1080638122.365416

ID SERIES_INSTANCE_UID
-----
1 1.2.392.200036.9116.2.2.2.1762929498.1080638122.503288

ID CONTENT_LENGTH
-----
1 525974

ID PATIENT_NAME          PATIENT_ID          MODALITY
-----
1 CANCIO 2HR           A-02-013           CT

```

5.4.2 Creating Thumbnail Images and Changing Image Formats

This section demonstrates some image processing operations that can be invoked within the database.

As an example, to create a JPEG thumbnail image from a DICOM image, you generate a new `ORDImage` object from the `ORDDicom` object. Before you can complete this task, you must describe the desired properties of the new `ORDImage` object.

The following description generates a JPEG thumbnail image of size 75x100 pixels:

```
'fileFormat=jfif fixedScale=75 100'
```

The code segment shown in [Example 5-5](#) defines the procedure `generate_thumb()`, which performs these tasks:

- Populates the column `imageThumb` of the table `medical_image_table` with the identifier `source_id`.
- Generates an `ORDImage` object in the column by invoking the `processCopy()` method on the `ORDDicom` object in the source row.

The code statements in [Example 5-5](#) where these tasks are performed are highlighted in bold.

Note: [Example 5-5](#) is a complete code sample that includes a call to the `setDataModel()` procedure. Keep in mind that this procedure call might not be required in all situations.

See the [setDataModel\(\) Procedure](#) for more information about when to call this procedure.

Example 5-5 *Generating and Processing the New `ORDImage` Object*

```

-- Set Data Model Repository
execute ordsys.ord_dicom.setDataModel();

create or replace procedure generate_thumb(source_id number, verb varchar2) is
    dcmSrc    ordsys.orddicom;
    imgDst    ordsys.ordimage;

```

```

begin
  select dicom, imageThumb into dcmSrc, imgDst from medical_image_table
     where id = source_id for update;
  dcmSrc.processCopy(verb, imgDst);

  update medical_image_table set imageThumb = imgDst where id = source_id;
  commit;
end;
/

-- Create a JPEG thumbnail image for our test DICOM
execute generate_thumb(1, 'fileFormat=jfif fixedScale=75 100');

-- look at our handiwork
column t.imageThumb.getFileFormat() format A20;
select id, t.imageThumb.getWidth(), t.imageThumb.getHeight(),
       t.imageThumb.getFileFormat()
from medical_image_table t;

```

Running [Example 5-5](#) generates the following output:

```

ID  T.IMAGETHUMB.GETWIDTH()  T.IMAGETHUMB.GETHEIGHT()  T.IMAGETHUMB.GETFILE
-----
1           75                100                JFIF

```

5.4.3 Making Anonymous Copies of ORDDicom Objects

This section shows how to protect patient privacy by making ORDDicom objects anonymous.

To make ORDDicom objects anonymous, you must create a new ORDDicom object in which certain user-specifiable DICOM attributes have either been removed or overwritten in both the new DICOM content and the associated ORDDicom object metadata. An XML anonymity document specifies which DICOM attributes to remove or replace and what action to take to make each attribute anonymous.

The default anonymity document, `orddman.xml`, is loaded during installation. You can create a customized anonymity document, but that topic is beyond the scope of this example. This example uses the default anonymity document.

The code segment in [Example 5-6](#) defines the procedure `generate_anon()`, which performs these tasks:

- Selects the original content `dicom` and the column `anonDicom` of the table `medical_image_table` with the identifier `source_id`.
- Generates an ORDDicom object in the column `anonDicom` by calling the `makeAnonymous()` method on the `dicom` in the source row.

If you run this code segment, replace the temporary UID for the variable `dest_sop_instance_uid` in the procedure `generate_anon` with a globally unique UID.

The code statement in [Example 5-6](#) where the `makeAnonymous()` method is called is highlighted in bold.

Note: Example 5–6 is a complete code sample that includes a call to the `setDataModel()` procedure. Keep in mind that this procedure call might not be required in all situations.

See the [setDataModel\(\) Procedure](#) for more information about when to call this procedure.

Example 5–6 Populating the Column and Generating an Anonymous ORDDicom Object

```
-- Set Data Model Repository

execute ordsys.ord_dicom.setDataModel();

create or replace procedure generate_anon(source_id number) is
    dcmSrc    ordsys.orddicom;
    anonDst   ordsys.orddicom;
    dest_sop_inst_uid varchar2(128) := '1.2.3';

begin
    select dicom, anonDicom into dcmSrc, anonDst from medical_image_table
        where id = source_id for update;
    dcmSrc.makeAnonymous(dest_sop_inst_uid, anonDst);
    update medical_image_table set anonDicom = anonDst where id = source_id;
    commit;
end;
/

-- Generate an Anonymous Copy of our test DICOM
execute generate_anon(1);

select m.id, t.PATIENT_NAME, t.PATIENT_ID
from medical_image_table m,
    xmltable
        (xmlnamespaces
            (default 'http://xmlns.oracle.com/ord/dicom/metadata_1_0'),
            '/DICOM_OBJECT'
        passing m.anonDicom.metadata
        columns
            patient_name varchar2(100)
                path '.*[@name="Patient"'s Name']/VALUE',
            patient_id   varchar2(100)
                path '.*[@name="Patient ID"]'
        ) t ;
```

Running Example 5–6 generates the following output:

ID	PATIENT_NAME	PATIENT_ID
1	anonymous	anonymous

5.4.4 Checking the Conformance of ORDDicom Objects

This section shows how to check the conformance of ORDDicom objects against a set of user-specified constraint rules. Constraint rules are specified in one or more constraint documents. These XML documents specify attribute relationships and semantic constraints that cannot be expressed by the DICOM metadata schema.

A default constraint document, `ordcmct.xml`, is loaded during installation. You can create a customized constraint document, but that topic is beyond the scope of this example. This example uses the default constraint document.

The code segment in [Example 5-7](#) defines the procedure `check_conform()`, which performs these tasks:

- Selects the original content `dicom` of the table `medical_image_table` with the identifier `source_id`.
- Displays a line of output text, which indicates either of these conditions for the DICOM content:

- Conformance valid

```
isconformanceValid(OracleOrdObject): 1
```

- Not conformance valid

```
isconformanceValid(OracleOrdObject): 0
```

Note: [Example 5-7](#) is a complete code sample that includes a call to the `setDataModel()` procedure. Keep in mind that this procedure call might not be required in all situations.

See the [setDataModel\(\) Procedure](#) for more information about when to call this procedure.

Example 5-7 Checking DICOM Conformance

```
-- Set Data Model Repository
execute ordsys.ord_dicom.setDataModel();

create or replace procedure check_conform(source_id number) is
    dcmSrc    ordsys.orddicom;
begin
    select dicom into dcmSrc from medical_image_table
        where id = source_id;
    dbms_output.put_line('isconformanceValid(OracleOrdObject): ' ||
        dcmSrc.isConformanceValid('OracleOrdObject'));
end;
/
show errors;
```

Running [Example 5-7](#) generates the following output:

```
SQL> execute check_conform(1) ;
isconformanceValid(OracleOrdObject): 1
```

The value of 1 indicates that the DICOM content used in this example was valid because it conformed to the Oracle default constraint rules. If the DICOM content had not been valid, running the example would have returned a value of 0. And, one or more constraint messages generated during the previous conformance validation would have been found in the view `orddcm_conformance_vld_msgs`.

The following code segment shows the description of this view:

```
SQL> describe orddcm_conformance_vld_msgs;
```

Name	Null?	Type
SOP_INSTANCE_UID		VARCHAR2(128 CHAR)

RULE_NAME	NOT NULL	VARCHAR2 (64 CHAR)
MESSAGE		VARCHAR2 (1999 CHAR)
MSG_TYPE	NOT NULL	VARCHAR2 (20 CHAR)
MSG_TIME	NOT NULL	TIMESTAMP (6)

You can query this view to examine any constraint messages that are generated during conformance validation. Because the DICOM content used in this example conformed with the Oracle constraint rules, there are no messages in the `orddcm_conformance_vld_msgs` view.

```
select * from orddcm_conformance_vld_msgs;
```

Thus, invoking the preceding select query generates the following output:

```
no rows selected
```

See [Section 3.2.8](#) for information about what to do if your DICOM content does not conform to the constraint rules defined for your organization. See [DICOM Repository Public Views](#) for information about the view `orddcm_conformance_vld_msgs`.

5.4.5 Handling Oracle Multimedia DICOM Exceptions in PL/SQL

Possible errors that can occur during run time should always be handled in your application. This practice enables the program to continue its operation even when it encounters a run-time error. This practice also enables you to know what went wrong during program operation. Proper error handling practices ensure that, whenever possible, you are always able to recover from an error while running an application. In addition, proper error handling provides you with the information you need so you always know what went wrong.

When handling exceptions, PL/SQL uses exception blocks. For example, in PL/SQL, the exception can appear as:

```
BEGIN
<some program logic>
EXCEPTION
    WHEN OTHERS THEN
        <some exception logic>
END;
```

When you design, code, and debug your application, you are aware of the places in your program where processing might stop due to a failure to anticipate an error. Those are the places in your program where you must add exception handling blocks to handle the potential errors.

See Also:

Oracle Database PL/SQL Language Reference for more information about handling PL/SQL exceptions

5.5 Developing DICOM Applications Using the DICOM Java API

Developers who are familiar with Java and Java Database Connectivity (JDBC) can write DICOM applications using Oracle Multimedia DICOM Java API. The `OrdDicom` class in Oracle Multimedia DICOM Java API is the Java proxy class for the `ORDDicom` database object. This class enables developers to write Java applications using the Oracle Multimedia object designed to store Digital Imaging and Communications in Medicine (DICOM) content.

This Java class is included in the `oracle.ord.dicom.*` package. This class is used to represent an instance of the ORDSYS.ORDDicom database object type in a Java application.

The following subsections describe tasks you must perform when developing DICOM applications in Java:

- [Setting Up Your Environment Variables](#)
- [Importing Oracle Java Classes into Your Application](#)
- [Handling Oracle Multimedia DICOM Exceptions in Java](#)

See Also:

See *Oracle Multimedia DICOM Java API Reference* for more information about the available methods in this class.

5.5.1 Setting Up Your Environment Variables

Before you can begin using Oracle Multimedia DICOM Java API, you must set up your environment to compile and run Java programs. First, you must specify the environment variable CLASSPATH. In addition, you must ensure that this variable includes the appropriate Oracle Java libraries for the Oracle Multimedia and other features that you intend to use.

See Also:

Oracle Multimedia User's Guide for complete details about this setup

5.5.2 Importing Oracle Java Classes into Your Application

After setting up your environment variables and including the appropriate Oracle Java libraries, you must include the appropriate import statements in your Java application before using Oracle Multimedia DICOM Java API.

Execute the following statements to import the required classes from the `oracle.ord.dicom.*` package and the `oracle.ord.im.*` package:

```
import oracle.ord.dicom.OrdDicom;
import oracle.ord.im.OrdImage;
```

Along with the standard JDBC classes included in the `java.sql` package, you must also import the Oracle JDBC extension class `oracle.jdbc.OracleResultSet`, with the following statement:

```
import oracle.jdbc.OracleResultSet;
```

5.5.3 Handling Oracle Multimedia DICOM Exceptions in Java

Possible errors that can occur during run time should always be handled in your application. This practice enables the program to continue its operation even when it encounters a run-time error. This practice also enables you to know what went wrong during program operation. Proper error handling practices ensure that, whenever possible, you are always able to recover from an error while running an application. In addition, proper error handling provides you with the information you need so you always know what went wrong.

When handling exceptions, Java uses the try/catch block. For example, in Java, the exception can appear as:

```
try {
```

```
        //<some program logic>
    }
    catch (exceptionName a) {
        //Exception logic
    }
    finally {
        //Execute logic if try block is executed even if an exception is caught
    }
}
```

When you design, code, and debug your application, you are aware of the places in your program where processing might stop due to a failure to anticipate an error. Those are the places in your program where you must add exception handling blocks to handle the potential errors.

See Also:

- *Oracle Database Java Developer's Guide* for more information about handling Java exceptions
- *Oracle Database JDBC Developer's Guide* for more information about handling Java exceptions using JDBC

DICOM Sample Application

This chapter describes the Oracle Multimedia DICOM Image Archive Demonstration, a DICOM sample application for Oracle Application Express, which you can use to build your own custom DICOM applications.

This chapter includes these sections:

- [Overview of the DICOM Sample Application](#) on page 6-1
- [Description of the DICOM Sample Application](#) on page 6-3

See [Appendix E](#) for the location of this sample application.

Note: This DICOM sample application is undergoing continuous improvements. Thus, the information in this chapter might not match the sample application shipped on OTN.

See the online `readme.txt` file that is included in the ZIP file with the Oracle Multimedia DICOM Image Archive Demonstration for information about the latest version of this sample application, and for complete requirements and instructions for installing, deinstalling, and using it.

6.1 Overview of the DICOM Sample Application

The DICOM sample application demonstrates many of the features of Oracle Multimedia DICOM. These features are as follows:

- Storing and retrieving medical imaging data in the database to synchronize the DICOM data with the associated business data
- Full object interfaces to Oracle Multimedia DICOM features
- Extracting DICOM metadata according to user-specifiable XML schemas
- Querying using extracted metadata, including querying using text that has a semantic relationship with text in the extracted metadata
- Image processing, such as generating thumbnail images
- Creating new DICOM objects
- Validating conformance based on a set of user-specified conformance rules
- Making DICOM objects anonymous based on user-defined rules that specify the set of attributes to be made anonymous and how to make those attributes anonymous

- The ability to update run-time behaviors, such as constraint rules and anonymity behavior, without installing a new release of Oracle Database

The DICOM sample application is demonstrated in a browser. Because a DICOM viewer is not available, full-size JPEG images and JPEG thumbnail images are created using Oracle Multimedia DICOM features. These images are stored in a main archive table, along with the original DICOM images. Additionally, all the metadata from the DICOM images is extracted from the DICOM images to enable searching based on DICOM metadata.

Creating the Main Archive Table

Upon installation of the sample application, the main archive table `dicom_archive` is created with the following columns:

- `id` - an integer used as the primary key
- `parent_id` - an integer that identifies the original image from which the DICOM image is created (used when new DICOM content is created; for example: when making an anonymous copy of a DICOM image)
- `dcm_filename` - the file name of the DICOM image that was imported
- `description` - a text description of the DICOM image
- `dicom` - the DICOM image stored as an ORDDicom object type
- `image` - the DICOM image in JPEG format
- `thumb` - a JPEG thumbnail image of the original DICOM image
- `metadata` - the user-defined metadata stored as a data type XMLType
- `isanonymous` - a flag that indicates whether the DICOM image and its associated metadata were made anonymous; if so, researchers have the ability to access the DICOM image and its metadata

The code segment shown in [Example 6-1](#) creates the table `dicom_archive`.

Example 6-1 Script to Create the Main Archive Table

```
--
-- the main archive table
--
create table dicom_archive
(
  id          integer not null primary key,
  parent_id  integer,          -- where this image is created from
  dcm_filename varchar2(60),    -- dicom image file name from import
  description varchar2(100),   -- description of the image
  dicom      orddicom,        -- dicom data
  image      ordimage,        -- dicom data in jpeg format
  thumb      ordimage,        -- dicom data in jpeg thumbnail
  metadata   xmltype,         -- user customized metadata
  isanonymous integer         -- accessible flag for the research role.
)
-- use pctfree parameter to avoid too many chained rows
pctfree 60
lob (dicom.source.localdata) store as securefile (nocache disable storage in row),
-- disable in row storage for the extension
-- so that it does not consume page space
-- it is usually < 4k in size
lob (dicom.extension) store as securefile (nocache disable storage in row),
-- store the metadata attribute of the orddicom object as clob,
```



```

-- which is full-text indexed. this is usually stored out of line
-- because it is size > 4k
xmltype dicom.metadata store as securefile clob (nocache),
lob (image.source.localdata) store as securefile (nocache disable storage in row),
lob (thumb.source.localdata) store as securefile (nocache),
-- bind the table with the schema so that the
-- the metadata column is shredded stored.
xmltype column metadata
xmlschema "http://xmlns.oracle.com/ord/meta/dia_demo"
element dicom_image;
/

```

Note: The preceding code segment uses SecureFile LOB storage for the media content.

After the main archive table is created, DICOM data can be loaded. When data is loaded, all the metadata from the DICOM images is extracted to enable searching. In addition, JPEG full-size images and thumbnail images are generated for display in a Web browser.

See Also:

- *Oracle Multimedia User's Guide* for tuning tips with SecureFile LOBs
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for more information about SecureFile LOBs

6.2 Description of the DICOM Sample Application

The DICOM sample application includes these user interfaces:

- DICOM Image Archive
- DICOM Image Archive Administration

Each user interface in the sample application has a separate login page. When logging in to the DICOM Image Archive interface, clinicians and researchers must identify their roles. Administrators are automatically identified when they log in to the DICOM Image Archive Administration interface.

The DICOM Image Archive interface demonstrates the kinds of user tasks that clinicians and researchers can perform to complete one or more operations on the sample DICOM content.

Table 6–1 lists the all the tasks that are demonstrated in the DICOM Image Archive interface of the DICOM sample application, and indicates which type of user can perform each task.

Table 6–1 *DICOM Image Archive Interface: Tasks and Designated Users*

Task	User
Adding DICOM images to the archive (import)	Clinician
Downloading selected DICOM images (export)	Clinician Researcher
Converting and downloading DICOM images	Clinician

Table 6–1 (Cont.) DICOM Image Archive Interface: Tasks and Designated Users

Task	User
Processing DICOM images (processCopy)	Clinician Researcher
Converting and loading scanned non-DICOM images (createDicomImage)	Clinician Researcher
Compressing and deleting selected DICOM images	Clinician
Editing metadata of selected DICOM images	Clinician
Deleting selected DICOM images from the archive	Clinician
Making selected DICOM images anonymous	Clinician
Validating the conformance of selected DICOM images	Clinician Researcher

The DICOM Image Archive Administration interface shows the tasks that only administrators can perform to manage the sample DICOM data model repository and the configuration documents stored within it.

[Table 6–2](#) lists the all the administrator tasks that are demonstrated in the DICOM Image Archive Administration interface of the DICOM sample application.

Table 6–2 DICOM Image Archive Administration Interface: Tasks

Task
Exporting documents from the DICOM repository (Download)
Inserting documents into the DICOM repository (Add)
Deleting documents from the DICOM repository (Delete)

In addition, several search capabilities are available to help users find specific DICOM images that are stored in the archive. [Table 6–3](#) summarizes the types of searches users can perform.

Table 6–3 DICOM Sample Application: Search Capabilities

Type of Search	Description
Searches on DICOM attributes	Require an exact (ignoring case) match for the text you enter in one or more DICOM attribute fields. (See Section 6.2.2.1 .)
Searches on keywords	Require an exact match (ignoring case) for the text you enter in the Keyword field. (See Section 6.2.2.2 .)
Searches on semantic data	Require a semantically related match for the text you enter in the Semantic field. (See Section 6.2.2.3). See also <i>Oracle Database Semantic Technologies Developer's Guide</i> .

The following subsections describe how to perform a few common tasks for each user interface in the DICOM sample application:

- [Logging In to the DICOM Image Archive Interface](#)
- [Searching for Specific DICOM Images](#)
- [Importing DICOM Images as a Clinician](#)
- [Processing DICOM Images as a Researcher](#)

- [Logging In to the DICOM Image Archive Administration Interface](#)
- [Inserting Configuration Documents](#)

6.2.1 Logging In to the DICOM Image Archive Interface

Before you can begin to work on any of the user tasks, you must log in to the DICOM Image Archive interface and identify your role as a clinician or researcher.


Logging In as a Clinician

Follow these steps to log in as a clinician:

1. In the **Login As** field, select **Clinician**.

[Figure 6–1](#) shows the login page for a user with the role of clinician selected.

Figure 6–1 Login Page for Clinicians and Researchers

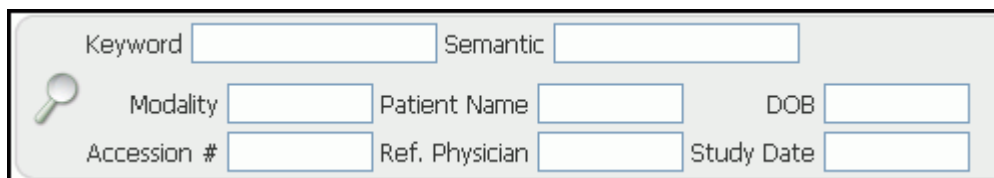


2. Enter your user name in the **User Name** field.
3. Enter your password in the **Password** field.
4. Click **Login**.

The DICOM Image Archive **Home** page appears. This **Home** page includes a search area, a **Thumbnails** area, a **Tasks** area, and an **Image Summary** area. The following four figures show each area on the **Home** page.

[Figure 6–2](#) shows the text input fields in the search area. The search area also includes a **Reset** button and a **Search** button, to the right of the text input fields.

Figure 6–2 Search Area on the DICOM Image Archive Home Page



After logging in successfully (as a clinician or a researcher), the text input fields in the search area enable you to access various search capabilities and find specific DICOM images that are shipped with this sample application.

[Figure 6–3](#) shows the **Thumbnails** area, which contains six thumbnail images representing the DICOM images that are installed with the sample application.

Figure 6-3 **Thumbnails Area on the DICOM Image Archive Home Page**

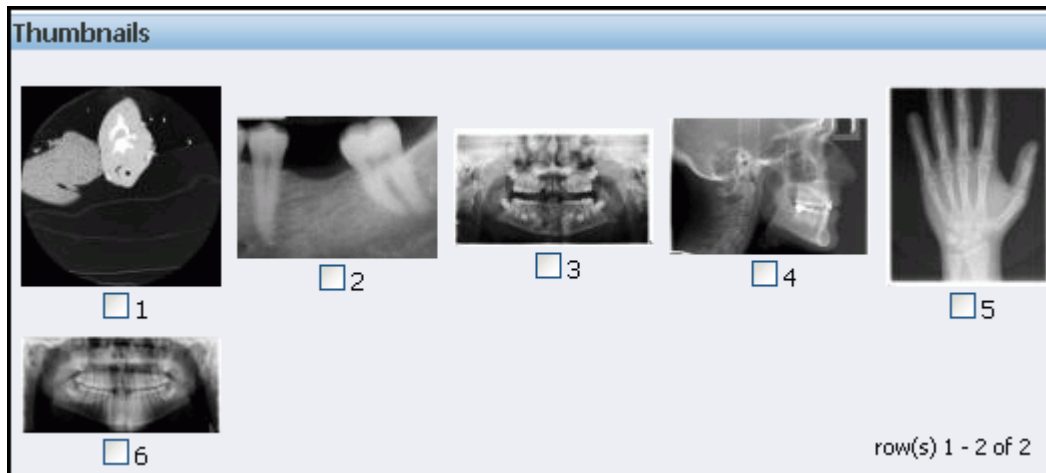
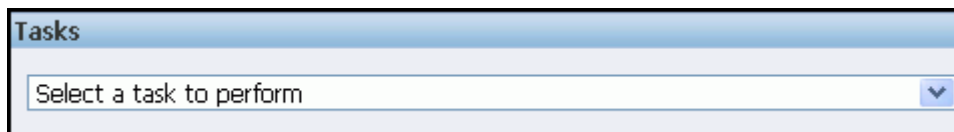


Figure 6-4 shows the **Tasks** area, with the default text in the selection field. The **Tasks** area also includes a **Start** button, to the right of the selection field.

Figure 6-4 **Tasks Area on the DICOM Image Archive Home Page**



To view all the designated user tasks for the clinician role, scroll through the items in the **Tasks** list on this **Home** page. (See [Table 6-1](#) for the list of clinician tasks.)

You can rest the mouse on each thumbnail image to view the metadata summary for that installed DICOM image. [Figure 6-5](#) shows the **Image Summary** area, with a list of fields containing the metadata summary for DICOM image 1 (**Image Summary: 1**).

Figure 6–5 Image Summary Area on the DICOM Image Archive Home Page

Image Summary
Image Summary: 1
Patient Name: CANCIO 2HR A-02-013
DOB:
Patient Sex:
Study Date: 2004-02-23
Study Time: 18:48:41.000000
Study Description:
Referring Physician:
Modality: CT
Anatomic Region:
Body Part Examined:
Image Row: 512
Image Column: 512
Bit Depth: 16
Number of Frames:
Accession #: 352
Transfer Syntax UID: 1.2.840.10008.1.2
Data Size: 525974

The metadata summary shows only a subset of the metadata for a selected image in the archive. Complete definitions of the metadata for each image are contained in the ORDDicom object in XML format. You can click each thumbnail image to view this XML metadata, and to view the full-size version of the DICOM image in JPEG format.

Logging In as a Researcher

Logging in as a researcher is similar to logging in as a clinician. Follow the same set of steps, with this exception: in Step 1, select **Researcher** in the **Login As** field.

To view all the designated user tasks for the researcher role, scroll through the items in the **Tasks** list on the DICOM Image Archive **Home** page for researchers. (See [Table 6–1](#) for the list of researcher tasks.)

Note: Researchers are permitted to access *only* images that have been made anonymous. The default images that are shipped with the DICOM sample application contain patient identifying information. Thus, the first time a researcher logs in to the DICOM sample application, no images are available. See [Section 6.2.4](#) for more information.

6.2.2 Searching for Specific DICOM Images

The DICOM sample application comes with a set of DICOM images. Clinicians and researchers can perform several kinds of searches for these images using text (such as patient name and anatomic region information), which is contained in the XML

metadata. You can use attributes, keywords, or semantic data to search on the metadata and find these DICOM images (see [Table 6-3](#)).

The following text input fields are included in the search area on the DICOM Image Archive **Home** page, as shown in [Figure 6-2](#):

- **Keyword**
- **Semantic**
- **Modality**
- **Patient Name**
- **DOB**
- **Accession #**
- **Ref. Physician**
- **Study Date**

The search area also includes a **Search** button, which you click to begin each search, and a **Reset** button, which you can click to restore the search area back to the full archive of images.

The search area is the same for clinicians and researchers. In this section, the search examples were performed as a clinician.

The following subsections describe the types of searches you can perform in the DICOM sample application:

- [Attribute Searches](#)
- [Keyword Searches](#)
- [Semantic Searches](#)

6.2.2.1 Attribute Searches

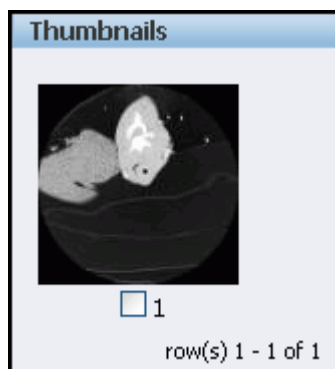
Attribute searches enable searching for DICOM images using XPath queries on the DICOM metadata, thus limiting the search to specific attributes in the DICOM metadata. These searches require a match between the text you enter in one or more DICOM attribute fields and the text contained in only those attributes in the DICOM metadata, regardless of case.

The DICOM sample application enables you to search for DICOM images using these DICOM attributes:

- **Modality**
- **Patient Name**
- **DOB**
- **Accession #**
- **Ref. Physician**
- **Study Date**

For example, to search for the DICOM image that contains the patient name CANCIO using an attribute search, enter **CANCIO** in the **Patient Name** field on the DICOM Image Archive **Home** page. Then, press Return or click **Search**.

The attribute search returns the matching image (numbered 1) of the lungs for the patient named CANCIO, as shown in the **Thumbnails** area in [Figure 6-6](#).

Figure 6–6 Thumbnail Image Result for an Attribute Search

Rest the mouse on thumbnail image 1 to view the **Image Summary** for DICOM image 1 for the patient named CANCIO (**Image Summary 1**: in [Figure 6–5](#)), as follows:

- Patient Name: CANCIO 2HR A-02-013
- DOB:
- Patient Sex:
- Study Date: 2004-02-23
- Study Time: 18:48:41.000000
- Study Description:
- Referring Physician:
- Modality: CT
- Anatomic Region:
- Body Part Examined:
- Image Row: 512
- Image Column: 512
- Bit Depth: 16
- Number of Frames:
- Accession #: 352
- Transfer Syntax UID: 1.2.840.10008.1.2
- Data Size: 525974

Click thumbnail image 1 to view the full-size version, or to view the XML metadata for DICOM image 1.

The metadata for the patient's name is defined in the <PERSON_NAME> tag, as shown in this code segment from the XML metadata for DICOM image 1:

```
<PERSON_NAME tag="00100010" definer="DICOM" name="Patient's Name" offset="710" length="22">
  <NAME type="unibyte">
    <FAMILY>CANCIO 2HR A-02-013</FAMILY>
  </NAME>
  <VALUE>CANCIO 2HR A-02-013</VALUE>
</PERSON_NAME>
```

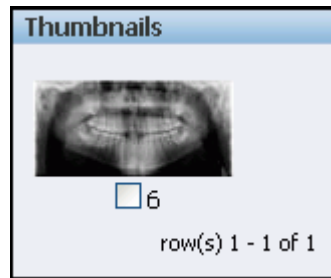
6.2.2.2 Keyword Searches

Keyword searches enable searching for DICOM images using keywords. These searches require a match between the text you enter in the **Keyword** field and the text contained in any attribute of the DICOM metadata, regardless of case.

For example, to search for dental images in the archive, enter the term `jaw` in the **Keyword** field on the DICOM Image Archive **Home** page. Then, press Return or click **Search**. This search finds images with the term `jaw` in any attribute of the DICOM metadata.

The keyword search on the term `jaw` returns one matching image (numbered 6), as shown in the **Thumbnails** area in [Figure 6-7](#).

Figure 6-7 Thumbnail Image Result for a Keyword Search



To view the metadata summary for DICOM image 6, as shown in [Figure 6-8](#), rest the mouse on thumbnail image 6.

Figure 6–8 Image Summary Result for a Keyword Search

Image Summary
Image Summary: 6
Patient Name: PLANMECA^Dimaxis
DOB: 1995-09-04
Patient Sex: M
Study Date: 2004-08-25
Study Time: 10:20:00.000000
Study Description: ProMax & Dixi
Referring Physician: D. Entist
Modality: PX
Anatomic Region: T-11100 Skull
Body Part Examined: JAW
Image Row: 516
Image Column: 1072
Bit Depth: 8
Number of Frames:
Accession #: ACCESSION99
Transfer Syntax UID: 1.2.840.10008.1.2.4.50
Data Size: 66296

A single image is returned as a match because it is the only DICOM image in the archive that contains text with the term `jaw` in the DICOM metadata, as shown in the `Body Part Examined:` field in the **Image Summary** area in [Figure 6–8](#).

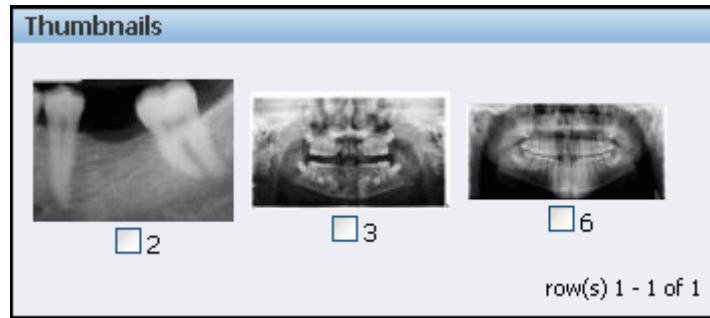
Click thumbnail image 6 to view the full-size version, or to view the XML metadata for DICOM image 6.

6.2.2.3 Semantic Searches

Semantic searches enable searching for DICOM images based on semantic relationships, which are loaded into the database during the installation of the sample application. These searches require a semantically related match between the text you enter in the **Semantic** field and the text contained in any attribute of the DICOM metadata.

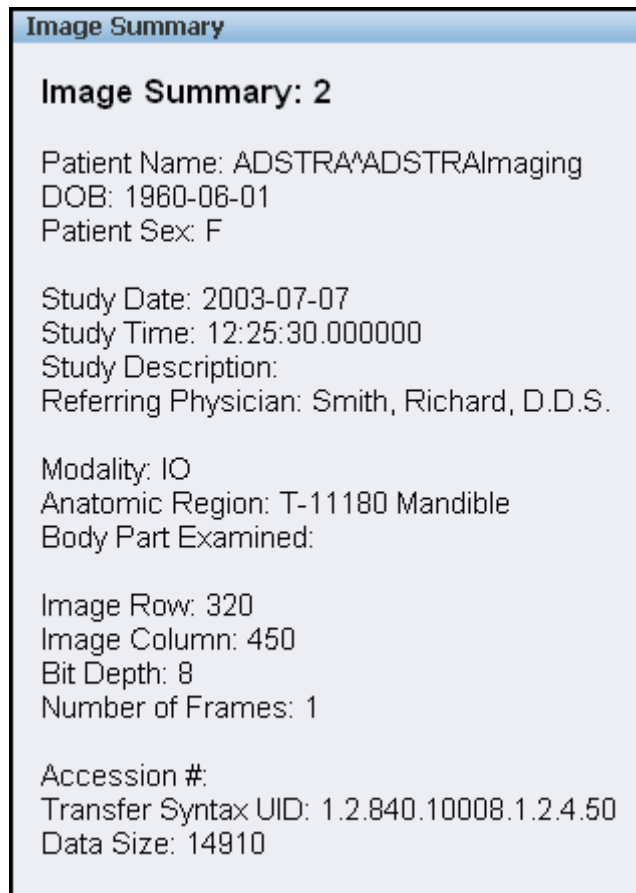
Note: Oracle Spatial must be installed before you attempt to perform a semantic search.

Using the example from the keyword search for dental images, enter the term `jaw` in the **Semantic** field on the DICOM Image Archive **Home** page. Then, press Return or click **Search**. The semantic search returns several matches, as shown in [Figure 6–9](#).

Figure 6–9 Thumbnail Image Results for a Semantic Search

Three thumbnail images are returned as matches because these three DICOM images in the archive contain metadata in the Anatomic Region attribute that is semantically related to the term `jaw`.

To view the metadata summary for a matching image, rest the mouse on its thumbnail image. For example, [Figure 6–10](#) shows the metadata summary for DICOM image 2.

Figure 6–10 Image Summary Result for a Semantic Search

For DICOM image 2, the metadata summary does not include the term `jaw`. However, the semantically related term `Mandible` is defined in the Anatomic Region Sequence section of the XML metadata for DICOM image 2. This metadata definition is highlighted in bold in the following code segment.

```

<SEQUENCE tag="00082218" definer="DICOM" name="Anatomic Region Sequence" offset="654" length="68">
  <ITEM>
    <SHORT_STRING tag="00080100" definer="DICOM" name="Code Value" offset="670"
      length="8">T-11180</SHORT_STRING>
    <SHORT_STRING tag="00080102" definer="DICOM" name="Coding Scheme Designator" offset="686"
      length="4">SNM3</SHORT_STRING>
    <LONG_STRING tag="00080104" definer="DICOM" name="Code Meaning" offset="698"
      length="8">Mandible</LONG_STRING>
  </ITEM>
</SEQUENCE>
<SEQUENCE tag="00082228" definer="DICOM" name="Primary Anatomic Structure Sequence" offset="734"
  length="176">
  <ITEM>
    <SHORT_STRING tag="00080100" definer="DICOM" name="Code Value" offset="750"
      length="8">T-54380</SHORT_STRING>
    <SHORT_STRING tag="00080102" definer="DICOM" name="Coding Scheme Designator" offset="766"
      length="4">SNM3</SHORT_STRING>
    <LONG_STRING tag="00080104" definer="DICOM" name="Code Meaning" offset="778"
      length="34">Mandibular left second molar tooth</LONG_STRING>
  </ITEM>
  <ITEM>
    <SHORT_STRING tag="00080100" definer="DICOM" name="Code Value" offset="828"
      length="8">T-54400</SHORT_STRING>
    <SHORT_STRING tag="00080102" definer="DICOM" name="Coding Scheme Designator" offset="844"
      length="4">SNM3</SHORT_STRING>
    <LONG_STRING tag="00080104" definer="DICOM" name="Code Meaning" offset="856"
      length="38">Mandibular left second premolar tooth</LONG_STRING>
  </ITEM>
</SEQUENCE>

```

For DICOM image 3, the following XML code segment shows the term **Maxilla** and **mandible** in the metadata (highlighted in bold, which is semantically related to the term jaw. (Similar to DICOM image 2, the metadata summary for DICOM image 3 does not include the term jaw.)

```

<SEQUENCE tag="00082218" definer="DICOM" name="Anatomic Region Sequence" offset="642" length="80">
  <ITEM>
    <SHORT_STRING tag="00080100" definer="DICOM" name="Code Value" offset="658"
      length="8">T-D1217</SHORT_STRING>
    <SHORT_STRING tag="00080102" definer="DICOM" name="Coding Scheme Designator" offset="674"
      length="4">SNM3</SHORT_STRING>
    <LONG_STRING tag="00080104" definer="DICOM" name="Code Meaning" offset="686"
      length="20">Maxilla and mandible</LONG_STRING>
  </ITEM>
</SEQUENCE>

```

See Also:

Oracle Database Semantic Technologies Developer's Guide for more information about Semantic Technologies

6.2.3 Importing DICOM Images as a Clinician

A common task that clinicians can perform is adding DICOM images to an archive of medical images. You can add your own DICOM images to the installed archive for the Oracle Multimedia DICOM Image Archive Demonstration.

For example, to import one or more of these images into the main archive of the DICOM sample application, follow these steps:

1. On the DICOM Image Archive **Home** page, select **Add DICOM image to the archive** from the **Tasks** list. Then, click **Start** to display the **Import DICOM Images** page shown in [Figure 6-11](#).

Figure 6–11 Import DICOM Images Page

- On the **Import DICOM Images** page, click **Browse** in the **DICOM Image File 1:** field to locate the first image to import. For this example, select dental image PLANMECA3 . dcm.

If this is the only image you want to import, click **Import** to upload it to the archive. Figure 6–12 shows the changes in some relevant areas on the DICOM Image Archive **Home** page, after importing dental image PLANMECA3 . dcm.

Figure 6–12 Importing an Image as a Clinician

In this example, the message "1 DICOM images are imported." appears at the top of the **Home** page, and thumbnail image 31 is added to the archive.

Note: For each image you add to the archive, a thumbnail image appears with the next higher number in the sequence.

3. Rest the mouse on thumbnail image 31 to view the metadata summary. Or, click thumbnail image 31 to display the full-size image and the XML metadata for DICOM image PLANMECA3 . dcm.
4. Use the remaining fields on the **Import DICOM Images** page (see [Figure 6-11](#)) to import additional images (up to five at a time), and then click **Import**. Or, click **Cancel** to terminate the operation or **Reset** to restore the archive to its previous state.

6.2.4 Processing DICOM Images as a Researcher

As a researcher, you can perform several processing operations on DICOM images in the archive, including scaling, rotating, and cropping. This example shows how to rotate an image of the lungs to better examine a specific region for further study.

Note: Researchers are permitted to access only images that contain *no* patient identifying information.

Thus, for use with the examples in this chapter:

- The set of DICOM images that is included with the installation software for the Oracle Multimedia DICOM Image Archive Demonstration was imported into the main archive.
 - Some DICOM images were made anonymous to enable researchers to see them.
-
-

For example, to process a DICOM image in the main archive of the DICOM sample application, follow these steps:

1. On the DICOM Image Archive **Home** page, select the thumbnail image of the lungs, which has been made anonymous. Then, select **Process (Rotate/Scale/Crop) the selected DICOM images** from the **Tasks** list. And, click **Start** to display the **Process DICOM Images** page shown in [Figure 6-13](#).

Figure 6–13 Process DICOM Images Page

The screenshot shows a web interface titled "Home > Process DICOM Images". Below the title is a section labeled "User Input 1".

The text in the "User Input 1" section reads: "The total number of images selected: 1" and "The IDs of the selected images are: 8".

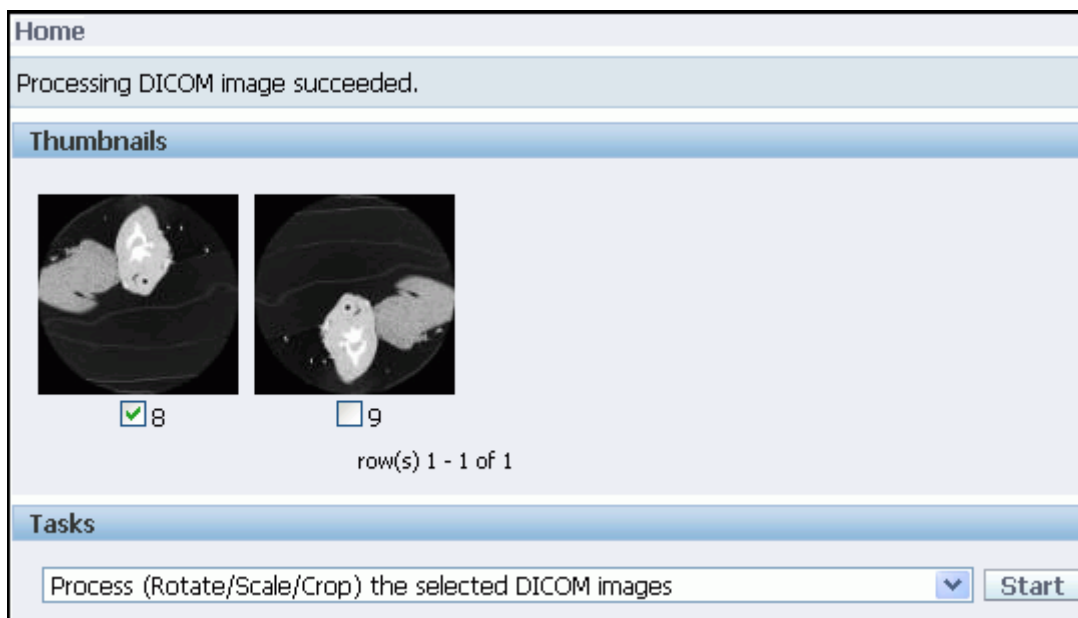
The interface is divided into four sections, each with a horizontal line above it:

- Scale:**
 - Operator: - Select Scale Operator - (dropdown menu)
 - Height: [text input field]
 - Width: [text input field]
 - Preserve aspect ratio
- Rotate:**
 - Operator: - Select Rotate Direction - (dropdown menu)
 - Degree: [text input field]
- Crop:**
 - Operator: No (dropdown menu)
 - Start X: [text input field]
 - End X: [text input field]
 - Start Y: [text input field]
 - End Y: [text input field]
- Frame:**
 - Number: [text input field]
 - Codec: -Select Codec - (dropdown menu)

At the bottom of the form are three buttons: "Cancel", "Reset", and "Process".

- On the **Process DICOM Images** page, select the processing operations you want to perform. For this example, in the **Rotate** area, select **Rotate counterclockwise** in the **Operator:** field, and enter **180** in the **Degree:** field. Then, click **Process**. Or, click **Cancel** to terminate the operation or **Reset** to restore the selected image to its previous state.

When you click **Process**, the DICOM Image Archive **Home** page displays the processed image, as shown in [Figure 6–14](#).

Figure 6–14 Processing an Image as a Researcher

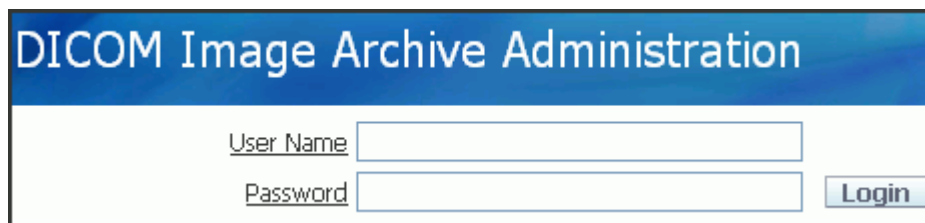
In this example, the message "Processing DICOM image succeeded." appears at the top of the **Home** page, and the processed image (thumbnail image 9, a rotated view of the lungs) appears in the archive.

3. Rest the mouse on thumbnail image 9 to view the anonymous metadata summary. Or, click thumbnail image 9 to display the full-size image and the XML metadata for DICOM image 9.
4. Use the remaining fields on the **Process DICOM Images** page (see [Figure 6–13](#)) to perform other processing operations on selected images, and then click **Process**. Or, click **Cancel** to terminate the operation or **Reset** to restore the archive to its previous state.

6.2.5 Logging In to the DICOM Image Archive Administration Interface

Before you can begin to work on any of the administrator tasks, you must log in to the DICOM Image Archive Administration interface.

[Figure 6–15](#) shows the login page for an administrator.

Figure 6–15 Login Page for Administrators

Follow these steps to log in:

1. Enter your user name in the **User Name** field.
2. Enter your password in the **Password** field.

3. Click **Login**.

The DICOM Image Archive Administration **Admin Home** page appears, as shown in [Figure 6-16](#).

Figure 6-16 Administrator Home Page

Admin Home			
Doc Name	Doc Type	Create Date	Installed By Oracle
ordcmsd.xml	STANDARD_DICTIONARY	02-OCT-08	1
ordcmpv.xml	PRIVATE_DICTIONARY	02-OCT-08	1
ordcmmp.xml	MAPPING	02-OCT-08	1
ordcman.xml	ANONYMITY	02-OCT-08	1
ordcmpf.xml	PREFERENCE	02-OCT-08	1
ordcmui.xml	UID_DEFINITION	02-OCT-08	1
ordcmcmc.xml	CONSTRAINT	02-OCT-08	1
ordcmcmd.xml	CONSTRAINT	02-OCT-08	1
ordcmct.xml	CONSTRAINT	02-OCT-08	1
dia_mp.xml	MAPPING	01-MAY-09	0

1 - 10

[Download](#) [Add](#) [Delete](#)

The list of configuration documents included with the Oracle Multimedia DICOM Image Archive Demonstration is shown in the table on the DICOM Image Archive Administration **Admin Home** page. In the last column, a 1 indicates the Oracle-defined and installed configuration documents, while a 0 indicates the user-defined configuration documents.

As an administrator, you can export, insert, or delete documents in the DICOM repository (see [Table 6-2](#)).

6.2.6 Inserting Configuration Documents

A common task that administrators can perform is inserting user-defined configuration documents into the DICOM repository. The installation software for the Oracle Multimedia DICOM Image Archive Demonstration includes a few DICOM configuration documents that you can export from and insert into the installed repository. To insert one or more user-defined configuration documents into the DICOM repository of the DICOM sample application, follow these steps:

1. On the DICOM Image Archive Administration **Admin Home** page, select **Add**. The **Add Repository Documents** page appears, with the following message:
Select the inserting document category:
2. In the **Document Category** field, from the **-Select Category-** list, select **CONSTRAINT**. Then, click **Add**. Or, click **Cancel** to terminate the operation.

If you click **Add**, the **Add Repository Documents** page reappears with the inserting document category indicated. The **Browse to select the document files:** region includes a list of input fields (**Document file 1;** **Document file 2;** and **Document file 3:**).

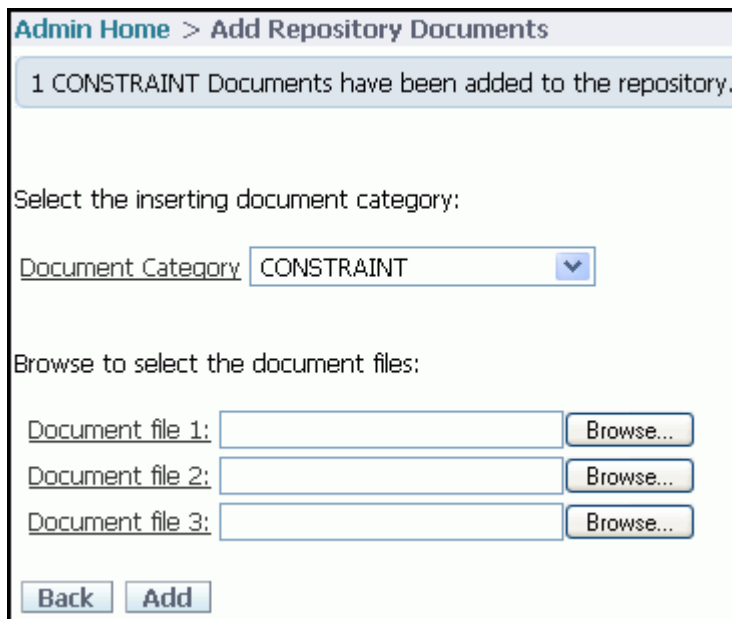
3. In the **Document file 1:** field, click **Browse...** to locate the first user-defined constraint document you want to insert into the repository. For this example, select the sample constraint file `pct_1.xml`.

If this is the only constraint document to be inserted into the repository, click **Add**. Otherwise, use the remaining **Document file** fields to insert additional constraint documents, and then click **Add**.

Optionally, you can select another type of configuration document in the **Document Category** field. Then, click **Browse...** to locate those files, and repeat these steps until all the documents have been inserted into the repository.

When a document has been inserted successfully, a message appears on the **Add Repository Documents** page, which indicates the number and type of configuration documents that you inserted into the repository, as shown in [Figure 6-17](#).

Figure 6-17 Add Repository Documents Page



The screenshot shows a web interface for adding repository documents. At the top, there is a breadcrumb trail: **Admin Home > Add Repository Documents**. Below this, a light blue message box states: "1 CONSTRAINT Documents have been added to the repository." The main form area contains the following elements:

- A label: "Select the inserting document category:"
- A dropdown menu labeled "Document Category" with "CONSTRAINT" selected.
- A label: "Browse to select the document files:"
- Three rows of input fields, each labeled "Document file 1:", "Document file 2:", and "Document file 3:", each followed by a "Browse..." button.
- At the bottom, there are two buttons: "Back" and "Add".

In this example, the message indicates that you inserted one constraint document.

When you are finished inserting documents into the repository, click **Back** to return to the updated DICOM Image Archive Administration **Admin Home** page, as shown in [Figure 6-18](#).

Figure 6–18 Administrator Home Page with a New Constraint Document

Admin Home			
Doc Name	Doc Type	Create Date	Installed By Oracle
ordcmsd.xml	STANDARD_DICTIONARY	02-OCT-08	1
ordcmpv.xml	PRIVATE_DICTIONARY	02-OCT-08	1
ordcmmp.xml	MAPPING	02-OCT-08	1
ordcman.xml	ANONYMITY	02-OCT-08	1
ordcmpf.xml	PREFERENCE	02-OCT-08	1
ordcmui.xml	UID_DEFINITION	02-OCT-08	1
ordcmcmc.xml	CONSTRAINT	02-OCT-08	1
ordcmcmd.xml	CONSTRAINT	02-OCT-08	1
ordcmct.xml	CONSTRAINT	02-OCT-08	1
dia_mp.xml	MAPPING	01-MAY-09	0
pct_1.xml	CONSTRAINT	29-MAY-09	0

1 - 11

Download Add Delete

The constraint document you inserted into the repository appears in a new row at the bottom of the table, which is highlighted in this figure. The new table row shows the name of the document (`pct_1.xml`), the type of document (`CONSTRAINT`), the creation date (`29-MAY-09`), and 0 in the **Installed By Oracle** column. The 0 indicates that the document is user-defined.

- To continue inserting other types of configuration documents into the repository from the **Add Repository Documents** page, select another type of document in the **Document Category** field, repeating the process described in Steps 2 and 3. Or, begin the process from Step 1 on the DICOM Image Archive Administration **Admin Home** page (See [Figure 6–16](#)).

When the entire process is complete, the documents you inserted into the repository appear at the bottom of the table on the DICOM Image Archive Administration **Admin Home** page. The new table row for each new document shows the name of the document, type of document, creation date, and a 0 in the **Installed By Oracle** column to indicate that the document is user-defined (see [Figure 6–18](#)).

ORDDicom Object Type Reference

Oracle Multimedia provides the ORDDicom object type, which supports the storage, management, and manipulation of DICOM format medical images and other data. The ORDDicom object is intended as an object that is written only once. To generate a new ORDDicom object by image processing or compression, create a new ORDDicom object, ORDImage object, or BLOB.

The ORDDicom object type is defined in the `ordcspec.sql` file. After installation, this file is available in the Oracle home directory at:

```
<ORACLE_HOME>/ord/im/admin (on Linux and UNIX)
```

```
<ORACLE_HOME>\ord\im\admin (on Windows)
```

This chapter describes the attributes, constructors, and methods in the ORDDicom object interface. See [Table 3-1](#) for information about other DICOM application programming interfaces (APIs).

This chapter contains these sections:

- [ORDDicom Object Examples](#) on page 7-1
- [ORDDicom Object Type](#) on page 7-3
- [ORDDicom Constructors](#) on page 7-4
- [ORDDicom Methods](#) on page 7-9

7.1 ORDDicom Object Examples

The examples in this chapter use the `MEDICAL_IMAGE_OBJ` table, which these examples create in the Product Media (PM) sample schema. See [Section 7.1.2](#) when reading through these examples.

Before using ORDDicom methods, you must load some data into the table. For example, you can use the SQL*Loader utility, a Java client, or the `import()` method. Substitute your DICOM files for those in the examples.

Note: If you manipulate the DICOM content itself (by either directly modifying the BLOB or changing the external source), call the `setProperties()` method to ensure that the object attributes stay synchronized. You must also ensure that the update time is modified. Otherwise, the object attributes will not match the DICOM content.

See the [setProperties\(\)](#) method for more information.

See Also:

See *Oracle Database Sample Schemas* for information about the PM and other sample schemas

7.1.1 Directory Definition and Setup for ORDDicom Object Examples

Issue the following statements before executing the examples, where `c:\mydir\work` is the directory where the user `pm` can find the DICOM files:

```
CREATE OR REPLACE DIRECTORY DICOMDIR as 'c:\mydir\work';
GRANT READ, WRITE ON DIRECTORY DICOMDIR TO pm;
```

Note: At the beginning of each database session, call the `setDataModel()` procedure. See the [setDataModel\(\) Procedure](#) for more information.

7.1.2 MEDICAL_IMAGE_OBJ Table Definition

Before loading data into the table, you must create the table and columns where the data is to be stored. The following PL/SQL code segment creates the `MEDICAL_IMAGE_OBJ` table with five columns.

```
CONNECT pm

Enter password: password

CREATE TABLE MEDICAL_IMAGE_OBJ
(
  id            integer primary key,
  dicom_src     ordsys.orddicom,
  dicom_dest    ordsys.orddicom,
  image_dest    ordsys.ordimage,
  blob_dest     blob
);
COMMIT;
```

where:

- `dicom_src`: the source DICOM content in the ORDDicom object.
- `dicom_dest`: the destination DICOM content in the ORDDicom object.
- `image_dest`: the destination DICOM content in the ORDImage object.
- `blob_dest`: the destination DICOM content in the BLOB.

ORDDicom Object Type

The ORDDicom object type supports the storage, management, and manipulation of DICOM format medical images and other data. The attributes for this object type are defined as follows in the `orddcspec.sql` file:

```
-----  
-- TYPE ATTRIBUTES  
-----  
SOP_INSTANCE_UID    VARCHAR2(128) ,  
SOP_CLASS_UID       VARCHAR2(64) ,  
STUDY_INSTANCE_UID  VARCHAR2(64) ,  
SERIES_INSTANCE_UID VARCHAR2(64) ,  
source              ORDDataSource ,  
metadata            SYS.XMLType ,  
contentLength       INTEGER ,  
flag                INTEGER ,  
extension           BLOB ,
```

where:

- `SOP_INSTANCE_UID`: the SOP instance UID of the embedded DICOM content.
- `SOP_CLASS_UID`: the SOP class UID of the embedded DICOM content.
- `STUDY_INSTANCE_UID`: the study instance UID of the embedded DICOM content.
- `SERIES_INSTANCE_UID`: the series instance UID of the embedded DICOM content.
- `source`: the original DICOM content stored within the database, under transaction control as a BLOB (recommended), or stored in an operating system-specific file in a local file system with pointer stored in the database.
- `metadata`: the XML metadata document extracted from the embedded DICOM content.
- `contentLength`: the length of the embedded DICOM content, in number of bytes.
- `flag`: an Oracle reserved attribute.
- `extension`: an Oracle reserved attribute.

ORDDicom Constructors

This section describes these ORDDicom constructor functions:

- [ORDDicom\(\) for BLOBs](#) on page 7-5
- [ORDDicom\(\) for ORDImage](#) on page 7-6
- [ORDDicom\(\) for Other Sources](#) on page 7-7

The ORDDicom object can be constructed using these constructors in a SQL statement or PL/SQL program.

The ORDDicom object has embedded BLOB attributes. BLOB locators must be initialized before they can be accessed. Thus, newly constructed ORDDicom objects (unless constructed from a temporary BLOB) must be inserted into a table before you can call object member methods on these ORDDicom objects.

ORDDicom() for BLOBs

Format

```
ORDDicom(SELF IN OUT NOCOPY ORDDicom,
         data IN BLOB,
         setproperties IN INTEGER DEFAULT 0)
RETURN SELF AS RESULT
```

Description

Constructs an ORDDicom object from a BLOB. The data stored in the BLOB is copied into the ORDDicom object when the constructed ORDDicom object is inserted or updated into a table. The metadata conforms to the XML schema defined by the default mapping document.

Parameters

data

Embedded DICOM content stored in a BLOB.

setproperties

Indicator flag that determines whether the DICOM attributes are extracted from the embedded DICOM content. If the value is 1, the DICOM attributes are extracted into the metadata attribute of the constructed ORDDicom object, and the attributes of the ORDDicom object are populated. If the value is 0, no DICOM attributes are extracted. The default is 0.

Pragmas

None.

Exceptions

None.

Usage Notes

Use this constructor to create an ORDDicom object when the DICOM content is stored in either a temporary or a persistent BLOB.

Examples

Create an ORDDicom object from a BLOB:

```
SQL> desc blob_tbl;
Name          Null?    Type
-----
ID             NUMBER(38)
DATA          BLOB

insert into medical_image_obj (id, dicom_src)
select s.id, ORDDicom(s.data) from blob_tbl s;
```

ORDDicom() for ORDImage

Format

```
ORDDicom(SELF IN OUT NOCOPY ORDDicom,  
         data IN ORDImage,  
         setproperties IN INTEGER DEFAULT 0)  
RETURN SELF AS RESULT
```

Description

Constructs an ORDDicom object from an ORDImage object that has either a local source (BLOB) or a file source (BFILE). If the DICOM content is stored originally in the BLOB of the ORDImage object, the data is copied into the BLOB in the ORDDicom object source attribute when the constructed ORDDicom object is inserted or updated into a table. If the DICOM content is stored originally as a BFILE source of the ORDImage object, the srcType, srcLocation, and srcName parameters from the ORDImage source are copied into the source attribute of the ORDDicom object. The metadata conforms to the XML schema defined by the default mapping document.

Parameters

data

Embedded DICOM content stored in an ORDImage object.

setproperties

Indicator flag that determines whether the DICOM attributes are extracted from the embedded DICOM content. If the value is 1, the DICOM attributes are extracted into the metadata attribute of the constructed ORDDicom object, and the attributes of the ORDDicom object are populated. If the value is 0, no DICOM attributes are extracted. The default is 0.

Pragmas

None.

Exceptions

None.

Usage Notes

Use this constructor to create an ORDDicom object when the DICOM content is stored in an ORDImage object. Or, use this constructor to migrate an ORDImage object to an ORDDicom object.

Examples

Create an ORDDicom object from an ORDImage object:

```
SQL> desc image_tbl;  
Name                Null?    Type  
-----  
ID                   NUMBER(38)  
IMAGE                ORDIMAGE  
  
insert into medical_image_obj (id, dicom_src)  
  select s.id, ORDDicom(s.image) from image_tbl s;
```


ORDDicom() for Other Sources

Format

```
ORDDicom( SELF IN OUT NOCOPY ORDDicom,
          source_type    IN VARCHAR2 DEFAULT 'LOCAL',
          source_location IN VARCHAR2 DEFAULT NULL,
          source_name    IN VARCHAR2 DEFAULT NULL,
          setproperties  IN INTEGER DEFAULT 0
        ) RETURN SELF AS RESULT
```

Description

Constructs an ORDDicom object from a specific source. By default, the value of the `source_type` parameter is set to `LOCAL`, which means that the source of the data is stored locally in the database in a BLOB. With the default values, an empty object with a local source is constructed. If the value of the `source_type` parameter is set to `FILE`, an ORDDicom object is constructed with the source stored as an external FILE. The metadata conforms to the XML schema defined by the default XML mapping document.

Parameters

source_type

The type of the source. Valid values are: `FILE` or `LOCAL`. The default is `LOCAL`.

source_location

The directory location of the source (used for `source_type=FILE`).

source_name

The file name of the source (used for `source_type=FILE`).

setproperties

Indicator flag that determines whether the DICOM attributes are extracted from the embedded DICOM content. If the value is 1, the DICOM attributes are extracted into the metadata attribute of the constructed ORDDicom object, and the attributes of the ORDDicom object are populated. If the value is 0, no DICOM attributes are extracted. The default is 0.

Pragmas

None.

Exceptions

None.

Usage Notes

Use this constructor to create an ORDDicom object when the DICOM content is stored in the file system. Use the empty constructor when uploading DICOM content from a client, such as a Web browser or a Java application. Or, use the empty constructor as a destination object for methods such as `processCopy()`, `makeAnonymous()`, and `writeMetadata()`.

Examples

Example 1:

Create an ORDDicom object from a file without populating the object attributes:

```
insert into medical_image_obj (id, dicom_src)
  values (1, ORDDicom('FILE', 'DICODEDIR', 'example.dcm'));
```

Example 2:

Create an ORDDicom object from a file with the setProperties flag set:

```
insert into medical_image_obj (id, dicom_src)
  values (2, ORDDicom('FILE', 'DICODEDIR', 'example.dcm', 1));
```

Example 3:

Create an empty ORDDicom object:

```
insert into medical_image_obj (id, dicom_src)
  values (3, ORDDicom());
```

ORDDicom Methods

This section presents reference information about these ORDDicom methods:

- [export\(\)](#) on page 7-10
- [extractMetadata\(\)](#) on page 7-12
- [getAttributeByName\(\)](#) on page 7-14
- [getAttributeByTag\(\)](#) on page 7-15
- [getContent\(\)](#) on page 7-16
- [getContentLength\(\)](#) on page 7-17
- [getSeriesInstanceUID\(\)](#) on page 7-18
- [getSOPClassUID\(\)](#) on page 7-19
- [getSOPInstanceUID\(\)](#) on page 7-20
- [getSourceInformation\(\)](#) on page 7-21
- [getSourceLocation\(\)](#) on page 7-22
- [getSourceName\(\)](#) on page 7-23
- [getSourceType\(\)](#) on page 7-24
- [getStudyInstanceUID\(\)](#) on page 7-25
- [import\(\)](#) on page 7-26
- [isAnonymous\(\)](#) on page 7-28
- [isConformanceValid\(\)](#) on page 7-29
- [isLocal\(\)](#) on page 7-31
- [makeAnonymous\(\)](#) on page 7-32
- [processCopy\(\)](#) to BLOBs on page 7-34
- [processCopy\(\)](#) to ORDDicom on page 7-35
- [processCopy\(\)](#) to ORDImage on page 7-37
- [setProperties\(\)](#) on page 7-39
- [writeMetadata\(\)](#) on page 7-40

Note: In this section, *<unique-UID>* represents a 64-byte, dot-concatenated, numeric string that represents a globally unique identifier for DICOM content worldwide. The UID is commonly constructed with a root that uniquely identifies the organization producing the DICOM content, and a suffix that uniquely identifies the DICOM content within that organization. For some examples in this section, you must replace *<unique-UID>* with the appropriate UID.

export()

Format

```
export(SELF IN ORDDicom,  
       dest_type IN VARCHAR2,  
       dest_location IN VARCHAR2,  
       dest_name IN VARCHAR2)
```

Description

Exports embedded DICOM content to a specified destination. The data remains in the source BLOB when it is copied to the destination.

Parameters

dest_type

The type of the destination (only FILE is supported).

dest_location

The location of the destination (must be a valid Oracle directory object).

dest_name

The name of the destination file.

Pragmas

None.

Exceptions

None.

Usage Notes

Use this method to export the embedded DICOM content to the local file system.

The export() method writes only to a database directory object that the user has privilege to access. That is, you can access a directory object that you have created using the SQL statement CREATE DIRECTORY, or one to which you have been granted READ and WRITE access.

For example, the following SQL*Plus commands create a directory object and grant the user pm permission to read and write any file within the directory

```
c:\mydir\work:
```

```
CONNECT sys as sysdba  
Enter password: password  
CREATE OR REPLACE DIRECTORY DICOMDIR AS 'c:\mydir\work';  
GRANT READ,WRITE ON DIRECTORY DICOMDIR TO pm;
```

See [Section 7.1](#) for more information about these directory and table definitions.

Examples

Export embedded DICOM content to a specified file:

```
declare  
  obj orddicom;  
begin
```

```
select dicom_src into obj from medical_image_obj where id = 1;
obj.export('FILE', 'DICOMDIR', 'exported.dcm');
end;
/
```

extractMetadata()

Format

```
extractMetadata(  
    extractOption IN VARCHAR2 DEFAULT 'ALL',  
    docName IN VARCHAR2 DEFAULT 'ordcmmp.xml')  
RETURN SYS.XMLTYPE
```

Description

Returns the DICOM metadata as XML code for a specified mapping document. The default mapping document refers to the default metadata namespace `http://xmlns.oracle.com/ord/dicom/metadata_1_0`. The metadata attribute of the ORDDicom object is not affected.

Parameters

extractOption

A string that specifies the types of metadata to extract from the DICOM content. Valid values are: `ALL`, `MAPPED`, and `STANDARD`. The default is `ALL`.

When the value of this parameter is `ALL`, all the attributes in the embedded DICOM content are extracted. When the value is set to `MAPPED`, only mapped attributes are extracted. And, when the value is set to `STANDARD`, only attributes that conform to the DICOM standard and mapped attributes are extracted.

docName

The name of the mapping document. The default mapping document `ordcmmp.xml` is loaded during installation. This document refers to the default metadata namespace `http://xmlns.oracle.com/ord/dicom/metadata_1_0`.

Pragmas

None.

Exceptions

None.

Usage Notes

Use this method to retrieve metadata from the embedded DICOM content as XML code, and then store it in a database table for searching or viewing.

Use the preference parameter `XML_SKIP_ATTR` to specify size limits for DICOM attributes to be omitted when encoding into XML. See [Section 10.2.6.8](#) for more information about this preference parameter.

Use the preference parameter `VALIDATE_METADATA` to specify whether the XML documents are validated against a specific XML schema that is registered with Oracle XML DB. See [Section 10.2.6.7](#) for more information about this preference parameter.

Examples

Extract metadata from the embedded DICOM content:

```
declare  
    obj orddicom;
```

```
    metadata xmltype;
begin
  select dicom_src into obj from medical_image_obj where id = 1;

  -- extract all the metadata using the default mapping document.
  metadata := obj.extractMetadata();

  -- extract the standard metadata using the default mapping document.
  metadata := obj.extractMetadata('standard');

  -- extract the standard metadata by specifying the mapping document.
  metadata := obj.extractMetadata('standard', 'ordcmmmp.xml');
end;
/
```

getAttributeByName()

Format

```
getAttributeByName(attributeName IN VARCHAR2,
                   definerName IN VARCHAR2 DEFAULT 'DICOM')
RETURN VARCHAR2 DETERMINISTIC PARALLEL_ENABLE
```

Description

Returns the value of a DICOM attribute, as a VARCHAR2 string, for DICOM attributes other than SQ type attributes. For SQ type attributes, this method returns a segment of XML code, as a VARCHAR2 string. Use this method to get a single attribute from the embedded DICOM content.

Parameters

attributeName

The name of a specified attribute.

definerName

The name of the attribute definer. The default name is DICOM.

Pragmas

None.

Exceptions

None.

Usage Notes

See [Section 3.2.4](#) for more information about the best methods to use for searching and retrieving DICOM attributes.

Examples

Return the name of a specified DICOM attribute:

```
declare
  obj orddicom;
  res varchar2(4000);
begin
  select dicom_src into obj from medical_image_obj where id = 1;
  obj.setProperties;

  -- Patient ID attribute, this will return patient ID value
  res := obj.getAttributeByName('Patient ID');
  dbms_output.put_line('Patient ID attribute: ' || res);

  -- attribute in SQ type, this will return an xml segment.
  res := obj.getAttributeByName('Source Image Sequence');
  dbms_output.put_line('Source Image Sequence attribute: ' || res);
end ;
/
```


getAttributeByTag()

Format

```
getAttributeByTag(tag IN VARCHAR2,
                 definerName IN VARCHAR2 DEFAULT 'DICOM')
RETURN VARCHAR2 DETERMINISTIC PARALLEL_ENABLE
```

Description

Returns the value of a DICOM attribute, as a VARCHAR2 string, for DICOM attributes other than SQ type attributes. For SQ type attributes, this method returns a segment of XML code, as a VARCHAR2 string. Use this method to get a single attribute of the embedded DICOM content.

Parameters

tag

The code value used to specify a DICOM attribute or item tag, as a hexadecimal string.

definerName

The name of the attribute definer. The default name is DICOM.

Pragmas

None.

Exceptions

None.

Usage Notes

See [Section 3.2.4](#) for more information about the best methods to use for searching and retrieving DICOM attributes.

Examples

Return the tag of a specified DICOM attribute:

```
declare
  obj orddicom;
  res varchar2(4000);
begin
  select dicom_src into obj from medical_image_obj where id = 1;
  obj.setProperties;

  -- Patient ID attribute, this will return patient ID value
  res := obj.getAttributeByTag('00100020');
  dbms_output.put_line('Patient ID attribute: ' || res);

  -- attribute in SQ type, this will return an xml segment.
  res := obj.getAttributeByTag('00082112');
  dbms_output.put_line('Source Image Sequence attribute: ' || res);
end ;
/
```

getContent()

Format

getContent() RETURN BLOB DETERMINISTIC

Description

Returns the embedded DICOM content stored in the source attribute of the ORDDicom object. This method returns the BLOB handle, or a null value if the DICOM content has not been imported.

Parameters

None.

Pragmas

None.

Exceptions

None.

Usage Notes

None.

Examples

Return the content of the source attribute for the ORDDicom object:

```
select t.dicom_src.getContent() from medical_image_obj t;
```

getContentLength()

Format

getContentLength() RETURN INTEGER DETERMINISTIC PARALLEL_ENABLE

Description

Returns the length of the embedded DICOM content. This method returns the value of the contentLength attribute of the ORDDicom object.

Parameters

None.

Pragmas

None.

Exceptions

None.

Usage Notes

None.

Examples

Return the value of the contentLength attribute for the ORDDicom object:

```
select t.dicom_src.getContentLength() from medical_image_obj t;
```

getSeriesInstanceUID()

Format

getSeriesInstanceUID() RETURN VARCHAR2 DETERMINISTIC PARALLEL_ENABLE,

Description

Returns the value of the SERIES_INSTANCE_UID attribute of the ORDDicom object.

Parameters

None.

Pragmas

None.

Exceptions

None.

Usage Notes

None.

Examples

Return the value of the SERIES_INSTANCE_UID attribute for the ORDDicom object:

```
select t.dicom_src.getSeriesInstanceUID() from medical_image_obj t;
```

getSOPClassUID()

Format

getSOPClassUID() RETURN VARCHAR2 DETERMINISTIC PARALLEL_ENABLE,

Description

Returns the value of the SOP_CLASS_UID attribute of the ORDDicom object.

Parameters

None.

Pragmas

None.

Exceptions

None.

Usage Notes

None.

Examples

Return the value of the SOP_CLASS_UID attribute for the ORDDicom object:

```
select t.dicom_src.getSOPClassUID() from medical_image_obj t;
```

getSOPInstanceUID()

Format

getSOPInstanceUID() RETURN VARCHAR2 DETERMINISTIC PARALLEL_ENABLE

Description

Returns the value of the SOP_INSTANCE_UID attribute of the ORDDicom object.

Parameters

None.

Pragmas

None.

Exceptions

None.

Usage Notes

None.

Examples

Return the value of the SOP_INSTANCE_UID attribute for the ORDDicom object:

```
select t.dicom_src.getSOPInstanceUID() from medical_image_obj t;
```

getSourceInformation()

Format

getSourceInformation() RETURN VARCHAR2 DETERMINISTIC PARALLEL_ENABLE

Description

Returns the source information from the source attribute of the ORDDicom object as a URL in the form "source_type://source_location/source_name".

Parameters

None.

Pragmas

None.

Exceptions

None.

Usage Notes

None.

Examples

Return the source information for the ORDDicom object:

```
select t.dicom_src.getSourceInformation() from medical_image_obj t;
```

getSourceLocation()

Format

getSourceLocation() RETURN VARCHAR2 DETERMINISTIC PARALLEL_ENABLE

Description

Returns the source location from the source attribute of the ORDDicom object.

Parameters

None.

Pragmas

None.

Exceptions

None.

Usage Notes

None.

Examples

Return the source location for the ORDDicom object:

```
select t.dicom_src.getSourceLocation() from medical_image_obj t;
```


getSourceName()

Format

```
getSourceName() RETURN VARCHAR2 DETERMINISTIC PARALLEL_ENABLE
```

Description

Returns the source name from the source attribute of the ORDDicom object.

Parameters

None.

Pragmas

None.

Exceptions

None.

Usage Notes

None.

Examples

Return the source name for the ORDDicom object:

```
select t.dicom_src.getSourceName() from medical_image_obj t;
```

getSourceType()

Format

getSourceType() RETURN VARCHAR2 DETERMINISTIC PARALLEL_ENABLE

Description

Returns the source type from the source attribute of the ORDDicom object.

Parameters

None.

Pragmas

None.

Exceptions

None.

Usage Notes

None.

Examples

Return the source type for the ORDDicom object:

```
select t.dicom_src.getSourceType() from medical_image_obj t;
```

getStudyInstanceUID()

Format

```
getStudyInstanceUID( ) RETURN VARCHAR2 DETERMINISTIC PARALLEL_ENABLE,
```

Description

Returns the value of the STUDY_INSTANCE_UID attribute of the ORDDicom object.

Parameters

None.

Pragmas

None.

Exceptions

None.

Usage Notes

None.

Examples

Return the value of the STUDY_INSTANCE_UID attribute for the ORDDicom object:

```
select t.dicom_src.getStudyInstanceUID() from medical_image_obj t;
```

import()

Format

```
import(SELF IN OUT NOCOPY ORDDicom,
       setproperties IN INTEGER DEFAULT 1)
```

Description

Imports DICOM content from the current source. This method assumes that the `source` attributes have been set in the ORDDicom object by passing the `source_type`, `source_location`, and `source_name` parameters to the constructor.

Parameters

setproperties

Indicator flag that determines whether the DICOM attributes are extracted into the metadata attribute of the ORDDicom object. If the value is 1, the DICOM attributes are extracted into the metadata attribute of the ORDDicom object, and the attributes of the ORDDicom object are populated. If the value is 0, no DICOM attributes are extracted. The default is 1.

Pragmas

None.

Exceptions

None.

Usage Notes

Use this method when the ORDDicom object is constructed from a source other than a BLOB, and must be imported into a BLOB.

The `import()` method reads only from a database directory object that the user has privilege to access. That is, you can access a directory object that you have created using the SQL statement `CREATE DIRECTORY`, or one to which you have been granted `READ` access.

For example, the following SQL*Plus commands create a directory object and grant the user `pm` permission to read any file within the directory `c:\mydir\work`:

```
CONNECT sys as sysdba
Enter password: password
CREATE OR REPLACE DIRECTORY DICOMDIR AS 'c:\mydir\work';
GRANT READ ON DIRECTORY DICOMDIR TO pm;
```

Assuming that the DICOM object is inserted as follows:

```
insert into medical_image_obj (id, dicom_src) values
(1, ORDDicom('FILE', 'DICOMDIR', 'imported.dcm'));
```

The user `pm` can import DICOM content from the `imported.dcm` file in this directory using the `import()` method of the ORDDicom object:

```
obj.import();
```

See [Section 7.1](#) for more information about these directory and table definitions.

Examples

Import the DICOM attributes:

```
declare
  obj orddicom;
begin
  select dicom_src into obj from medical_image_obj where id = 1 for update;
  if (obj.isLocal() = 0) then
    obj.import();
  end if;
  update medical_image_obj set dicom_src = obj where id = 1;
end;
/
```

isAnonymous()

Format

```
isAnonymous(anonymityDocName IN VARCHAR2 DEFAULT 'ordcman.xml')  
RETURN INTEGER
```

Description

Determines whether the embedded DICOM content is anonymous using a specified anonymity document, which is stored in the data model repository. This method returns a value of 1 if the data is anonymous; otherwise it returns a value of 0.

Parameters

anonymityDocName

The name of the anonymity document. The default name is `ordcman.xml`.

Pragmas

None.

Exceptions

None.

Usage Notes

Use this method before calling the `makeAnonymous()` method to find out whether patient identifying information has been removed from the embedded DICOM content (see also the [makeAnonymous\(\)](#) method).

Examples

Check if the embedded DICOM content is anonymous:

```
select t.dicom_src.isAnonymous('ordcman.xml') from medical_image_obj t;
```

isConformanceValid()

Format

```
isConformanceValid(  
    constraintName IN VARCHAR2  
) RETURN INTEGER
```

Description

Performs a conformance validation check to determine whether the embedded DICOM content conforms to a specific set of constraints identified by the `constraintName` parameter. This method returns a value of 1 if conformance is valid; otherwise it returns a value of 0.

This method also logs error messages from the constraint documents, which can be viewed by querying the public view [orddcm_conformance_vld_msgs](#). The public view [orddcm_constraint_names](#) contains the list of constraint names.

Parameters

constraintName

The name of the constraint to be used for conformance validation checking.

Pragmas

None.

Exceptions

None.

Usage Notes

If this method is called from a SQL query, ensure that the constraint rule definition does not contain any <ACTION> elements.

Use the preference parameter `EXP_IF_NULL_ATTR_IN_CONSTRAINT` to indicate whether to throw exceptions when encountering missing attributes or null attribute values during conformance validation. See [Section 10.2.6.2](#) for more information about this preference parameter.

Examples

Check if the ORDDicom objects are conformance valid. Then, show any conformance validation messages that are generated.

```
declare  
    cursor dicom_src_cur is  
        select dicom_src from medical_image_obj order by id;  
begin  
    for rec in dicom_src_cur loop  
        dbms_output.put_line('isConformanceValid(PatientModule): ' ||  
            rec.dicom_src.isConformanceValid('PatientModule'));  
    end loop;  
end;  
/  
  
select t1.id, t2.message, t2.msg_time time
```

```
from medical_image_obj t1, orddcm_conformance_vld_msgs t2
where t1.dicom_src.sop_instance_uid = t2.sop_instance_uid and
      t2.rule_name = 'PatientModule';
```


isLocal()

Format

isLocal() RETURN INTEGER DETERMINISTIC PARALLEL_ENABLE

Description

Returns the local status of the source. If the DICOM content is stored in the source BLOB, the object is defined as local. If the DICOM content is stored externally in an operating system-specific file, the object is defined as not local. This method returns a value of 1 if the object is local; otherwise it returns a value of 0.

Parameters

None.

Pragmas

None.

Exceptions

None.

Usage Notes

None.

Examples

Check if the DICOM content is local:

```
select t.dicom_src.isLocal() from medical_image_obj t;
declare
  obj orddicom;
begin
  select dicom_src into obj from medical_image_obj where id = 1 for update;
  if (obj.isLocal() = 0) then
    obj.import();
  end if;
  update medical_image_obj set dicom_src = obj where id = 1;
end;
/
```

makeAnonymous()

Format

```
makeAnonymous(SELF IN ORDDicom,  
              dest_SOP_INSTANCE_UID IN VARCHAR2,  
              dest IN OUT NOCOPY ORDDicom,  
              anonymityDocName IN VARCHAR2 DEFAULT 'ordcman.xml')
```

Description

Removes patient identifying information from the ORDDicom object after copying it into another ORDDicom object, based on a specified anonymity document. Both the embedded DICOM content and the metadata attribute in the destination ORDDicom object are made anonymous.

Parameters

dest_SOP_INSTANCE_UID

The SOP instance UID of the destination ORDDicom object. It must ensure that the destination DICOM content is globally unique.

dest

An empty ORDDicom object in which to store the anonymous ORDDicom object.

anonymityDocName

The name of the anonymity document. The default name is `ordcman.xml`.

Pragmas

None.

Exceptions

None.

Usage Notes

Use this method to remove patient identifying information from the embedded DICOM content for use in data sharing and research.

Examples

Remove patient identifying information from the destination ORDDicom object:

Note: Replace `<unique-UID>` with the UID that identifies the organization producing the DICOM content and the DICOM content within that organization.

```
declare  
  obj_src orddicom;  
  obj_dest orddicom;  
  dest_sop_instance_uid varchar2(128) := '<unique-UID>';  
begin  
  select dicom_src, dicom_dest into obj_src, obj_dest  
  from medical_image_obj where id = 1 for update;  
  obj_src.makeAnonymous(dest_sop_instance_uid, obj_dest, 'ordcman.xml');
```

```
update medical_image_obj set dicom_dest = obj_dest where id = 1;  
end;  
/
```

processCopy() to BLOBs

Format

```
processCopy(SELF IN ORDDicom,  
            command IN VARCHAR2,  
            dest IN OUT NOCOPY BLOB)
```

Description

Copies the ORDDicom object that is input into the destination BLOB, and then performs the specified processing operations on the destination BLOB. The original ORDDicom object that was input remains unchanged.

Parameters

command

A command string that accepts a processing operator as input. Valid values include: `frame`, `contentFormat`, `fileFormat`, `compressionFormat`, `cut`, `scale`, and `rotate`. See [Appendix D](#) for information about processing operators.

dest

The destination BLOB that contains the output of the process command on the ORDDicom object.

Pragmas

None.

Exceptions

None.

Usage Notes

Use this method to process an ORDDicom object into a BLOB after copying it from the ORDDicom object. In this case, the output in the BLOB is image or video content.

See [Appendix C](#) for information about the encoding rules that support DICOM content processing. See [Appendix D](#) for more information about DICOM processing.

Use the preference parameter `VALIDATE_METADATA` to specify whether the XML documents are validated against the Oracle default DICOM metadata schema. See [Section 10.2.6.7](#) for more information about this preference parameter.

Examples

Copy the DICOM content into a BLOB and then process it:

```
declare  
    obj orddicom;  
    dest blob;  
begin  
    select dicom_src, blob_dest into obj, dest  
    from medical_image_obj where id = 1 for update;  
  
    obj.processCopy('fileFormat=jpeg maxScale=100 100', dest);  
end;  
/
```

processCopy() to ORDDicom

Format

```
processCopy(SELF IN ORDDicom,  
            command IN VARCHAR2,  
            dest_SOP_INSTANCE_UID IN VARCHAR2,  
            dest IN OUT NOCOPY ORDDicom,  
            metadata IN SYS.XMLTYPE DEFAULT NULL)
```

Description

Copies the ORDDicom object that is input into a destination ORDDicom object, then performs the specified processing operations on the destination ORDDicom object and updates it with the new metadata. The original ORDDicom object that was input remains unchanged.

Parameters

command

A command string that accepts a processing operator as input. Valid values include: `compressionFormat`, `frame`, `contentFormat`, `cut`, `scale`, and `rotate`. See [Appendix D](#) for information about processing operators.

dest_SOP_INSTANCE_UID

The SOP instance UID of the destination ORDDicom object. It must ensure that the destination DICOM content is globally unique.

dest

An empty ORDDicom object in which to store the new DICOM content with the new metadata.

metadata

The new metadata to be written into the new DICOM content.

Pragmas

None.

Exceptions

None.

Usage Notes

Use this method to process DICOM content into a destination ORDDicom object after copying it from the source ORDDicom object. In this case, the output is image or video content.

See [Appendix C](#) for information about the encoding rules that support DICOM content processing. See [Appendix D](#) for more information about DICOM processing.

Use the preference parameter `VALIDATE_METADATA` to specify whether the XML documents are validated against the Oracle default DICOM metadata schema. See [Section 10.2.6.7](#) for more information about this preference parameter.

Examples

Copy the DICOM content into an ORDDicom object and then process it:

Note: Replace *<unique-UID>* with the UID that identifies the organization producing the DICOM content and the DICOM content within that organization.

```
declare
  obj_src orddicom;
  obj_dest orddicom;
  dest_sop_instance_uid varchar2(128) := '<unique-UID>';
begin
  select dicom_src, dicom_dest into obj_src, obj_dest
  from medical_image_obj where id = 1 for update;
  obj_src.processcopy('compressionFormat=jpeg',
                    dest_sop_instance_uid,
                    obj_dest);

  update medical_image_obj set dicom_dest = obj_dest where id = 1;
end;
/
```

processCopy() to ORDImage

Format

```
processCopy(SELF IN ORDDicom,
            command IN VARCHAR2,
            dest IN OUT NOCOPY ORDImage)
```

Description

Copies the ORDDicom object that is input into the destination ORDImage object, and then performs the specified processing operations on the destination ORDImage object. The original ORDDicom object that was input remains unchanged.

Parameters

command

A command string that accepts an image processing operator as input. Valid values include: `frame`, `contentFormat`, `fileFormat`, `compressionFormat`, `cut`, `scale`, and `rotate`. See [Appendix D](#) for information about image processing operators.

dest

An empty ORDImage object in which to store the new, processed ORDImage object without the DICOM metadata.

Pragmas

None.

Exceptions

None.

Usage Notes

Use this method to get a non-DICOM image that is suitable for presentation on the Web from the embedded DICOM content.

See [Appendix C](#) for information about the encoding rules that support image content processing. See [Appendix D](#) for more information about DICOM processing.

Use the preference parameter `VALIDATE_METADATA` to specify whether the XML documents are validated against the Oracle default DICOM metadata schema. See [Section 10.2.6.7](#) for more information about this preference parameter.

Examples

Copy the DICOM content into an ORDImage object and then process it:

```
declare
  obj_src orddicom;
  obj_dest ordimage;
begin
  select dicom_src, image_dest into obj_src, obj_dest
  from medical_image_obj where id = 1 for update;
  obj_src.processcopy('fileFormat=jpeg maxScale=100 100', obj_dest);

  update medical_image_obj set image_dest = obj_dest where id = 1;
end;
```

/

setProperties()

Format

```
setProperties(SELF IN OUT NOCOPY ORDDicom)
```

Description

Sets the attributes of the ORDDicom object. The attributes of the ORDDicom object are populated and the embedded DICOM content attributes are extracted into the metadata attribute of the ORDDicom object. The XML metadata conforms to the default metadata schema namespace

```
http://xmlns.oracle.com/ord/dicom/metadata_1_0.
```

If the repository contains a stored tag list document, the XML metadata contains only the DICOM attributes specified by the stored tag list. Otherwise, the XML metadata contains all the attributes from the embedded DICOM content.

Parameters

None.

Pragmas

None.

Exceptions

None.

Usage Notes

Use this method to populate ORDDicom object attributes and to get the metadata from the embedded DICOM content.

Use the preference parameter XML_SKIP_ATTR to specify size limits for DICOM attributes to be omitted when encoding into XML. See [Section 10.2.6.8](#) for more information about this preference parameter.

Use the preference parameter VALIDATE_METADATA to specify whether the XML documents are validated against a specific XML schema that is registered with Oracle XML DB. See [Section 10.2.6.7](#) for more information about this preference parameter.

Examples

Set the attributes of the ORDDicom object:

```
declare
  obj orddicom;
begin
  select dicom_src into obj from medical_image_obj where id = 1 for update;
  obj.setProperties();
  update medical_image_obj set dicom_src = obj where id = 1;
end;
/
```

writeMetadata()

Format

```
writeMetadata(SELF IN ORDDicom,  
             metadata IN SYS.XMLTYPE,  
             dest IN OUT NOCOPY ORDDicom)
```

Description

Modifies the current ORDDicom object with the metadata provided by making a copy of the existing ORDDicom object in the destination ORDDicom object, and then modifying the metadata. The original ORDDicom object remains unchanged. The attributes in the embedded DICOM content of the destination ORDDicom object are copied from the metadata that was input.

Parameters

metadata

The input metadata stored in data type XMLType. In the destination ORDDicom object, the input metadata is used to update the values for attributes that are identical to attributes in the source ORDDicom object, or to add any new attributes. The metadata must conform to the default metadata schema with the namespace `http://xmlns.oracle.com/ord/dicom/metadata_1_0`. The SOP instance UID in the metadata must ensure that the destination DICOM content is globally unique.

dest

An empty ORDDicom object in which to store the new embedded DICOM content with the new metadata.

Pragmas

None.

Exceptions

None.

Usage Notes

Use this method to update attributes in the embedded DICOM content.

In addition, you can use this method to add private attributes to the embedded DICOM content.

See [Appendix C](#) for information about the encoding rules that support metadata extraction.

Use the preference parameter `VALIDATE_METADATA` to specify whether the XML documents are validated against the Oracle default DICOM metadata schema. See [Section 10.2.6.7](#) for more information about this preference parameter.

Use the preference parameter `SQ_WRITE_LEN` to specify how the DICOM sequence (SQ) types are encoded. See [Section 10.2.6.6](#) for more information about this preference parameter.

Examples

Write the new metadata to the copy of the embedded DICOM content:

```
declare
  obj_src orddicom;
  obj_dest orddicom;
  metadata xmltype;
begin
  metadata := xmltype(bfilename('DICOMDIR', 'wm_meta.xml'),
                    nls_charset_id('AL32UTF8'),
                    'http://xmlns.oracle.com/ord/dicom/metadata_1_0');

  select dicom_src, dicom_dest into obj_src, obj_dest
  from medical_image_obj where id = 1 for update;

  obj_src.writeMetadata(metadata, obj_dest);

  update medical_image_obj set dicom_dest = obj_dest where id = 1;

end;
/
```

DICOM Relational Interface Reference

Oracle Multimedia DICOM provides a relational interface, which is defined in the ORD_DICOM PL/SQL package. This package provides the same features as those provided by the ORDDicom object interface, in the relational environment. The DICOM relational interface adds Oracle Multimedia support to medical image data stored in BLOBs and BFILEs, rather than in the ORDDicom object type.

Application developers who create medical imaging applications without using the Oracle Multimedia ORDDicom object type to store and manage medical image data in relational tables can use the Oracle Multimedia DICOM relational interface to manage their medical image data. In addition, application developers who do not want to migrate their existing medical image applications to use Oracle Multimedia ORDDicom objects can use this relational interface to manage their medical image data.

The ORD_DICOM package is defined in the `ordcpxksp.sql` file. After installation, this file is available in the Oracle home directory at:

```
<ORACLE_HOME>/ord/im/admin (on Linux and UNIX)
```

```
<ORACLE_HOME>\ord\im\admin (on Windows)
```

This chapter describes the functions and procedures in the DICOM relational interface. See [Table 3-1](#) for information about other DICOM application programming interfaces (APIs).

This chapter contains these sections:

- [Examples for DICOM Relational Functions and Procedures](#) on page 8-1
- [DICOM Relational Functions](#) on page 8-3
- [DICOM Relational Procedures](#) on page 8-19

8.1 Examples for DICOM Relational Functions and Procedures

The functions and procedures for the DICOM relational interface described in this chapter show examples based on the MEDICAL_IMAGE_REL table, which these examples create in the Product Media (PM) sample schema. See [Section 8.1.2](#) when reading through these examples.

Before using DICOM relational interface functions and procedures, you must load some data into the table. For example, you can use SQL*Loader or the `importFrom()` procedure. Substitute your DICOM files for those in the examples.

See Also:

Oracle Database Sample Schemas for information about the PM and other sample schemas

8.1.1 Directory Definition and Setup for DICOM Relational Examples

Issue the following statements before executing the examples, where `c:\mydir\work` is the directory where the user `pm` can find the DICOM files:

```
CREATE OR REPLACE DIRECTORY DICOMDIR as 'c:\mydir\work';
GRANT READ, WRITE ON DIRECTORY DICOMDIR TO pm;
```

Note: At the beginning of each database session, call the `setDataModel()` procedure. See the [setDataModel\(\) Procedure](#) for more information.

8.1.2 MEDICAL_IMAGE_REL Table Definition

Before loading data into the table, you must create the table and columns where the data is to be stored. The following PL/SQL code segment creates the `MEDICAL_IMAGE_REL` table with five columns.

```
CONNECT pm

Enter password: password

CREATE TABLE MEDICAL_IMAGE_REL
(
  id          integer primary key,
  blob_src   blob,
  bfile_src  bfile,
  image_src  ordsys.ordimage,
  blob_dest  blob
);
COMMIT;
```

where:

- `blob_src`: the source DICOM content in the BLOB.
- `bfile_src`: the source DICOM content in the BFILE.
- `image_src`: the source DICOM content in the ORDImage object.
- `blob_dest`: the destination DICOM content in the BLOB.

DICOM Relational Functions

The ORD_DICOM package defines these DICOM relational functions:

- [extractMetadata\(\)](#) for BFILES on page 8-4
- [extractMetadata\(\)](#) for BLOBs on page 8-6
- [extractMetadata\(\)](#) for ORDImage on page 8-8
- [isAnonymous\(\)](#) for BFILES on page 8-10
- [isAnonymous\(\)](#) for BLOBs on page 8-11
- [isAnonymous\(\)](#) for ORDImage on page 8-12
- [isConformanceValid\(\)](#) for BFILES on page 8-13
- [isConformanceValid\(\)](#) for BLOBs on page 8-15
- [isConformanceValid\(\)](#) for ORDImage on page 8-17

extractMetadata() for BFILES

Format

```
extractMetadata(  
  data IN BFILE,  
  extractOption IN VARCHAR2 DEFAULT 'ALL',  
  docName IN VARCHAR2 DEFAULT 'ordcmmp.xml')  
RETURN SYS.XMLTYPE
```

Description

Returns the DICOM metadata as an XML document for a specified mapping document. The default mapping document refers to the default metadata namespace `http://xmlns.oracle.com/ord/dicom/metadata_1_0`.

Parameters

data

The input DICOM content stored in a BFILE.

extractOption

A string that specifies the types of metadata to extract from the DICOM content. Valid values are: `ALL`, `MAPPED`, and `STANDARD`. The default is `ALL`.

When the value of the `extractOption` parameter is `ALL`, all attributes in the DICOM content are extracted. When the value is set to `MAPPED`, only mapped attributes are extracted. And, when the value is set to `STANDARD`, only attributes that conform to the DICOM standard and mapped attributes are extracted.

docName

The name of the mapping document. The default mapping document `ordcmmp.xml` is loaded during installation.

Pragmas

None.

Exceptions

None.

Usage Notes

Use the preference parameter `XML_SKIP_ATTR` to specify size limits for DICOM attributes to be omitted when encoding into XML. See [Section 10.2.6.8](#) for more information about this preference parameter.

Use the preference parameter `VALIDATE_METADATA` to specify whether the XML documents are validated against a specific XML schema that is registered with Oracle XML DB. See [Section 10.2.6.7](#) for more information about this preference parameter.

Examples

Extract metadata from the DICOM content stored in a BFILE:

```
declare  
  src bfile;  
  metadata xmltype;
```



```
begin
  select bfile_src into src from medical_image_rel where id = 1 for update;
  metadata := ord_dicom.extractMetadata(src, 'all', 'ordcmmmp.xml');
end;
/
```

extractMetadata() for BLOBs

Format

```
extractMetadata(  
  data IN BLOB,  
  extractOption IN VARCHAR2 DEFAULT 'ALL',  
  docName IN VARCHAR2 DEFAULT 'ordcmmp.xml')  
RETURN SYS.XMLTYPE
```

Description

Returns the DICOM metadata as an XML document for a specified mapping document. The default mapping document refers to the default metadata namespace `http://xmlns.oracle.com/ord/dicom/metadata_1_0`.

Parameters

data

The input DICOM content stored in a BLOB.

extractOption

A string that specifies the types of metadata to extract from the DICOM content. Valid values are: `ALL`, `MAPPED`, and `STANDARD`. The default is `ALL`.

When the value of the `extractOption` parameter is `ALL`, all attributes in the DICOM content are extracted. When the value is set to `MAPPED`, only mapped attributes are extracted. And, when the value is set to `STANDARD`, only attributes that conform to the DICOM standard and mapped attributes are extracted.

docName

The name of the mapping document. The default mapping document `ordcmmp.xml` is loaded during installation.

Pragmas

None.

Exceptions

None.

Usage Notes

Use the preference parameter `XML_SKIP_ATTR` to specify size limits for DICOM attributes to be omitted when encoding into XML. See [Section 10.2.6.8](#) for more information about this preference parameter.

Use the preference parameter `VALIDATE_METADATA` to specify whether the XML documents are validated against a specific XML schema that is registered with Oracle XML DB. See [Section 10.2.6.7](#) for more information about this preference parameter.

Examples

Extract metadata from the DICOM content stored in a BLOB:

```
declare  
  src blob;  
  metadata xmltype;
```

```
begin
  select blob_src into src from medical_image_rel where id = 1 for update;
  metadata := ord_dicom.extractMetadata(src, 'all', 'ordcmmmp.xml');
end;
```

extractMetadata() for ORDImage

Format

```
extractMetadata(  
  data IN ORDSYS.ORDImage,  
  extractOption IN VARCHAR2 DEFAULT 'ALL',  
  docName IN VARCHAR2 DEFAULT 'ordcmmmp.xml')  
RETURN SYS.XMLTYPE
```

Description

Returns the DICOM metadata as an XML document for a specified mapping document. The default mapping document refers to the default metadata namespace `http://xmlns.oracle.com/ord/dicom/metadata_1_0`.

Parameters

data

The input DICOM content stored in an ORDImage object.

extractOption

A string that specifies the types of metadata to extract from the DICOM content. Valid values are: `ALL`, `MAPPED`, and `STANDARD`. The default is `ALL`.

When the value of the `extractOption` parameter is `ALL`, all attributes in the DICOM content are extracted. When the value is set to `MAPPED`, only mapped attributes are extracted. And, when the value is set to `STANDARD`, only attributes that conform to the DICOM standard and mapped attributes are extracted.

docName

The name of the mapping document. The default mapping document `ordcmmmp.xml` is loaded during installation.

Pragmas

None.

Exceptions

None.

Usage Notes

Use the preference parameter `XML_SKIP_ATTR` to specify size limits for DICOM attributes to be omitted when encoding into XML. See [Section 10.2.6.8](#) for more information about this preference parameter.

Use the preference parameter `VALIDATE_METADATA` to specify whether the XML documents are validated against a specific XML schema that is registered with Oracle XML DB. See [Section 10.2.6.7](#) for more information about this preference parameter.

Examples

Extract metadata from the DICOM content stored in an ORDImage object:

```
declare  
  src ordimage;  
  metadata xmltype;
```

```
begin
  select image_src into src from medical_image_rel where id = 1 for update;
  metadata := ord_dicom.extractMetadata(src, 'all', 'ordcmmmp.xml');
end;
/
```

isAnonymous() for BFILEs

Format

```
isAnonymous(  
  src IN BFILE,  
  anonymityDocName IN VARCHAR2 DEFAULT 'ordcman.xml')  
RETURN INTEGER
```

Description

Determines whether the input DICOM content is anonymous, using a specified anonymity document. This function returns a value of 1 if the data is anonymous; otherwise it returns a value of 0.

Parameters

src

The input DICOM content stored in a BFILE.

anonymityDocName

The name of the anonymity document. The default name is `ordcman.xml`.

Pragmas

None.

Exceptions

None.

Usage Notes

None.

Examples

Check if the DICOM content stored in a BFILE is anonymous:

```
select ord_dicom.isAnonymous(t.bfile_src, 'ordcman.xml')  
  from medical_image_rel t where id = 1;
```

isAnonymous() for BLOBs

Format

```
isAnonymous(  
  src IN BLOB,  
  anonymityDocName IN VARCHAR2 DEFAULT 'ordcman.xml')  
RETURN INTEGER
```

Description

Determines whether the input DICOM content is anonymous, using a specified anonymity document. This function returns a value of 1 if the data is anonymous; otherwise it returns a value of 0.

Parameters

src

The input DICOM content stored in a BLOB.

anonymityDocName

The name of the anonymity document. The default name is `ordcman.xml`.

Pragmas

None.

Exceptions

None.

Usage Notes

None.

Examples

Check if the DICOM content stored in a BLOB is anonymous:

```
select ord_dicom.isAnonymous(t.blob_src, 'ordcman.xml')  
  from medical_image_rel t where id = 1;
```

isAnonymous() for ORDImage

Format

```
isAnonymous(  
  src IN ORDSYS.ORDImage,  
  anonymityDocName IN VARCHAR2 DEFAULT 'ordcman.xml')  
RETURN INTEGER
```

Description

Determines whether the input DICOM content is anonymous, using a specified anonymity document. This function returns a value of 1 if the data is anonymous; otherwise it returns a value of 0.

Parameters

src

The input DICOM content stored in an ORDImage object.

anonymityDocName

The name of the anonymity document. The default name is `ordcman.xml`.

Pragmas

None.

Exceptions

None.

Usage Notes

None.

Examples

Check if the DICOM content stored in an ORDImage object is anonymous:

```
select ord_dicom.isAnonymous(t.image_src, 'ordcman.xml')  
  from medical_image_rel t where id = 1;
```


isConformanceValid() for BFILEs

Format

```
isConformanceValid(  
  src IN BFILE,  
  constraintName IN VARCHAR2  
) RETURN INTEGER
```

Description

Performs a conformance validation check to determine whether the input DICOM content conforms to a specified set of constraints identified by the `constraintName` parameter. This method returns a value of 1 if conformance is valid; otherwise it returns a value of 0.

This method also logs error messages from the constraint documents, which can be viewed by querying the public view [orddcm_conformance_vld_msgs](#). The public view [orddcm_constraint_names](#) contains the list of constraint names.

Parameters

src

The input DICOM content stored in a BFILE.

constraintName

The name of the constraint to be used for conformance validation checking.

Pragmas

None.

Exceptions

None.

Usage Notes

If this method is called from a SQL query, ensure that the constraint rule definition does not contain any <ACTION> elements.

Use the preference parameter `EXP_IF_NULL_ATTR_IN_CONSTRAINT` to indicate whether to throw exceptions when encountering missing attributes or null attribute values during conformance validation. See [Section 10.2.6.2](#) for more information about this preference parameter.

Examples

Delete any existing conformance validation messages. Check if the DICOM content stored in a BFILE is conformance valid. Then, show any new conformance validation messages that are generated.

```
-- clear the previous conformance validation messages  
delete from orddcm_conformance_vld_msgs;  
  
-- conformance validate the DICOM content  
declare  
  src bfile;  
begin
```

```
select bfile_src into src from medical_image_rel where id = 1;
dbms_output.put_line('isConformanceValid(PatientModule) ' ||
ord_dicom.isConformanceValid(src, 'PatientModule'));
end;
/

-- get the logged conformance validation messages
select message, msg_time time from orddcm_conformance_vld_msgs
where rule_name='PatientModule';
```

isConformanceValid() for BLOBs

Format

```
isConformanceValid(
  src IN BLOB,
  constraintName IN VARCHAR2
) RETURN INTEGER
```

Description

Performs a conformance validation check to determine whether the input DICOM content conforms to a specified set of constraints identified by the `constraintName` parameter. This method returns a value of 1 if conformance is valid; otherwise it returns a value of 0.

This method also logs error messages from the constraint documents, which can be viewed by querying the public view [orddcm_conformance_vld_msgs](#). The public view [orddcm_constraint_names](#) contains the list of constraint names.

Parameters

src

The input DICOM content stored in a BLOB.

constraintName

The name of the constraint to be used for conformance validation checking.

Pragmas

None.

Exceptions

None.

Usage Notes

If this method is called from a SQL query, ensure that the constraint rule definition does not contain any `<ACTION>` elements.

Use the preference parameter `EXP_IF_NULL_ATTR_IN_CONSTRAINT` to indicate whether to throw exceptions when encountering missing attributes or null attribute values during conformance validation. See [Section 10.2.6.2](#) for more information about this preference parameter.

Examples

Delete any existing conformance validation messages. Check if the DICOM content stored in a BLOB is conformance valid. Then, show any new conformance validation messages that are generated.

```
-- clear the previous conformance validation messages
delete from orddcm_conformance_vld_msgs;

-- conformance validate the DICOM content
declare
  src blob;
begin
```

```
select blob_src into src from medical_image_rel where id = 1;
dbms_output.put_line('isConformanceValid(PatientModule) ' ||
ord_dicom.isConformanceValid(src, 'PatientModule'));
end;
/

-- get the logged conformance validation messages
select message, msg_time time from orddcm_conformance_vld_msgs
where rule_name='PatientModule';
```

isConformanceValid() for ORDImage

Format

```
isConformanceValid(  
  src IN ORDSYS.ORDImage,  
  constraintName IN VARCHAR2  
) RETURN INTEGER
```

Description

Performs a conformance validation check to determine whether the input DICOM content conforms to a specified set of constraints identified by the `constraintName` parameter. This method returns a value of 1 if conformance is valid; otherwise it returns a value of 0.

This method also logs error messages from the constraint documents, which can be viewed by querying the public view [orddcm_conformance_vld_msgs](#). The public view [orddcm_constraint_names](#) contains the list of constraint names.

Parameters

src

The input DICOM content stored in an ORDImage object.

constraintName

The name of the constraint to be used for conformance validation checking.

Pragmas

None.

Exceptions

None.

Usage Notes

If this method is called from a SQL query, ensure that the constraint rule definition does not contain any <ACTION> elements.

Use the preference parameter `EXP_IF_NULL_ATTR_IN_CONSTRAINT` to indicate whether to throw exceptions when encountering missing attributes or null attribute values during conformance validation. See [Section 10.2.6.2](#) for more information about this preference parameter.

Examples

Delete any existing conformance validation messages. Check if the DICOM content stored in an ORDImage object is conformance valid. Then, show any new conformance validation messages that are generated.

```
-- clear the previous conformance validation messages  
delete from orddcm_conformance_vld_msgs;  
  
-- conformance validate the DICOM content  
declare  
  src ordimage;  
begin
```

```
select image_src into src from medical_image_rel where id = 1;
dbms_output.put_line('isConformanceValid(PatientModule) ' ||
ord_dicom.isConformanceValid(src, 'PatientModule'));
end;
/

-- get the logged conformance validation messages
select message, msg_time time from orddcm_conformance_vld_msgs
where rule_name='PatientModule';
```

DICOM Relational Procedures

The ORD_DICOM package defines these DICOM relational procedures:

- [createDicomImage\(\)](#) for BFILES on page 8-20
- [createDicomImage\(\)](#) for BLOBs on page 8-22
- [createDicomImage\(\)](#) for ORDImage on page 8-24
- [export\(\)](#) on page 8-26
- [importFrom\(\)](#) on page 8-28
- [makeAnonymous\(\)](#) for BFILES on page 8-30
- [makeAnonymous\(\)](#) for BLOBs on page 8-32
- [makeAnonymous\(\)](#) for ORDImage on page 8-34
- [processCopy\(\)](#) for BFILES on page 8-36
- [processCopy\(\)](#) for BFILES with SOP Instance UID on page 8-37
- [processCopy\(\)](#) for BLOBs on page 8-39
- [processCopy\(\)](#) for BLOBs with SOP Instance UID on page 8-40
- [processCopy\(\)](#) for ORDImage on page 8-42
- [processCopy\(\)](#) for ORDImage with SOP Instance UID on page 8-44
- [writeMetadata\(\)](#) for BFILES on page 8-46
- [writeMetadata\(\)](#) for BLOBs on page 8-48
- [writeMetadata\(\)](#) for ORDImage on page 8-50

Note: In this section, *<unique-UID>* represents a 64-byte, dot-concatenated, numeric string that represents a globally unique identifier for DICOM content worldwide. The UID is commonly constructed with a root that uniquely identifies the organization producing the DICOM content, and a suffix that uniquely identifies the DICOM content within that organization. For some examples in this section, you must replace *<unique-UID>* with the appropriate UID.

createDicomImage() for BFILEs

Format

```
createDicomImage(  
    src IN BFILE,  
    metadata IN SYS.XMLTYPE,  
    dest IN OUT NOCOPY BLOB)
```

Description

Creates a DICOM image from a source image and DICOM metadata.

Parameters

src

The source raster image stored in a BFILE.

metadata

DICOM metadata stored in data type XMLType. The metadata must include all standard and private attributes. It must include a new SOP instance UID for the destination DICOM image, ensuring that the destination DICOM content is globally unique.

dest

An empty BLOB in which to store the DICOM image created from the source image and metadata.

Pragmas

None.

Exceptions

None.

Usage Notes

See [Appendix C](#) for information about the encoding rules that support image content processing.

Use the preference parameter `VALIDATE_METADATA` to specify whether the XML documents are validated against the Oracle default DICOM metadata schema. See [Section 10.2.6.7](#) for more information about this preference parameter.

Examples

Create a DICOM image from a source BFILE and DICOM metadata:

```
declare  
    src bfile;  
    dest blob;  
    metadata xmltype;  
begin  
    metadata := xmltype(bfilename('DICOMDIR', 'wm_meta.xml'),  
                        nls_charset_id('AL32UTF8'),  
                        'http://xmlns.oracle.com/ord/dicom/metadata_1_0');  
  
    select bfile_src, blob_dest into src, dest from medical_image_rel
```



```
    where id = 1 for update;  
  
    ord_dicom.createDicomImage(src, metadata, dest);  
end;  
/
```

createDicomImage() for BLOBs

Format

```
createDicomImage(  
    src IN BLOB,  
    metadata IN SYS.XMLTYPE,  
    dest IN OUT NOCOPY BLOB)
```

Description

Creates a DICOM image from a source image and DICOM metadata.

Parameters

src

The source raster image stored in a BLOB.

metadata

DICOM metadata stored in data type XMLType. The metadata must include all standard and private attributes. It must include a new SOP instance UID for the destination DICOM image, ensuring that the destination DICOM content is globally unique.

dest

An empty BLOB in which to store the DICOM image created from the source image and metadata.

Pragmas

None.

Exceptions

None.

Usage Notes

See [Appendix C](#) for information about the encoding rules that support image content processing.

Use the preference parameter `VALIDATE_METADATA` to specify whether the XML documents are validated against the Oracle default DICOM metadata schema. See [Section 10.2.6.7](#) for more information about this preference parameter.

Examples

Create a DICOM image from a source BLOB and DICOM metadata:

```
declare  
    src blob;  
    dest blob;  
    metadata xmltype;  
begin  
    metadata := xmltype(bfilename('DICOMDIR', 'wm_meta.xml'),  
                        nls_charset_id('AL32UTF8'),  
                        'http://xmlns.oracle.com/ord/dicom/metadata_1_0');  
  
    select blob_src, blob_dest into src, dest from medical_image_rel
```

```
        where id = 1 for update;

        ord_dicom.createDicomImage(src, metadata, dest);
end;
/
```

createDicomImage() for ORDImage

Format

```
createDicomImage(  
    src IN ORDSYS.ORDImage,  
    metadata IN SYS.XMLTYPE,  
    dest IN OUT NOCOPY BLOB)
```

Description

Creates a DICOM image from a source image and DICOM metadata.

Parameters

src

The source raster image stored in an ORDImage object.

metadata

DICOM metadata stored in data type XMLType. The metadata must include all standard and private attributes. It must include a new SOP instance UID for the destination DICOM image, ensuring that the destination DICOM content is globally unique.

dest

An empty BLOB in which to store the DICOM image created from the source image and metadata.

Pragmas

None.

Exceptions

None.

Usage Notes

See [Appendix C](#) for information about the encoding rules that support image content processing.

Use the preference parameter `VALIDATE_METADATA` to specify whether the XML documents are validated against the Oracle default DICOM metadata schema. See [Section 10.2.6.7](#) for more information about this preference parameter.

Examples

Create a DICOM image from a source ORDImage object and DICOM metadata:

```
declare  
    src ordimage;  
    dest blob;  
    metadata xmltype;  
begin  
    metadata := xmltype(bfilename('DICOMDIR', 'wm_meta.xml'),  
                        nls_charset_id('AL32UTF8'),  
                        'http://xmlns.oracle.com/ord/dicom/metadata_1_0');  
  
    select image_src, blob_dest into src, dest from medical_image_rel
```

```
    where id = 1 for update;  
  
    ord_dicom.createDicomImage(src, metadata, dest);  
end;  
/
```

export()

Format

```
export(  
  src          IN  BLOB,  
  dest_type   IN  VARCHAR2,  
  dest_location IN VARCHAR2,  
  dest_name   IN  VARCHAR2)
```

Description

Exports DICOM content in a BLOB to a specified destination. The data remains in the source BLOB when it is copied to the destination.

Parameters

src

The source location of the DICOM content.

dest_type

The type of the destination (only `FILE` is supported).

dest_location

The location of the destination (must be a valid Oracle directory object).

dest_name

The name of the destination file.

Pragmas

None.

Exceptions

None.

Usage Notes

The `export()` procedure writes only to a database directory object that the user has privilege to access. That is, you can access a directory object that you have created using the SQL statement `CREATE DIRECTORY`, or one to which you have been granted `READ` and `WRITE` access.

For example, the following SQL*Plus commands create a directory object and grant the user `pm` permission to read and write to any file within the directory

```
c:\mydir\work:
```

```
CONNECT sys as sysdba  
Enter password: password  
CREATE OR REPLACE DIRECTORY DICOMDIR AS 'c:\mydir\work';  
GRANT READ,WRITE ON DIRECTORY DICOMDIR TO pm;
```

See [Section 8.1](#) for more information about these directory and table definitions.

Examples

Export DICOM content from a BLOB to a specified file:

```
declare
  src blob;
begin
  select blob_src into src from medical_image_rel where id = 1;
  ord_dicom.export(src, 'FILE', 'DICOMDIR', 'exported.dcm');
end;
/
```

importFrom()

Format

```
importFrom(  
  dest      IN OUT NOCOPY BLOB,  
  source_type IN  VARCHAR2,  
  source_location IN  VARCHAR2,  
  source_name IN  VARCHAR2)
```

Description

Imports DICOM content from a specified source into a BLOB.

Parameters

dest

The storage destination of the imported DICOM file.

source_type

The type of the source (only FILE is supported).

source_location

The location of the source (must be a valid Oracle directory object).

source_name

The name of the source file.

Pragmas

None.

Exceptions

None.

Usage Notes

The importFrom() procedure reads only from a database directory object that the user has privilege to access. That is, you can access a directory object that you have created using the SQL statement CREATE DIRECTORY, or one to which you have been granted READ access.

For example, the following SQL*Plus commands create a directory object and grant the user pm permission to read any file within the directory c:\mydir\work:

```
CONNECT sys as sysdba  
Enter password: password  
CREATE OR REPLACE DIRECTORY DICOMDIR AS 'c:\mydir\work';  
GRANT READ ON DIRECTORY DICOMDIR TO pm;
```

See [Section 8.1](#) for more information about these directory and table definitions.

Examples

Import the DICOM content into a BLOB:

```
declare  
  dest blob;
```



```
begin
  select blob_dest into dest from medical_image_rel where id = 1 for update;
  ord_dicom.importFrom(dest, 'file', 'DICOMDIR', 'example.dcm');
end;
/
```

makeAnonymous() for BFILES

Format

```
makeAnonymous(  
  src IN BFILE,  
  dest_sop_instance_uid IN VARCHAR2,  
  dest IN OUT NOCOPY BLOB,  
  anonymityDocName IN VARCHAR2 DEFAULT 'ordcman.xml')
```

Description

Removes patient identifying information from the source DICOM content after copying it into another DICOM content BLOB, based on a specified anonymity document.

Parameters

src

The input DICOM content stored in a BFILE.

dest_sop_instance_uid

The SOP instance UID of the destination DICOM content. It must ensure that the destination DICOM content is globally unique.

dest

An empty BLOB in which to store the anonymous DICOM content.

anonymityDocName

The name of the anonymity document. The default name is `ordcman.xml`.

Pragmas

None.

Exceptions

None.

Usage Notes

None.

Examples

Remove patient identifying information from the embedded DICOM content stored in a BFILE:

Note: Replace `<unique-UID>` with the UID that identifies the organization producing the DICOM content and the DICOM content within that organization.

```
declare  
  src bfile;  
  dest blob;  
  dest_sop_instance_uid varchar2(128) := '<unique-UID>';
```

```
begin
  select bfile_src, blob_dest into src, dest from medical_image_rel
  where id = 1 for update;
  ord_dicom.makeAnonymous(src, dest_sop_instance_uid, dest, 'ordcman.xml');
end;
/
```

makeAnonymous() for BLOBs

Format

```
makeAnonymous(  
  src IN BLOB,  
  dest_sop_instance_uid IN VARCHAR2,  
  dest IN OUT NOCOPY BLOB,  
  anonymityDocName IN VARCHAR2 DEFAULT 'ordcman.xml')
```

Description

Removes patient identifying information from the source DICOM content after copying it into another DICOM content BLOB, based on a specified anonymity document.

Parameters

src

The input DICOM content stored in a BLOB.

dest_sop_instance_uid

The SOP instance UID of the destination DICOM content. It must ensure that the destination DICOM content is globally unique.

dest

An empty BLOB in which to store the anonymous DICOM content.

anonymityDocName

The name of the anonymity document. The default name is `ordcman.xml`.

Pragmas

None.

Exceptions

None.

Usage Notes

None.

Examples

Remove patient identifying information from the embedded DICOM content stored in a BLOB:

Note: Replace `<unique-UID>` with the UID that identifies the organization producing the DICOM content and the DICOM content within that organization.

```
declare  
  src blob;  
  dest blob;  
  dest_sop_instance_uid varchar2(128) := '<unique-UID>';
```

```
begin
  select blob_src, blob_dest into src, dest from medical_image_rel
  where id = 1 for update;
  ord_dicom.makeAnonymous(src, dest_sop_instance_uid, dest, 'ordcman.xml');
end;
/
```

makeAnonymous() for ORDImage

Format

```
makeAnonymous(  
  src IN ORDSYS.ORDImage,  
  dest_sop_instance_uid IN VARCHAR2,  
  dest IN OUT NOCOPY BLOB,  
  anonymityDocName IN VARCHAR2 DEFAULT 'ordcman.xml')
```

Description

Removes patient identifying information from the source DICOM content after copying it into another DICOM content BLOB, based on a specified anonymity document.

Parameters

src

The input DICOM content stored in an ORDImage object.

dest_sop_instance_uid

The SOP instance UID of the destination DICOM content. It must ensure that the destination DICOM content is globally unique.

dest

An empty BLOB in which to store the anonymous DICOM content.

anonymityDocName

The name of the anonymity document. The default name is `ordcman.xml`.

Pragmas

None.

Exceptions

None.

Usage Notes

None.

Examples

Remove patient identifying information from the embedded DICOM content stored in an ORDImage object:

Note: Replace `<unique-UID>` with the UID that identifies the organization producing the DICOM content and the DICOM content within that organization.

```
declare  
  src ordimage;  
  dest blob;  
  dest_sop_instance_uid varchar2(128) := '<unique-UID>';
```

```
begin
  select image_src, blob_dest into src, dest from medical_image_rel
  where id = 1 for update;
  ord_dicom.makeAnonymous(src, dest_sop_instance_uid, dest, 'ordcman.xml');
end;
/
```

processCopy() for BFILEs

Format

```
processCopy(  
  src IN BFILE,  
  command IN VARCHAR2,  
  dest IN OUT NOCOPY BLOB)
```

Description

Reads the DICOM content that is input from the source BFILE, writes it to the destination BLOB, and then performs the specified processing operations on the destination BLOB. The original DICOM content that was input remains unchanged.

Parameters

src

The input DICOM content stored in the source BFILE.

command

A command string that accepts a processing operator as input. Valid values include: `fileFormat`, `frame`, `contentFormat`, `compressionFormat`, `cut`, `scale`, and `rotate`. See [Appendix D](#) for information about processing operators.

dest

An empty BLOB in which to store the destination content.

Pragmas

None.

Exceptions

None.

Usage Notes

See [Appendix C](#) for information about the encoding rules that support DICOM content processing. See [Appendix D](#) for more information about DICOM processing.

Use the preference parameter `VALIDATE_METADATA` to specify whether the XML documents are validated against the Oracle default DICOM metadata schema. See [Section 10.2.6.7](#) for more information about this preference parameter.

Examples

Copy the DICOM content from a BFILE into a BLOB and then process it:

```
declare  
  src bfile;  
  dest blob;  
begin  
  select bfile_src, blob_dest into src, dest from medical_image_rel  
  where id = 1 for update;  
  ord_dicom.processCopy(src, 'fileFormat=jpeg maxScale=100 100', dest);  
end;  
/
```


processCopy() for BFILEs with SOP Instance UID

Format

```
processCopy(src IN BFILE,  
           command IN VARCHAR2,  
           dest_sop_instance_uid IN VARCHAR2,  
           dest IN OUT NOCOPY BLOB,  
           metadata IN SYS.XMLTYPE DEFAULT NULL)
```

Description

Reads the DICOM content that is input from the source BFILE, writes it to the destination BLOB, performs the specified processing operations on the destination BLOB, and then updates it with the new metadata. The original DICOM content that was input remains unchanged.

Parameters

src

The input DICOM content stored in the source BFILE.

command

A command string that accepts a processing operator as input. Valid values include: `frame`, `contentFormat`, `compressionFormat`, `cut`, `scale`, and `rotate`. See [Appendix D](#) for information about processing operators.

dest_sop_instance_uid

The SOP instance UID of the destination DICOM content. It must ensure that the destination DICOM content is globally unique.

dest

An empty BLOB in which to store the destination content.

metadata

The new metadata to be written into the new DICOM content.

Pragmas

None.

Exceptions

None.

Usage Notes

See [Appendix C](#) for information about the encoding rules that support DICOM content processing. See [Appendix D](#) for more information about DICOM processing.

Use the preference parameter `VALIDATE_METADATA` to specify whether the XML documents are validated against the Oracle default DICOM metadata schema. See [Section 10.2.6.7](#) for more information about this preference parameter.

Examples

Copy the DICOM content from a BFILE into a BLOB with a specified SOP instance UID and then process it:

Note: Replace *<unique-UID>* with the UID that identifies the organization producing the DICOM content and the DICOM content within that organization.

```
src bfile;
dest blob;
dest_sop_instance_uid varchar2(128) := '<unique-UID>';
begin
  select bfile_src, blob_dest into src, dest from medical_image_rel
  where id = 1 for update;
  ord_dicom.processCopy(
    src, 'CompressionFormat=jpeg', dest_sop_instance_uid, dest);
end;
/
```

processCopy() for BLOBs

Format

```
processCopy(
  src IN BLOB,
  command IN VARCHAR2,
  dest IN OUT NOCOPY BLOB)
```

Description

Reads the DICOM content that is input from the source BLOB, writes it to the destination BLOB, and then performs the specified processing operations on the destination BLOB. The original DICOM content that was input remains unchanged.

Parameters

src

The input DICOM content stored in the source BLOB.

command

A command string that accepts a processing operator as input. Valid values include: `fileFormat`, `frame`, `contentFormat`, `compressionFormat`, `cut`, `scale`, and `rotate`. See [Appendix D](#) for information about processing operators.

dest

An empty BLOB in which to store the destination content.

Pragmas

None.

Exceptions

None.

Usage Notes

See [Appendix C](#) for information about the encoding rules that support DICOM content processing. See [Appendix D](#) for more information about DICOM processing.

Use the preference parameter `VALIDATE_METADATA` to specify whether the XML documents are validated against the Oracle default DICOM metadata schema. See [Section 10.2.6.7](#) for more information about this preference parameter.

Examples

Copy the DICOM content from a BLOB into another BLOB and then process it:

```
src blob;
dest blob;
begin
  select blob_src, blob_dest into src, dest from medical_image_rel
  where id = 1 for update;
  ord_dicom.processCopy(src, 'fileFormat=jpeg maxScale=100 100' dest);
end;
/
```

processCopy() for BLOBs with SOP Instance UID

Format

```
processCopy(src IN BLOB,  
            command IN VARCHAR2,  
            dest_sop_instance_uid IN VARCHAR2,  
            dest IN OUT NOCOPY BLOB,  
            metadata IN SYS.XMLTYPE DEFAULT NULL)
```

Description

Reads the DICOM content that is input from the source BLOB, writes it to the destination BLOB, performs the specified processing operations on the destination BLOB, and then updates it with the new metadata. The original DICOM content that was input remains unchanged.

Parameters

src

The input DICOM content stored in the source BLOB.

command

A command string that accepts a processing operator as input. Valid values include: `frame`, `contentFormat`, `compressionFormat`, `cut`, `scale`, and `rotate`. See [Appendix D](#) for information about processing operators.

dest_sop_instance_uid

The SOP instance UID of the destination DICOM content. It must ensure that the destination DICOM content is globally unique.

dest

An empty BLOB in which to store the destination content.

metadata

The new metadata to be written into the new DICOM content.

Pragmas

None.

Exceptions

None.

Usage Notes

See [Appendix C](#) for information about the encoding rules that support DICOM content processing. See [Appendix D](#) for more information about DICOM processing.

Use the preference parameter `VALIDATE_METADATA` to specify whether the XML documents are validated against the Oracle default DICOM metadata schema. See [Section 10.2.6.7](#) for more information about this preference parameter.

Examples

Copy the DICOM content from a BLOB into another BLOB with a specified SOP instance UID and then process it:

Note: Replace *<unique-UID>* with the UID that identifies the organization producing the DICOM content and the DICOM content within that organization.

```
declare
  src blob;
  dest blob;
  dest_sop_instance_uid varchar2(128) := '<unique-UID>';
begin
  select blob_src, blob_dest into src, dest from medical_image_rel
  where id = 1 for update;
  ord_dicom.processCopy(
    src, 'CompressionFormat=jpeg', dest_sop_instance_uid, dest);
end;
/
```

processCopy() for ORDImage

Format

```
processCopy(  
  src IN ORDSYS.ORDImage,  
  command IN VARCHAR2,  
  dest IN OUT NOCOPY BLOB)
```

Description

Reads the DICOM content that is input from the source ORDImage object, writes it to the destination BLOB, and then performs the specified processing operations on the destination BLOB. The original DICOM content that was input remains unchanged.

Parameters

src

The input DICOM content stored in the source ORDImage object.

command

A command string that accepts an image processing operator as input. Valid values include: `fileFormat`, `frame`, `contentFormat`, `compressionFormat`, `cut`, `scale`, and `rotate`. See [Appendix D](#) for information about image processing operators.

dest

An empty BLOB in which to store the destination content.

Pragmas

None.

Exceptions

None.

Usage Notes

Use this method to get a non-DICOM image that is suitable for presentation on the Web from the embedded DICOM content.

See [Appendix C](#) for information about the encoding rules that support image content processing. See [Appendix D](#) for more information about DICOM processing.

Use the preference parameter `VALIDATE_METADATA` to specify whether the XML documents are validated against the Oracle default DICOM metadata schema. See [Section 10.2.6.7](#) for more information about this preference parameter.

Examples

Copy the DICOM content from an ORDImage object into a BLOB and then process it:

```
declare  
  src ordimage;  
  dest blob;  
begin  
  select image_src, blob_dest into src, dest from medical_image_rel  
  where id = 1 for update;  
  ord_dicom.processCopy(src, 'fileFormat=jpeg maxScale=100 100' dest);
```

```
end;  
/
```

processCopy() for ORDImage with SOP Instance UID

Format

```
processCopy(src IN ORDSYS.ORDImage,  
            command IN VARCHAR2,  
            dest_sop_instance_uid IN VARCHAR2,  
            dest IN OUT NOCOPY BLOB,  
            metadata IN SYS.XMLTYPE DEFAULT NULL)
```

Description

Reads the DICOM content that is input from the source ORDImage object, writes it to the destination BLOB, performs the specified processing operations on the destination BLOB, and then updates it with the new metadata. The original DICOM content that was input remains unchanged.

Parameters

src

The input DICOM content stored in the source ORDImage object.

command

A command string that accepts an image processing operator as input. Valid values include: `frame`, `contentFormat`, `compressionFormat`, `cut`, `scale`, and `rotate`. See [Appendix D](#) for information about image processing operators.

dest_sop_instance_uid

The SOP instance UID of the destination DICOM image. It must ensure that the destination DICOM content is globally unique.

dest

An empty BLOB in which to store the destination content.

metadata

The new metadata to be written into the new DICOM content.

Pragmas

None.

Exceptions

None.

Usage Notes

Use this method to get a non-DICOM image that is suitable for presentation on the Web from the embedded DICOM content.

See [Appendix C](#) for information about the encoding rules that support image content processing. See [Appendix D](#) for more information about DICOM processing.

Use the preference parameter `VALIDATE_METADATA` to specify whether the XML documents are validated against the Oracle default DICOM metadata schema. See [Section 10.2.6.7](#) for more information about this preference parameter.

Examples

Copy the DICOM content from an ORDImage object into a BLOB with a specified SOP instance UID and then process it:

Note: Replace *<unique-UID>* with the UID that identifies the organization producing the DICOM content and the DICOM content within that organization.

```
declare
  src ordimage;
  dest blob;
  dest_sop_instance_uid varchar2(128) := '<unique-UID>';
begin
  select image_src, blob_dest into src, dest from medical_image_rel
  where id = 1 for update;
  ord_dicom.processCopy(
    src, 'CompressionFormat=jpeg', dest_sop_instance_uid, dest);
end;
/
```

writeMetadata() for BFILES

Format

```
writeMetadata(  
    src IN BFILE,  
    metadata IN SYS.XMLTYPE,  
    dest IN OUT NOCOPY BLOB),
```

Description

Writes or modifies the current DICOM content with the metadata provided by making a copy of the existing DICOM content in the destination BLOB, and then modifying the metadata. The original DICOM content remains unchanged. The attributes in the destination DICOM content are copied from the metadata that was input.

Parameters

src

The input DICOM content stored in a BFILE.

metadata

The input metadata stored in data type XMLType. In the destination DICOM content, the input metadata is used to update the values for attributes that are identical to attributes in the source DICOM content, or to add any new attributes. The metadata must conform to the default metadata schema with the namespace `http://xmlns.oracle.com/ord/dicom/metadata_1_0`. The SOP instance UID in the metadata must ensure that the destination DICOM content is globally unique.

dest

An empty BLOB in which to store the new DICOM content with the new metadata.

Pragmas

None.

Exceptions

None.

Usage Notes

See [Appendix C](#) for information about the encoding rules that support metadata extraction.

Use the preference parameter `VALIDATE_METADATA` to specify whether the XML documents are validated against the Oracle default DICOM metadata schema. See [Section 10.2.6.7](#) for more information about this preference parameter.

Use the preference parameter `SQ_WRITE_LEN` to specify how the DICOM sequence (SQ) types are encoded. See [Section 10.2.6.6](#) for more information about this preference parameter.

Examples

Write the new metadata to the copy of the DICOM content in the destination BLOB:

```
declare  
    src bfile;
```

```
dest blob;
metadata xmltype;
begin
  metadata := xmltype(bfilename('DICOMDIR', 'wm_meta.xml'),
    nls_charset_id('AL32UTF8'),
    'http://xmlns.oracle.com/ord/dicom/metadata_1_0');

  select bfile_src, blob_dest into src, dest from medical_image_rel
    where id = 1 for update;

  ord_dicom.writeMetadata(src, metadata, dest);

end;
/
```

writeMetadata() for BLOBs

Format

```
writeMetadata(  
  src IN BLOB,  
  metadata IN SYS.XMLTYPE,  
  dest IN OUT NOCOPY BLOB),
```

Description

Writes or modifies the current DICOM content with the metadata provided by making a copy of the existing DICOM content in the destination BLOB, and then modifying the metadata. The original DICOM content remains unchanged. The attributes in the destination DICOM content are copied from the metadata that was input.

Parameters

src

The input DICOM content stored in a BLOB.

metadata

The input metadata stored in data type XMLType. In the destination DICOM content, the input metadata is used to update the values for attributes that are identical to attributes in the source DICOM content, or to add any new attributes. The metadata must conform to the default metadata schema with the namespace `http://xmlns.oracle.com/ord/dicom/metadata_1_0`. The SOP instance UID in the metadata must ensure that the destination DICOM content is globally unique.

dest

An empty BLOB in which to store the new DICOM content with the new metadata.

Pragmas

None.

Exceptions

None.

Usage Notes

See [Appendix C](#) for information about the encoding rules that support metadata extraction.

Use the preference parameter `VALIDATE_METADATA` to specify whether the XML documents are validated against the Oracle default DICOM metadata schema. See [Section 10.2.6.7](#) for more information about this preference parameter.

Use the preference parameter `SQ_WRITE_LEN` to specify how the DICOM sequence (SQ) types are encoded. See [Section 10.2.6.6](#) for more information about this preference parameter.

Examples

Write the new metadata to the copy of the DICOM content in the destination BLOB:

```
declare  
  src blob;
```

```
dest blob;
metadata xmltype;
begin
  metadata := xmltype(bfilename('DICOMDIR', 'wm_meta.xml'),
    nls_charset_id('AL32UTF8'),
    'http://xmlns.oracle.com/ord/dicom/metadata_1_0');

  select blob_src, blob_dest into src, dest from medical_image_rel
    where id = 1 for update;

  ord_dicom.writeMetadata(src, metadata, dest);

end;
/
```

writeMetadata() for ORDImage

Format

```
writeMetadata(  
  src IN ORDSYS.ORDImage,  
  metadata IN SYS.XMLTYPE,  
  dest IN OUT NOCOPY BLOB),
```

Description

Writes or modifies the current DICOM content with the metadata provided by making a copy of the existing DICOM content in the destination BLOB, and then modifying the metadata. The original DICOM content remains unchanged. The attributes in the destination DICOM content are copied from the metadata that was input.

Parameters

src

The input DICOM content stored in an ORDImage object.

metadata

The input metadata stored in data type XMLType. In the destination DICOM content, the input metadata is used to update the values for attributes that are identical to attributes in the source DICOM content, or to add any new attributes. The metadata must conform to the default metadata schema with the namespace `http://xmlns.oracle.com/ord/dicom/metadata_1_0`. The SOP instance UID in the metadata must ensure that the destination DICOM content is globally unique.

dest

An empty BLOB in which to store the new DICOM content with the new metadata.

Pragmas

None.

Exceptions

None.

Usage Notes

See [Appendix C](#) for information about the encoding rules that support metadata extraction.

Use the preference parameter `VALIDATE_METADATA` to specify whether the XML documents are validated against the Oracle default DICOM metadata schema. See [Section 10.2.6.7](#) for more information about this preference parameter.

Use the preference parameter `SQ_WRITE_LEN` to specify how the DICOM sequence (SQ) types are encoded. See [Section 10.2.6.6](#) for more information about this preference parameter.

Examples

Write the new metadata to the copy of the DICOM content in the destination BLOB:

```
declare  
  src ordimage;
```

```
dest blob;
metadata xmltype;
begin
  metadata := xmltype(bfilename('DICOMDIR', 'wm_meta.xml'),
    nls_charset_id('AL32UTF8'),
    'http://xmlns.oracle.com/ord/dicom/metadata_1_0');

  select image_src, blob_dest into src, dest from medical_image_rel
    where id = 1 for update;

  ord_dicom.writeMetadata(src, metadata, dest);

end;
/
```


Part III

DICOM Administration Usage and Reference

This part contains user and reference information for administrators of the DICOM data model repository.

Part III includes these chapters:

- [Chapter 9, "Overview of DICOM Administration"](#)
- [Chapter 10, "Creating DICOM Configuration Documents"](#)
- [Chapter 11, "Administering the DICOM Repository"](#)
- [Chapter 12, "ORD_DICOM_ADMIN Package Reference"](#)

Overview of DICOM Administration

This chapter briefly describes the tasks that are related to the administration of the Oracle Multimedia DICOM data model repository.

The Oracle Multimedia DICOM data model repository is a set of collectively managed, user-configurable XML documents that defines the run-time behavior of Oracle Multimedia DICOM (see [Section 2.4.1](#)). Because Oracle Multimedia DICOM is fully functional after installing Oracle Multimedia, administrators need not access the repository unless they want to update it to configure Oracle Multimedia DICOM for a particular database instance. Each database has its own set of configuration documents. Administrators can customize the repository by adding or deleting configuration documents. Only one administrator at a time is permitted to make changes in the data model repository. Administrators of the DICOM data model repository are assigned the ORDADMIN role.

Administrators can perform several repository management tasks. This chapter provides guidelines for loading the repository, and retrieving, adding, and removing content from the repository. See [Chapter 10](#) for more information about writing configuration documents.

The DICOM data model repository provides an administrative (PL/SQL) application programming interface (API) for managing configuration documents. See [Chapter 12](#) for reference information about the ORD_DICOM_ADMIN data model repository API. Administrators can also use several other application programming interfaces (APIs), which are provided by Oracle Multimedia DICOM (see [Table 3-1](#)).

This chapter includes these sections:

- [Assigning Administrator Roles and Privileges](#) on page 9-2
- [Managing XML Schemas](#) on page 9-2
- [Loading the Data Model Repository](#) on page 9-4
- [Browsing the Repository with Views](#) on page 9-4
- [Exporting Documents from the Repository](#) on page 9-5
- [Inserting Documents into the Repository](#) on page 9-5
- [Updating Documents in the Repository](#) on page 9-7
- [Deleting Documents from the Repository](#) on page 9-8
- [Oracle Data Pump Utilities Support for the Data Model Repository](#) on page 9-9

9.1 Assigning Administrator Roles and Privileges

After installing Oracle Multimedia DICOM, the ORDADMIN role is created, with the database system privileges required for administration of the DICOM data model repository.

The ORDADMIN role must be assigned to the administrator of the DICOM data model repository. The following code segment shows a sample GRANT statement for the administrator `dcmadmin`:

```
GRANT ORDADMIN to dcmadmin;
```

Because of the way database roles behave, tasks for which the administrator must write PL/SQL named procedures require explicit privileges. The following code segment shows a sample of a GRANT statement for an administrator named `dcmadmin`:

```
GRANT EXECUTE on ORD_DICOM_ADMIN to dcmadmin;  
GRANT SELECT on orddcm_document_refs to dcmadmin;
```

Administrators must also be granted the WRITE privilege for specified directories. For example, during operations where configuration documents are exported, administrators must have WRITE access to the directory where those documents are to be stored.

All users can load the data model repository into memory structures and view several public views. Only administrators can export, insert, or delete configuration documents from the data model repository. And, only administrators can query administrator-only views.

Note: Keep these administrative guidelines in mind:

- The `editDataModel()` procedure enables administrators to lock the data model while making changes.
- Changes in the data model repository are committed only by using the `publishDataModel()` procedure. In addition to committing the changes, this procedure unlocks the data model, making it available to other administrators.

See the [editDataModel\(\) Procedure](#) and the [publishDataModel\(\) Procedure](#) for more information.

9.2 Managing XML Schemas

The DICOM XML schemas in Oracle Multimedia DICOM are registered as global XML schemas in Oracle Database with Oracle XML DB. If you define any custom XML schemas, they must be also registered as global schemas with Oracle XML DB. You can find and examine the registered XML schemas by querying the dictionary view `ALL_XML_SCHEMAS`.

The following subsections briefly describe how you can use XML DB with your DICOM XML schemas:

- [Registering XML Schemas](#)
- [Finding User-Defined XML Schemas](#)

See Also:

- *Oracle XML DB Developer's Guide* for more information about registering XML schemas
- *Oracle Database Reference* for more information about the dictionary view ALL_XML_SCHEMAS

9.2.1 Registering XML Schemas

The DICOM XML schemas that correspond to the default configuration documents supplied by Oracle are automatically registered during installation. (See [Appendix B](#) for listings of the installed DICOM XML schemas.) Oracle requires that you manually register all user-defined (custom) XML schemas as global schemas, using Oracle XML DB.

Note: Setting the parameter LOCAL to FALSE indicates that the schema is global.

See Also:

Oracle XML DB Developer's Guide for an example of how to register a custom schema as a global schema

9.2.2 Finding User-Defined XML Schemas

After registering the custom XML schemas for your DICOM data model repository, you can search for and examine them. Query the dictionary view ALL_XML_SCHEMAS to find all the registered XML schemas.

[Example 9–1](#) shows a sample of how to list the namespaces for the metadata schemas associated with the user-defined mapping documents in your DICOM data model repository.

Note: [Example 9–1](#) includes a call to the setDataModel() procedure. This procedure call might not be required in all situations.

See the [setDataModel\(\) Procedure](#) and [Section 9.3](#) for more information.

Example 9–1 Finding User-Registered XML Schemas

```
exec ord_dicom.setdatamodel;

select t.doc_name, xmlcast(xmlquery(
  'declare default element namespace
  "http://xmlns.oracle.com/ord/dicom/mapping_1_0"; $x//NAMESPACE'
  passing ord_dicom_admin.getdocumentcontent(t.doc_name) as "x"
  returning content) as varchar2(4000) ) as schema_url
  from orddcm_documents t
  where t.installed_by_oracle=0 and t.doc_type= 'MAPPING'
  order by t.doc_id asc;
```

DOC_NAME	SCHEMA_URL
sample_map_10.xml	http://www.mycompany.com/metatest10
sample_map_11.xml	http://www.mycompany.com/metatest11

See Also:

Oracle XML DB Developer's Guide for more information about finding registered XML schemas

9.3 Loading the Data Model Repository

At the start of every database session, administrators and users must load the data model repository from the database into memory structures. Users load the data model by calling the `setDataModel()` procedure. Administrators load the data model by calling either the `setDataModel()` procedure or the `editDataModel()` procedure.

After loading the repository into memory, administrators and users can call the `setDataModel()` procedure whenever the application requires the new data model changes.

Note: Keep these guidelines in mind:

- Administrators and users must call the `setDataModel()` procedure before calling any other DICOM methods, functions, or procedures.
- Administrators can call the `setDataModel()` procedure when no changes are being made to the data model (for example: when calling the `getDocumentContent()` procedure or the `exportDocument()` procedure only).
- Administrators can call the `editDataModel()` procedure when making changes to the data model (for example: when inserting or deleting documents).

See the [setDataModel\(\) Procedure](#) and the [editDataModel\(\) Procedure](#) for more information.

Using the DICOM data model utility in the `ORD_DICOM` package, administrators and users call the `setDataModel()` procedure as follows:

```
exec ord_dicom.setdatamodel;
```

See the [setDataModel\(\) Procedure](#) for more information.

Using the `ORD_DICOM_ADMIN` package, administrators call the `editDataModel()` procedure as follows:

```
exec ord_dicom_admin.editDataModel();
```

See the [editDataModel\(\) Procedure](#) for more information.

9.4 Browsing the Repository with Views

Several public views are available to all DICOM users for browsing the DICOM repository. Oracle Multimedia DICOM also provides the administrator view `orddcm_document_refs`, which lists the documents that are referenced by other documents in the repository.

Administrators commonly use the views `orddcm_documents`, `orddcm_document_types`, and `orddcm_document_refs` when inserting, updating, and deleting documents from the repository.

See [Table 3–2](#) for the list of repository views that are available to administrators.

9.5 Exporting Documents from the Repository

Before exporting documents from the repository (and possibly before making any changes to the configuration documents), administrators should perform the following tasks:

1. Call the `setDataModel()` procedure at the beginning of each database session to load the repository from the database into memory structures. Locking the repository at this point is not required.

Administrators (and other users) can also call this procedure whenever the application requires the new data model changes.

See the [setDataModel\(\) Procedure](#) for more information.

2. Obtain copies of the existing documents in the repository, using either the `getDocumentContent()` function or the `exportDocument()` procedure.

The `getDocumentContent()` function returns the specified document as data type `XMLType`.

The `exportDocument()` procedure writes the contents of the specified document to a specified file in a directory for which the administrator has been granted the `WRITE` privilege.

See the [getDocumentContent\(\) Function](#) and the [exportDocument\(\) Procedure](#) for more information.

9.6 Inserting Documents into the Repository

The process of inserting documents into the repository can involve the use of these procedures:

- `editDataModel()`
- `insertDocument()`
- `publishDataModel()`
- `rollbackDataModel()`

See [Chapter 12](#) for reference information about these procedures. See [Chapter 11](#) for examples of the insertion process.

The following subsections briefly describe the insertion process for different types of documents:

- [Inserting Anonymity, Mapping, and Constraint Documents](#)
- [Inserting Dictionary Documents](#)
- [Inserting Preference and UID Definition Documents](#)
- [Inserting Stored Tag List Documents](#)

9.6.1 Inserting Anonymity, Mapping, and Constraint Documents

For anonymity documents and mapping documents, the order of insertion is irrelevant. For constraint documents, however, the order of insertion is important. If there are dependencies among any of the constraint documents to be inserted, insert the documents without dependencies first. Then, insert the remaining documents.

9.6.2 Inserting Dictionary Documents

Inserting standard or private dictionary documents requires merging all dictionary attributes each time a new dictionary document is inserted, in accordance with the following rules:

- Attribute tags must be unique, and must not match existing wildcard tags.
- Attribute tags must not be used in other document types.

In addition, for private dictionary documents, attribute tags must not be included in any existing range tags.

Note: Oracle recommends limiting insertions of standard dictionary documents to reflect changes or additions to the DICOM standard only.

See the XML schemas `ordcmsd.xsd` and `ordcmpv.xsd` in [Appendix B](#) for more information about DICOM attributes and attribute tags in dictionary documents.

9.6.3 Inserting Preference and UID Definition Documents

The installed preference document (`ordcmpf.xml`) assigns default preference values. Administrators can insert a user-defined preference document into the repository to override some or all of the default preference values. Preference values that are not overridden by a user-defined preference document retain their Oracle-defined default values.

Administrators can also insert user-defined UID definition documents to add new private UID values, or to reflect updates in the DICOM standard.

To insert a user-defined preference or UID definition document, follow these steps:

1. Export the installed, Oracle-defined document and rename it.
2. Update the exported, renamed document, changing the parameter values as required.
3. Insert the updated user-defined document into the repository.

If a user-defined preference or UID definition document is later deleted, the Oracle-installed document is reapplied.

Note: Only one user-defined preference or UID definition document is permitted in the repository.

9.6.4 Inserting Stored Tag List Documents

Administrators can use the `generateTagListDocument()` function to create a stored tag list document, which is a list of the attribute tags used in a specified set of mapping and constraint documents in the repository.

The stored tag list document is an optional XML document that specifies the DICOM attributes to be extracted from the embedded DICOM content and stored in the XML metadata attribute of the ORDDicom object when the `setProperties()` method is called. Generally, stored tag list documents contain the attribute tags used in mapping and constraint documents.

See the [generateTagListDocument\(\) Function](#) for more information. See [Section 11.4](#) for an example of the stored tag list document insertion process.

9.7 Updating Documents in the Repository

The process of updating documents in the repository can involve the use of these procedures:

- `editDataModel()`
- `exportDocument()`
- `deleteDocument()`
- `insertDocument()`
- `publishDataModel()`
- `rollbackDataModel()`

See [Chapter 12](#) for reference information about these procedures. See [Section 11.2](#) for an example of the update process.

The following subsections briefly describe the update process for different types of documents:

- [Updating Anonymity, Mapping, and Constraint Documents](#)
- [Updating Dictionary Documents](#)
- [Updating Preference and UID Definition Documents](#)

9.7.1 Updating Anonymity, Mapping, and Constraint Documents

Updating anonymity documents, mapping documents, and constraint documents involves a similar set of actions. For both anonymity documents and mapping documents, follow these steps:

1. Export the existing document.
2. Edit the exported document.
3. Delete the existing document.
4. Insert the edited document.

Constraint documents are updated in reverse order from their insertion order. In addition, if there are dependencies among any of the constraint documents to be updated, update the documents with no dependencies first. Then, update the remaining documents.

9.7.2 Updating Dictionary Documents

Updating standard or private dictionary documents requires checking the attribute tags and dependencies with other documents in the repository. Standard and private dictionary documents can be updated only if no other documents are using the attribute tags that are defined in the new documents. To update the attribute tags that

are being used by other documents, first update the dependent documents to remove the referenced attribute tags. Then, update the dictionary tags in accordance with the following rules:

- Attribute tags must be unique, and must not match existing wildcard tags.
- Attribute tags must not be used in other document types.

In addition, for private dictionary documents, DICOM attribute tags must not be included in any existing range tags.

Note: Oracle recommends limiting updates of standard dictionary documents to reflect changes or additions to the DICOM standard only.

See the XML schemas `ordcmsd.xsd` and `ordcmpv.xsd` in [Appendix B](#) for more information about DICOM attributes and attribute tags in dictionary documents.

9.7.3 Updating Preference and UID Definition Documents

To update existing user-defined preference documents or UID definition documents, follow these steps:

1. Export the user-defined document.
2. Edit the exported document.
3. Delete the existing user-defined document.
4. Insert the edited document.

Note: UID values defined by Oracle or the DICOM standard must not be changed.

9.8 Deleting Documents from the Repository

The process of deleting documents from the repository can involve the use of these procedures:

- `editDataModel()`
- `exportDocument()`
- `deleteDocument()`
- `publishDataModel()`
- `rollbackDataModel()`

Use the `exportDocument()` procedure to save a copy of an original document before deleting it from the repository.

Only user-defined documents can be deleted. Documents installed by Oracle are default documents that cannot be removed. Additionally, to ensure conformance with referenced constraints, remove documents in the reverse order from the order in which they were loaded.

See [Chapter 12](#) for reference information about these procedures. See [Section 11.3](#) for an example of the deletion process.

The following subsections briefly describe the deletion process for different types of documents:

- [Deleting Anonymity, Mapping, and Constraint Documents](#)
- [Deleting Dictionary Documents](#)
- [Deleting Preference and UID Definition Documents](#)

9.8.1 Deleting Anonymity, Mapping, and Constraint Documents

For anonymity documents and mapping documents, the order of deletion is irrelevant. Constraint documents, however, must be deleted in reverse order from their insertion order.

9.8.2 Deleting Dictionary Documents

Deleting standard or private dictionary documents requires checking dependencies with other documents in the repository. For example, a user-defined dictionary document can be deleted only if no other documents reference it.

9.8.3 Deleting Preference and UID Definition Documents

When a user-defined preference document is deleted, the values of the preference parameters revert to the installed values from the default Oracle-defined preference document (`ordcmpf.xml`). Similarly, when a UID definition document is deleted, the UID values revert to the installed values from the default Oracle-defined UID definition document (`ordcmui.xml`).

9.9 Oracle Data Pump Utilities Support for the Data Model Repository

Export and import using Oracle Data Pump is described in *Oracle Database Utilities*. This section provides guidelines and examples for using the Oracle Data Pump commands `expdp` and `impdp` to perform export and import operations on the DICOM data model repository, which is stored in the ORDDATA schema. Do so *only* if you have made changes to your data model repository.

Caution: Because the tables in the ORDDATA schema can change with each database release, Oracle Data Pump Export and Import utilities for the ORDDATA schema are supported within the *same* database release only.

These examples use the commands `expdp` and `impdp`. After you enter the command with your user name and the appropriate parameters, you are prompted for a password. The examples in this section do not show these prompts.

The following subsections provide information about roles and modes for export and import operations on the DICOM data model repository:

- [Roles for Export and Import Operations](#)
- [Modes for Export and Import Operations](#)

See Also:

Oracle Database Utilities for more information about Oracle Data Pump and its command-line clients, `expdp` and `impdp`

9.9.1 Roles for Export and Import Operations

To perform export and import operations on the ORDDATA schema, administrators must have these roles, respectively:

- DATAPUMP_EXP_FULL_DATABASE
- DATAPUMP_IMP_FULL_DATABASE

See Also:

Oracle Database Utilities for more information about the required roles for the Oracle Data Pump Export and Import utilities

9.9.2 Modes for Export and Import Operations

Export and import operations for Oracle Multimedia are supported in the following modes:

- Schema (recommended)
- Full

The following subsections describe these modes for export and import operations:

- [Exporting in Schema Mode](#)
- [Exporting in Full Mode](#)
- [Importing in Schema Mode](#)
- [Importing in Full Mode](#)

9.9.2.1 Exporting in Schema Mode

Use schema mode to export the DICOM data model repository in the ORDDATA schema.

[Example 9–2](#) shows how to perform a schema mode export.

Example 9–2 Schema Mode Export

```
expdp SCHEMAS=ORDDATA CONTENT=ALL DIRECTORY=DUMP_DIR
      DUMPFILE=exp_orddata.dmp LOGFILE=exp_orddata.log
```

[Example 9–2](#) defines the following parameters and values:

Parameter and Value	Definition
SCHEMAS=ORDDATA	Specifies the schema to export. There is no default value.
CONTENT=ALL	Specifies the content to export. The default value ALL specifies to export data and metadata.
DIRECTORY=DUMP_DIR	Specifies the default directory location for the dump file and the log file. The directory DUMP_DIR must already exist, and you must have access to it.
DUMPFILE=exp_orddata.dmp	Specifies the name of the dump file for the export job.
LOGFILE=exp_orddata.dmp	Specifies the name of the log file for the export job.

Note: This operation does not export the corresponding metadata schemas, which are associated with any user-defined mapping documents in the ORDDATA schema. Without these metadata schemas, the data model repository is incomplete. (See [Section 9.2.2](#) for information about how to find these schemas.)

See Also:

Oracle Database Utilities for more information about schema mode export

9.9.2.2 Exporting in Full Mode

No recommendations apply for the ORDDATA schema.

See Also:

Oracle Database Utilities for more information about full mode export

9.9.2.3 Importing in Schema Mode

Use schema mode to import configuration documents into the DICOM data model repository.

[Example 9-3](#) shows how to perform a schema mode import.

Example 9-3 Schema Mode Import

```
impdp SCHEMAS=ORDDATA CONTENT=ALL TABLE_EXISTS_ACTION=REPLACE
      DIRECTORY=DUMP_DIR DUMPFILE=exp_orddata.dmp LOGFILE=imp_orddata.log
```

[Example 9-3](#) defines the following parameters and values:

Parameter and Value	Definition
SCHEMAS=ORDDATA	Specifies the schema to import. There is no default value.
CONTENT=ALL	Specifies the content to import. The default value ALL specifies to import data and metadata.
TABLE_EXISTS_ACTION=REPLACE	Specifies the action to take when the table already exists. Oracle supports only the value REPLACE.
DIRECTORY=DUMP_DIR	Specifies the default directory location for the dump file and the log file. The directory DUMP_DIR must already exist, and you must have access to it.
DUMPFILE=exp_orddata.dmp	Specifies the name of the dump file for the import job.
LOGFILE=imp_orddata.log	Specifies the name of the log file for the import job.

Note: This operation does not import the corresponding metadata schemas, which are associated with any user-defined mapping documents in the ORDDATA schema. Without these metadata schemas, the data model repository is incomplete. (See [Section 9.2.2](#) for information about how to find these schemas.)

This operation generates several errors; however, this behavior is expected because the metadata already exists.

See Also:

Oracle Database Utilities for more information about schema mode import

9.9.2.4 Importing in Full Mode

Oracle recommends using the following parameter setting to import the ORDDATA schema: `TABLE_EXISTS_ACTION=REPLACE`. If this setting does not apply to a full database import operation, Oracle recommends using the following two-step process:

1. Exclude the ORDDATA schema from the full import operation (see *Oracle Database Utilities*).
2. Perform a schema mode import of the ORDDATA schema with the supported setting for the parameter `TABLE_EXISTS_ACTION` (see [Example 9-3](#)).

See Also:

Oracle Database Utilities for more information about full mode import

Creating DICOM Configuration Documents

Administrators can perform several repository management tasks, including creating their own configuration documents. This chapter describes the characteristics of configuration documents and provides instructions on how to write configuration documents that are specific to a particular organization or enterprise.

In Oracle Database 11g Release 2 (11.2), Oracle Multimedia DICOM supports the processing of DICOM files that contain private attribute definer names consisting of special characters. To enable referencing of DICOM metadata tags that refer to private attribute definers using these special characters, Oracle Multimedia DICOM extends the syntax for value locators used in the DICOM configuration documents to enable the expression of these definers. The **DICOM value locator** type identifies DICOM attributes or their child components within the DICOM content. See [General Format for DICOM Value Locators](#) for reference information.

Administrators can use DICOM value locators with anonymity, constraint, and mapping documents. The DICOM value locator is also used in the mid-tier Java API. See [Section 10.2.1.5](#), [Section 10.2.2.5](#), and [Section 10.2.3.6](#), respectively, for more information about using DICOM value locators in anonymity, constraint, and mapping documents. See [Chapter 9](#) for more information about inserting, updating, and deleting configuration documents. See [Chapter 11](#) for examples of these processes.

Note: In this chapter, names of attributes for XML elements appear in monospace type.

This chapter includes these sections:

- [Characteristics of Configuration Documents](#) on page 10-2
- [Writing Configuration Documents](#) on page 10-5

For more information about the data structure and encoding definitions of DICOM content, see the DICOM standard. This standard is available on the Web site for the National Electrical Manufacturers Association (NEMA) at

<http://medical.nema.org/>

See Also:

Oracle Multimedia Mid-Tier Java API Reference for reference information about the Java classes that support DICOM value locators in the middle tier

10.1 Characteristics of Configuration Documents

After installation, the Oracle Multimedia DICOM data model repository contains the following set of default configuration documents:

- Anonymity (`ordcman.xml`)
- Constraint (`ordcmct.xml`, `ordcmcmd.xml`, `ordcmcmc.xml`)
- Mapping (`ordcmap.xml`)
- Standard Dictionary (`ordcmsd.xml`)
- Private Dictionary (`ordcmpv.xml`)
- Preference (`ordcmpf.xml`)
- UID Definition (`ordcmui.xml`)

Note: In addition, users can install an optional stored tag list document that specifies the DICOM attributes to be extracted from the embedded DICOM content and stored in the XML metadata attribute of the ORDDicom object when the `setProperties()` method is called.

See [Section 2.4.1](#) for a brief definition of each configuration document.

Each type of configuration document has a specific set of characteristics, which are briefly described in these sections:

- [Characteristics of Anonymity Documents](#)
- [Characteristics of Constraint Documents](#)
- [Characteristics of Mapping Documents](#)
- [Characteristics of Standard Dictionary Documents](#)
- [Characteristics of Private Dictionary Documents](#)
- [Characteristics of Preference Documents](#)
- [Characteristics of UID Definition Documents](#)
- [Characteristics of Stored Tag List Documents](#)

10.1.1 Characteristics of Anonymity Documents

Anonymity documents have these characteristics:

- No other document types depend on anonymity documents.
- There are no dependencies between anonymity documents.
- Anonymity documents depend on the standard and private dictionary documents.
- There is no maximum limit on the number of anonymity documents in the repository.
- Changes made to anonymity documents affect the results of the `isAnonymous()` and `makeAnonymous()` methods.

10.1.2 Characteristics of Constraint Documents

Constraint documents have these characteristics:

- No other document types depend on constraint documents.
- There may be dependencies between constraint documents.
- Constraint documents depend on the standard and private dictionary documents and the preference document.
- There is no limit on the number of constraint documents in the repository.
- Constraint documents can be written in such a way that later constraint documents depend on previously inserted constraint documents. As an example using installed constraint documents, `ordcmct.xml` depends on `ordcmcmd.xml`, and both of those documents depend on `ordcmcmc.xml`.
- The `isConformanceValid()` method depends on the specified constraint rule and the DICOM content. If the constraint rule (defined in a constraint document) is changed, the DICOM content may be validated differently.
- The preference parameter `EXP_IF_NULL_ATTR_IN_CONSTRAINT` in the preference document can be used to specify whether to raise an exception when encountering a null value for the attributes specified in the constraint rules. If this preference parameter is set to `true`, the `isConformanceValid()` method raises an exception; otherwise, the predicate that was encountered is evaluated to `false`.

10.1.3 Characteristics of Mapping Documents

Mapping documents have these characteristics:

- No other document types depend on mapping documents.
- There are no dependencies between mapping documents.
- Mapping documents depend on the standard and private dictionary documents and the preference document.
- There is no limit on the number of mapping documents in the repository.
- Changes made to mapping documents affect the results of the `extractMetadata()` method, when the mapping document is used as a parameter.
- If a mapping document specifies a metadata namespace value in the `<NAMESPACE>` element, the XML schema that corresponds to the metadata namespace value must be registered with Oracle XML DB.

See Also:

Oracle XML DB Developer's Guide for more information about registering XML schemas

To enable the `extractMetadata()` method to function correctly, the structure of the mapping document must be consistent with the XML schema that corresponds to the value of the `<NAMESPACE>` element.

- Extracted metadata is schema validated only if both of these conditions exist:
 - The mapping document specifies a value in the `<NAMESPACE>` element.
 - The value of the preference parameter `VALIDATE_METADATA` in the preference document is `true`.

10.1.4 Characteristics of Standard Dictionary Documents

Standard dictionary documents have these characteristics:

- One standard dictionary document (`ordcmsd.xml`) is installed by Oracle.
- Changes to standard dictionary documents must be limited to updates in the DICOM standard.
- There is no limit on the number of standard dictionary documents in the repository.

10.1.5 Characteristics of Private Dictionary Documents

Private dictionary documents have these characteristics:

- One private dictionary document (`ordcmpv.xml`) is installed by Oracle.
- There is no limit on the number of private dictionary documents in the repository.

10.1.6 Characteristics of Preference Documents

Preference documents have these characteristics:

- A maximum of two (one Oracle-defined and one user-defined) preference documents are permitted in the repository. The installed, Oracle-defined preference document (`ordcmpf.xml`) includes Oracle-fixed preference parameter names and lists of values. The user-defined preference document can update the default preference parameter values that were set in the Oracle-defined preference document.
- Changes to preference parameter values in preference documents change the behavior of DICOM methods, functions, and procedures. Specifically, preference parameter values that are defined in the user-defined preference document override the default values defined in the Oracle-defined preference document.

10.1.7 Characteristics of UID Definition Documents

UID definition documents have these characteristics:

- A maximum of two (one Oracle-defined and one user-defined) UID definition documents are permitted in the repository. The installed, Oracle-defined UID definition document (`ordcmui.xml`) includes the UID definitions listed in Part 6 of the DICOM standard.
- Changes to user-defined UID definition documents must be limited to updates in the DICOM standard or additions of new UID values.

Note: Existing UID values must not be changed.

10.1.8 Characteristics of Stored Tag List Documents

Stored tag list documents have these characteristics:

- Only one stored tag list document can exist in the repository.
- The stored tag list document depends on the standard and private dictionary documents and the preference document.
- The preference parameter `MANDATE_ATTR_TAGS_IN_STL` enforces the rule that all tags used by the constraint and mapping documents are listed in the stored tag list document. The value of the `MANDATE_ATTR_TAGS_IN_STL` preference parameter in the preference document is `false`, by default. If this preference parameter is set to `true`, the rule is enforced; otherwise it is not enforced.

10.2 Writing Configuration Documents

Administrators can write one or more configuration documents to support specific applications or organizations. These subsections describe how to create each type of configuration document:

- [Creating Anonymity Documents](#)
- [Creating Constraint Documents](#)
- [Creating Mapping Documents and Metadata XML Schemas](#)
- [Creating Standard Dictionary Documents](#)
- [Creating Private Dictionary Documents](#)
- [Creating Preference Documents](#)
- [Creating UID Definition Documents](#)
- [Creating Stored Tag List Documents](#)

10.2.1 Creating Anonymity Documents

Anonymity documents specify the set of attributes to be made anonymous, and the actions to be taken to make those attributes anonymous. In the ORDDicom object, anonymity documents are used by the methods `isAnonymous()` and `makeAnonymous()` to create new objects in which personally identifying information has been removed or replaced.

The XML schema `ordcman.xsd` defines the XML schema that constrains anonymity documents. (See [Section B.1](#) for a listing of the anonymity document schema `ordcman.xsd`.)

The default anonymity document, `ordcman.xml`, lists a subset of the attributes defined in the Basic Application Level Confidentiality Profile in Part 15 of the DICOM standard. These attributes are either removed or replaced with the string "anonymous" in the DICOM content. In addition, the default anonymity document removes all undefined standard attributes and all private attributes from the DICOM content.

Within each anonymity document, the `<ANONYMITY_ACTION>` element includes an `action` attribute. The value of the `action` attribute can apply to a single attribute or to a set of attributes within that anonymity document. Global actions apply to a set of attributes, such as all private attributes.

[Table 10–1](#) describes the valid values for the `action` attribute.

Table 10–1 *action Attribute Values*

Value	Description
none	No action is taken. The unchanged value of the specified attribute or the set of attributes appears in the resulting DICOM content.
remove	The default value. The specified attribute or the set of attributes is removed from the resulting DICOM content. Note: Some imaging applications may depend on certain attributes (for example: <code>SOP_INSTANCE_UID</code> or <code>SOP_CLASS_UID</code>). In these cases, use the <code>replace</code> action with an appropriate value in place of the <code>remove</code> action.

Table 10–1 (Cont.) action Attribute Values

Value	Description
replace	<p>The attribute value is replaced by the specified value in the resulting DICOM content. The specified replacement value must be a string representation that matches the data type of the tag as defined by the <VR> element in the data dictionaries.</p> <p>Note: This action value is not permitted for global actions.</p> <p>For example:</p> <p>The standard tag 00100022 represents the Patient ID, which has a <VR> value of CS (CODE_STRING) in the data dictionary. The data type CS is defined in the XML schema <code>ordcmrdt.xsd</code> as <code>xs:token of length 16</code>. The replacement value must be a string representation that conforms to this definition.</p> <p>The standard tag 00081160 represents the Referenced Frame Number, which has a <VR> value of IS (INTEGER_STRING). The data type IS is defined in the XML schema <code>ordcmrdt.xsd</code> as <code>xs:integer</code>. The replacement value must be a string representation that conforms to this definition.</p>

Attributes in anonymity documents can be standard or private. Standard and private attributes can be either defined or undefined. Defined standard attributes are defined in the DICOM standard and in the standard dictionary in the data model repository. Defined private attributes are defined by and specific to a particular organization. Defined private attributes known to Oracle Multimedia are defined in the private dictionary in the data model repository. Undefined attributes are defined in neither the standard dictionary nor the private dictionary in the data model repository.

The <INDIVIDUAL_ATTRIBUTE> element defines the action taken to make a defined standard or private attribute anonymous. For private attributes, the action specified in this element always overrides the global action defined by the <PRIVATE_ATTRIBUTES> element. In addition, standard or private attributes that are undefined cannot be specified as <INDIVIDUAL_ATTRIBUTE> element values in an anonymity document.

Table 10–2 describes the elements that define the global actions in the anonymity document.

Table 10–2 Global Action Elements

Element	Description
<PRIVATE_ATTRIBUTES>	<p>Specifies the action performed on all defined and undefined private attributes.</p> <p>Note: The action for a private attribute defined by the <INDIVIDUAL_ATTRIBUTE> element always overrides the global action defined by the <PRIVATE_ATTRIBUTES> element.</p>
<UNDEFINED_STANDARD_ATTRIBUTES>	<p>Specifies the action performed on all standard attributes that are not defined in the standard dictionaries in the data model repository.</p> <p>A DICOM attribute tag contains a group number and an element number.</p> <p>A standard attribute tag is identified by an even-numbered group number.</p>

Table 10–2 (Cont.) Global Action Elements

Element	Description
<UNDEFINED_PRIVATE_ATTRIBUTES>	<p>Specifies the action performed on all private attributes that are not defined in the private dictionaries in the data model repository.</p> <p>Note: The action value defined by this element takes precedence over the action value defined by the <PRIVATE_ATTRIBUTES > element.</p>

Examples of valid values for an <ATTRIBUTE_TAG> element are as follows:
00100010, 00100010 (DICOM), 10871100 (PRIVATE ORG).

Note: Currently, only these values are permitted for the definer name in a private attribute:

- The uppercase and lowercase letters: A–Z
- The numbers: 0–9
- The characters: ' . ' (dot), ' ' (space), and ' / ' (forward slash)

The following subsections contain examples that show how to create anonymity documents:

- [Making Standard Attributes Anonymous](#)
- [Making Undefined Standard Attributes Anonymous](#)
- [Making Selected Private Attributes Anonymous](#)
- [Making All Private Attributes Anonymous](#)
- [Using DICOM Value Locators in Anonymity Documents](#)

10.2.1.1 Making Standard Attributes Anonymous

This subsection shows how to replace a single standard attribute with specified values that make that attribute anonymous in the resulting DICOM content. [Example 10–1](#) shows a code segment for the standard attribute *Patient's Name*. The XML statements where these actions are defined are highlighted in bold.

Example 10–1 Making a Standard Attribute Anonymous

```
<INDIVIDUAL_ATTRIBUTE>
  <ATTRIBUTE_TAG>00100010</ATTRIBUTE_TAG>
  <DESCRIPTION>Patient's Name </DESCRIPTION>
  <ANONYMITY_ACTION action="replace">anonymous</ANONYMITY_ACTION>
</INDIVIDUAL_ATTRIBUTE>
```

The following code segment shows how the <ATTRIBUTE_TAG> value 00100010 is defined in the standard dictionary in the data model repository. If the definer name is not specified in the <ATTRIBUTE_TAG> value, the default value of "DICOM" is assumed. The value of the 00100010 tag is replaced with the value "anonymous" in the resulting DICOM content.

The standard attribute tag 00100010 is defined as follows in the standard dictionary:

```
<STANDARD_ATTRIBUTE_DEFINITION>
  <TAG>00100010</TAG>
```

```

    <NAME>Patient's Name</NAME>
    <VR>PN</VR>
    <VM>1</VM>
</STANDARD_ATTRIBUTE_DEFINITION>

```

The tag definition shown in the preceding code segment represents the standard attribute `Patient's Name`, of data type `PN` as defined by the `<VR>` element. The `<VR>` element is the value representation element used to specify standard data types, as defined in Part 5 of the DICOM standard. The data type `PN` is defined in the XML schema `ordcmrdt.xsd`. The replacement value for the attribute `Patient's Name` must be a string representation of the value defined by the `<VR>` element.

10.2.1.2 Making Undefined Standard Attributes Anonymous

This subsection shows how to use the action value `remove` to remove all undefined standard attributes from the resulting DICOM content. [Example 10–2](#) shows the XML statement where this action is defined.

Example 10–2 Removing All Undefined Standard Attributes

```
<UNDEFINED_STANDARD_ATTRIBUTES action="remove" />
```

10.2.1.3 Making Selected Private Attributes Anonymous

This subsection shows how to replace a single private attribute with specified values that make that attribute anonymous in the resulting DICOM content. [Example 10–3](#) shows a code segment for the private attribute `10871100 (PRIVATE ORG)`. The XML statements where these actions are defined are highlighted in bold.

Example 10–3 Making a Private Attribute Anonymous

```

<INDIVIDUAL_ATTRIBUTE>
  <ATTRIBUTE_TAG>10871100 (PRIVATE ORG)</ATTRIBUTE_TAG>
  <DESCRIPTION>Media Type </DESCRIPTION>
  <ANONYMITY_ACTION action="replace"> replaced</ANONYMITY_ACTION>
</INDIVIDUAL_ATTRIBUTE>

```

In [Example 10–3](#), the `<ATTRIBUTE_TAG>` value `10871100` with the definer name `PRIVATE ORG` must be defined in a private dictionary in the data model repository. The value of the private attribute `10871100 (PRIVATE ORG)` is replaced with the specified value in the resulting DICOM content.

The private attribute tag `10871100` is defined as follows in the private dictionary:

```

<PRIVATE_ATTRIBUTE_DEFINITION>
  <TAG>10871100</TAG>
  <NAME>Media Type</NAME>
  <DEFINER>PRIVATE ORG</DEFINER>
  <VR>CS</VR>
  <VM>1</VM>
</PRIVATE_ATTRIBUTE_DEFINITION>

```

This tag definition shown in the preceding code segment represents the private attribute `10871100 (PRIVATE ORG)`, of data type `CS` and name `Media Type` as defined by the `<VR>` element. The data type `CS` is defined in the XML schema `ordcmrdt.xsd` as `xs:token of length 16`. The replacement value must be a string representation that conforms to the value defined by the `<VR>` element.

10.2.1.4 Making All Private Attributes Anonymous

This subsection shows how to make several private attributes anonymous by removing or replacing those attributes with specified values that make them anonymous in the resulting DICOM content.

Note: Keep these guidelines in mind:

- The action value for a specified private attribute defined by the `<INDIVIDUAL_ATTRIBUTE>` element always overrides the global action defined by the `<PRIVATE_ATTRIBUTES>` element.
 - The global action specified by the `<UNDEFINED_PRIVATE_ATTRIBUTES>` element overrides the global action defined by the `<PRIVATE_ATTRIBUTE>` element.
-

[Example 10-4](#) shows a code segment that uses the action value `remove` to remove all private attributes in the resulting DICOM content.

Example 10-4 Removing All Private Attributes

```
<PRIVATE_ATTRIBUTES action="remove"></PRIVATE_ATTRIBUTES>
<UNDEFINED_PRIVATE_ATTRIBUTES action="remove" />
```

Similarly, [Example 10-5](#) shows a code segment that uses the action value `remove` to remove all undefined private attributes from the resulting DICOM content. And, this code segment uses the action value `replace` to replace the value of the tag for the defined private attribute 10871100 (`PRIVATE ORG`) with the string `"anonymous"` in the resulting DICOM content. The XML statements where these actions are defined are highlighted in bold.

Example 10-5 Removing All Undefined Private Attributes

```
<PRIVATE_ATTRIBUTES action="remove"></PRIVATE_ATTRIBUTES>
<INDIVIDUAL_ATTRIBUTE>
  <ATTRIBUTE_TAG>10871100(PRIVATE ORG)</ATTRIBUTE_TAG>
  <DESCRIPTION>Media Type </DESCRIPTION>
  <ANONYMITY_ACTION action="replace"> anonymous</ANONYMITY_ACTION>
</INDIVIDUAL_ATTRIBUTE>
```

[Example 10-6](#) shows a code segment that uses the action value `none` to include all defined private attributes in the resulting DICOM content. And, this code segment uses the action value `remove` to remove all undefined private attributes from the resulting DICOM content.

Example 10-6 Including Defined Private Attributes and Removing Undefined Private Attributes

```
<PRIVATE_ATTRIBUTES action="none"></PRIVATE_ATTRIBUTES>
<UNDEFINED_PRIVATE_ATTRIBUTES action="remove" />
```

[Example 10-7](#) shows a code segment that uses the action value `remove` to remove all defined private attributes from the resulting DICOM content. And, this code segment uses the action value `none` to include all undefined private attributes in the resulting DICOM content.

Example 10–7 Removing Defined Private Attributes and Including Undefined Private Attributes

```
<PRIVATE_ATTRIBUTES action="remove"></PRIVATE_ATTRIBUTES>
<UNDEFINED_PRIVATE_ATTRIBUTES action="none" />
```

10.2.1.5 Using DICOM Value Locators in Anonymity Documents

This subsection shows how to use DICOM value locators to specify attributes in anonymity documents. [Example 10–8](#) shows how to specify the attribute Person Name with a DICOM value locator to make the attribute anonymous in the resulting DICOM content. This attribute is a child attribute of the attribute Content Sequence (data type SQ). The XML statement where this action is performed is highlighted in bold.

Example 10–8 Simple DICOM Value Locator in an Anonymity Document

```
<INDIVIDUAL_ATTRIBUTE>
  <ATTRIBUTE_TAG>0040A730[1].0040A123</ATTRIBUTE_TAG>
  <DESCRIPTION> Person Name </DESCRIPTION>
  <ANONYMITY_ACTION action="replace">anonymous</ANONYMITY_ACTION>
</INDIVIDUAL_ATTRIBUTE>
```

The standard attribute tags 0040A730 and 0040A123 are defined as follows in the standard dictionary:

```
<STANDARD_ATTRIBUTE_DEFINITION>
  <TAG>0040A730</TAG>
  <NAME>Content Sequence</NAME>
  <VR>SQ</VR>
  <VM>1</VM>
</STANDARD_ATTRIBUTE_DEFINITION>
<STANDARD_ATTRIBUTE_DEFINITION>
  <TAG>0040A123</TAG>
  <NAME>Person Name</NAME>
  <VR>PN</VR>
  <VM>1</VM>
</STANDARD_ATTRIBUTE_DEFINITION>
```

[Example 10–9](#) shows how to use a DICOM value locator with a wildcard character in an anonymity document to remove attributes with multiple values in the resulting DICOM content. The XML statement where the DICOM value locator is used is highlighted in bold.

Example 10–9 DICOM Value Locator with a Wildcard Character in an Anonymity Document

```
<INDIVIDUAL_ATTRIBUTE>
  <ATTRIBUTE_TAG>00081050[*]</ATTRIBUTE_TAG>
  <DESCRIPTION> Performing physician's name </DESCRIPTION>
  <ANONYMITY_ACTION action="remove"></ANONYMITY_ACTION>
</INDIVIDUAL_ATTRIBUTE>
```

The standard attribute tag 00081050 is defined as follows in the standard dictionary:

```
<STANDARD_ATTRIBUTE_DEFINITION>
  <TAG>00081050</TAG>
  <NAME>Performing Physician's Name</NAME>
  <VR>PN</VR>
  <VM>1-n</VM>
</STANDARD_ATTRIBUTE_DEFINITION>
```


In the tag definition shown in the preceding code segment, the `<VM>` element indicates that this attribute can contain multiple values. Using a wildcard character in the DICOM value locator ensures that all the values for this attribute are made anonymous.

Note: DICOM value locators with the TAG_FIELD component are not supported in anonymity documents.

See [General Format for DICOM Value Locators](#) for more information about the DICOM value locator type.

10.2.2 Creating Constraint Documents

Constraint documents define one or more constraint rules. The XML schema `ordcmct.xsd` defines the XML schema that constrains constraint documents. (See [Section B.2](#) for a listing of the constraint document schema `ordcmct.xsd`.)

The default constraint documents (`ordcmct.xml`, `ordcmcmd.xml`, `ordcmcmc.xml`), are XML representations of the rules to check the conformance of DICOM content, according to the DICOM standard and other organization-wide guidelines.

At run time, users can invoke a PL/SQL or Java function to check the conformance of DICOM content according to one or more invocable constraint rules. Each invocable constraint rule is defined as a global rule (using the `<GLOBAL_RULE>` element). Global rules are constraint rules that define requirements to be met by the DICOM content.

Constraint rules can comprise individual constraint predicates (using the `<PREDICATE>` element). A **constraint predicate** defines conditions on DICOM content. A predicate can be a logical statement, a relational statement that compares values, a function call evaluation that returns a Boolean type, or a reference to other predicate definitions. Predicate definitions are recursive. For example, when used as a logical statement, a predicate includes the logical OR of two other predicates. Each of the other predicates, in turn, can be a relational predicate.

Predicates operate on individual attributes or sets of attributes in the DICOM content. Within a predicate, attributes are specified with the `<ATTRIBUTE_TAG>` element. Attributes are defined by the DICOM standard or by the private organizations or enterprises that create DICOM content. Attributes can be specified using simple DICOM value locators or more complex DICOM value locators, including those with macro substitution strings (see [General Format for DICOM Value Locators](#)).

Complex constraint rules can be defined more simply using constraint macros. Each **constraint macro** can be defined as a global macro (using the `<GLOBAL_MACRO>` element). Constraint macros follow the same predicate definition grammar as constraint rules. They differ from constraint rules in that the predicate operands in constraint macros can contain macro parameters rather than the fixed values contained in constraint rules. The macro parameters in a constraint macro are replaced with parameter values when the macro is invoked (using the `<INVOKE_MACRO>` element). Constraint macros can also be recursive. As an example, constraint macro A is recursive if it invokes itself or other constraint macros that invoke constraint macro A. Recursive constraint macros are useful when specifying validation requirements for hierarchical or recursive structured DICOM content, such as the content in a DICOM structured report.

Constraint definitions can be separated into multiple constraint document files, with each constraint file defining one or more constraint rules or constraint macros. Global rules and global macros can reference other internal and external global rules and global macros. Internal rules and internal macros are defined within the same constraint file. External rules and external macros are imported from other constraint document files that define those rules and macros. Before referencing a set of external constraint rules or external constraint macros in your constraint file, you must specify those rules or macros in your file (using the <EXTERNAL_RULE_INCLUDE> element or the <EXTERNAL_MACRO_INCLUDE> element, respectively). In addition, DICOM administrators must insert the *referenced* constraint document files into the repository before inserting the *referencing* constraint files.

In the XML constraint schema `ordcmct.xsd`, <ACTION> elements are defined to associate conformance validation messages with a predicate, a constraint rule, or a constraint macro. If predicates are evaluated to the conditions specified in the <ACTION> elements associated with the predicates, you can see these messages after conformance validation by querying the view `orddcm_conformance_vld_msgs`. (See [orddcm_conformance_vld_msgs](#) for reference information.)

The following subsections contain examples that show how to create a variety of constraint documents:

- [Defining a Simple Constraint Rule](#)
- [Defining Constraint Rules by Importing Other Constraint Rules](#)
- [Defining and Referencing Constraint Macros](#)
- [Defining Recursive Constraint Macros](#)
- [Using DICOM Value Locators in Constraint Documents](#)

Note: These subsections contain references to tables in the DICOM standard. For more information about these tables, see the DICOM standard, which is available on the Web site for the National Electrical Manufacturers Association (NEMA) at

<http://medical.nema.org/>

10.2.2.1 Defining a Simple Constraint Rule

This subsection shows how to construct a simple constraint rule that checks two conditions required by the SOP Common Module, which is defined in the DICOM standard, in PS 3.3, Table C.12-1.

The following table shows the SOP Class UID and SOP Instance UID conditions as they are defined in the SOP Common Module of the DICOM standard.

Attribute Name	Tag	Type	Attribute Description
SOP Class UID	(0008,0016)	1	Uniquely identifies the SOP Class. See C.12.1.1.1 for further explanation. See also PS 3.4.
SOP Instance UID	(0008,0018)	1	Uniquely identifies the SOP Instance. See C.12.1.1.1 for further explanation. See also PS 3.4.

The two entries in the preceding table indicate that the attributes SOP Class UID (0008,0016) and SOP Instance UID (0008,0018) must exist and cannot be empty.

The code segment in [Example 10–10](#) shows a predicate that checks whether the attribute SOP Class UID (0008,0016) is not empty:

Example 10–10 Predicate for One Condition on SOP Class UID

```
<PREDICATE>
  <BOOLEAN_FUNC operator="notEmpty">
    <ATTRIBUTE_TAG>00080016</ATTRIBUTE_TAG>
  </BOOLEAN_FUNC>
</PREDICATE>
```

The code segment in [Example 10–11](#) shows a predicate that checks whether the attribute SOP Instance UID (0008,0018) is not empty:

Example 10–11 Predicate for One Condition on SOP Instance UID

```
<PREDICATE>
  <BOOLEAN_FUNC operator="notEmpty">
    <ATTRIBUTE_TAG>00080018</ATTRIBUTE_TAG>
  </BOOLEAN_FUNC>
</PREDICATE>
```

Checking whether both of these attributes are not empty is equivalent to doing a logical AND operation for the two predicates shown in [Example 10–10](#) and [Example 10–11](#). The prescribed way to undertake this operation is to define another predicate that performs a logical AND operation on the preceding two predicates. [Example 10–12](#) shows a predicate with a logical AND operation for these two conditions:

Example 10–12 Predicate for Two Conditions

```
<PREDICATE>
  <LOGICAL operator="and">
    <PREDICATE>
      <BOOLEAN_FUNC operator="notEmpty">
        <ATTRIBUTE_TAG>00080016</ATTRIBUTE_TAG>
      </BOOLEAN_FUNC>
    </PREDICATE>
    <PREDICATE>
      <BOOLEAN_FUNC operator="notEmpty">
        <ATTRIBUTE_TAG>00080018</ATTRIBUTE_TAG>
      </BOOLEAN_FUNC>
    </PREDICATE>
  </LOGICAL>
</PREDICATE>
```

A simpler way to define predicates having logical AND relations is by omitting the outer predicate that was shown in [Example 10–12](#). Thus, taking the complete constraint rule from the constraint document `ordcmcmd.xml`, the global rule `SOPCommonModule` can be defined as shown in [Example 10–13](#):

Example 10–13 Global Rule for Two Boolean Functions

```
<GLOBAL_RULE name="SOPCommonModule">
  <DESCRIPTION>
    A subset of SOP Common Module defined in DICOM standard,
```

```

    PS 3.3-2007, Table C.12-1
</DESCRIPTION>
<PREDICATE>
  <BOOLEAN_FUNC operator="notEmpty">
    <ATTRIBUTE_TAG>00080016</ATTRIBUTE_TAG>
  </BOOLEAN_FUNC>
</PREDICATE>
<PREDICATE>
  <BOOLEAN_FUNC operator="notEmpty">
    <ATTRIBUTE_TAG>00080018</ATTRIBUTE_TAG>
  </BOOLEAN_FUNC>
</PREDICATE>
</GLOBAL_RULE>

```

Each global rule must have a unique name. Furthermore, each global rule can include an optional <DESCRIPTION> element to provide descriptive information about that rule.

The code segment in [Example 10-13](#) shows how to define predicates that represent Boolean functions or logical relations. Predicates that represent relational relations can be defined similarly.

[Example 10-14](#) is a partial code segment that shows two sample predicates within a global rule. These predicates ensure that no error is raised when the DICOM content that is being validated does not contain attributes in the global rule. [Example 3-1](#) shows the complete code segment for this global rule (for the Patient Module).

Example 10-14 Global Rule for Two Conditions on Patient's Sex

```

<GLOBAL_RULE name="PatientModule">
.
.
.
  <PREDICATE>
    <BOOLEAN_FUNC operator="notEmpty">
      <ATTRIBUTE_TAG>00100040</ATTRIBUTE_TAG>
    </BOOLEAN_FUNC>
  </PREDICATE>
  <PREDICATE>
    <DESCRIPTION>Patient's Sex</DESCRIPTION>
    <RELATIONAL operator="in">
      <ATTRIBUTE_TAG>00100040</ATTRIBUTE_TAG>
      <STRING_VALUE>M</STRING_VALUE>
      <STRING_VALUE>F</STRING_VALUE>
      <STRING_VALUE>O</STRING_VALUE>
    </RELATIONAL>
  </PREDICATE>
.
.
.
</GLOBAL_RULE>

```

The first predicate in [Example 10-14](#) tests the condition that the attribute Patient's Sex (0010, 0040) is not empty. This predicate ensures that all subsequent predicates that reference the attribute Patient's Sex (0010, 0040) are tested only when this predicate returns true during conformance validation. Thus, the second predicate is tested only when the DICOM content that is being validated contains the attribute Patient's Sex (0010, 0040) *and* the value is not empty. Therefore, no error is raised if the DICOM content does not contain the attribute Patient's Sex (0010, 0040).

See [Section 10.2.2.4](#) for more examples of predicates that represent relational relations.

10.2.2.2 Defining Constraint Rules by Importing Other Constraint Rules

This subsection shows how to construct constraint rules hierarchically by referencing other external constraint rules. This section also shows how to reference external constraint rules.

The constraint document `ordcmct.xml` defines the global rule `OracleOrdObject`. This constraint rule is defined as the logical AND relation of three constraint rules: `SOPCommonModule`, `GeneralSeriesModule`, and `GeneralStudyModule`. [Example 10–15](#) shows a global rule with a logical AND relation for these three constraint rules:

Example 10–15 Global Rule for a Logical Relation

```
<GLOBAL_RULE name="OracleOrdObject">
  <PREDICATE>
    <GLOBAL_RULE_REF>SOPCommonModule</GLOBAL_RULE_REF>
  </PREDICATE>
  <PREDICATE>
    <GLOBAL_RULE_REF>GeneralSeriesModule</GLOBAL_RULE_REF>
  </PREDICATE>
  <PREDICATE>
    <GLOBAL_RULE_REF>GeneralStudyModule</GLOBAL_RULE_REF>
  </PREDICATE>
</GLOBAL_RULE>
```

These three constraint rules are defined in the constraint document `ordcmcmd.xml` and imported into the DICOM constraint document `ordcmct.xml`. [Example 10–16](#) shows an external rule for these constraint rules:

Example 10–16 External Rule for Three Constraint Rules

```
<EXTERNAL_RULE_INCLUDE name="GeneralStudyModule">
  A subset of General Study Module defined in DICOM standard,
  PS 3.3-2007, Table C.7-3
</EXTERNAL_RULE_INCLUDE>
<EXTERNAL_RULE_INCLUDE name="GeneralSeriesModule">
  A subset of General Series Module defined in DICOM standard,
  PS 3.3-2007, Table C.7-5a
</EXTERNAL_RULE_INCLUDE>
<EXTERNAL_RULE_INCLUDE name="SOPCommonModule">
  A subset of SOP Common Module defined in DICOM standard,
  PS 3.3-2007, Table C.12-1
</EXTERNAL_RULE_INCLUDE>
```

Thus, the global rule `OracleOrdObject` references the global rule `SOPCommonModule` that is defined in [Section 10.2.2.1](#). Other constraint rules can also reference the global rule `SOPCommonModule`. In this way, constraint documents can be written in a modular and structured fashion.

10.2.2.3 Defining and Referencing Constraint Macros

This subsection shows how to construct a constraint macro that checks whether a DICOM attribute is a code sequence attribute. The constraint macro performs this checking operation by following the first two conditions required by the Code Sequence Macro, which is defined in the DICOM standard, in PS 3.3, Table 8.8-1.

The following table shows the Code Value and Coding Scheme Designator conditions as they are defined in the Code Sequence Macro of the DICOM standard.

Attribute Name	Tag	Type	Attribute Description
Code Value	(0008,0100)	1C	See Section 8.1. Required if a sequence item is present.
Coding Scheme Designator	(0008,0102)	1C	See Section 8.2. Required if a sequence item is present.

The two entries in the preceding table indicate that the mandatory child attributes Code Value (0008,0100) and Coding Scheme Designator (0008,0102) must not be empty.

Example 10-17 shows a global macro definition (from the default constraint document `ordcmcmc.xml`) that checks whether the attributes Code Value (0008,0100) and Coding Scheme Designator (0008,0102) are not empty:

Example 10-17 Global Macro for a Condition

```
<GLOBAL_MACRO name="CodeSequenceMacro">
  <DESCRIPTION>
    A subset of Code Sequence Macro defined in DICOM standard,
    PS 3.3-2007, Table 8.8-1
  </DESCRIPTION>
  <PARAMETER_DECLARATION>CodeAttr</PARAMETER_DECLARATION>
  <PREDICATE>
    <DESCRIPTION>Code value must not be empty</DESCRIPTION>
    <BOOLEAN_FUNC operator="notEmpty">
      <ATTRIBUTE_TAG>${CodeAttr}.00080100</ATTRIBUTE_TAG>
    </BOOLEAN_FUNC>
  </PREDICATE>
  <PREDICATE>
    <DESCRIPTION>Coding scheme designator must not be empty</DESCRIPTION>
    <BOOLEAN_FUNC operator="notEmpty">
      <ATTRIBUTE_TAG>${CodeAttr}.00080102</ATTRIBUTE_TAG>
    </BOOLEAN_FUNC>
  </PREDICATE>
</GLOBAL_MACRO>
```

Predicates that are included in constraint macros include operands that contain parameters. These parameters must be defined in `<PARAMETER_DECLARATION>` elements. When the parameters are referenced in these operands, the parameters must be preceded by the dollar sign symbol and enclosed within braces as in: `${ }`. In the preceding code segment, the parameter `CodeAttr` represents the code sequence to be examined. Thus, checking whether the code value of the parameter `CodeAttr` is not empty is equivalent to checking whether the parameter `${CodeAttr}.00080100` is not empty.

Constraint macros can be invoked by one or more constraint rules, with the macro parameters being set to different values. To invoke a constraint macro, you must specify the name of the macro with the name value pairs of all parameters for that macro.

Example 10-18 shows the definition for the global rule `GeneralSeriesModule` (from the default constraint document `ordcmcmd.xml`), which invokes the constraint macro `CodeSequenceMacro`. The code statements where this macro is invoked are highlighted in bold.

Example 10–18 Global Rule with a Constraint Macro

```

<GLOBAL_RULE name="GeneralSeriesModule">
  <DESCRIPTION>
    A subset of General Series Module defined in DICOM standard,
    PS 3.3-2007, Table C.7-5a
  </DESCRIPTION>
  <PREDICATE>
    <BOOLEAN_FUNC operator="notEmpty">
      <ATTRIBUTE_TAG>00080060</ATTRIBUTE_TAG>
    </BOOLEAN_FUNC>
    <ACTION action="warning" when="false">
      missing attribute 00080060
    </ACTION>
  </PREDICATE>
  <PREDICATE>
    <BOOLEAN_FUNC operator="notEmpty">
      <ATTRIBUTE_TAG>0020000E</ATTRIBUTE_TAG>
    </BOOLEAN_FUNC>
    <ACTION action="warning" when="false">
      missing attribute 0020000E
    </ACTION>
  </PREDICATE>
  <PREDICATE>
    <LOGICAL operator="derive">
      <PREDICATE>
        <BOOLEAN_FUNC operator="notEmpty">
          <ATTRIBUTE_TAG>00400260</ATTRIBUTE_TAG>
        </BOOLEAN_FUNC>
      </PREDICATE>
      <PREDICATE>
        <INVOKE_MACRO>
          <MACRO_NAME>CodeSequenceMacro</MACRO_NAME>
          <PARAMETER>
            <NAME>CodeAttr</NAME>
            <VALUE>00400260</VALUE>
          </PARAMETER>
        </INVOKE_MACRO>
        <ACTION action="warning" when="false">
          missing attribute 00400260.00080100 or 00400260.00080102
        </ACTION>
      </PREDICATE>
    </LOGICAL>
  </PREDICATE>
  <ACTION action="warning" when="false">
    GeneralSeriesModule is not satisfied
  </ACTION>
</GLOBAL_RULE>

```

Because the constraint macro `CodeSequenceMacro` is defined in a different constraint file, it is imported in the beginning of the constraint document `ordcmcmd.xml`, as follows:

```

<EXTERNAL_MACRO_INCLUDE name="CodeSequenceMacro">
  Defines a code sequence macro</EXTERNAL_MACRO_INCLUDE>

```

When users check whether the DICOM content conforms to the constraint rule `GeneralSeriesModule`, the DICOM content is checked against the constraint macro `CodeSequenceMacro`, where the parameter `CodeAttr` is substituted as `00400260`. Specifically, the predicate to check the `<ATTRIBUTE_TAG>` element `${CodeAttr}.00080100 not empty` becomes `00400260.00080100 not`

empty. And, the <ATTRIBUTE_TAG> element `${CodeAttr}.00080102` not empty becomes `00400260.00080102` not empty.

If the constraint macro `CodeSequenceMacro` with the parameter `CodeAttr` substituted as `00400260` evaluates to `false` on the DICOM content, the view `orddcm_conformance_vld_msgs` contains the message "missing attribute `00400260.00080100` or `00400260.00080102`" for the DICOM content. These conformance validation messages can be used to provide information about specific attributes in the DICOM content that do not conform to the constraint rules.

See the [orddcm_conformance_vld_msgs](#) view for more information about these messages.

10.2.2.4 Defining Recursive Constraint Macros

This subsection shows two examples that check a subset of attributes required by the SR Document Content Module, which is defined in the DICOM standard, in PS 3.3, Section C.17.3.

The following table shows the Content Sequence and Relationship Type attributes as they are defined in the Document Relationship Macro Attributes in Table C.17-6. The table also shows the Value Type attribute as it is defined in the Document Content Macro Attributes in Table C.17-5.

Attribute Name	Tag	Type	Attribute Description
Content Sequence	(0040,A730)	1C	<p>A potentially recursively nested Sequence of items that conveys content that is the Target of Relationships with the enclosing Source Content Item.</p> <p>One or more Items may be included in this sequence.</p> <p>Required if the enclosing Content Item has relationships.</p> <p>Notes:</p> <ol style="list-style-type: none"> 1. If this Attribute is not present then the enclosing Item is a leaf. 2. The order of Items within this Sequence is semantically significant for presentation. <p>See C.17.3.2.4 for further explanation.</p>

Attribute Name	Tag	Type	Attribute Description
Relationship Type	(0040,A010)	1	<p>The type of relationship between the (enclosing) Source Content Item and the Target Content Item.</p> <p>IODs specify additional constraints on Relationships (including lists of Enumerated Values).</p> <p>Defined Terms:</p> <p>CONTAINS HAS PROPERTIES HAS OBS CONTEXT HAS ACQ CONTEXT INFERRED FROM SELECTED FROM HAS CONCEPT MOD</p> <p>See C.17.3.2.4 for further explanation.</p>
Value Type	(0040,A040)	1	<p>The type of the value encoded in this Content Item.</p> <p>Defined Terms:</p> <p>TEXT NUM CODE DATETIME DATE TIME UIDREF PNAME COMPOSITE IMAGE WAVEFORM SCoord TCOORD CONTAINER</p> <p>See C.17.3.2.1 for further explanation.</p>

For example, the attribute `Content Sequence (0040,A730)` provides the hierarchical structuring of the Content Tree (DICOM standard, PS 3.3, Section C.17.3.1) by recursively nesting Content Items. A parent (or source) Content Item has an explicit relationship to each child (or target) Content Item, which is conveyed by the attribute `Relationship Type (0040,A010)` (DICOM standard, PS 3.3, Section C.17.3.2.4).

To avoid infinite loops and control how many levels of the hierarchy structure to evaluate, use the preference parameter `MAX_RECURSION_DEPTH` in the preference document to specify the number of levels of recursion for a constraint macro. See [Section 10.2.6.4](#) for information about defining this preference parameter in a preference document.

The code segment in [Example 10-19](#) is an example of recursion with a single constraint macro that invokes itself. The code segment in [Example 10-20](#) is an example of recursion between two different constraint macros, with the first macro invoking a second macro, which then invokes the first macro.

Note: The examples in this section are not shipped with the installation software.

Recursion with a Single Constraint Macro

Example 10–19 defines the recursive global constraint macro `DocumentContentMacro`. This macro is invoked by the global constraint rule `DocumentContentModule`. The code statements where this macro is defined and invoked are highlighted in bold.

Example 10–19 One Constraint Macro with Recursion

```

<GLOBAL_MACRO name="DocumentContentMacro">
  <PARAMETER_DECLARATION>tag</PARAMETER_DECLARATION>
  <PREDICATE>
    <BOOLEAN_FUNC operator="notEmpty">
      <ATTRIBUTE_TAG>${tag}.0040A010</ATTRIBUTE_TAG>
    </BOOLEAN_FUNC>
  </PREDICATE>
  <PREDICATE>
    <RELATIONAL operator="in">
      <ATTRIBUTE_TAG>${tag}.0040A010</ATTRIBUTE_TAG>
      <STRING_VALUE>CONTAINS</STRING_VALUE>
      <STRING_VALUE>HAS PROPERTIES</STRING_VALUE>
      <STRING_VALUE>HAS OBS CONTEXT</STRING_VALUE>
      <STRING_VALUE>HAS ACQ CONTEXT</STRING_VALUE>
      <STRING_VALUE>INFERRED FROM</STRING_VALUE>
      <STRING_VALUE>SELECTED FROM</STRING_VALUE>
      <STRING_VALUE>HAS CONCEPT MOD</STRING_VALUE>
    </RELATIONAL>
  </PREDICATE>
  <PREDICATE>
    <BOOLEAN_FUNC operator="notEmpty">
      <ATTRIBUTE_TAG>${tag}.0040A040</ATTRIBUTE_TAG>
    </BOOLEAN_FUNC>
    <ACTION action="log" when="true">${RUNTIME_TAG} is not empty</ACTION>
  </PREDICATE>
  <PREDICATE>
    <LOGICAL operator="derive">
      <PREDICATE>
        <RELATIONAL operator="eq">
          <ATTRIBUTE_TAG>${tag}.0040A040</ATTRIBUTE_TAG>
          <STRING_VALUE>CONTAINER</STRING_VALUE>
        </RELATIONAL>
        <ACTION action="log" when="false">${RUNTIME_TAG} value is not
          CONTAINER</ACTION>
      </PREDICATE>
    </PREDICATE>
    <INVOKE_MACRO>
      <MACRO_NAME>DocumentContentMacro</MACRO_NAME>
      <PARAMETER>
        <NAME>tag</NAME>
        <VALUE>${tag}.0040A730[*]</VALUE>
      </PARAMETER>
    </INVOKE_MACRO>
  </PREDICATE>
</LOGICAL>
</PREDICATE>
</GLOBAL_MACRO>

```

```

<GLOBAL_RULE name="DocumentContentModule">
  <PREDICATE>
    <INVOKE_MACRO>
      <MACRO_NAME>DocumentContentMacro</MACRO_NAME>
    <PARAMETER>
      <NAME>tag</NAME>
      <VALUE>0040A730[*]</VALUE>
    </PARAMETER>
  </INVOKE_MACRO>
</PREDICATE>
</GLOBAL_RULE>

```

During conformance validation, the $\${RUNTIME_TAG}$ tags in **<ACTION>** elements are replaced by the DICOM value locators whose corresponding attributes in the DICOM content are evaluated to meet the condition specified by the *when* attribute of the **<ACTION>** element. The first action creates a message that contains a list of DICOM value locators, all ending with tag 0040A040, whose corresponding attributes in the DICOM content are not empty. The second action creates a message that contains a list of DICOM value locators, all ending with tag 0040A040, whose corresponding attributes in the DICOM content do not match CONTAINER.

In [Example 10–19](#), the recursion occurs when the recursive constraint macro invokes itself by specifying the name value pair of the parameter defined for that macro.

In the definition of the macro `DocumentContentMacro`, the first predicate checks whether the attribute $\${tag}.0040A010$ is not empty. The second predicate uses a relational operation to check whether this attribute has a valid value. The third predicate checks whether the attribute $\${tag}.0040A040$ is not empty. The fourth predicate checks whether the value of $\${tag}.0040A040$ is CONTAINER. If so, then the macro `DocumentContentMacro` is invoked on the content sequence of $\${tag}.0040A730[*]$. The code `[*]` is a wildcard character that represents the numbered series of attributes in the constraint document to be checked by the macro.

Recursion with Two Constraint Macros

[Example 10–20](#) defines two recursive global constraint macros:

`DocumentContentMacroA` and `DocumentContentMacroB`. The code statements where these macros are defined and invoked are highlighted in bold.

Example 10–20 Two Constraint Macros with Recursion

```

<GLOBAL_MACRO name="DocumentContentMacroA">
  <PARAMETER_DECLARATION>attr</PARAMETER_DECLARATION>
  <PREDICATE>
    <BOOLEAN_FUNC operator="notEmpty">
      <ATTRIBUTE_TAG>${attr}.0040A040</ATTRIBUTE_TAG>
    </BOOLEAN_FUNC>
  </PREDICATE>
  <PREDICATE>
    <LOGICAL operator="derive">
      <PREDICATE>
        <RELATIONAL operator="eq">
          <ATTRIBUTE_TAG>${attr}.0040A040</ATTRIBUTE_TAG>
          <STRING_VALUE>CONTAINER</STRING_VALUE>
        </RELATIONAL>
      </PREDICATE>
    </PREDICATE>
  </PREDICATE>
  <INVOKE_MACRO>
    <MACRO_NAME>DocumentContentMacroB</MACRO_NAME>
  <PARAMETER>

```

```

        <NAME>tag</NAME>
        <VALUE>${attr}.0040A730[*]</VALUE>
    </PARAMETER>
</INVOKE_MACRO>
</PREDICATE>
</LOGICAL>
</PREDICATE>
</GLOBAL_MACRO>

<GLOBAL_MACRO name="DocumentContentMacroB">
<PARAMETER_DECLARATION>tag</PARAMETER_DECLARATION>
<PREDICATE>
    <BOOLEAN_FUNC operator="notEmpty">
        <ATTRIBUTE_TAG>${tag}.0040A010</ATTRIBUTE_TAG>
    </BOOLEAN_FUNC>
</PREDICATE>
<PREDICATE>
    <RELATIONAL operator="in">
        <ATTRIBUTE_TAG>${tag}.0040A010</ATTRIBUTE_TAG>
        <STRING_VALUE>CONTAINS</STRING_VALUE>
        <STRING_VALUE>HAS PROPERTIES</STRING_VALUE>
        <STRING_VALUE>HAS OBS CONTEXT</STRING_VALUE>
        <STRING_VALUE>HAS ACQ CONTEXT</STRING_VALUE>
        <STRING_VALUE>INFERRED FROM</STRING_VALUE>
        <STRING_VALUE>SELECTED FROM</STRING_VALUE>
        <STRING_VALUE>HAS CONCEPT MOD</STRING_VALUE>
    </RELATIONAL>
</PREDICATE>
<PREDICATE>
    <INVOKE_MACRO>
        <MACRO_NAME>DocumentContentMacroA</MACRO_NAME>
        <PARAMETER>
            <NAME>attr</NAME>
            <VALUE>${tag}</VALUE>
        </PARAMETER>
    </INVOKE_MACRO>
</PREDICATE>
</GLOBAL_MACRO>

```

In [Example 10–20](#), the recursion occurs when `DocumentContentMacroB` invokes `DocumentContentMacroA`, and then `DocumentContentMacroA` invokes `DocumentContentMacroB`.

10.2.2.5 Using DICOM Value Locators in Constraint Documents

This subsection shows how to use DICOM value locators to specify attributes in constraint documents. DICOM value locators can be used to specify predicates that include private attributes and their definers. DICOM value locators can contain macros for substitution when the constraint document is loaded into the repository.

The following list shows a few examples of valid macro substitution strings:

- A complete series of sublocators
- A tag, including the special wildcard tag ". "
- A definer name (DICOM, the default name, or the name of a private definer)
- An item number (a positive integer or the wildcard character '*')

See [General Format for DICOM Value Locators](#) for complete reference information.

[Example 10–21](#) shows how to specify the attribute `Content Sequence` with a DICOM value locator in a global constraint macro and create a log of the results at run time. (This example uses some code from [Example 10–19](#).) The DICOM value locators are highlighted in bold.

Example 10–21 DICOM Value Locator with a Wildcard Character in a Constraint Macro

```
<GLOBAL_MACRO name="DocumentContentMacro">
  <PARAMETER_DECLARATION>tag</PARAMETER_DECLARATION>
  <PREDICATE>
    <BOOLEAN_FUNC operator="notEmpty">
      <ATTRIBUTE_TAG>#{tag}.0040A010</ATTRIBUTE_TAG>
    </BOOLEAN_FUNC>
  </PREDICATE>
  <PREDICATE>
    <RELATIONAL operator="in">
      <ATTRIBUTE_TAG>#{tag}.0040A010</ATTRIBUTE_TAG>
      <STRING_VALUE>CONTAINS</STRING_VALUE>
      <STRING_VALUE>HAS PROPERTIES</STRING_VALUE>
      <STRING_VALUE>HAS OBS CONTEXT</STRING_VALUE>
      <STRING_VALUE>HAS ACQ CONTEXT</STRING_VALUE>
      <STRING_VALUE>INFERRED FROM</STRING_VALUE>
      <STRING_VALUE>SELECTED FROM</STRING_VALUE>
      <STRING_VALUE>HAS CONCEPT MOD</STRING_VALUE>
    </RELATIONAL>
  </PREDICATE>
  <PREDICATE>
    <BOOLEAN_FUNC operator="notEmpty">
      <ATTRIBUTE_TAG>#{tag}.0040A040</ATTRIBUTE_TAG>
    </BOOLEAN_FUNC>
    <ACTION action="log" when="true">#{RUNTIME_TAG} is not empty</ACTION>
  </PREDICATE>
  <PREDICATE>
    <LOGICAL operator="derive">
      <PREDICATE>
        <RELATIONAL operator="eq">
          <ATTRIBUTE_TAG>#{tag}.0040A040</ATTRIBUTE_TAG>
          <STRING_VALUE>CONTAINER</STRING_VALUE>
        </RELATIONAL>
        <ACTION action="log" when="false">#{RUNTIME_TAG} value is not
          CONTAINER</ACTION>
      </PREDICATE>
    </PREDICATE>
    <INVOKE_MACRO>
      <MACRO_NAME>DocumentContentMacro</MACRO_NAME>
      <PARAMETER>
        <NAME>tag</NAME>
        <VALUE>#{tag}.0040A730[*]</VALUE>
      </PARAMETER>
    </INVOKE_MACRO>
  </PREDICATE>
</GLOBAL_MACRO>
<GLOBAL_RULE name="DocumentContentModule">
  <PREDICATE>
    <INVOKE_MACRO>
      <MACRO_NAME>DocumentContentMacro</MACRO_NAME>
      <PARAMETER>
        <NAME>tag</NAME>
```

```

        <VALUE>0040A730 [*]</VALUE>
    </PARAMETER>
</INVOKE_MACRO>
</PREDICATE>
</GLOBAL_RULE>

```

The standard attribute tags 0040A010, 0040A040, and 0040A730 are defined as follows in the standard dictionary:

```

<STANDARD_ATTRIBUTE_DEFINITION>
  <TAG>0040A010</TAG>
  <NAME>Relationship Type</NAME>
  <VR>CS</VR>
  <VM>1</VM>
</STANDARD_ATTRIBUTE_DEFINITION>
<STANDARD_ATTRIBUTE_DEFINITION>
  <TAG>0040A040</TAG>
  <NAME>Value Type</NAME>
  <VR>CS</VR>
  <VM>1</VM>
</STANDARD_ATTRIBUTE_DEFINITION>
<STANDARD_ATTRIBUTE_DEFINITION>
  <TAG>0040A730</TAG>
  <NAME>Content Sequence</NAME>
  <VR>SQ</VR>
  <VM>1</VM>
</STANDARD_ATTRIBUTE_DEFINITION>

```

In the tag definition shown in the preceding code segment, the <VR> element of tag 0040A730 indicates that this tag is a sequence tag and the corresponding attribute in the DICOM content can contain multiple sequence items. Using a wildcard character in the DICOM value locator ensures that all the items for this attribute are checked.

[Example 10–22](#) shows how to specify all the Relationship Type attributes with a DICOM value locator in a global rule. The DICOM value locator is highlighted in bold.

Example 10–22 DICOM Value Locator with a Special Wildcard Tag in a Global Rule

```

<GLOBAL_RULE name="MagicTagTest">
  <PREDICATE>
    <RELATIONAL operator="in">
      <ATTRIBUTE_TAG>..0040A010</ATTRIBUTE_TAG>
      <STRING_VALUE>CONTAINS</STRING_VALUE>
      <STRING_VALUE>HAS PROPERTIES</STRING_VALUE>
      <STRING_VALUE>HAS OBS CONTEXT</STRING_VALUE>
      <STRING_VALUE>HAS ACQ CONTEXT</STRING_VALUE>
      <STRING_VALUE>INFERRED FROM</STRING_VALUE>
      <STRING_VALUE>HAS CONCEPT MOD</STRING_VALUE>
    </RELATIONAL>
    <ACTION action="log" when="false">${RUNTIME_TAG} does not contain required
      values</ACTION>
  </PREDICATE>
</GLOBAL_RULE>

```

In the preceding code segment, the DICOM value locator for all 0040A010 attributes includes the special wildcard tag "." to indicate that the predicate performs a relational operation to check all the Relational Type attributes in the DICOM content.

Note: The following restrictions apply to the use of the special wildcard tag ".*" in a DICOM value locator:

- It can be used by one predicate operand only.
 - It cannot be used within a recursive constraint macro, or to call a recursive constraint macro.
-
-

See [General Format for DICOM Value Locators](#) for more information about the DICOM value locator type.

10.2.3 Creating Mapping Documents and Metadata XML Schemas

Mapping documents define how the DICOM attributes are mapped into an XML document. The metadata XML document can be constrained by an XML schema, or it can be a well-formed XML document with no XML schema constraints.

The XML schema `ordcmmmp.xsd` defines the XML schema that constrains mapping documents. (See [Section B.6](#) for a listing of the mapping document schema `ordcmmmp.xsd`.)

The following subsections describe the structure of mapping documents and metadata XML schemas. The subsections also contain code examples that show how to create mapping documents and corresponding metadata schemas:

- [Structure of a Mapping Document](#)
- [Structure of a Metadata XML Schema](#)
- [Creating a Mapping Document for Metadata with No Schema Constraints](#)
- [Creating a Mapping Document for Metadata with Schema Constraints and a Mapped Section Only](#)
- [Creating a Mapping Document for Metadata with Schema Constraints](#)
- [Using DICOM Value Locators in Mapping Documents](#)

10.2.3.1 Structure of a Mapping Document

Each mapping document must include a root element and a namespace declaration, similar to the following:

```
<XML_MAPPING_DOCUMENT xmlns="http://xmlns.oracle.com/ord/dicom/mapping_1_0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.oracle.com/ord/dicom/mapping_1_0
http://xmlns.oracle.com/ord/dicom/mapping_1_0">
```

The remaining elements in the mapping document, in order of appearance, are as follows:

- `<dt:DOCUMENT_HEADER>`: This element is optional. It is used to keep the update record of the XML document. If this element is specified, add the following namespace declaration to the root element:

```
namespace: xmlns:dt=http://xmlns.oracle.com/ord/dicom/datatype_1_0"
```

See [Section B.3](#) for more information about the data type definition schema `ordcmrdt.xsd`.

- `<NAMESPACE>`: This mandatory element specifies the namespace of the metadata XML document. This namespace must match the target namespace of

the corresponding metadata schema. If this element is empty, the extracted metadata XML is not schema constrained.

- `<ROOT_ELEM_TAG>`: This mandatory element specifies the root element name of the metadata XML document.
- `<UNMAPPED_ELEM>`: This element specifies the XML path to the element that is the parent element of all unmapped attributes. This XML path is relative to the root element (specified by the element `<ROOT_ELEM_TAG>`) of the metadata XML. If this element is omitted or empty, the parent element of the unmapped attributes is the root element of the metadata XML document. See [Section 2.5](#) for more information about unmapped attributes.
- `<MAPPED_ELEM>`: This element specifies the XML path to the element that is the parent element of all mapped attributes. This XML path is relative to the root element of the metadata XML (specified by the element `<ROOT_ELEM_TAG>`). If this element is omitted or empty, the parent element of the mapped attributes is the root element of the metadata XML document. See [Section 2.5](#) for more information about mapped attributes.
- `<MAPPED_PATH>`: This element specifies the XML path to a mapped attribute. This XML path is relative to the parent element of the mapped attributes as defined by the element `<MAPPED_ELEM>`. This element can be used multiple times within a mapping document to specify the XML paths to all mapped attributes. The order of appearance of each `<MAPPED_PATH>` element reflects the order of appearance of the mapped attributes in a metadata XML document. Within the `<MAPPED_PATH>` element, the `<ATTRIBUTE_TAG>` and `<PATH>` elements are used to define each mapped attribute tag and its XML path, respectively. See [Section B.6](#) for more information about the mapping document schema `ordcmmp.xsd`.

10.2.3.2 Structure of a Metadata XML Schema

During the process of extracting DICOM metadata and encoding it into an XML document, a mapping document is used to encode the DICOM attributes into an XML document, and an XML schema is used to validate the encoded metadata XML document. To extract DICOM metadata into a schema-valid XML document, the mapping document and the XML schema definition for the metadata XML document must be synchronized.

The general rules for creating an XML schema for a metadata XML document that is specific to a particular application are as follows:

- The order of the elements defined in the XML schema must match the order of the mapped XML paths for those elements.
- The XML schema must use data types that are either compatible with or less restrictive than the data types defined by Oracle in the schema `ordcmddt.xsd`.
- To extract unmapped DICOM attributes into the XML document, the parent element of the unmapped attributes must be defined as type `DATASET_T` (as in the schema `ordcmddt.xsd`).
- If the attributes `writeTag`, `writeName`, `writeDefiner`, and `writeRawValue` of the `<MAPPED_PATH>` element are set to `"true"` in the mapping document, the XML schema must define the corresponding attributes `tag`, `name`, `definer`, and `rawValue` for each element that is described by the `<MAPPED_PATH>` element in the mapping document.

- If the attribute `occurs` of the `<MAPPED_PATH>` element in the mapping document is set to `"false"` either implicitly or explicitly, the XML schema definition of the element must set the attribute `minOccurs` to `"0"`.
- If the attribute `notEmpty` of the `<MAPPED_PATH>` element in the mapping document is set to `"false"`, the XML schema definition for the element must set the attribute `xsi:nillable` to `"true"`. Otherwise, the element defined by the `<MAPPED_PATH>` element must permit an empty value.

See [Section B.7](#) for more information about the data type definition schema `ordcmddt.xsd`.

10.2.3.3 Creating a Mapping Document for Metadata with No Schema Constraints

This subsection describes how to create a mapping document and a well-formed metadata XML document with no XML schema constraints. For applications that require only a well-formed metadata XML document, the mapping document can contain an empty `<NAMESPACE>` element because the extracted metadata need not conform to any XML schema.

This subsection also includes a code example of a mapping document, followed by a code example of the resulting metadata XML document. Appropriate XML statements are highlighted in bold to show where the main actions occur in both the mapping document and the related XML document. Descriptions of each bolded XML statement follow the examples.

[Example 10–23](#) shows a sample mapping document with an empty `<NAMESPACE>` element. This example also shows the values to be specified for the `occurs`, `notEmpty`, `writeTag`, `writeDefiner`, and `writeName` attributes in the `<MAPPED_PATH>` element to define the extracted metadata XML document. If these attributes are not specified, they are set to `"false"`, the default value.

Example 10–23 Sample Mapping Document for Metadata with No Schema Constraints

```
<?xml version="1.0" encoding="UTF-8"?>
<XML_MAPPING_DOCUMENT
  xmlns="http://xmlns.oracle.com/ord/dicom/mapping_1_0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.oracle.com/ord/dicom/mapping_1_0
    http://xmlns.oracle.com/ord/dicom/mapping_1_0">
  <NAMESPACE></NAMESPACE>
  <ROOT_ELEM_TAG>DICOM_OBJECT</ROOT_ELEM_TAG>
  <UNMAPPED_ELEM>OTHER_ATTRIBUTES</UNMAPPED_ELEM>
  <MAPPED_ELEM>KEY_ATTRIBUTES</MAPPED_ELEM>

  <MAPPED_PATH occurs="true" notEmpty="true">
    <ATTRIBUTE_TAG>00100010</ATTRIBUTE_TAG>
    <PATH>PATIENT/NAME</PATH>
  </MAPPED_PATH>

  <MAPPED_PATH occurs="true" notEmpty="true">
    <ATTRIBUTE_TAG>00100020</ATTRIBUTE_TAG>
    <PATH>PATIENT/ID</PATH>
  </MAPPED_PATH>

  <MAPPED_PATH occurs="true" notEmpty="false">
    <ATTRIBUTE_TAG>00100030</ATTRIBUTE_TAG>
    <PATH>PATIENT/BIRTH_DATE</PATH>
  </MAPPED_PATH>
```

```

<MAPPED_PATH occurs="false" notEmpty="false">
  <ATTRIBUTE_TAG>00100040</ATTRIBUTE_TAG>
  <PATH>PATIENT/SEX</PATH>
</MAPPED_PATH>

<MAPPED_PATH writeTag="true" writeDefiner="true" writeName="true">
  <ATTRIBUTE_TAG>00200010</ATTRIBUTE_TAG>
  <PATH>STUDY/ID</PATH>
</MAPPED_PATH>

<MAPPED_PATH>
  <ATTRIBUTE_TAG>00080030</ATTRIBUTE_TAG>
  <PATH>STUDY/TIME</PATH>
</MAPPED_PATH>

<MAPPED_PATH>
  <ATTRIBUTE_TAG>00081080</ATTRIBUTE_TAG>
  <PATH>STUDY/ADMITTING_DIAGNOSES_DESCRIPTION</PATH>
</MAPPED_PATH>

</XML_MAPPING_DOCUMENT>

```

Example 10–24 shows the resulting metadata XML document whose XML metadata was extracted with the value of the `extractOption` parameter in the `extractMetadata()` method set to `MAPPED`.

Example 10–24 Resulting XML Document for Example 10–23

```

<?xml version="1.0" encoding="DEC-MCS"?>
<DICOM_OBJECT xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <KEY_ATTRIBUTES>
    <PATIENT>
      <NAME>
        <NAME type="unibyte">
          <FAMILY>CANCIO 2HR A-02-013</FAMILY>
        </NAME>
        <VALUE>CANCIO 2HR A-02-013</VALUE>
      </NAME>
      <ID>ISR SCT610b</ID>
      <BIRTH_DATE xsi:nil="true"/>
      <SEX xsi:nil="true"/>
    </PATIENT>
    <STUDY>
      <ID definer="DICOM" tag="00200010" name="Study ID">352</ID>
      <TIME>18:48:41.000000</TIME>
    </STUDY>
  </KEY_ATTRIBUTES>
</DICOM_OBJECT>

```

In Example 10–23 and Example 10–24, these actions occur:

- In the mapping document, the `<NAMESPACE>` element is empty. As a result, the resulting extracted XML document does not include the default namespace declaration because it is not constrained by an XML schema.
- The `<ROOT_ELEM_TAG>` element in the mapping document matches the root element tag (shown in the `<DICOM_OBJECT>` element) in the extracted XML document.
- In the mapping document, the value of the `notEmpty` attribute of the `<MAPPED_PATH>` element for the DICOM attribute with tag 00100040 is "false". In the

extracted XML document, the value of the `xsi:nil` attribute in the `<BIRTH_DATE>` element is set to `"true"` because this DICOM attribute is empty in the DICOM content.

- In the mapping document, the `<MAPPED_PATH>` element for the DICOM attribute with tag `00200010`, the `writeTag`, `writeName`, and `writeDefiner` attributes are set to `"true"`. In the extracted XML document, the child element `<ID>` under the `<STUDY>` element has corresponding `definer`, `tag`, and `name` attributes.
- In the mapping document, the `<MAPPED_PATH>` element for the DICOM attribute with tag `00081080` uses the default value (`"false"`) for the `occurs` attribute. Because this DICOM attribute does not exist in the DICOM content, the element `<ADMITTING_DIAGNOSES_DESCRIPTION>` does not exist in the extracted XML document.

See [extractMetadata\(\)](#) for reference information about this method and the supported values for the `extractOption` parameter.

10.2.3.4 Creating a Mapping Document for Metadata with Schema Constraints and a Mapped Section Only

This subsection describes how to create a mapping document and a metadata XML document with XML schema constraints, discarding the unmapped section. This example can be used in applications that require only the DICOM attributes defined in the mapped section of the XML metadata, and in metadata XML documents with XML schema constraints.

This subsection also includes code examples of a mapping document, an XML schema, and the resulting metadata XML document constrained by the schema. Appropriate XML statements are highlighted in bold to show where the main actions occur in both the mapping document and the related XML document. Descriptions of each bolded XML statement follow the examples.

The `extractMetadata()` method of the `ORDDicom` object can extract all or part of the metadata into an XML document based on the value of the `extractOption` parameter.

See [extractMetadata\(\)](#) for reference information about this method and the supported values for the `extractOption` parameter.

[Example 10–25](#) shows a sample mapping document that specifies the namespace of the metadata XML document, so that the metadata document can be constrained by an XML schema.

Example 10–25 Sample Mapping Document for Metadata with Schema Constraints and a Mapped Section Only

```
<?xml version="1.0" encoding="UTF-8"?>
<XML_MAPPING_DOCUMENT
  xmlns="http://xmlns.oracle.com/ord/dicom/mapping_1_0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.oracle.com/ord/dicom/mapping_1_0
    http://xmlns.oracle.com/ord/dicom/mapping_1_0">
  <NAMESPACE>http://www.mycompany.com/dicom/example2</NAMESPACE>
  <ROOT_ELEM_TAG>DICOM_OBJECT</ROOT_ELEM_TAG>
  <UNMAPPED_ELEM></UNMAPPED_ELEM>
  <MAPPED_ELEM></MAPPED_ELEM>

  <MAPPED_PATH occurs="true" notEmpty="true">
    <ATTRIBUTE_TAG>00100010</ATTRIBUTE_TAG>
    <PATH>PATIENT/NAME</PATH>
```

```

</MAPPED_PATH>

<MAPPED_PATH occurs="true" notEmpty="true">
  <ATTRIBUTE_TAG>00100020</ATTRIBUTE_TAG>
  <PATH>PATIENT/ID</PATH>
</MAPPED_PATH>
<MAPPED_PATH occurs="true" notEmpty="false">
  <ATTRIBUTE_TAG>00100030</ATTRIBUTE_TAG>
  <PATH>PATIENT/BIRTH_DATE</PATH>
</MAPPED_PATH>

<MAPPED_PATH occurs="false" notEmpty="false">
  <ATTRIBUTE_TAG>00100040</ATTRIBUTE_TAG>
  <PATH>PATIENT/SEX</PATH>
</MAPPED_PATH>

<MAPPED_PATH writeTag="true" writeDefiner="true" writeName="true">
  <ATTRIBUTE_TAG>00200010</ATTRIBUTE_TAG>
  <PATH>STUDY/ID</PATH>
</MAPPED_PATH>

<MAPPED_PATH>
  <ATTRIBUTE_TAG>00080030</ATTRIBUTE_TAG>
  <PATH>STUDY/TIME</PATH>
</MAPPED_PATH>

<MAPPED_PATH>
  <ATTRIBUTE_TAG>00081080</ATTRIBUTE_TAG>
  <PATH>STUDY/ADMITTING_DIAGNOSES_DESCRIPTION</PATH>
</MAPPED_PATH>

</XML_MAPPING_DOCUMENT>

```

The mapping document in [Example 10–25](#) differs from the mapping document in [Example 10–23](#) as follows:

- The <NAMESPACE> element contains a value.
- The <UNMAPPED_ELEM> and the <MAPPED_ELEM> elements are empty, thus the mapped path is relative to the <ROOT_ELEM_TAG> element.

The XML schema for the mapping document in [Example 10–25](#) is defined in [Example 10–26](#):

Example 10–26 Sample XML Schema for Example 10–25

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  xmlns="http://www.mycompany.com/dicom/example2"
  xmlns:xdb="http://xmlns.oracle.com/xdb"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.mycompany.com/dicom/example2"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xs:element name="DICOM_OBJECT">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="PATIENT">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="NAME" type="PERSON_NAME"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

```

        <xs:element name="ID" type="xs:string"/>
        <xs:element name="BIRTH_DATE" type="xs:date" nillable="true"/>
        <xs:element name="SEX" type="xs:string" minOccurs="0"
            nillable="true"/>
    </xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="STUDY">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="ID" type="ID_TYPE" minOccurs="0"
                nillable="true"/>
            <xs:element name="TIME" type="xs:time" minOccurs="0"
                nillable="true"/>
            <xs:element name="ADMITTING_DIAGNOSES_DESCRIPTION" type="xs:string"
                minOccurs="0" nillable="true"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:complexType name="PERSON_NAME">
    <xs:sequence>
        <xs:element name="NAME">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="FAMILY" type="xs:string" minOccurs="0"
                        nillable="true"/>
                    <xs:element name="GIVEN" type="xs:string" minOccurs="0"
                        nillable="true"/>
                    <xs:element name="MIDDLE" type="xs:string" minOccurs="0"
                        nillable="true"/>
                    <xs:element name="PREFIX" type="xs:string" minOccurs="0"
                        nillable="true"/>
                    <xs:element name="SUFFIX" type="xs:string" minOccurs="0"
                        nillable="true"/>
                </xs:sequence>
                <xs:attribute name="type" default="unibyte">
                    <xs:simpleType>
                        <xs:restriction base="xs:token">
                            <xs:enumeration value="unibyte"/>
                            <xs:enumeration value="ideographic"/>
                            <xs:enumeration value="phonetic"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:attribute>
            </xs:complexType>
        </xs:element>
        <xs:element name="VALUE" minOccurs="0" nillable="true">
            <xs:simpleType>
                <xs:restriction base="xs:token">
                    <xs:maxLength value="64"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="ID_TYPE">
    <xs:simpleContent>

```

```

    <xs:extension base="xs:string">
      <xs:attribute name="tag" type="xs:string"/>
      <xs:attribute name="definer" type="xs:string"/>
      <xs:attribute name="name" type="xs:string"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
</xs:schema>

```

In [Example 10–25](#) and the associated XML schema (in [Example 10–26](#)), the following elements and data types are defined:

- The namespace declaration of the metadata XML schema has the same value as the `<NAMESPACE>` element in the mapping document.
- For the `<MAPPED_PATH>` element in the mapping document that has the value of the `notEmpty` attribute set to `"false"` either explicitly, as for tag `00100030`, or implicitly using the default value, as for tag `00800030`, the corresponding element defined in the XML schema has the value of the `nullable` attribute set to `"true"`, as in the `<BIRTH_DATE>` element and the `<TIME>` element.
- Any `<MAPPED_PATH>` element in the mapping document that has the value of the `occurs` attribute set to `"false"` either explicitly, as for tag `00100040`, or implicitly using the default value, as for tag `00800030`, the corresponding element defined in the XML schema must have the value of the attribute `minOccurs` set to `"0"`, as in the `<SEX>` element and the `<TIME>` element.
- The data type defined in the XML schema must be compatible with the data type defined by Oracle in the schema `ordcmddt.xsd`. In [Example 2](#), the `"PERSON_NAME"` data type definition is copied from the schema `ordcmddt.xsd`, while the `"ID_TYPE"` data type is defined so that the `<ID>` element under the `<STUDY>` element can have the `tag`, `definer`, and `name` attributes. The `"ID_TYPE"` data type is compatible with the `"SH_ATTR_T"` data type defined by Oracle. The `<SEX>` element is defined to use the `"xs:string"` data type, which is compatible with the Oracle-defined data type `"CS"`.
- The `<UNMAPPED_ELEM>` element is empty in the mapping document, and the root element `<DICOM_OBJECT>` of the metadata XML is not defined as the `"DATASET_T"` type. With this schema constraint, a valid metadata XML document can include a mapped section only. Thus, if the application attempts to extract metadata using a value of `"ALL"` or `"STANDARD"` for the `extractOption` parameter, the attempt returns this error:

```
ORA-53259: cannot extract metadata that conforms to the
schema definition
```

[Example 10–27](#) shows the resulting metadata XML document whose XML metadata was extracted with the value of the `extractOption` parameter in the `extractMetadata()` method set to `MAPPED`.

Example 10–27 Resulting XML Document for [Example 10–25](#)

```

<?xml version="1.0" encoding="DEC-MCS"?>
<DICOM_OBJECT xmlns="http://www.mycompany.com/dicom/example2"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.mycompany.com/dicom/example2
  http://www.mycompany.com/dicom/example2">
  <PATIENT>
    <NAME>
      <NAME type="unibyte">
        <FAMILY>CANCIO 2HR A-02-013</FAMILY>

```

```

    </NAME>
    <VALUE>CANCIO 2HR A-02-013</VALUE>
  </NAME>
  <ID>ISRSCT610b</ID>
  <BIRTH_DATE xsi:nil="true"/>
  <SEX xsi:nil="true"/>
</PATIENT>
<STUDY>
  <ID definer="DICOM" tag="00200010" name="Study ID">352</ID>
  <TIME>18:48:41.000000</TIME>
</STUDY>
</DICOM_OBJECT>

```

10.2.3.5 Creating a Mapping Document for Metadata with Schema Constraints

This subsection describes how to create a mapping document and a metadata XML document with XML schema constraints, including the unmapped section.

This subsection also includes code examples of a mapping document, an XML schema, and the resulting metadata XML document constrained by the schema. Appropriate XML statements are highlighted in bold to show where the main actions occur in both the mapping document and the related XML document. Descriptions of each bolded XML statement follow the examples.

[Example 10–28](#) shows a sample mapping document that includes a sequence type attribute in the mapped section.

Example 10–28 Sample Mapping Document for Metadata with Schema Constraints

```

<?xml version="1.0" encoding="UTF-8"?>
<XML_MAPPING_DOCUMENT
  xmlns="http://xmlns.oracle.com/ord/dicom/mapping_1_0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.oracle.com/ord/dicom/mapping_1_0
    http://xmlns.oracle.com/ord/dicom/mapping_1_0">
  <NAMESPACE>http://www.mycompany.com/dicom/example3</NAMESPACE>
  <ROOT_ELEM_TAG>DICOM_OBJECT</ROOT_ELEM_TAG>
  <UNMAPPED_ELEM>OTHER_ATTRIBUTES</UNMAPPED_ELEM>
  <MAPPED_ELEM>KEY_ATTRIBUTES</MAPPED_ELEM>

  <MAPPED_PATH occurs="true" notEmpty="true">
    <ATTRIBUTE_TAG>00100010</ATTRIBUTE_TAG>
    <PATH>PATIENT/NAME</PATH>
  </MAPPED_PATH>

  <MAPPED_PATH occurs="true" notEmpty="true">
    <ATTRIBUTE_TAG>00100020</ATTRIBUTE_TAG>
    <PATH>PATIENT/ID</PATH>
  </MAPPED_PATH>

  <MAPPED_PATH occurs="true" notEmpty="false">
  <ATTRIBUTE_TAG>00100030</ATTRIBUTE_TAG>
    <PATH>PATIENT/BIRTH_DATE</PATH>
  </MAPPED_PATH>

  <MAPPED_PATH occurs="false" notEmpty="false">
    <ATTRIBUTE_TAG>00100040</ATTRIBUTE_TAG>
    <PATH>PATIENT/SEX</PATH>
  </MAPPED_PATH>

  <MAPPED_PATH writeTag="true" writeDefiner="true" writeName="true">

```

```

    <ATTRIBUTE_TAG>00200010</ATTRIBUTE_TAG>
    <PATH>STUDY/ID</PATH>
  </MAPPED_PATH>

  <MAPPED_PATH>
    <ATTRIBUTE_TAG>00081084</ATTRIBUTE_TAG>
    <PATH>
      STUDY/ADMITTING_DIAGNOSES_CODE_SEQUENCE
    </PATH>
  </MAPPED_PATH>

</XML_MAPPING_DOCUMENT>

```

The mapping document in [Example 10–28](#) differs from the mapping document in [Example 10–25](#) as follows:

- The <UNMAPPED_ELEM> and <MAPPED_ELEM> elements are not empty.
- There is a <MAPPED_PATH> element for tag 00081084, which is of DICOM sequence type.

The XML schema code segment in [Example 10–29](#) shows how to define the unmapped root element for the mapping document in [Example 10–28](#) to include the unmapped section in the extracted metadata XML document. This code segment also shows how to use Oracle-defined data types from the schema `ordcmddt.xsd`.

Example 10–29 Sample XML Schema for Example 10–28

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="http://www.mycompany.com/dicom/example3"
  xmlns:xdb="http://xmlns.oracle.com/xdb"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.mycompany.com/dicom/example3"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xs:include schemaLocation="http://www.mycompany.com/dicom/datatype_3"/>
  <xs:element name="DICOM_OBJECT">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="KEY_ATTRIBUTES">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="PATIENT">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="NAME" type="PN">
                      </xs:element>
                    <xs:element name="ID" type="xs:string">
                      </xs:element>
                    <xs:element name="BIRTH_DATE" type="xs:date" nillable="true">
                      </xs:element>
                    <xs:element name="SEX" type="xs:string" minOccurs="0"
                      nillable="true">
                      </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            <xs:element name="STUDY">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="ID" type="SH_ATTR_T" minOccurs="0"

```



```

        nillable="true" />
        <xs:element name="ADMITTING_DIAGNOSES_CODE_SEQUENCE" type="SQ"
            nillable="true" minOccurs="0" />
    </xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:element>
<xs:element name="OTHER_ATTRIBUTES" type="DATASET_T" minOccurs="0"
    maxOccurs="unbounded">
    </xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>

```

In [Example 10–28](#) and the associated XML schema (in [Example 10–29](#)), the following actions occur:

- Instead of defining its own data types as in [Section 10.2.3.4](#), this XML schema uses the Oracle-defined data types by copying the schema `ordcmmdt.xsd` into an application-specific XSD file and then changing the namespace declaration, as in this XML code segment:

```

<xs:schema
xmlns="http://www.mycompany.com/dicom/example3"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xdb="http://xmlns.oracle.com/xdb"
targetNamespace="http://www.mycompany.com/dicom/example3"
elementFormDefault="qualified"
attributeFormDefault="unqualified">

```

This data type XML schema is registered as a global XML schema with Oracle XML DB, using the following namespace URL:

```
"http://www.mycompany.com/dicom/datatype_3"
```

Then, this registered namespace URL is included in the metadata XML schema.

See *Oracle XML DB Developer's Guide* for more information about registering XML schemas.

- Because the Oracle-defined data types are included in the XML schema, they can be referenced directly, as shown in the `<NAME>`, `<ADMITTING_DIAGNOSES_CODE_SEQUENCE>`, and `<OTHER_ATTRIBUTES>` elements.
- The `<OTHER_ATTRIBUTES>` element is defined as type "DATASET_T" to enable the inclusion of all unmapped DICOM attributes under this element when a value of `ALL` or `STANDARD` is passed to the `extractMetadata()` method for the `extractOption` parameter. Because the `minOccurs` attribute is defined as 0, when the value `MAPPED` is passed to the `extractMetadata()` method, only the mapped section is included in the extracted XML document.

[Example 10–30](#) shows the resulting metadata XML document whose XML metadata was extracted with the value of the `extractOption` parameter in the `extractMetadata()` method set to `MAPPED`.

Example 10–30 Resulting XML Document for [Example 10–28](#) (MAPPED)

```
<?xml version="1.0" encoding="DEC-MCS"?>
```

```

<DICOM_OBJECT xmlns="http://www.mycompany.com/dicom/example3"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.mycompany.com/dicom/example3
  http://www.mycompany.com/dicom/example3">
  <KEY_ATTRIBUTES>
    <PATIENT>
      <NAME>
        <NAME type="unibyte">
          <FAMILY>CANCIO 2HR A-02-013</FAMILY>
        </NAME>
        <VALUE>CANCIO 2HR A-02-013</VALUE>
      </NAME>
      <ID>ISRSCT610b</ID>
      <BIRTH_DATE xsi:nil="true"/>
      <SEX xsi:nil="true"/>
    </PATIENT>
    <STUDY>
      <ID definer="DICOM" tag="00200010" name="Study ID">352</ID>
      <ADMITTING_DIAGNOSES_CODE_SEQUENCE>
        <ITEM>
          <SHORT_STRING tag="00080100" definer="DICOM" name="Code Value"
            offset="692" length="0"/>
          <SHORT_STRING tag="00080102" definer="DICOM" name="Coding Scheme
            Designator" offset="700" length="0"/>
          <SHORT_STRING tag="00080103" definer="DICOM" name="Coding Scheme
            Version" offset="708" length="0"/>
          <LONG_STRING tag="00080104" definer="DICOM" name="Code Meaning"
            offset="716" length="0"/>
          <CODE_STRING tag="00080105" definer="DICOM" name="Mapping Resource"
            offset="724" length="0"/>
          <DATE_TIME tag="00080106" definer="DICOM" name="Context Group Version"
            offset="732" length="0" xsi:nil="true" rawValue=""
            byteOrderLE="true"/>
          <DATE_TIME tag="00080107" definer="DICOM" name="Context Group Local
            Version" offset="740" length="0" xsi:nil="true" rawValue=""
            byteOrderLE="true"/>
          <CODE_STRING tag="0008010B" definer="DICOM" name="Context Group
            Extension Flag" offset="748" length="0"/>
          <UNIQUE_ID tag="0008010D" definer="DICOM" name="Context Group Extension
            Creator UID" offset="756" length="0" xsi:nil="true" rawValue=""
            byteOrderLE="true"/>
          <CODE_STRING tag="0008010F" definer="DICOM" name="Context Identifier"
            offset="764" length="0"/>
        </ITEM>
      </ADMITTING_DIAGNOSES_CODE_SEQUENCE>
    </STUDY>
  </KEY_ATTRIBUTES>
</DICOM_OBJECT>

```

Example 10–31 shows the resulting metadata XML document whose XML metadata was extracted with the value of the `extractOption` parameter in the `extractMetadata()` method set to `ALL`.

Example 10–31 Resulting XML Document for Example 10–28 (ALL)

```

<?xml version="1.0" encoding="DEC-MCS"?>
<DICOM_OBJECT xmlns="http://www.mycompany.com/dicom/example3"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.mycompany.com/dicom/example3
  http://www.mycompany.com/dicom/example3">

```

```

<KEY_ATTRIBUTES>
  <PATIENT>
    <NAME>
      <NAME type="unibyte">
        <FAMILY>CANCIO 2HR A-02-013</FAMILY>
      </NAME>
      <VALUE>CANCIO 2HR A-02-013</VALUE>
    </NAME>
    <ID>ISR SCT610b</ID>
    <BIRTH_DATE xsi:nil="true"/>
    <SEX xsi:nil="true"/>
  </PATIENT>
  <STUDY>
    <ID definer="DICOM" tag="00200010" name="Study ID">352</ID>
    <ADMITTING_DIAGNOSES_CODE_SEQUENCE>
      <ITEM>
        <SHORT_STRING tag="00080100" definer="DICOM" name="Code Value"
          offset="692" length="0"/>
        <SHORT_STRING tag="00080102" definer="DICOM" name="Coding Scheme
          Designator" offset="700" length="0"/>
        <SHORT_STRING tag="00080103" definer="DICOM" name="Coding Scheme
          Version" offset="708" length="0"/>
        <LONG_STRING tag="00080104" definer="DICOM" name="Code Meaning"
          offset="716" length="0"/>
        <CODE_STRING tag="00080105" definer="DICOM" name="Mapping Resource"
          offset="724" length="0"/>
        <DATE_TIME tag="00080106" definer="DICOM" name="Context Group Version"
          offset="732" length="0" xsi:nil="true" rawValue=""
          byteOrderLE="true"/>
        <DATE_TIME tag="00080107" definer="DICOM" name="Context Group Local
          Version" offset="740" length="0" xsi:nil="true" rawValue=""
          byteOrderLE="true"/>
        <CODE_STRING tag="0008010B" definer="DICOM" name="Context Group
          Extension Flag" offset="748" length="0"/>
        <UNIQUE_ID tag="0008010D" definer="DICOM" name="Context Group Extension
          Creator UID" offset="756" length="0" xsi:nil="true" rawValue=""
          byteOrderLE="true"/>
        <CODE_STRING tag="0008010F" definer="DICOM" name="Context Identifier"
          offset="764" length="0"/>
      </ITEM>
    </ADMITTING_DIAGNOSES_CODE_SEQUENCE>
  </STUDY>
</KEY_ATTRIBUTES>
<OTHER_ATTRIBUTES>
  <OTHER_BYTE tag="00020001" definer="DICOM" name="File Meta Information
    Version" offset="156" length="2">AQA=
</OTHER_BYTE>
  <UNIQUE_ID tag="00020002" definer="DICOM" name="Media Storage SOP Class UID"
    offset="166" length="26">1.2.840.10008.5.1.4.1.1.2</UNIQUE_ID>
  <UNIQUE_ID tag="00020003" definer="DICOM" name="Media Storage SOP Instance
    UID" offset="200" length="54">1.2.392.200036.91</UNIQUE_ID>
  <UNIQUE_ID tag="00020010" definer="DICOM" name="Transfer Syntax UID"
    offset="262" length="18">1.2.840.10008.1.2</UNIQUE_ID>
  <UNIQUE_ID tag="00020012" definer="DICOM" name="Implementation Class UID"
    offset="288" length="16">1.2.804.114118.3</UNIQUE_ID>
  <APPLICATION_ENTITY tag="00020016" definer="DICOM" name="Source Application Entity
    Title" offset="326" length="0" xsi:nil="true" rawValue="" byteOrderLE="true"/>
  <CODE_STRING tag="00080008" definer="DICOM" name="Image Type" offset="334"
    length="22">ORIGINAL</CODE_STRING>

```

```

<CODE_STRING tag="00080008" definer="DICOM" name="Image Type" offset="334"
  length="22">PRIMARY</CODE_STRING>
<CODE_STRING tag="00080008" definer="DICOM" name="Image Type" offset="334"
  length="22">AXIAL</CODE_STRING>
<UNIQUE_ID tag="00080016" definer="DICOM" name="SOP Class UID" offset="364"
  length="26">1.2.840.10008.5.1.4.1.1.2</UNIQUE_ID>
<UNIQUE_ID tag="00080018" definer="DICOM" name="SOP Instance UID" offset="398"
  length="54">1.2.392.200036.91</UNIQUE_ID>
<DATE tag="00080020" definer="DICOM" name="Study Date" offset="460"
length="8">2004-02-23</DATE>
<DATE tag="00080022" definer="DICOM" name="Acquisition Date" offset="476"
length="8">2004-02-23</DATE>
<DATE tag="00080023" definer="DICOM" name="Content Date" offset="492"
length="8">2004-02-23</DATE>
<TIME tag="00080030" definer="DICOM" name="Study Time" offset="508"
length="14">18:48:41.000000</TIME>
<TIME tag="00080032" definer="DICOM" name="Acquisition Time" offset="530"
length="14">18:50:52.100000</TIME>
<TIME tag="00080033" definer="DICOM" name="Content Time" offset="552"
length="14">18:50:52.725000</TIME>
<SHORT_STRING tag="00080050" definer="DICOM" name="Accession Number"
offset="574" length="4">352</SHORT_STRING>
<CODE_STRING tag="00080060" definer="DICOM" name="Modality" offset="586"
length="2">CT</CODE_STRING>
.
.
.

<SHORT_STRING tag="00400253" definer="DICOM" name="Performed Procedure Step
  ID" offset="1742" length="4">351</SHORT_STRING>
<OTHER_WORD tag="7FE00010" definer="DICOM" name="Pixel Data" offset="1766"
  length="524288" truncated="true" xsi:nil="true" endian="little"/>
</OTHER_ATTRIBUTES>
</DICOM_OBJECT>

```

10.2.3.6 Using DICOM Value Locators in Mapping Documents

This subsection shows how to use DICOM value locators to specify attributes in mapping documents. [Example 10–32](#) shows how to use a DICOM value locator to specify the family name field of the Patient's Name attribute with unibyte encoding in a mapping document. The XML statement where this action is performed is highlighted in bold.

Example 10–32 DICOM Value Locator for an Attribute with Unibyte Encoding in a Mapping Document

```

<MAPPED_PATH>
  <ATTRIBUTE_TAG>00100010#UnibyteFamily</ATTRIBUTE_TAG>
  <PATH>FAMILY_NAME</PATH>
</MAPPED_PATH>

```

Using the mapping document that contains the code shown in [Example 10–32](#) in the `extractMetadata()` method results in an XML document that contains the tag `<FAMILY_NAME>` with the value that corresponds to the family name field of the attribute 00100010.

[Example 10–33](#) shows how to use a DICOM value locator to specify an attribute embedded within a sequence attribute in a mapping document.

Example 10–33 DICOM Value Locator for an Embedded Sequence Attribute in a Mapping Document

```
<MAPPED_PATH>
  <ATTRIBUTE_TAG>0040A730[2].0040A010</ATTRIBUTE_TAG>
  <PATH>RELATIONSHIP</PATH>
</MAPPED_PATH>
```

Using the mapping document that contains the code shown in [Example 10–33](#) in the `extractMetadata()` method results in an XML document that contains the tag `<RELATIONSHIP>` with the value that corresponds to the value of attribute `0040A010`. Attribute `0040A010` is embedded within the second item of the sequence attribute `0040A730`.

The standard attribute tags `0040A730` and `0040A010` are defined as follows in the standard dictionary:

```
<STANDARD_ATTRIBUTE_DEFINITION>
  <TAG>0040A730</TAG>
  <NAME>Content Sequence</NAME>
  <VR>SQ</VR>
  <VM>1</VM>
</STANDARD_ATTRIBUTE_DEFINITION>
```

```
<STANDARD_ATTRIBUTE_DEFINITION>
  <TAG>0040A010</TAG>
  <NAME>Relationship Type</NAME>
  <VR>CS</VR>
  <VM>1</VM>
</STANDARD_ATTRIBUTE_DEFINITION>
```

In the tag definition shown in the preceding code segment, the `<VR>` element shows that attribute `0040A730` is an attribute of sequence type. By using the DICOM value locator, any attribute that is embedded within a sequence type attribute can be retrieved individually into the resulting XML metadata document.

[Example 10–34](#) shows how to use a DICOM value locator to specify a single value of an attribute that contains multiple values in a mapping document.

Example 10–34 DICOM Value Locator for a Single Attribute Value in a Mapping Document

```
<MAPPED_PATH>
  <ATTRIBUTE_TAG>00080008[3]</ATTRIBUTE_TAG>
  <PATH>IMAGE_TYPE</PATH>
</MAPPED_PATH>
```

Using the mapping document that contains the code shown in [Example 10–34](#) in the `extractMetadata()` method results in an XML document that contains the tag `<IMAGE_TYPE>` with the value that corresponds to the third value of the attribute `00080008`.

The standard attribute tag `00080008` is defined as follows in the standard dictionary:

```
<STANDARD_ATTRIBUTE_DEFINITION>
  <TAG>00080008</TAG>
  <NAME>Image Type</NAME>
  <VR>CS</VR>
  <VM>1-n</VM>
</STANDARD_ATTRIBUTE_DEFINITION>
```

In the tag definition shown in the preceding code segment, the `<VM>` element shows that the attribute can contain multiple values. Using the DICOM value locator with an item number such as `00080008[3]` ensures that the specific (third) value is retrieved.

See [General Format for DICOM Value Locators](#) for more information about the DICOM value locator type.

10.2.4 Creating Standard Dictionary Documents

Standard dictionary documents are XML representations of the data dictionary defined by the DICOM standard.

The XML schema `ordcmsd.xsd` defines the XML schema that constrains standard dictionary documents. (See [Section B.10](#) for a listing of the standard dictionary document schema `ordcmsd.xsd`. See the text included within the `<xs:annotation>` element for a detailed description of the schema elements.)

The default standard dictionary document `ordcmsd.xml` is an XML representation of the data dictionary, as defined in Part 6 of the DICOM standard.

The XML schema `ordcmrdt.xsd` defines the DICOM standard data types and Oracle extensions to those data types, which are used by all other DICOM XML schemas. (See [Section B.3](#) for a listing of the data type definition schema `ordcmrdt.xsd`.)

Note: Standard dictionary documents must not include any other attributes, such as attributes defined by manufacturers of modalities or organizations other than the DICOM Standards Committee.

Through the Oracle Technology Network, Oracle may release new standard dictionary documents along with installation instructions for DICOM administrators. These new standard dictionary documents reflect new releases of, or addendum to, the DICOM standard.

The following subsections contain examples that show how to create standard dictionary documents:

- [Defining Standard Attributes](#)
- [Retiring Standard Attributes](#)

10.2.4.1 Defining Standard Attributes

The standard attribute definition in the standard dictionary permits two types of attribute tags: simple tags and wildcard tags. An attribute tag value consists of the group number followed by an element number.

A simple attribute tag is 4-byte hexadecimal number that consists of a group number followed by an element number (for example: `10871100`).

A wildcard attribute tag represents a set of hexadecimal numbers. It also consists of a group number followed by an element number, where portions of the value are replaced by the character `x` or `X` (for example: `1087xx00`).

Each standard attribute in the standard dictionary document is represented by a `<STANDARD_ATTRIBUTE_DEFINITION>` element. Each standard attribute listed in the XML document corresponds to a related tag that is defined in Part 6 of the DICOM standard.

Note: The definer name "DICOM" and the UID "1.2.840.10008.1" are reserved by Oracle to refer to DICOM standard attributes.

Defining a Standard Attribute Using a Simple Attribute Tag

[Example 10–35](#) shows how to define a standard attribute with a simple attribute tag. The XML statements where this action is defined are highlighted in bold.

Example 10–35 Definition of a Standard Attribute with a Simple Attribute Tag

```
<STANDARD_ATTRIBUTE_DEFINITION>
  <TAG>00080008</TAG>
  <NAME>Image Type</NAME>
  <VR>CS</VR>
  <VM>1-n</VM>
</STANDARD_ATTRIBUTE_DEFINITION>
```

In [Example 10–35](#), the `<TAG>` element represents the hexadecimal value of the attribute tag 00080008, which is defined by the DICOM standard. This attribute tag value consists of a group number, followed by an element number. The `<NAME>` element represents the name of the attribute tag that is defined by the DICOM standard. The `<VR>` element represents the value representation of the attribute tag that is defined by the DICOM standard. The `<VM>` element represents the value multiplicity of the attribute tag defined by the DICOM standard. (See the type `<VR_T>` and the type `<VM_T>` listed in the XML schema `ordcmrtd.xsd` for more information about the value representation and value multiplicity data types supported by Oracle Multimedia DICOM.)

Defining a Standard Attribute Using a Wildcard Attribute Tag

[Example 10–36](#) shows how to define a standard attribute with a wildcard attribute tag. The XML statements where this action is defined are highlighted in bold.

Example 10–36 Definition of a Standard Attribute with a Wildcard Attribute Tag

```
<STANDARD_ATTRIBUTE_DEFINITION>
  <TAG>60xx0010</TAG>
  <NAME>Overlay Rows</NAME>
  <VR>US</VR>
  <VM>1</VM>
</STANDARD_ATTRIBUTE_DEFINITION>
```

In [Example 10–36](#), the standard attribute definition contains the wildcard tag `60xx0010`.

10.2.4.2 Retiring Standard Attributes

Standard attributes included in the standard dictionary document are retired in the DICOM standard when they are no longer needed. [Example 10–37](#) shows the definition of a standard attribute as retired. The XML statement where this action is defined is highlighted in bold.

Example 10–37 Definition of a Standard Attribute As Retired

```
<STANDARD_ATTRIBUTE_DEFINITION>
  <TAG>00080010</TAG>
  <NAME>Recognition Code</NAME>
```

```
<VR>CS</VR>
<VM>1</VM>
<RETIRED>true</RETIRED>
</STANDARD_ATTRIBUTE_DEFINITION>
```

In [Example 10-37](#), for the `<RETIRED>` element a value of `true` indicates that the attribute is retired and corresponds to the `RET` value used in the DICOM standard. Oracle recommends providing the `<VR>` and `<VM>` element values for a retired attribute to enable the metadata to be extracted from the DICOM content.

10.2.5 Creating Private Dictionary Documents

Private dictionary documents are XML documents that list the attributes defined by modality manufacturers or organizations other than the DICOM Standards Committee.

Private organizations or manufacturers of modalities can include attributes that are specific to their particular organizations in their DICOM content. These private attributes must be defined in a private dictionary document and stored in the data model repository to enable Oracle Multimedia to process metadata associated with the private attributes in the DICOM content.

The XML schema `ordcmpv.xsd` defines the XML schema that constrains private dictionary documents. (See [Section B.9](#) for a listing of the private dictionary document schema `ordcmpv.xsd`.)

The default private dictionary document `ordcmpv.xml` lists the private attributes defined by Oracle.

The XML schema `ordcmrdt.xsd` defines the DICOM standard data types and Oracle extensions to those data types, which are used by all other DICOM XML schemas. (See [Section B.3](#) for a listing of the data type definition schema `ordcmrdt.xsd`.)

The following subsections contain examples that show how to create private dictionary documents:

- [Defining Private Attributes](#)
- [Defining Attribute Definers](#)
- [Retiring Private Attributes](#)

10.2.5.1 Defining Private Attributes

The private attribute definition in the private dictionary permits three types of attribute tags: simple tags, wildcard tags, and range tags. An attribute tag value consists of the group number followed by an element number.

A simple attribute tag is 4-byte hexadecimal number that consists of a group number followed by an element number (for example: `10871100`).

A wildcard attribute tag represents a set of hexadecimal numbers. It also consists of a group number followed by an element number, where portions of the value are replaced by the character `x` or `X` (for example: `1087xx00`).

A range attribute tag represents a range of hexadecimal numbers. It consists of two simple attribute tags that define a range of values. The starting range value must be less than the ending range value.

Each private attribute in the private dictionary document is represented by a `<PRIVATE_ATTRIBUTE_DEFINITION>` element.

Note: Keep these additional guidelines in mind:

- Do not use wildcard attribute tags within range attribute tags.
 - The definer name "DICOM" and the UID "1.2.840.10008.1" are reserved by Oracle to refer to DICOM standard attributes. Do not use them in private dictionaries.
 - Oracle recommends associating all DICOM private attributes in the DICOM content with a UID-qualified name.
 - The value of a tag-definer pair must be unique within the private dictionary document.
-
-

Defining a Private Attribute Using a Simple Attribute Tag

[Example 10–38](#) shows how to define a private attribute with a simple attribute tag. The XML statements where this action is defined are highlighted in bold.

Example 10–38 Definition of a Private Attribute with a Simple Attribute Tag

```
<PRIVATE_ATTRIBUTE_DEFINITION>
  <TAG>FFFF1001</TAG>
  <NAME>locator macro tag</NAME>
  <DEFINER>ORACLE</DEFINER>
  <VR>SQ</VR>
  <VM>1</VM>
</PRIVATE_ATTRIBUTE_DEFINITION>
```

In [Example 10–38](#), the `<TAG>` element represents the hexadecimal value of the attribute tag `FFFF1001`. The `<NAME>` element represents the name of the organization defining the tag, `ORACLE`. The `<VR>` element represents the value representation of the attribute tag. The `<VR>` element is defined as type `VR_T` in the XML schema `ordcmrdt.xsd`. The `<VR>` element value must be one of the values that are listed for type `<VR_T>` in the XML schema `ordcmrdt.xsd`. The `<VM>` element represents the value multiplicity of the attribute tag. The `<VM>` element is defined as type `VM_T` in the XML schema `ordcmrdt.xsd`. The `<VM>` element value must be one of the values listed for type `VM_T` in the XML schema `ordcmrdt.xsd`. (See the type `<VR_T>` and the type `<VM_T>` listed in the XML schema `ordcmrdt.xsd` for more information about the value representation and value multiplicity data types supported by Oracle Multimedia DICOM.)

Defining a Private Attribute Using a Wildcard Attribute Tag

[Example 10–39](#) shows how to define a private attribute with a wildcard attribute tag. The XML statements where this action is defined are highlighted in bold.

Example 10–39 Definition of a Private Attribute with a Wildcard Attribute Tag

```
<PRIVATE_ATTRIBUTE_DEFINITION>
  <TAG>0119XX00</TAG>
  <NAME>PRIVATE_TAG1</NAME>
  <DEFINER>PRIVATE_ORG</DEFINER>
  <VR>IS</VR>
  <VM>1</VM>
</PRIVATE_ATTRIBUTE_DEFINITION>
```

In [Example 10–39](#), the private attribute definition contains the wildcard tag `0119XX00` and the definer name `PRIVATE_ORG`.

Defining a Private Attribute Using a Range Attribute Tag

[Example 10–40](#) shows how to define a private attribute with a range attribute tag. The XML statements where this action is defined are highlighted in bold.

Example 10–40 Definition of a Private Attribute with a Range Attribute Tag

```
<PRIVATE_ATTRIBUTE_DEFINITION>
  <TAG_RANGE>
    <dt:STARTING_TAG>A0110010</dt:STARTING_TAG>
    <dt:ENDING_TAG>A01AAA10</dt:ENDING_TAG>
  </TAG_RANGE>
  <NAME>Private Tag</NAME>
  <DEFINER>1.2.840.423</DEFINER>
  <VR>ST</VR>
  <VM>1</VM>
</PRIVATE_ATTRIBUTE_DEFINITION>
```

In [Example 10–40](#), the private attribute definition contains a range tag that specifies all tag ranges from A0110010 to A01AAA10. The definer name 1.2.840.423 is a UID value.

Note: An attribute definition cannot match another attribute definition of the same tag-definer pair within a private dictionary document.

The following XML code segment shows an example of an attribute definition that is *not* permitted in the private dictionary:

```
<PRIVATE_ATTRIBUTE_DEFINITION>
  <TAG>0119XX00</TAG>
  <NAME>private tag</NAME>
  <DEFINER>PRIVATE_ORG</DEFINER>
  .
  .
  .
</PRIVATE_ATTRIBUTE_DEFINITION>

<PRIVATE_ATTRIBUTE_DEFINITION>
  <TAG>01191100</TAG>
  <NAME>private tag</NAME>
  <DEFINER>PRIVATE_ORG</DEFINER>
  .
  .
  .
</PRIVATE_ATTRIBUTE_DEFINITION>
```

In the preceding code segment, the tag-definer pair (01191100, PRIVATE_ORG) matches the tag-definer pair (0119XX00, PRIVATE_ORG).

10.2.5.2 Defining Attribute Definers

An attribute definer refers to the definer name and UID of an organization defining private attributes.

The <ATTRIBUTE_DEFINERS> element is an optional element that represents all private definer names and definer UIDs used in a private dictionary document.

Note: The definer name "DICOM" and the UID "1.2.840.10008.1" are reserved by Oracle to refer to DICOM standard attributes. Do not use them in private dictionaries.

The <ATTRIBUTE_DEFINERS> element is of type ATTR_DEFINERS_T. This type is defined in the XML schema ordcmrdt.xsd.

[Example 10–41](#) shows how to define the definer name and UID for an organization that defines private attributes:

Example 10–41 Definition of an Attribute Definer

```
<ATTRIBUTE_DEFINERS>
  <dt:ATTR_DEFINER>
    <dt:NAME>1.2.1234.12345.123456.2.0</dt:NAME>
    <dt:UID>1.2.1234.12345.123456.2.0</dt:UID>
  </dt:ATTR_DEFINER>
  <dt:ATTR_DEFINER>
    <dt:NAME>PRIVATE_ORG</dt:NAME>
    <dt:UID>1.2.123.1234</dt:UID>
  </dt:ATTR_DEFINER>
</ATTRIBUTE_DEFINERS>
```

In [Example 10–41](#), the prefix dt is mapped to the namespace `http://xmlns.oracle.com/ord/dicom/datatype_1_0` where the <ATTR_DEFINER> element is defined. The value of the <dt:NAME> element represents the name of the private organization. The value of the <dt:UID> element represents the UID of the private organization.

10.2.5.3 Retiring Private Attributes

Private attributes included in private dictionary documents can be retired when they are no longer needed. [Example 10–42](#) shows how to define a private attribute as retired. The XML statement where this action is defined is highlighted in bold.

Example 10–42 Definition of a Private Attribute as Retired

```
<PRIVATE_ATTRIBUTE_DEFINITION>
  <TAG>01191100</TAG>
  <NAME>locator macro tag</NAME>
  <DEFINER>PRIVATE_ORG</DEFINER>
  <VR>SQ</VR>
  <VM>1</VM>
  <RETIRED>true</RETIRED>
</PRIVATE_ATTRIBUTE_DEFINITION>
```

In [Example 10–42](#), for the <RETIRED> element, a value of true indicates that the attribute is retired and corresponds to the RET value used in the DICOM standard. Oracle recommends providing the <VR> and <VM> element values for a retired attribute to enable the metadata to be extracted from the DICOM content.

10.2.6 Creating Preference Documents

Preference documents are XML documents that list the set of preference parameters that define the run-time behavior of Oracle Multimedia DICOM.

The XML document `ordcmpf.xsd` defines the XML schema that constrains preference documents. See [Section B.8](#) for a listing of the preference document schema `ordcmpf.xsd`.

The default preference document `ordcmpf.xml` includes the default values for the run-time preference parameters for Oracle Multimedia DICOM.

Within a preference document, a `<PREFERENCE_DEF>` element is used to define each preference parameter and its value.

Note: Keep these additional guidelines in mind:

- A maximum of two (one Oracle-defined and one user-defined) preference documents are permitted in the repository.
 - Changes to preference parameter values in preference documents change the behavior of DICOM methods, functions, and procedures. Specifically, preference parameter values that are defined in the user-defined preference document override the default values defined in the Oracle-defined preference document `ordcmpf.xml`.
 - All preference parameter names and their associated values are defined by Oracle. User-defined preference documents can change the values of the Oracle-defined preference parameters.
-

See the text included within the `<xs:annotation>` element in [Example B-8](#) on page B-47 for a detailed description of the preference parameters and their valid values.

The following subsections contain examples that show how to define preference parameter values in a preference document:

- [Defining the BINARY_SKIP_INVALID_ATTR Preference Parameter](#)
- [Defining the EXP_IF_NULL_ATTR_IN_CONSTRAINT Preference Parameter](#)
- [Defining the MANDATE_ATTR_TAGS_IN_STL Preference Parameter](#)
- [Defining the MAX_RECURSION_DEPTH Preference Parameter](#)
- [Defining the SPECIFIC_CHARACTER_SET Preference Parameter](#)
- [Defining the SQ_WRITE_LEN Preference Parameter](#)
- [Defining the VALIDATE_METADATA Preference Parameter](#)
- [Defining the XML_SKIP_ATTR Preference Parameter](#)

10.2.6.1 Defining the BINARY_SKIP_INVALID_ATTR Preference Parameter

The `BINARY_SKIP_INVALID_ATTR` preference parameter is used to specify whether Oracle Multimedia DICOM includes or skips invalid attributes and their values in the binary output of DICOM content when making a copy of the DICOM Part 10 file. Invalid attributes are attributes that do not conform to the specified DICOM standard. [Example 10-43](#) shows how to define this preference parameter in a preference document.

Example 10-43 Definition of the BINARY_SKIP_INVALID_ATTR Preference Parameter

```
<PREFERENCE_DEF>
  <PARAMETER>BINARY_SKIP_INVALID_ATTR</PARAMETER>
```

```

<DESCRIPTION>
  Do not skip the value of attributes that do not conform to the
  DICOM specification when making a copy of a DICOM Part 10 file.
</DESCRIPTION>
<VALUE>false</VALUE>
</PREFERENCE_DEF>

```

The `<PARAMETER>` element specifies the Oracle-defined preference parameter `BINARY_SKIP_INVALID_ATTR`.

The `<VALUE>` element represents the value of the `BINARY_SKIP_INVALID_ATTR` preference parameter, which is `false` by default.

If the value of this preference parameter is `false`, invalid attributes and their values are included in the corresponding output. If the preference parameter value is set to `true`, the attribute values are included in the output with length 0.

10.2.6.2 Defining the `EXP_IF_NULL_ATTR_IN_CONSTRAINT` Preference Parameter

The `EXP_IF_NULL_ATTR_IN_CONSTRAINT` preference parameter is used to specify whether Oracle Multimedia DICOM throws an exception when encountering a missing attribute or a null attribute value during conformance validation.

[Example 10-44](#) shows how to define this preference parameter in a preference document.

Example 10-44 Definition of the `EXP_IF_NULL_ATTR_IN_CONSTRAINT` Preference Parameter

```

<PREFERENCE_DEF>
  <PARAMETER>EXP_IF_NULL_ATTR_IN_CONSTRAINT</PARAMETER>
  <DESCRIPTION>
    Throw an exception if a missing attribute or a null
    attribute value is encountered during conformance
    validation.
  </DESCRIPTION>
  <VALUE>true</VALUE>
</PREFERENCE_DEF>

```

The `<PARAMETER>` element specifies the Oracle-defined preference parameter `EXP_IF_NULL_ATTR_IN_CONSTRAINT`.

The `<VALUE>` element represents the value of the `EXP_IF_NULL_ATTR_IN_CONSTRAINT` preference parameter, which is `true` by default.

In a constraint predicate that is not preceded by a `notEmpty` Boolean function predicate on an attribute, when the value of this preference parameter is `true`, an exception is thrown at the first occurrence of either of these conditions:

- The attribute is missing.
- The attribute has a null value.

See [Example 10-14](#) for more information.

If the value is set to `false`, no exception is thrown.

This preference parameter affects the results of the method `isConformanceValid()`. See the [isConformanceValid\(\)](#) method for more information.

10.2.6.3 Defining the `MANDATE_ATTR_TAGS_IN_STL` Preference Parameter

The `MANDATE_ATTR_TAGS_IN_STL` preference parameter is used to specify whether Oracle Multimedia DICOM enforces the rule that all tags used by the

constraint and mapping documents must be listed in the stored tag list document. [Example 10–45](#) shows how to define this preference parameter in a preference document.

Example 10–45 Definition of the MANDATE_ATTR_TAGS_IN_STL Preference Parameter

```
<PREFERENCE_DEF>
  <PARAMETER>MANDATE_ATTR_TAGS_IN_STL</PARAMETER>
  <DESCRIPTION>
    All tags in the constraint and mapping docs must be listed in the
    STORED_TAG_LIST document.
  </DESCRIPTION>
  <VALUE>true</VALUE>
</PREFERENCE_DEF>
```

The `<PARAMETER>` element specifies the Oracle-defined preference parameter `MANDATE_ATTR_TAGS_IN_STL`.

The `<VALUE>` element represents the value of the `MANDATE_ATTR_TAGS_IN_STL` preference parameter, which is `false` by default.

The value of this preference parameter determines whether to enforce the rule that all tags used by the constraint and mapping documents are listed in the stored tag list document. If the value of this preference parameter is set to `true`, the rule is enforced. If the value is `false`, the rule is not enforced.

However, if an existing stored tag list document does not satisfy the rule, the value of the preference parameter cannot be set to `true` until the stored tag list document is updated. Also, if the value of this preference parameter is `true`, only constraint and mapping documents that use tags listed in the stored tag list document can be inserted into the repository.

10.2.6.4 Defining the MAX_RECURSION_DEPTH Preference Parameter

The `MAX_RECURSION_DEPTH` preference parameter is used to specify the maximum level of recursion Oracle Multimedia DICOM uses to evaluate recursive constraint macros during conformance validation. [Example 10–46](#) shows how to define this preference parameter in a preference document.

Example 10–46 Definition of the MAX_RECURSION_DEPTH Preference Parameter

```
<PREFERENCE_DEF>
  <PARAMETER>MAX_RECURSION_DEPTH</PARAMETER>
  <DESCRIPTION>
    The maximum level of recursion when evaluating a recursive constraint
    during conformation validation.
  </DESCRIPTION>
  <VALUE>16</VALUE>
</PREFERENCE_DEF>
```

The `<PARAMETER>` element specifies the Oracle-defined preference parameter `MAX_RECURSION_DEPTH`.

The `<VALUE>` element represents the value of the `MAX_RECURSION_DEPTH` preference parameter. Valid values are integers from 1 to 32767. The default is 16. If the recursion level exceeds the value of this parameter, an exception is thrown.

See [Section 10.2.2.4](#) for more information about recursive constraint macros.

10.2.6.5 Defining the SPECIFIC_CHARACTER_SET Preference Parameter

The SPECIFIC_CHARACTER_SET preference parameter is used to specify the character set Oracle Multimedia DICOM uses to decode certain data elements (those with value representations of SH, LO, ST, LT, PN, and UT) when the standard attribute Specific Character Set is not specified. [Example 10–47](#) shows how to define this preference parameter in a preference document.

Example 10–47 Definition of the SPECIFIC_CHARACTER_SET Preference Parameter

```
<PREFERENCE_DEF>
  <PARAMETER>SPECIFIC_CHARACTER_SET</PARAMETER>
  <DESCRIPTION>
    Use the ASCII character set to decode data elements SH (Short String),
    LO (Long String), ST (Short Text), LT (Long Text), PN (Person Name),
    and UT (Unlimited Text) when the Specific Character Set Attribute is
    missing.
  </DESCRIPTION>
  <VALUE>ASCII</VALUE>
</PREFERENCE_DEF>
```

The <PARAMETER> element specifies the Oracle-defined preference parameter SPECIFIC_CHARACTER_SET.

The <VALUE> element represents the value of the SPECIFIC_CHARACTER_SET preference parameter. Valid values are:

Character Sets

ASCII(default value)

ISO_IR 100

ISO_IR 101

ISO_IR 109

ISO_IR 110

ISO_IR 144

ISO_IR 127

ISO_IR 126

ISO_IR 138

ISO_IR 148

ISO_IR 13

ISO_IR 166

ISO_IR 192

GB18030

The value of this preference parameter determines how data elements with value representations of SH (Short String), LO (Long String), ST (Short Text) LT (Long Text), PN (Person Name), and UT (Unlimited Text) are decoded when the standard attribute Specific Character Set (0008,0005) is missing. The DICOM standard mandates using the default character set ISO-IR 6 or ASCII for decoding when the standard attribute Specific Character Set is not specified. This preference

parameter enables applications to specify a character set other than the DICOM standard default character set in these cases.

10.2.6.6 Defining the SQ_WRITE_LEN Preference Parameter

The SQ_WRITE_LEN preference parameter is used to specify how the method `writeMetadata()` encodes DICOM sequence (SQ) types. [Example 10–48](#) shows how to define this preference parameter in a preference document.

Example 10–48 Definition of the SQ_WRITE_LEN Preference Parameter

```
<PREFERENCE_DEF>
  <PARAMETER>SQ_WRITE_LEN</PARAMETER>
  <DESCRIPTION>
    All sequence types are encoded in XML with an explicit length and
    without sequence delimiters.
  </DESCRIPTION>
  <VALUE>true</VALUE>
</PREFERENCE_DEF>
```

The `<PARAMETER>` element specifies the Oracle-defined preference parameter `SQ_WRITE_LEN`.

The `<VALUE>` element represents the value of the `SQ_WRITE_LEN` preference parameter, which is `true` by default.

The value of this preference parameter specifies how the DICOM sequence (SQ) types are encoded by the method `writeMetadata()`. See the [writeMetadata\(\)](#) method for more information.

If the value of this preference parameter is `true`, the SQ types are encoded with explicit lengths and without item or sequence delimiters. This default behavior enables DICOM viewers to skip the sequence attributes.

If the value is set to `false`, the SQ types are encoded with variable (undefined) lengths and terminated with sequence delimiters. This option enables backward compatibility with older DICOM viewers and DICOM applications that support only undefined lengths for SQ types.

10.2.6.7 Defining the VALIDATE_METADATA Preference Parameter

The VALIDATE_METADATA preference parameter is used to specify whether to validate XML documents that are used in DICOM functions and procedures. [Example 10–49](#) shows how to define this preference parameter in a preference document.

Example 10–49 Definition of the VALIDATE_METADATA Preference Parameter

```
<PREFERENCE_DEF>
  <PARAMETER>VALIDATE_METADATA</PARAMETER>
  <DESCRIPTION>
    Do not validate XML metadata documents.
  </DESCRIPTION>
  <VALUE>false</VALUE>
</PREFERENCE_DEF>
```

The `<PARAMETER>` element specifies the Oracle-defined preference parameter `VALIDATE_METADATA`.

The `<VALUE>` element represents the value of the preference parameter `VALIDATE_METADATA`, which is `false` by default.

The value of this preference parameter determines whether to validate the XML documents used in the DICOM functions and procedures. All XML documents used in the DICOM functions and procedures, except those that are generated by the method `extractMetadata()`, are validated against the Oracle default DICOM metadata schema. The XML documents generated by the method `extractMetadata()` are validated against the XML schema whose namespace is defined in the specified mapping document. See the `extractMetadata()` method and [Section 10.1.3](#) for more information.

If the value of this preference parameter is `false`, the XML documents are not validated.

If the value is set to `true`, the XML documents are validated against a specific XML schema that is registered with Oracle XML DB.

10.2.6.8 Defining the XML_SKIP_ATTR Preference Parameter

The `XML_SKIP_ATTR` preference parameter is used to specify size limits for DICOM attributes to omit when encoding into XML. [Example 10–50](#) shows how to define this preference parameter in a preference document.

Example 10–50 Definition of the XML_SKIP_ATTR Preference Parameter

```
<PREFERENCE_DEF>
  <PARAMETER>XML_SKIP_ATTR</PARAMETER>
  <DESCRIPTION>
    When encoding DICOM attributes into XML, omit the DICOM attributes
    with sizes larger than 512 bytes.
  </DESCRIPTION>
  <VALUE>512</VALUE>
</PREFERENCE_DEF>
```

The `<PARAMETER>` element specifies the Oracle-defined preference parameter `XML_SKIP_ATTR`.

The `<VALUE>` element represents the value of the `XML_SKIP_ATTR` preference parameter. Valid values are integers from 128 to $2^{32}-1$. The default is 512.

The value of this preference parameter is used to specify size limits for DICOM attributes to be omitted when encoding into XML. Omit DICOM attributes whose size (in bytes) in the DICOM content is larger than the value of this preference parameter. If an attribute of type SQ is omitted, its child items are also omitted.

In addition, the value of this preference parameter affects the following:

- The metadata generated by the method `extractMetadata()`. See the [extractMetadata\(\)](#) method for more information.
- The behavior of the `setProperties()` method. See the [setProperties\(\)](#) method for more information.

10.2.7 Creating UID Definition Documents

UID definition documents are XML representations of the unique identifiers (UIDs) defined by the DICOM standard. The UID definitions are used by Oracle Multimedia DICOM to parse the DICOM content.

The XML schema `ordcmui.xsd` defines the XML schema that constrains UID definition documents. (See [Section B.12](#) for a listing of the UID definition document schema `ordcmui.xsd`. See the text included within the `<xs:annotation>` element for a detailed description of the attributes used in the schema.)

The default UID definition document `ordcmui.xml` includes the UIDs listed in Part 6 of the DICOM standard.

Within a UID definition document, a `<UID_DEF>` element is used to define each UID definition.

Note: Keep these additional guidelines in mind:

- A maximum of two (one Oracle-defined and one user-defined) UID definition documents are permitted in the repository.
 - Changes in user-defined UID definition documents must be limited to updates in the DICOM standard or additions of new UID values. See Part 5 of the DICOM standard for more information about creating privately defined unique identifiers.
 - Existing UIDs defined by the DICOM standard can be changed *only* to include updates in the DICOM standard.
-
-

The following subsections contain examples that show how to create UID definitions in UID definition documents:

- [Defining UID Definitions](#)
- [Retiring UID Definitions](#)

10.2.7.1 Defining UID Definitions

[Example 10–51](#) shows how to define a storage class UID definition. The XML statements where this action is defined are highlighted in bold.

Example 10–51 Definition of a Storage Class UID Definition

```
<UID_DEF classification="storageClass" contentType="image">
  <UID>1.2.840.10008.5.1.4.1.1.2</UID>
  <NAME>CT Image Storage</NAME>
</UID_DEF>
```

In [Example 10–51](#), the `<UID>` element value `1.2.840.10008.5.1.4.1.1.2` represents a UID value that is defined by the DICOM standard. The `<NAME>` element value `CT Image Storage` represents the UID name that is defined by the DICOM standard. The value of the `classification` attribute `"storageClass"` in the `<UID_DEF>` element corresponds to the UID type SOP Class defined by the DICOM standard. The value of the `contentType` attribute `"image"` indicates that the DICOM content contains pixel data.

10.2.7.2 Retiring UID Definitions

[Example 10–52](#) shows how to define a UID definition for transfer syntax as retired. The XML statements where this action is defined are highlighted in bold.

Example 10–52 Definition of a UID Definition as Retired

```
<UID_DEF classification="transferSyntax" isCompressed="true" isEVR="true"
  isLE="true" retired="true">
  <UID>1.2.840.10008.1.2.4.52</UID>
  <NAME>JPEG Extended (Process 3 and 5) (Retired)</NAME>
</UID_DEF>
```

In [Example 10-52](#), the <UID> element value 1.2.840.10008.1.2.4.52 represents a UID value that is defined by the DICOM standard. The <NAME> element value JPEG Extended (Process 3 and 5) (Retired) represents the UID name that is defined by the DICOM standard. The value of the `classification` attribute "transferSyntax" corresponds to the UID type Transfer Syntax defined by the DICOM standard. The values of the attributes `isCompressed`, `isEVR` (explicit VR), and `isLE` (little endian) are derived from the transfer syntax definitions listed in Part 5 of the DICOM standard. The value of the `retired` attribute "true" indicates that this UID definition is retired.

10.2.8 Creating Stored Tag List Documents

Stored tag list documents specify the DICOM attributes to be extracted from the embedded DICOM content and stored in the XML metadata attribute of the ORDDicom object when the `setProperties()` method is called. The XML schema `ordcmstl.xsd` defines the XML schema that constrains stored tag list documents. (See [Section B.11](#) for a listing of the stored tag list document schema `ordcmstl.xsd`.)

Generally, stored tag list documents contain the attribute tags used in mapping and constraint documents. Each attribute tag value in a stored tag list document is defined by an <ATTRIBUTE_TAG> element.

Oracle recommends using the [generateTagListDocument\(\) Function](#) to create a stored tag list document. For example, a stored tag list might include some tags shown in the following code segment:

```
<ATTRIBUTE_TAG>00080008[3]</ATTRIBUTE_TAG>
<ATTRIBUTE_TAG>00081155 (DICOM)</ATTRIBUTE_TAG>
<ATTRIBUTE_TAG>00082112.00081150</ATTRIBUTE_TAG>
<ATTRIBUTE_TAG>00082112.0040A170.00080104</ATTRIBUTE_TAG>
<ATTRIBUTE_TAG>0040A730.0040A123[1]#PersonName</ATTRIBUTE_TAG>
<ATTRIBUTE_TAG>0040A730[2].0040A160</ATTRIBUTE_TAG>
```

See [Section 11.4](#) for examples of generating and inserting stored tag list documents into the repository. See [Section 3.2.10](#) for information about using a stored tag list to extract specific attributes from DICOM content.

Administering the DICOM Repository

This chapter uses code examples to show how to manage configuration documents in the DICOM data model repository. The code examples are written in PL/SQL to match the ORD_DICOM_ADMIN package, which is provided in PL/SQL only.

Oracle Multimedia DICOM provides capabilities for several administrative operations related to the data model repository. For example, administrators can review the Oracle-defined configuration documents in the DICOM data model repository before determining whether to add custom configuration documents for a particular organization (Steps 2 and 4 in [Section 11.2](#) describe this process). Using views or invoking data model utility functions, administrators can obtain attributes and other detailed information about these configuration documents. Working with the procedures and functions in the ORD_DICOM_ADMIN package, administrators can insert, update, export, or delete configuration documents from the repository. Administrators of the DICOM data model repository are assigned the ORDADMIN role.

This chapter includes these sections:

- [Sample Session 1: Inserting Two Documents](#) on page 11-1
- [Sample Session 2: Updating a Mapping Document](#) on page 11-3
- [Sample Session 3: Deleting a Constraint Document](#) on page 11-4
- [Sample Session 4: Inserting a Stored Tag List Document](#) on page 11-5

See [Chapter 12](#) for reference information about the ORD_DICOM_ADMIN package.

For additional examples, articles, and other information about Oracle Multimedia DICOM and Oracle Multimedia, see the Oracle Multimedia Software section of the Oracle Technology Network Web site at

<http://www.oracle.com/technology/products/multimedia/>

11.1 Sample Session 1: Inserting Two Documents

The following sample session shows the steps for inserting a mapping document and a constraint document into the repository. This sample assumes that the following prerequisite tasks have been completed:

- The directory object (DICOMDIR) has been created, and the administrator has been granted READ access.
- The metadata schema associated with the mapping document has been registered with Oracle XML DB as a global XML schema.

See Also:

Oracle XML DB Developer's Guide for more information about registering XML schemas

Perform the following steps to insert a mapping document and a constraint document into the repository:

Step 1 Edit the Data Model

Prepare the data model for editing and lock it to prevent other administrators from making changes at the same time. For example:

```
exec ord_dicom_admin.editDataModel();
```

The data model remains locked until you publish the changes, perform a rollback operation, or exit the session.

Optionally, you can enter the following command to query the `orddcm_documents` view:

```
select * from orddcm_documents order by doc_id asc;
```

Review the list of documents.

Step 2 Insert the New Documents

First, insert the sample mapping document (`sample_map.xml`), as follows:

```
exec ord_dicom_admin.insertDocument('sample_map.xml', 'MAPPING',  
  xmltype(bfilename('DICOMDIR', 'sample_map.xml'), nls_charset_id('AL32UTF8')));
```

See [Example 2-1](#) for the sample mapping document (`sample_map.xml`).

Then, insert the sample constraint document (`sample_ct.xml`), as follows:

```
exec ord_dicom_admin.insertDocument('sample_ct.xml', 'CONSTRAINT',  
  xmltype(bfilename('DICOMDIR', 'sample_ct.xml'), nls_charset_id('AL32UTF8')));
```

See [Example 2-3](#) for the sample constraint document (`sample_ct.xml`).

Optionally, you can enter the following command to query the `orddcm_documents` view:

```
select * from orddcm_documents order by doc_id asc;
```

Review the list of documents to ensure that the inserted documents appear.

As another option, you can enter the following command to query the `orddcm_document_refs` view, and then review the document references:

```
select * from orddcm_document_refs order by doc_name, ref_by_doc_name asc;
```

At this point, you can run administrative tests as needed. In addition, you can call the `rollbackDataModel()` procedure to terminate the operation without publishing the changes. Or, you can continue with the next step.

Step 3 Publish the Changes

Publish the changes and unlock the data model, as follows:

```
exec ord_dicom_admin.publishDataModel();
```

The data model includes the inserted configuration documents.

11.2 Sample Session 2: Updating a Mapping Document

The following sample session shows the steps for updating a mapping document in the repository. This sample assumes that the following prerequisite tasks have been completed:

- The directory object (DICOMDIR) has been created, and the administrator has been granted READ and WRITE access.
- The referenced metadata schema associated with the mapping document has been registered with Oracle XML DB as a global XML schema.

See *Oracle XML DB Developer's Guide* for information about registering XML schemas.

Perform the following steps to update a mapping document in the repository:

Step 1 Edit the Data Model

As in [Section 11.1](#), prepare the data model for editing and lock it to prevent other administrators from making changes at the same time. For example:

```
exec ord_dicom_admin.editDataModel();
```

The data model remains locked until you publish the changes, perform a rollback operation, or exit the session.

Optionally, you can enter the following command to query the orddcm_documents view:

```
select * from orddcm_documents order by doc_id asc;
```

Review the list of documents.

Step 2 Export the Existing Document

Export the sample mapping document (sample_map.xml) from the repository into a specified file (sample_map_export.xml) for editing. For example:

```
exec ord_dicom_admin.exportDocument('sample_map.xml', 'DICOMDIR',
  'sample_map_export.xml');
```

See [Example 2-1](#) for the sample mapping document (sample_map.xml).

The sample_map_export.xml file is available for editing with an XML editor.

Step 3 Delete the Existing Document

Delete the existing mapping document (sample_map.xml) from the repository, as follows:

```
exec ord_dicom_admin.deleteDocument('sample_map.xml');
```

The repository no longer includes the sample mapping document.

Optionally, you can enter the following command to query the orddcm_documents view:

```
select * from orddcm_documents order by doc_id asc;
```

Review the list of documents to ensure that the deleted document no longer appears.

Step 4 Edit the Exported Document

Use an XML editor to edit the file that was exported in Step 2. Then, save the changes and check the permissions on the file before inserting it into the repository.

Step 5 Insert the Edited Document

Insert the edited document (`sample_map_edited.xml`) into the repository, as follows:

```
exec ord_dicom_admin.insertDocument('sample_map_edited.xml', 'MAPPING',
  xmltype(bfilename('DICOMDIR', 'sample_map_edited.xml'),
  nls_charset_id('AL32UTF8')));
```

Optionally, you can enter the following command to query the `orddcm_documents` view:

```
select * from orddcm_documents order by doc_id asc;
```

Review the list of documents to ensure that the updated document appears.

At this point, you can test your changes, run the `extractMetadata()` method on the edited mapping document to retrieve metadata from the embedded DICOM content as XML code, and then store it in a database table for searching or viewing. In addition, you can call the `rollbackDataModel()` procedure to terminate the operation without publishing the changes. Or, you can continue with the next step.

Step 6 Publish the Changes

As in [Section 11.1](#), publish the changes and unlock the data model, as follows:

```
exec ord_dicom_admin.publishDataModel();
```

The data model includes the updated configuration document.

11.3 Sample Session 3: Deleting a Constraint Document

The following sample session shows the steps for deleting a constraint document from the repository. This sample assumes that the following prerequisite task has been completed:

The directory object (`DICOMDIR`) has been created, and the administrator has been granted `WRITE` access.

Perform the following steps to delete a constraint document from the repository:

Step 1 Edit the Data Model

As in [Section 11.1](#), prepare the data model for editing and lock it to prevent other administrators from making changes at the same time. For example:

```
exec ord_dicom_admin.editDataModel();
```

The data model remains locked until you publish the changes, perform a rollback operation, or exit the session.

Optionally, you can enter the following command to query the `orddcm_documents` view:

```
select * from orddcm_documents order by doc_id asc;
```

Review the list of documents.

Step 2 Export the Existing Document (Optional and Recommended)

Export the sample constraint document (`sample_ct.xml`) from the repository into a specified file (`sample_ct_export.xml`) for editing. For example:

```
exec ord_dicom_admin.exportDocument('sample_ct.xml', 'DICOMDIR',
'sample_ct_export.xml');
```

See [Example 2-3](#) for the sample constraint document (`sample_ct.xml`).

Optionally, you can enter the following command to query the `orddcm_document_refs` view, which shows the list of documents that are referenced by other documents in the repository:

```
select * from orddcm_document_refs order by doc_name, ref_by_doc_name asc;
```

Review the list of documents to check the references for the constraint document that you intend to delete.

Step 3 Delete the Document

Delete the sample constraint document (`sample_ct.xml`), as follows:

```
exec ord_dicom_admin.deleteDocument('sample_ct.xml');
```

The repository no longer includes the sample constraint document.

Optionally, you can enter the following command to query the `orddcm_documents` view:

```
select * from orddcm_documents order by doc_id asc;
```

Review the list of documents to ensure that the deleted document no longer appears.

At this point, you can call the `rollbackDataModel()` procedure to terminate the operation without publishing the changes. Or, you can continue with the next step.

Step 4 Publish the Changes

Publish the changes and unlock the data model, as follows:

```
exec ord_dicom_admin.publishDataModel();
```

The data model does not include the deleted configuration document.

11.4 Sample Session 4: Inserting a Stored Tag List Document

This section describes two different approaches you can take to insert a stored tag list document into the repository:

- [Section 11.4.1](#) describes how to insert a stored tag list document with a known set of attribute tags.

Follow the example in this section to create, and then insert, your own stored tag list document if you believe that you know the full set of attribute tags that all the constraint and mapping documents in your repository are likely to use.

- [Section 11.4.2](#) describes how to generate and insert a stored tag list document from an existing repository that is populated with constraint and mapping documents.

Follow the examples in this section if you have an existing repository and want to generate, and then insert, a stored tag list document from tags used by your existing constraint and mapping documents.

Select the appropriate approach to take advantage of the storage and performance benefits of the stored tag list feature.

The following subsections describe each approach:

- [Inserting a Stored Tag List Document with a Known Set of Tags](#)
- [Generating and Inserting a Stored Tag List Document](#)

11.4.1 Inserting a Stored Tag List Document with a Known Set of Tags

You can create your own stored tag list document, and then insert it into the repository. Include all the attribute tags to be defined in the constraint and mapping documents in the repository in your document. [Example 11–1](#) is a sample file (`sample_stl.xml`) that you can insert as the stored tag list document.

Example 11–1 Inserting a Stored Tag List Document with Known Attribute Tags

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
Copyright (c) 2008, Oracle. All rights reserved.

NAME
sample_stl.xml - sample stored tag list

-->
<ATTRIBUTE_TAG_LIST xmlns="http://xmlns.oracle.com/ord/dicom/attributeTag_1_0"
xmlns:dt="http://xmlns.oracle.com/ord/dicom/datatype_1_0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.oracle.com/ord/dicom/attributeTag_1_0
http://xmlns.oracle.com/ord/dicom/attributeTag_1_0">
<DOCUMENT_HEADER>
<dt:DOCUMENT_CHANGE_LOG>
<dt:DOCUMENT_MODIFIER>ORACLE</dt:DOCUMENT_MODIFIER>
<dt:DOCUMENT_MODIFICATION_DATE>2009-02-25</dt:DOCUMENT_MODIFICATION_DATE>
<dt:DOCUMENT_VERSION>1.0</dt:DOCUMENT_VERSION>
</dt:DOCUMENT_CHANGE_LOG>
</DOCUMENT_HEADER>
<ATTRIBUTE_TAG>00020002</ATTRIBUTE_TAG>
<ATTRIBUTE_TAG>00020003</ATTRIBUTE_TAG>
<ATTRIBUTE_TAG>00100010</ATTRIBUTE_TAG>
</ATTRIBUTE_TAG_LIST>
```

This approach assumes that the following prerequisite task has been completed:

The directory object (DICOMDIR) has been created, and the administrator has been granted READ access.

Perform the following steps to insert a stored tag list document with a known set of attribute tags into the repository:

Step 1 Edit the Data Model

As in [Section 11.1](#), prepare the data model for editing and lock it to prevent other administrators from making changes at the same time. For example:

```
exec ord_dicom_admin.editDataModel();
```

The data model remains locked until you publish the changes, perform a rollback operation, or exit the session.

At this point, to see which documents are stored in the repository, enter the following command to query the `orddcm_documents` view:

```
select doc_name from orddcm_documents order by doc_id asc;
```

Step 2 Insert the Stored Tag List Document

Insert the sample stored tag list document file (`sample_stl.xml`), as follows:

```
exec ord_dicom_admin.insertDocument(
    'sample_stl.xml',
    'STORED_TAG_LIST',
    xmltype(bfilename('DICOMDIR', 'sample_stl.xml'),
    nls_charset_id('AL32UTF8') ) );
```

Step 3 Publish the Changes

Publish the changes and unlock the data model, as follows:

```
exec ord_dicom_admin.publishDataModel;
```

The data model includes the inserted configuration documents.

11.4.2 Generating and Inserting a Stored Tag List Document

If you have an existing repository that is populated with constraint and mapping documents, you can generate a stored tag list document from tags used by your existing constraint and mapping documents, make changes to the tags as needed, and insert the stored tag list document into the repository.

To enforce specific rules for the attributes you want to include in your stored tag list, insert a preference document that defines those attribute rules. [Example 11–2](#) shows a preference document (`sample_pf.xml`) that enforces the mandate attribute tags rule for mapping and constraint documents. The code statements where this operation is performed are highlighted in bold.

Example 11–2 Sample Preference Document for the Mandate Attribute Tags Rule

```
<?xml version="1.0" encoding="UTF-8"?>
<DICOM_RUNTIME_PREFERENCES xmlns="http://xmlns.oracle.com/ord/dicom/preference_1_0"
  xmlns:dt="http://xmlns.oracle.com/ord/dicom/datatype_1_0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.oracle.com/ord/dicom/preference_1_0
  http://xmlns.oracle.com/ord/dicom/preference_1_0">
  <DOCUMENT_HEADER>
    <dt:DOCUMENT_CHANGE_LOG>
      <dt:DOCUMENT_MODIFIER>Developer 1</dt:DOCUMENT_MODIFIER>
      <dt:DOCUMENT_MODIFICATION_DATE>2009-02-20</dt:DOCUMENT_MODIFICATION_DATE>
      <dt:DOCUMENT_VERSION>0.2</dt:DOCUMENT_VERSION>
      <dt:MODIFICATION_COMMENT> enable MANDATE_ATTR_TAGS_IN_STL</dt:MODIFICATION_COMMENT>
    </dt:DOCUMENT_CHANGE_LOG>
  </DOCUMENT_HEADER>
  <PREFERENCE_DEF>
    <PARAMETER>MANDATE_ATTR_TAGS_IN_STL</PARAMETER>
    <DESCRIPTION> allowed values true or false</DESCRIPTION>
    <VALUE>true</VALUE>
  </PREFERENCE_DEF>
</DICOM_RUNTIME_PREFERENCES>
```

Because the value of the `MANDATE_ATTR_TAGS_IN_STL` parameter is set to `true`, all the tags used by the constraint and mapping documents are listed in the stored tag list document. See [Section 10.2.6.3](#) for more information about defining this parameter in preference documents.

This approach assumes that the following prerequisite task has been completed:

The directory object (`DICOMDIR`) has been created, and the administrator has been granted `READ` and `WRITE` access.

Perform the following steps to generate and insert a stored tag list document into the repository:

Step 1 Edit the Data Model

As in [Section 11.1](#), prepare the data model for editing and lock it to prevent other administrators from making changes at the same time. For example:

```
exec ord_dicom_admin.editDataModel();
```

The data model remains locked until you publish the changes, perform a rollback operation, or exit the session.

Optionally, you can enter the following command to query the orddcm_documents view:

```
select doc_name from orddcm_documents order by doc_id asc;
```

Review the list of documents to see which documents exist in your repository.

Step 2 Generate and Export the Stored Tag List Document to a File

Generate and export the stored tag list document to the sample stored tag list document file (sample_stl.xml), as shown in [Example 11-3](#):

Example 11-3 *Generating and Exporting a Stored Tag List Document to a File*

```
--
-- Generate a tag list document and store in sample_stl.xml
-- in DICOMDIR
--
-- A stored tag list document must include all tags used by
-- mapping and constraint documents in the repository
-- to enforce the mandate_attr_tags_in_stl rule using a
-- preference document.
--
-- Use the 'ALL' option to ensure that the generated tag list
-- document contains all the tags used by mapping and constraint
-- documents.
--
set serveroutput on;
declare
  fid          UTL_FILE.FILE_TYPE;
  buf          VARCHAR2(32767);
  bytesLeft    INTEGER;
  pos          INTEGER;
  chunk        INTEGER;
  xml_clob     CLOB;
  xt           XMLType;
BEGIN

  xt := ORD_DICOM_ADMIN.generateTagListDocument('ALL');
  if (xt is null) then
    dbms_output.put_line('Repository has no documents');
    dbms_output.put_line('Tag list document is null');
    return;
  end if;

  -- write xml document to file using UTL_FILE
  -- let UTL_FILE report errors for directory and filename
  fid := utl_file.fopen( 'DICOMDIR', 'sample_stl.xml', 'w', 32767 );
```

```

-- Read from clob and write to file
select XMLSerialize(document xt as clob)
       into xml_clob from dual;
bytesLeft := dbms_lob.getLength( xml_clob );
pos       := 1;
WHILE bytesLeft > 0 loop
  chunk := 32767;
  IF bytesLeft < chunk THEN
    chunk := bytesLeft;
  END IF;
  dbms_lob.read(xml_clob, chunk, pos, buf );
  utl_file.put( fid, buf);
  utl_file.fflush(fid);
  bytesLeft := bytesLeft - chunk;
  pos := pos + chunk;
end loop;

utl_file.fclose( fid );

--free the temporary blob returned by xmltype
if(dbms_lob.isTemporary(xml_clob) <> 0) then
  dbms_lob.freeTemporary(xml_clob);
end if;

exception
WHEN OTHERS then
  -- free temp lob
  if(dbms_lob.isTemporary(xml_clob) <> 0) then
    dbms_lob.freeTemporary(xml_clob);
  end if;

  raise;  -- original exception
END;
/

```

Step 3 Optional: Edit the Stored Tag List Document

Use an XML editor to edit the stored tag list document file that was exported in Step 2. Make changes to the tags as needed, save the changes, and check the permissions on the file before inserting it into the repository.

Step 4 Insert the Stored Tag List Document

Insert the generated stored tag list document (sample_stl.xml), as follows:

```

ord_dicom_admin.insertDocument(
    'sample_stl.xml',
    'STORED_TAG_LIST',
    xmltype(bfilename('DICOMDIR', 'sample_stl.xml'),
    nls_charset_id('AL32UTF8') ) );

```

Step 5 Optional: Enforce the Mandate Attribute Tags Rule

To enforce the mandate attribute tags rule, insert a preference document with the value of the MANDATE_ATTR_TAGS_IN_STL parameter set to true, as follows:

```

exec ord_dicom_admin.insertDocument(
    'sample_pf.xml',
    'PREFERENCE',
    xmltype(bfilename('DICOMDIR', 'sample_pf.xml'),

```

```
nls_charset_id('AL32UTF8') ) );
```

See [Example 11–2](#) for a sample preference document that enforces this rule.

Note: If the `MANDATE_ATTR_TAGS_IN_STL` parameter is set to `true` in your preference document, and you attempt to insert any mapping or constraint documents that contain tags that are not listed in your stored tag list document, those documents are not inserted into the repository. See [Section 10.2.6.3](#) for more information about this preference parameter.

Optionally, you can enter the following command to query the `orddcm_documents` view:

```
select * from orddcm_documents order by doc_id asc;
```

Review the list of documents.

Step 6 Publish the Changes

Publish the changes and unlock the data model, as follows:

```
exec ord_dicom_admin.publishDataModel;
```

ORD_DICOM_ADMIN Package Reference

Oracle Multimedia DICOM provides the ORD_DICOM_ADMIN data model repository interface in the ORD_DICOM_ADMIN package. This package provides the data model repository interface, which is intended for use by DICOM administrators to maintain the Oracle Multimedia DICOM repository. The data model repository is a collection of documents. One set of documents is loaded during installation. After installation, DICOM administrators can add more documents to the data model repository using the procedures and functions in the ORD_DICOM_ADMIN package.

Oracle Multimedia DICOM also defines a view for the DICOM repository to be used by DICOM administrators.

In addition, Oracle Multimedia provides DICOM value locators, which administrators can use to specify DICOM attributes or their child components within the DICOM content and with anonymity, constraint, and mapping documents.

The ORD_DICOM_ADMIN package is defined in the `ordcrpsp.sql` file. After installation, this file is available in the Oracle home directory at:

```
<ORACLE_HOME>/ord/im/admin (on Linux and UNIX)
```

```
<ORACLE_HOME>\ord\im\admin (on Windows)
```

This chapter describes the functions, procedures, and views in the ORD_DICOM_ADMIN data model repository interface. It also describes the syntax for DICOM value locators. See [Table 3-1](#) for information about other DICOM application programming interfaces (APIs).

This chapter contains these sections:

- [Directory Definition and Setup for ORD_DICOM_ADMIN Examples](#) on page 12-1
- [Important Notes for DICOM Repository Administrators](#) on page 12-2
- [DICOM Data Model Repository Administrator Functions](#) on page 12-3
- [DICOM Data Model Repository Administrator Procedures](#) on page 12-6
- [DICOM Repository Administrator Views](#) on page 12-15
- [General Format for DICOM Value Locators](#) on page 12-17

12.1 Directory Definition and Setup for ORD_DICOM_ADMIN Examples

In this chapter, some examples use `c:\mydir\work` to represent the directory specification where your test files can be located. Other examples use `DICOMDIR` to represent the directory specification for the test files. And, a few examples use `dcmadmin` to represent the user, who is acting as an administrator. See the example

for each function or procedure for specific directory definitions for DICOM data files and other details specific to that function or procedure.

12.2 Important Notes for DICOM Repository Administrators

The following guidelines apply to administrative operations related to the Oracle Multimedia DICOM data model repository:

- Administrators must be assigned the ORDADMIN role. See [Section 9.1](#) for more information about administrator privileges.
- At the start of each database session, administrators must load the data model repository from the database into memory structures. To load the data model, administrators call either the `setDataModel()` procedure or the `editDataModel()` procedure, depending on the following situations:
 - When you *are not* making changes to the data model, call the `setDataModel()` procedure (see the [setDataModel\(\) Procedure](#)).
 - When you *are* making changes to the data model, call the `editDataModel()` procedure (see the [editDataModel\(\) Procedure](#)).

DICOM Data Model Repository Administrator Functions

The ORD_DICOM_ADMIN package defines these ORD_DICOM_ADMIN data model repository functions:

- [generateTagListDocument\(\) Function](#) on page 12-4
- [getDocumentContent\(\) Function](#) on page 12-5

generateTagListDocument() Function

Format

generateTagListDocument(docSet in varchar2 default 'USER') return XMLType

Description

Returns a stored tag list document as a data type XMLType that contains the attribute tags from the mapping and constraint documents stored in the data model repository, and which conforms to the XML schema `ordcmst1.xsd`. (See [Section B.11](#) for a listing of this schema.)

Parameters

docSet

A string that lists a specified set of documents in the repository. Valid values are: `USER`, `ALL`, and `ORACLE`. The default is `USER`.

When the value of this parameter is `USER`, the attribute tags from the user-defined mapping and constraint documents are listed in the generated stored tag list document. If there are no user-defined constraint or mapping documents in the repository, a null value is returned. When the value is set to `ALL`, the attribute tags from all the mapping and constraint documents are listed. And, when the value is set to `ORACLE`, only the attribute tags from the Oracle-defined mapping and constraint documents are listed.

Pragmas

None.

Exceptions

None.

Usage Notes

Use this function to generate a stored tag list document that is consistent with the repository documents. (See [Section 10.2.8](#) for more information about creating stored tag list documents.)

Before calling this function, call the `setDataModel()` procedure (see the [setDataModel\(\) Procedure](#)).

Examples

Generate a stored tag list document for the user-defined mapping and constraint documents in the repository:

```
exec ord_dicom.setDataModel;
set long 50000;
set pagesize 1000;
select ord_dicom_admin.generateTagListDocument from dual;
```

getDocumentContent() Function

Format

```
getDocumentContent(docName IN VARCHAR2) RETURN XMLTYPE
```

Description

Returns the document as data type XMLType. This function can be used to make copies of documents in the data model repository without changing the content of the original documents.

Parameters

docName

The name of the document whose content is to be retrieved. The document content is returned as data type XMLType. The orddcm_documents view includes the column DOC_NAME, where docName is one of the documents listed.

Pragmas

None.

Exceptions

None.

Usage Notes

Before calling this function, call either the setDataModel() procedure or the editDataModel() procedure in these situations:

- Call the setDataModel() procedure when you are not making changes to the data model, such as when calling the getDocumentContent() procedure or the exportDocument() procedure only.
- Call the editDataModel() procedure when you are making changes to the data model, such as when inserting or deleting documents.

See the [setDataModel\(\) Procedure](#) and the [editDataModel\(\) Procedure](#) for more information.

Examples

Get the content of a specified document (ordcmpf.xml) in the repository:

```
exec ord_dicom.setDataModel;
set long 5000;
set pagesize 1000;
select ord_dicom_admin.getDocumentContent('ordcmpf.xml') from dual;
```

DICOM Data Model Repository Administrator Procedures

The ORD_DICOM_ADMIN package defines these ORD_DICOM_ADMIN data model repository procedures:

- [deleteDocument\(\) Procedure](#) on page 12-7
- [editDataModel\(\) Procedure](#) on page 12-8
- [exportDocument\(\) Procedure](#) on page 12-9
- [insertDocument\(\) Procedure](#) on page 12-11
- [publishDataModel\(\) Procedure](#) on page 12-13
- [rollbackDataModel\(\) Procedure](#) on page 12-14

deleteDocument() Procedure

Format

```
deleteDocument(docName IN VARCHAR2)
```

Description

Deletes the specified document from the data model repository. Documents installed by Oracle are treated as default documents that cannot be removed.

See [Section 9.8](#) for more information about deleting configuration documents.

Parameters

docName

The name of the document to be removed from the repository, where docName is one of the documents listed in the orddcm_documents view.

Pragmas

None.

Exceptions

None.

Usage Notes

Before calling this procedure, call the editDataModel() procedure (see the [editDataModel\(\) Procedure](#)).

In addition, Oracle recommends calling the exportDocument() procedure to make a copy of the specified document before deleting it from the repository (see the [exportDocument\(\) Procedure](#)).

Examples

Copy and then delete a document (sample_pf.xml) from the repository:

```
exec ord_dicom_admin.editDataModel();
select doc_name from orddcm_documents order by doc_id asc;
exec ord_dicom_admin.exportDocument('sample_pf.xml', 'DICODEDIR',
                                     'sample_pf_export.xml');
exec ord_dicom_admin.deleteDocument('sample_pf.xml');
select doc_name from orddcm_documents order by doc_id asc;
exec ord_dicom_admin.publishDataModel();
```

where:

- DICODEDIR: an Oracle directory object for which the administrator has been granted the WRITE privilege, and to which the contents of the specified file are to be copied.

editDataModel() Procedure

Format

editDataModel()

Description

Begins an administrator editing session for making changes to the data model repository. The administrator session maintains a lock on the repository until the `publishDataModel()` or `rollbackDataModel()` procedure is called, or until the session exits. Call this procedure before making changes to the repository. Upon successful completion, this procedure implicitly commits the current transaction.

Parameters

None.

Pragmas

None.

Exceptions

None.

Usage Notes

None.

Examples

Begin an editing session in the repository, while locking it from other administrators:

```
exec ord_dicom_admin.editDataModel( );
```

exportDocument() Procedure

Format

```
exportDocument(docName IN VARCHAR2, dirName IN VARCHAR2, fileName IN VARCHAR2)
```

Description

Exports the contents of the document specified by the docName parameter to the specified file. This procedure writes the data to a file in a directory for which the administrator has been granted the WRITE privilege.

Parameters

docName

The name of the specified document in the repository, where docName is one of the documents listed in the orddcm_documents view.

dirName

The directory location of the specified file. This string is an Oracle directory object name, and it is case-sensitive. The default is uppercase. The WRITE privilege must be granted to the administrator for this directory object.

fileName

The name of the file, including the file extension (file type), but excluding the directory path.

Pragmas

None.

Exceptions

None.

Usage Notes

Before calling this procedure, call either the setDataModel() or the editDataModel() procedure in these situations:

- Call the setDataModel() procedure when you are not making changes to the data model, such as when calling the getDocumentContent() procedure or the exportDocument() procedure only.
- Call the editDataModel() procedure when you are making changes to the data model, such as when inserting or deleting documents.

See the [setDataModel\(\) Procedure](#) and the [editDataModel\(\) Procedure](#) for more information.

Examples

Export the contents of an existing file in the repository (sample_map.xml) into an external file (sample_map_export.xml):

```
exec ord_dicom.setDataModel();  
exec ord_dicom_admin.exportDocument('ordcmpf.xml', 'DICOMDIR', 'ordcmpf_exp.xml');
```

where:

- DICOMDIR: an Oracle directory object that contains the file where the contents of the specified file are to be copied.

insertDocument() Procedure

Format

```
insertDocument(docName IN VARCHAR2, docType IN VARCHAR2, xmlDoc IN XMLType)
```

Description

Loads the specified XML configuration document into the data model repository. The document name must be unique. Supported document types are listed in the public view [orddcm_document_types](#). The document is validated against the registered schema that is associated with the document type.

Documents must be loaded into the repository in the following order:

1. Standard data dictionary documents
2. Private data dictionary documents
3. Other configuration documents, including the following:
 - Constraint documents
 - XML mapping documents
 - Anonymity documents
 - Preference documents
 - UID definition documents

Other configuration documents can be loaded in any order, unless there are dependencies between constraint documents.

There are semantic dependencies between the documents. For example, elements referenced in an XML mapping document must exist in the standard or private data dictionary documents. The view [orddcm_documents](#) contains the details of the documents in the repository.

Parameters

docName

The unique name of the specified document. The name cannot exceed 100 characters in length, and it must not contain the reserved prefix `ORD`.

docType

The type of the document to be loaded into the repository. Supported values are listed in the public view [orddcm_document_types](#). The value of this parameter must not be `NULL`.

xmlDoc

The XML document to be loaded into the repository. The value of this parameter must not be `NULL`.

Pragmas

None.

Exceptions

None.

Usage Notes

Before calling this procedure, call the `editDataModel()` procedure (see the [editDataModel\(\) Procedure](#)).

Before inserting a user-defined mapping document into the repository, its associated metadata schema must have been registered as a global schema with Oracle XML DB. (The metadata schema associated with the default mapping document is registered with Oracle XML DB upon installation of Oracle Multimedia DICOM. See [Appendix B](#) for more information.)

See Also:

Oracle XML DB Developer's Guide for information about registering XML schemas

Examples

Insert a mapping document (`sample_map.xml`) into the repository:

```
exec ord_dicom_admin.editDataModel();
exec ord_dicom_admin.insertDocument('sample_map.xml', 'MAPPING',
  xmltype(bfilename('DICOMDIR', 'sample_map.xml'),
    nls_charset_id('AL32UTF8')));
select * from orddcm_documents order by doc_id asc;
exec ord_dicom_admin.publishDataModel();
```

where:

- **MAPPING**: the type of the document to be loaded into the repository.
- **DICOMDIR**: the Oracle directory object that contains the file to be loaded.

publishDataModel() Procedure

Format

```
publishDataModel( )
```

Description

Publishes changes to the data model repository. This procedure also unlocks the repository, making it available for updating by other administrators. By calling the `setDataModel()` procedure to refresh the data model repository, users can access the latest published changes to the repository. Upon successful completion, this procedure implicitly commits the current transaction.

Parameters

None.

Pragmas

None.

Exceptions

None.

Usage Notes

Before calling this procedure, call the `editDataModel()` procedure (see the [editDataModel\(\) Procedure](#)).

Depending on when the administrator publishes changes to the data model repository, users who are connected to the repository may have to call the `setDataModel()` procedure more than once to see the latest changes. Also, based on when they call the `setDataModel()` procedure, it is possible for two users who are connected to the same data model repository to access different versions of the repository. See [Figure 2–5](#) for more information about this possible scenario. See also the [setDataModel\(\) Procedure](#).

Examples

Publish the changes to the repository:

```
exec ord_dicom_admin.editDataModel();
select doc_name from orddcm_documents order by doc_id asc;
exec ord_dicom_admin.insertDocument( 'sample_pf.xml', 'PREFERENCE',
  xmltype(bfilename('DICOMDIR', 'sample_pf.xml'),
  nls_charset_id('AL32UTF8') ) );
select doc_name from orddcm_documents order by doc_id asc;
exec ord_dicom_admin.publishDataModel();
```

rollbackDataModel() Procedure

Format

rollbackDataModel()

Description

Terminates changes to the data model since the previous call to the editDataModel() procedure. Call this procedure to roll back the changes and unlock the data model, making it available for updating by other administrators. Upon successful completion, this procedure implicitly commits the current transaction.

Parameters

None.

Pragmas

None.

Exceptions

None.

Usage Notes

Before calling this procedure, call the editDataModel() procedure (see the [editDataModel\(\) Procedure](#)).

At end of the database session, call this procedure to roll back the data model. Because the changes to the data model were terminated rather than published, the views are populated with the data from the previously published data model.

Examples

Terminate the data model changes without publishing them to the repository:

```
exec ord_dicom_admin.editDataModel();
select doc_name from orddcm_documents order by doc_id asc;
exec ord_dicom_admin.deleteDocument('sample_pf.xml');
select doc_name from orddcm_documents order by doc_id asc;
exec ord_dicom_admin.rollbackDataModel();
```

DICOM Repository Administrator Views

This section describes this Oracle Multimedia DICOM repository view for administrators:

- [orddcm_document_refs](#) on page 12-16

See [DICOM Repository Public Views](#) for information about the Oracle Multimedia DICOM repository public views.

orddcm_document_refs

Format

Column Name	Data Type	Description
doc_name	VARCHAR2(100)	Document name
ref_by_doc_name	VARCHAR2(100)	Referenced by document name

Description

This read-only view lists the documents that are referenced by other documents. The SELECT privilege is granted to the ORDADMIN role for this view. This view is available to administrators only.

Usage Notes

Before querying this view, call either the `setDataModel()` or the `editDataModel()` procedure as follows:

- Call the `setDataModel()` procedure when you are not making changes to the data model, such as when calling the `getDocumentContent()` procedure or the `exportDocument()` procedure only.
- Call the `editDataModel()` procedure when you are making changes to the data model, such as when inserting or deleting documents.

If you call the `setDataModel()` procedure, call it again whenever the application requires the new data model changes.

See the [setDataModel\(\) Procedure](#) and the [editDataModel\(\) Procedure](#) for more information.

Examples

Show the references between a set of Oracle-installed configuration documents:

```

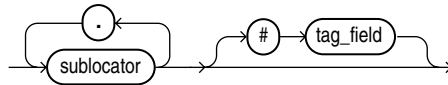
-----
DOC_NAME          REF_BY_DOC_NAME
-----
ordcmpv.xml      ordcmcmc.xml
ordcmpv.xml      ordcmcmd.xml
ordcmsd.xml      ordcman.xml
ordcmsd.xml      ordcmcmc.xml
ordcmsd.xml      ordcmcmd.xml

```

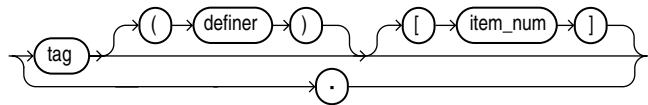
General Format for DICOM Value Locators

Syntax

value_locator ::=



sublocator ::=



Description

Identifies an attribute in the DICOM content, either at the root level or from the root level down. Each level in the tree hierarchy is represented by a sublocator. DICOM value locators can include one or more sublocators, depending on the level of the attribute in the DICOM content.

DICOM value locators can be used within anonymity documents to specify the attributes to be made anonymous in the resulting DICOM content.

When DICOM value locators are used within constraint documents, they can contain macros for substitution when the constraint document is loaded into the repository. Define macro substitution strings either as valid DICOM value locators, or as one of the DICOM value locator parameters with no token separators.

DICOM value locators can be used within mapping documents to specify the attributes to be retrieved or extracted from the metadata in the resulting DICOM XML document.

Parameters

tag

A DICOM standard compliant tag in the format of an 8-digit hexadecimal number.

Example 1: A hexadecimal tag for a root-level attribute

```
0040A730
```

Example 2: A hexadecimal tag for a root-level attribute

```
0040A123
```

. (the dot character)

A sublocator separator or a special wildcard tag that matches to zero or a series of sublocators. (See [DICOM Value Locator Example 4](#).)

definer

A string that identifies the organization creating the tag. For tags that are defined by the DICOM standard, the default (which can be omitted) is DICOM. For private tags, the value must include a definer string (which can be empty) to uniquely identify each private attribute. This parameter is optional.

Definer strings cannot contain the following unescaped token separator characters: '(', ')', '[', ']', '{', '}', '\$', '#', and '*'. Although it is possible to refer to definer strings that contain these characters, you must use an escape mechanism to express definer strings with these characters. Specifically, a single '#' (the escape character) must precede each token separator character in the definer string for a specified attribute.

Example 1: A definer string for a private organization

ORACLE

Example 2: A definer string with token separators that have been escaped. The original string was \$abc (#def) [ghi].

#\$abc##def#[ghi#]

item_num

An integer that identifies a data element within an attribute, or a wildcard character ("*") that identifies all data elements within an attribute. The default is 1, the first data element of an attribute. This parameter is optional.

- For the binary data types OF, OW, OB, and UN (defined in the data type definition schema `ordcmrdt.xsd`):
 This parameter identifies one data value within a DICOM attribute that contains a list of data values, as shown in the following table.

Binary Data Type	Data Value Identification
OF	A floating-point number
OW	A word
OB	A byte
UN	A byte

- For the data type SQ:
 This parameter identifies one sequence item.
- For the data types ST, LT, and UT:
 The value of this parameter is always 1 because attributes of these types always contain single values.

Note: The concept of an item number is not well-defined by the DICOM standard. Do not confuse it with DICOM value multiplicity.

DICOM value locators with a special wildcard tag ". " cannot be immediately followed by an `item_num` parameter.

Data elements are defined in Part 5 of the DICOM standard.

Example 1: An integer that represents a specific sequence item in an attribute

The second sequence item in attribute 0040A730:

0040A730 [2]

Example 2: An integer with a wildcard character that represents all the data elements in an attribute

All items in the sequence attribute 0040A730:

0040A730[*]

tag_field

A string that identifies a derived value within an attribute. A tag that contains this string must be the last tag of a DICOM value locator. The default is NONE. This parameter is optional.

The following table shows the supported values for the tag_field parameter:

Value	Meaning
NONE	N/A
UnibyteFamily	Family name, in unibyte characters ¹
UnibyteGiven	Given name, in unibyte characters ¹
UnibyteMiddle	Middle name, in unibyte characters ¹
UnibytePrefix	Prefix, in unibyte characters ¹
UnibyteSuffix	Suffix, in unibyte characters ¹
IdeographicFamily	Family name, in ideographic characters ¹
IdeographicGiven	Given name, in ideographic characters ¹
IdeographicMiddle	Middle name, in ideographic characters ¹
IdeographicPrefix	Prefix, in ideographic characters ¹
IdeographicSuffix	Suffix, in ideographic characters ¹
PhoneticFamily	Family name, in phonetic characters ¹
PhoneticGiven	Given name, in phonetic characters ¹
PhoneticMiddle	Middle name, in phonetic characters ¹
PhoneticPrefix	Prefix, in phonetic characters ¹
PhoneticSuffix	Suffix, in phonetic characters ¹
PersonName	Person name, as a concatenated string of characters in the convention of three groups of five components
AgeString	Age, as a string
AgeInDays	Age, in number of days
ByteLength	Binary length of the path
ByteOffset	Byte offset of the path
NumEntry	Total number of data elements within an attribute
VR ²	In-stream value representation (VR) of an attribute

¹ This value is a component of the attribute Person Name.

² See the description of the item_num parameter for more information about data types.

For complete definitions of the values in the preceding table, see Part 5 of the DICOM standard on the Web site for the National Electrical Manufacturers Association (NEMA) at

<http://medical.nema.org/>

Note: DICOM value locators with a tag_field parameter are not supported in anonymity and constraint documents.

DICOM value locators with a special wildcard tag ". ." cannot be immediately followed by a tag_field parameter.

Examples

DICOM Value Locator Example 1

A DICOM value locator with the required tag parameters only:

```
00080096.00401101.00080100
```

where:

- 00080096.00401101.00080100: the code value of the Person Identification Code Sequence in the Referring Physician Identification Sequence.

DICOM Value Locator Example 2

A DICOM value locator with the required tag parameter and the optional item_num parameter:

```
00080096[1].00401101[1].00080100[1]
```

where:

- [1]: the first data element of an attribute. This DICOM value locator is equivalent to the DICOM value locator shown in **DICOM Value Locator Example 1**.

DICOM Value Locator Example 3

A DICOM value locator with the required tag parameter and the optional definer and item_num parameters:

```
00080096 (DICOM) [1].00401101 (DICOM) [1].00080100 (DICOM) [1]
```

where:

- (DICOM): the default definer string. Because the tag is defined by the DICOM standard, the DICOM value locator can be represented without the definer string, as in **DICOM Value Locator Example 2**. This DICOM value locator is equivalent to the DICOM value locator in **DICOM Value Locator Example 1**.

DICOM Value Locator Example 4

A DICOM value locator with a special wildcard tag (" . ") for all code values in the DICOM content:

```
..00080100
```

DICOM Value Locator Example 5

A DICOM value locator with a special wildcard tag (" . ") for all code values in all the 0040A730 sequence items in the DICOM content:

```
0040A730[*]...00080100
```

DICOM Value Locator Example 6

A DICOM value locator with a private definer and escape characters:

```
00431028(#$abc#(##def#)#[ghi#])
```

where:

- (`#$abc#(##def#)#[ghi#]`): the private definer string. Because the definer string contains token separator characters, it must be represented by preceding each token separator character with the escape character ("`#`").

Part IV

DICOM Appendixes

This part contains supplementary information related to the Oracle Multimedia DICOM feature.

Part IV includes these appendixes:

- [Appendix A, "DICOM Configuration Documents"](#)
- [Appendix B, "DICOM XML Schemas"](#)
- [Appendix C, "DICOM Encoding Rules"](#)
- [Appendix D, "DICOM Processing and Supported Formats"](#)
- [Appendix E, "DICOM Sample Applications"](#)
- [Appendix F, "Migrating from Release 10.2 DICOM Support"](#)

DICOM Configuration Documents

This appendix lists the names of the DICOM configuration documents. These documents are loaded into the DICOM data model repository when Oracle Multimedia DICOM is installed.

The DICOM configuration documents are as follows:

- Anonymity document (`ordcman.xml`)
- Constraint document (`ordcmct.xml`, `ordcmcmd.xml`, `ordcmcmc.xml`)
- Mapping document (`ordcmmp.xml`)
- Preference document (`ordcmpf.xml`)
- Private Dictionary document (`ordcmpv.xml`)
- Standard Dictionary document (`ordcmsd.xml`)
- Stored tag list document (optional)
- UID Definition document (`ordcmui.xml`)

The latest versions of these documents are available as files in the `ord/xml` directory under `<ORACLE_HOME>` when Oracle Multimedia DICOM is installed.

The DICOM configuration documents are associated with one or more DICOM XML schemas. See [Appendix B](#) for information about these XML schemas.

DICOM XML Schemas

This appendix lists the DICOM XML schemas used by the methods of the ORDDicom object type. When Oracle Multimedia DICOM is installed, these schemas are registered as global XML schemas in Oracle Database with Oracle XML DB (see [Section 9.2.1](#)).

Note: The schemas in this appendix might not match the code shipped with the Oracle installation. For the final versions of these schemas, use the files provided with the installation.

The latest versions of these schemas are available as files in the `ord/xml/xsd` directory under `<ORACLE_HOME>`. To locate and examine the schemas, query the dictionary view `ALL_XML_SCHEMAS` (see [Section 9.2.2](#)). In addition, read the documentation embedded within each schema file for more information.

The DICOM XML schemas are associated with one or more DICOM configuration documents. See [Appendix A](#) for a list of configuration documents.

This appendix includes these DICOM XML schemas:

- [Anonymity Document Schema](#) (`ordcman.xsd`) on page B-2
- [Constraint Document Schema](#) (`ordcmct.xsd`) on page B-4
- [Data Type Definition Schema](#) (`ordcmrdt.xsd`) on page B-12
- [Default DICOM Metadata Schema](#) (`ordcmmd.xsd`) on page B-27
- [Manifest Document Schema](#) (`ordcmmt.xsd`) on page B-27
- [Mapping Document Schema](#) (`ordcmmp.xsd`) on page B-29
- [Metadata Data Type Definition Schema](#) (`ordcmddt.xsd`) on page B-32
- [Preference Document Schema](#) (`ordcmpf.xsd`) on page B-47
- [Private Dictionary Document Schema](#) (`ordcmpv.xsd`) on page B-52
- [Standard Dictionary Document Schema](#) (`ordcmsd.xsd`) on page B-53
- [Stored Tag List Document Schema](#) (`ordcmstl.xsd`) on page B-55
- [UID Definition Document Schema](#) (`ordcmui.xsd`) on page B-56

See Also:

- *Oracle XML DB Developer's Guide* for more information about registering XML schemas
- <http://www.w3.org/XML/Schema> for more information about XML schemas
- *Oracle Database Reference* for more information about the dictionary view ALL_XML_SCHEMAS

B.1 Anonymity Document Schema

The anonymity document schema `ordcman.xsd`, shown in [Example B-1](#), defines the structure of the anonymity documents. The namespace for this schema is

`http://xmlns.oracle.com/ord/dicom/anonymity_1_0`

Example B-1 Anonymity Document Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
Copyright (c) 2007, Oracle. All rights reserved.

NAME
ordcman.xsd - XML schema for DICOM anonymity documents
-->

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://xmlns.oracle.com/ord/dicom/anonymity_1_0"
xmlns:dt="http://xmlns.oracle.com/ord/dicom/datatype_1_0"
targetNamespace="http://xmlns.oracle.com/ord/dicom/anonymity_1_0" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:import namespace="http://xmlns.oracle.com/ord/dicom/datatype_1_0"
schemaLocation="http://xmlns.oracle.com/ord/dicom/datatype_1_0"/>
  <xs:annotation>
    <xs:documentation>
      Introduction
      This schema defines the DICOM anonymity document.

      Structure Overview
      Question mark "?" means optional items.
      Plus "+" means one or more items.
      Asterisk "*" means zero or more items.

      ANONYMITY_RULE_DOCUMENT
      DOCUMENT_HEADER?
      DOCUMENT_CHANGE_LOG*
      DOCUMENT_MODIFIER
      DOCUMENT_MODIFICATION_DATE
      DOCUMENT_VERSION?
      MODIFICATION_COMMENT?
      BASE_DOCUMENT?
      BASE_DOCUMENT_RELEASE_DATE?
      BASE_DOCUMENT_DESCRIPTION?
      PRIVATE_ATTRIBUTES
      UNDEFINED_STANDARD_ATTRIBUTES
      UNDEFINED_PRIVATE_ATTRIBUTES
      INDIVIDUAL_ATTRIBUTE*

      The preceding element values specify the actions required to make
      a DICOM attribute, or a selected group of DICOM attributes,
      anonymous.

    </xs:documentation>
  </xs:annotation>
```

```

</xs:annotation>
<xs:element name="ANONYMITY_RULE_DOCUMENT">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="DOCUMENT_HEADER" type="dt:DOCUMENT_HEADER_T" nillable="true" minOccurs="0"/>
      <xs:element name="PRIVATE_ATTRIBUTES" type="ANONYM_G_T">
        <xs:annotation>
          <xs:documentation>
            Specify the action required to make all private
            attributes anonymous.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="UNDEFINED_STANDARD_ATTRIBUTES" type="ANONYM_G_T">
        <xs:annotation>
          <xs:documentation>
            Specify the action required to make all undefined
            standard attributes anonymous. Undefined standard
            attributes are not defined by the standard data dictionaries
            when makeAnonymous or isAnonymous functions are invoked.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="UNDEFINED_PRIVATE_ATTRIBUTES" type="ANONYM_G_T">
        <xs:annotation>
          <xs:documentation>
            Specify the action required to make all undefined private
            attributes anonymous. Undefined private attributes are
            not defined by the private data dictionaries when
            makeAnonymous or isAnonymous functions are invoked.
            This element takes priority over the previous
            element PRIVATE_ATTRIBUTES.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="INDIVIDUAL_ATTRIBUTE" minOccurs="0" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>
            Specify the action required to make an attribute anonymous.
            This element overwrites the group specifications
            specified in the preceding elements.
          </xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:sequence>
            <xs:element name="ATTRIBUTE_TAG" type="dt:VALUE_LOCATOR_T"/>
            <xs:element name="DESCRIPTION" type="dt:SHORT_TEXT_T" minOccurs="0"/>
            <xs:element name="ANONYMITY_ACTION" type="ANONYM_T" nillable="true"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:complexType name="ANONYM_T">
  <xs:simpleContent>
    <xs:annotation>
      <xs:documentation>
        The anonymity action type has an attribute action,
        which defines the action used to make an
        attribute anonymous.
        If the value of the action attribute is "none", no
        action will be taken.
        If the value of the action attribute is "remove", then
        the element does not require a value.(The default value
        of the action attribute is "remove").
      </xs:documentation>
    </xs:annotation>
  </xs:simpleContent>
</xs:complexType>

```

The selected candidate attribute will be removed from the DICOM object to make it anonymous.

If the value of the action attribute is "replace", then the string value encoded in the attribute will be cast into the corresponding type of the attribute and the new value replaces the original.

If the value of the action attribute is "empty" , then the attribute will be changed into zero length attribute (for future use only).

If the value of the action attribute is "encrypt", then the string value encoded in the action attribute will be replaced with an encrypted value (for future use only).

```

</xs:documentation>
</xs:annotation>
<xs:extension base="dt:SHORT_STRING_T">
  <xs:attribute name="action" default="remove">
    <xs:simpleType>
      <xs:restriction base="xs:token">
        <xs:enumeration value="none"/>
        <xs:enumeration value="remove"/>
        <xs:enumeration value="replace"/>
        <xs:enumeration value="empty"/>
        <xs:enumeration value="encrypt"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:extension>
</xs:simpleContent>
</xs:complexType>
<xs:complexType name="ANONYM_G_T">
  <xs:simpleContent>
    <xs:annotation>
      <xs:documentation>
        The anonymity action type for a group attribute is similar to
        ANONYM_T except that it does not allow "replace" action.
      </xs:documentation>
    </xs:annotation>
    <xs:extension base="dt:SHORT_STRING_T">
      <xs:attribute name="action" default="remove">
        <xs:simpleType>
          <xs:restriction base="xs:token">
            <xs:enumeration value="none"/>
            <xs:enumeration value="remove"/>
            <xs:enumeration value="empty"/>
            <xs:enumeration value="encrypt"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
</xs:schema>

```

B.2 Constraint Document Schema

The constraint document schema `ordcmct.xsd`, shown in [Example B-2](#), defines the structure of the constraint documents. The namespace for this schema is

```
http://xmlns.oracle.com/ord/dicom/constraint_1_0
```

Example B-2 Constraint Document Schema

```

<?xml version="1.0" encoding="UTF-8"?>
<!--
  Copyright (c) 2007, Oracle. All rights reserved.

```

```

NAME
  ordcmct.xsd - XML schema for DICOM constraint documents
-->

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="http://xmlns.oracle.com/ord/dicom/constraint_1_0"
  xmlns:ct="http://xmlns.oracle.com/ord/dicom/constraint_1_0"
  xmlns:dt="http://xmlns.oracle.com/ord/dicom/datatype_1_0"
  targetNamespace="http://xmlns.oracle.com/ord/dicom/constraint_1_0" elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xs:import namespace="http://xmlns.oracle.com/ord/dicom/datatype_1_0"
    schemaLocation="http://xmlns.oracle.com/ord/dicom/datatype_1_0"/>
  <xs:annotation>
    <xs:documentation>

```

Introduction

This schema defines the DICOM constraint document.

A DICOM constraint document defines rules to check the conformance of a DICOM content with respect to the DICOM standard and other organization-wide guidelines. This XML schema document defines the XML schema constraining constraint documents.

A constraint document defines one or more constraint rules. A constraint rule is the unit of invocation for conformance checking. At run time, a user may invoke a PL/SQL or Java function to check the conformance of a DICOM content with respect to a particular rule. Each invocable rule is defined as a GLOBAL_RULE, which can reference other global rules internally.

A constraint rule can be decomposed into individual predicates. A predicate can be a logical statement, a relational statement comparing values, a function call evaluation that returns a Boolean type, or a reference to other predicate definitions. Predicate definitions are recursive. A predicate can be a logical statement, which includes the logical OR of two other predicates. Each predicate can be a relational predicate. For example:
(patientName=="Joe Smith" AND patientSex=="M")

After being translated into a predicate, the preceding example becomes:

```

<PREDICATE>
  <DESCRIPTION>An example to find an object that has
    (patientName="Joe Smith" AND patientSex=="M")
  </DESCRIPTION>
  <LOGICAL operator="and">
    <PREDICATE>
      <RELATIONAL operator="eq">
        <DICOM_ATTRIBUTE>00100010</DICOM_ATTRIBUTE>
        <XML_VALUE>
          <dt:PERSON_NAME>
            <dt:NAME>
              <dt:FAMILY>Smith</dt:FAMILY>
              <dt:GIVEN>Joe</dt:GIVEN>
            </dt:NAME>
          </dt:PERSON_NAME>
        </XML_VALUE>
      </RELATIONAL>
    </PREDICATE>
    <PREDICATE>
      <RELATIONAL operator="eq">
        <DICOM_ATTRIBUTE>00100040</DICOM_ATTRIBUTE>
        <XML_VALUE>
          <dt:CODE_STRING>M</dt:CODE_STRING>
        </XML_VALUE>
      </RELATIONAL>

```

```

    </PREDICATE>
</LOGICAL>
</PREDICATE>

```

Constraint macros can be used to simplify the definition of complex constraint rules. Constraint macros follow the same predicate definition grammar as constraint rules. The operands in constraint macros can be variables rather than fixed values, as they are in constraint rules. The variables in a macro are substituted when the macro is invoked. For example, you can define a macro to compare patient names (patientName == \$NAME). When this macro is invoked, the parameter NAME is assigned the value "Joe Smith" and the macro is transformed into the predicate:(patientName == "Joe Smith"). As another example, you can define a macro to check if a DICOM attribute is a code sequence attribute. A code sequence attribute must contain the mandatory child attributes, code value and code scheme. This macro checks whether the specified code sequence attribute contains these mandatory child attributes.

```

<GLOBAL_MACRO name="CSMacro">
  <DESCRIPTION>
    A subset of Code Sequence Macro defined in DICOM standard,
    PS3.3-2007, Table 8.8-1
  </DESCRIPTION>
  <PARAMETER_DECLARATION>
    CodeAttr
  </PARAMETER_DECLARATION>
  <PREDICATE>
    <DESCRIPTION>Code value must not be empty</DESCRIPTION>
    <BOOLEAN_FUNC operator="notEmpty">
      <DICOM_ATTRIBUTE>${CodeAttr}.00080100
    </DICOM_ATTRIBUTE>
    </BOOLEAN_FUNC>
  </PREDICATE>
  <PREDICATE>
    <DESCRIPTION>Code scheme designator must not be empty
  </DESCRIPTION>
    <BOOLEAN_FUNC operator="notEmpty">
      <DICOM_ATTRIBUTE>${CodeAttr}.00080102
    </DICOM_ATTRIBUTE>
    </BOOLEAN_FUNC>
  </PREDICATE>
  <!-- other predicates follow -->
</GLOBAL_MACRO>

```

You can separate a constraint definition into multiple files. Each file defines one or more constraint rules or macros. A file can import the macros and constraint rules that are defined in a difference file. You must specify the set of external rules or macros before referencing them in a file. EXTERNAL_RULE_INCLUDE and EXTERNAL_MACRO_INCLUDE statements serve this purpose.

Structure Overview

Question mark "?" means optional items.
 Plus "+" means one or more items.
 Asterisk "*" means zero or more items.

```

CONFORMANCE_CONSTRAINT_DEFINITION
DOCUMENT_HEADER?
DOCUMENT_CHANGE_LOG+
DOCUMENT_MODIFIER
DOCUMENT_MODIFICATION_DATE
DOCUMENT_VERSION?

```

```

        MODIFICATION_COMMENT?
        BASE_DOCUMENT?
        BASE_DOCUMENT_RELEASE_DATE?
        BASE_DOCUMENT_DESCRIPTION?
EXTERNAL_MACRO_INCLUDE*
EXTERNAL_RULE_INCLUDE*
(GLOBAL_MACRO|GLOBAL_RULE)+

GLOBAL_RULE (name) | PREDICATE_DEFINITION (name)
    DESCRIPTION?
    PREDICATE_DEFINITION*
    PREDICATE+
    ACTION (when, action) *

GLOBAL_MACRO (name)
    DESCRIPTION?
    PARAMETER_DECLARATION+
    PREDICATE_DEFINITION*
    PREDICATE+
    ACTION (when, action) *

PREDICATE
    DESCRIPTION?
    (LOGICAL|RELATIONAL|BOOLEAN_FUNC|INVOKE_MACRO|PREDICATE_REF|GLOBAL_RULE_REF)
    ACTION (when, action) *

LOGICAL(operator)
    PREDICATE+

RELATIONAL(operator)
    (ATTRIBUTE_TAG|FUNCTION) (ATTRIBUTE_TAG|STRING_VALUE|XML_VALUE|FUNCTION) +

BOOLEAN_FUNC(operator)
    (ATTRIBUTE_TAG|STRING_VALUE|XML_VALUE|FUNCTION) *

INVOKE_MACRO
    MACRO_NAME
    PARAMETER+
        NAME
        VALUE

FUNCTION(operator)
    (ATTRIBUTE_TAG|STRING_VALUE|XML_VALUE|FUNCTION) *

</xs:documentation>
</xs:annotation>
<xs:element name="CONFORMANCE_CONSTRAINT_DEFINITION">
  <xs:annotation>
    <xs:documentation>A constraint document defines groups of predicates to validate the conformance
      of a DICOM content or a DICOM metadata document.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="DOCUMENT_HEADER" type="dt:DOCUMENT_HEADER_T" minOccurs="0"/>
      <xs:element name="EXTERNAL_MACRO_INCLUDE" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="dt:SHORT_TEXT_T">
              <xs:attribute name="name" type="dt:SHORT_ID_T" use="required"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
      <xs:element name="EXTERNAL_RULE_INCLUDE" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:simpleContent>

```

```

        <xs:extension base="dt:SHORT_TEXT_T">
            <xs:attribute name="name" type="dt:SHORT_ID_T" use="required"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
</xs:element>
<xs:choice maxOccurs="unbounded">
    <xs:element name="GLOBAL_MACRO" type="PREDICATE_MACRO_T">
        <xs:key name="LOCAL_PRED_KEY1">
            <xs:selector xpath="ct:PREDICATES_DEFINITION"/>
            <xs:field xpath="@name"/>
        </xs:key>
        <xs:keyref name="LOCAL_PREDICATE_REF1" refer="ct:LOCAL_PRED_KEY1">
            <xs:selector xpath="//ct:LOGICAL"/>
            <xs:field xpath="//ct:PREDICATE_REF"/>
        </xs:keyref>
    </xs:element>
    <xs:element name="GLOBAL_RULE" type="PREDICATE_GROUP_T">
        <xs:key name="LOCAL_PRED_KEY2">
            <xs:selector xpath="ct:PREDICATES_DEFINITION"/>
            <xs:field xpath="@name"/>
        </xs:key>
        <xs:keyref name="LOCAL_PREDICATE_REF2" refer="ct:LOCAL_PRED_KEY2">
            <xs:selector xpath="//ct:LOGICAL"/>
            <xs:field xpath="//ct:PREDICATE_REF"/>
        </xs:keyref>
    </xs:element>
</xs:choice>
</xs:sequence>
</xs:complexType>
<!-- predicate group defined under the root element is global -->
<xs:key name="GLOBAL_PRED1">
    <xs:selector xpath="ct:GLOBAL_RULE|ct:EXTERNAL_RULE_INCLUDE"/>
    <xs:field xpath="@name"/>
</xs:key>
<xs:keyref name="PREDICATE_REF1" refer="ct:GLOBAL_PRED1">
    <xs:selector xpath="//ct:LOGICAL"/>
    <xs:field xpath="ct:GLOBAL_RULE_REF"/>
</xs:keyref>
<xs:key name="GLOBAL_MACRO1">
    <xs:selector xpath="ct:GLOBAL_MACRO|ct:EXTERNAL_MACRO_INCLUDE"/>
    <xs:field xpath="@name"/>
</xs:key>
<xs:keyref name="MACRO_USE1" refer="ct:GLOBAL_MACRO1">
    <xs:selector xpath="//ct:INVOKE_MACRO"/>
    <xs:field xpath="ct:MACRONAME"/>
</xs:keyref>
</xs:element>
<xs:complexType name="PREDICATE_GROUP_T">
    <xs:annotation>
        <xs:documentation>A predicate group is the logical AND
            of a collection of predicates or predicate groups.
            Each predicate group has a name that is unique within
            its parent. Any other predicates can reference
            this predicate group by its name. The value of the reference
            is the Boolean of the predicate group.
            Optionally, a predicate group can contain a set of
            predicate definitions. These definitions are not part of the
            logical AND component of the predicate group, but they
            are meant to be referenced within the predicate group.
            A predicate group has an optional action element that
            specifies what action to take when the predicate evaluates to true
            or false.
        </xs:documentation>
    </xs:annotation>
</xs:sequence>

```



```

<xs:element name="DESCRIPTION" type="dt:SHORT_TEXT_T" minOccurs="0"/>
<xs:element name="PREDICATES_DEFINITION" type="PREDICATE_GROUP_T" minOccurs="0" maxOccurs="unbounded"/>
<xs:choice maxOccurs="unbounded">
  <xs:element name="PREDICATE" type="PREDICATE_T"/>
</xs:choice>
<xs:element name="ACTION" type="ACTION_T" nillable="true" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="name" type="dt:SHORT_ID_T"/>
</xs:complexType>
<xs:complexType name="PREDICATE_MACRO_T">
  <xs:annotation>
    <xs:documentation>
      </xs:documentation>
    </xs:annotation>
  <xs:sequence>
    <xs:element name="DESCRIPTION" type="dt:SHORT_TEXT_T" minOccurs="0"/>
    <xs:element name="PARAMETER_DECLARATION" type="dt:SHORT_NAME_T" nillable="false" maxOccurs="unbounded"/>
    <xs:element name="PREDICATES_DEFINITION" type="PREDICATE_GROUP_T" minOccurs="0" maxOccurs="unbounded"/>
    <xs:choice maxOccurs="unbounded">
      <xs:element name="PREDICATE" type="PREDICATE_T"/>
    </xs:choice>
    <xs:element name="ACTION" type="ACTION_T" nillable="true" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="name" type="dt:SHORT_ID_T"/>
</xs:complexType>
<xs:complexType name="ACTION_T">
  <xs:annotation>
    <xs:documentation>
      A type to specify an action for a predicate value.
      The "when" attribute specifies the predicate value.
      The "action" attribute specifies the type of action.
      When the action type is "log", "warning", or "error",
      the string value of this attribute is returned
      in a log file or as part of warning or error message.
    </xs:documentation>
  </xs:annotation>
  <xs:simpleContent>
    <xs:extension base="dt:SHORT_TEXT_T">
      <xs:attribute name="when" type="xs:boolean" use="required"/>
      <xs:attribute name="action" type="ACTION_LIST_T" use="required"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:simpleType name="ACTION_LIST_T">
  <xs:restriction base="xs:token">
    <xs:enumeration value="none"/>
    <xs:enumeration value="log"/>
    <xs:enumeration value="warning"/>
    <xs:enumeration value="error"/>
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="PREDICATE_T">
  <xs:sequence>
    <xs:element name="DESCRIPTION" type="dt:SHORT_TEXT_T" minOccurs="0"/>
    <xs:choice>
      <xs:element name="LOGICAL" type="LOGICAL_PREDICATE_T"/>
      <xs:element name="RELATIONAL" type="RELATIONAL_PREDICATE_T"/>
      <xs:element name="BOOLEAN_FUNC" type="BOOLEAN_FUNC_PREDICATE_T"/>
      <xs:element name="INVOKE_MACRO" type="MACRO_USE_T"/>
      <xs:element name="PREDICATE_REF" type="xs:IDREF"/>
      <xs:element name="GLOBAL_RULE_REF" type="xs:IDREF"/>
    </xs:choice>
    <xs:element name="ACTION" type="ACTION_T" nillable="true" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="MACRO_USE_T">

```

```

<xs:sequence>
  <xs:element name="MACRO_NAME" type="xs:IDREF"/>
  <xs:element name="PARAMETER" maxOccurs="unbounded">
    <xs:complexType>
      <xs:sequence maxOccurs="unbounded">
        <xs:element name="NAME" type="dt:SHORT_STRING_T" nillable="false"/>
        <xs:element name="VALUE" type="dt:SHORT_TEXT_T"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:sequence>
</xs:complexType>
<xs:complexType name="LOGICAL_PREDICATE_T">
  <xs:sequence maxOccurs="unbounded">
    <!--Boolean type, static inline predicate definition -->
    <xs:element name="PREDICATE" type="PREDICATE_T"/>
  </xs:sequence>
  <xs:attribute name="operator" use="required">
    <xs:simpleType>
      <xs:restriction base="xs:token">
        <xs:annotation>
          <xs:documentation>
            A derive B ( NOT A ) OR B )
          </xs:documentation>
        </xs:annotation>
        <xs:enumeration value="and"/>
        <xs:enumeration value="or"/>
        <xs:enumeration value="derive"/>
        <xs:enumeration value="not"/>
        <xs:enumeration value="xor"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:complexType>
<xs:complexType name="RELATIONAL_PREDICATE_T">
  <xs:sequence>
    <xs:choice>
      <xs:element name="ATTRIBUTE_TAG" type="dt:VALUE_LOCATOR_MACRO_T"/>
      <xs:element name="FUNCTION" type="FUNCTION_T"/>
    </xs:choice>
    <xs:choice maxOccurs="unbounded">
      <xs:element name="ATTRIBUTE_TAG" type="dt:VALUE_LOCATOR_MACRO_T"/>
      <xs:element name="STRING_VALUE" type="dt:MIXED_TEXT_T"/>
      <xs:element name="XML_VALUE" type="dt:ATTR_VALUE_T"/>
      <xs:element name="FUNCTION" type="FUNCTION_T"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="operator" use="required">
    <xs:simpleType>
      <xs:restriction base="xs:token">
        <xs:annotation>
          <xs:documentation>
            gt    greater than
            ge    greater than or equal to
            lt    less than
            le    less than or equal to
            eq    equal to
            ne    not equal to
            in    value in the set of
            match attribute value matches pattern
            The second operand must be a Java regular expression
            pattern as specified by JDK1.5 java.lang.String class
            documentation. The first operator should be a DICOM
            attribute tag. The tag should identify an attribute
            that belongs to one of the following value representation
            types:
          </xs:documentation>
        </xs:annotation>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>

```

```

        AE, AS, AT, CS, DA, DT, LO, LT, PN,
        SH, ST, TM, UI and UT
    Note that the operands must be compatible with each other
    when a predicate invokes relational operator. For example,
    (patientAge > 005M) is a valid predicate. But
    (patientAge > "Joe Smith") is not a valid predicate, because
    the operand "Joe Smith" cannot be cast into an instance
    of the patient age attribute.
    </xs:documentation>
</xs:annotation>
<xs:enumeration value="gt"/>
<xs:enumeration value="ge"/>
<xs:enumeration value="lt"/>
<xs:enumeration value="le"/>
<xs:enumeration value="eq"/>
<xs:enumeration value="ne"/>
<xs:enumeration value="in"/>
<xs:enumeration value="match"/>
</xs:restriction>
</xs:simpleType>
</xs:attribute>
</xs:complexType>
<xs:complexType name="BOOLEAN_FUNC_PREDICATE_T">
  <xs:choice maxOccurs="unbounded" minOccurs="0">
    <xs:element name="ATTRIBUTE_TAG" type="dt:VALUE_LOCATOR_MACRO_T"/>
    <xs:element name="STRING_VALUE" type="dt:MIXED_TEXT_T"/>
    <xs:element name="XML_VALUE" type="dt:ATTR_VALUE_T"/>
    <xs:element name="FUNCTION" type="FUNCTION_T"/>
  </xs:choice>
  <xs:attribute name="operator" use="required">
    <xs:annotation>
      <xs:documentation>
        To allow future extensions, the set of allowed operators for Boolean
        function types are not fixed. Operator names are case-sensitive.
        The current values for this operator
        are: "notEmpty", "occurs", "true", and "false".
        "occurs" takes a single operand ATTRIBUTE_TAG,
        and returns true if an attribute matching the tag exists. (The
        attribute value can be an empty string or null. For example,
        a DICOM type 2 attribute may be empty.); Otherwise, it returns
        false.
        "notEmpty" takes a single operand ATTRIBUTE_TAG.
        It returns true if an attribute matching the tag exists in
        a DICOM content and has a non-null value (e.g. a DICOM type 1
        attribute); otherwise, it returns false.
        "true" takes no operand and it always returns true.
        "false" takes no operand and it always returns false.
      </xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:simpleType>
    <xs:restriction base="xs:token">
      <xs:maxLength value="64"/>
    </xs:restriction>
  </xs:simpleType>
</xs:complexType>
</xs:complexType>
<xs:complexType name="FUNCTION_T">
  <xs:choice minOccurs="0" maxOccurs="unbounded">
    <xs:element name="ATTRIBUTE_TAG" type="dt:VALUE_LOCATOR_MACRO_T"/>
    <xs:element name="STRING_VALUE" type="dt:MIXED_TEXT_T"/>
    <xs:element name="XML_VALUE" type="dt:ATTR_VALUE_T"/>
    <xs:element name="FUNCTION" type="FUNCTION_T"/>
  </xs:choice>
  <xs:attribute name="operator" use="required">
    <xs:annotation>
      <xs:documentation>

```

```

        To allow future extensions, the set of allowed operators for
        function types are not fixed. Operator names are case-sensitive.
        This feature is not supported for Oracle Database 11g Release 1.
    </xs:documentation>
</xs:annotation>
<xs:simpleType>
    <xs:restriction base="xs:token">
        <xs:maxLength value="64"/>
    </xs:restriction>
</xs:simpleType>
</xs:attribute>
</xs:complexType>
</xs:schema>

```

B.3 Data Type Definition Schema

The schema `ordcmrtdt.xsd`, shown in [Example B-3](#), defines the DICOM standard data types. The namespace for this schema is

`http://xmlns.oracle.com/ord/dicom/datatype_1_0`

Example B-3 Data Type Definition Schema

```

<?xml version="1.0" encoding="UTF-8"?>
<!--
Copyright (c) 2007, 2009, Oracle and/or its affiliates. All rights reserved.

NAME
    ordcmrtdt.xsd - XML schema for DICOM standard data types.
-->

<xs:schema xmlns="http://xmlns.oracle.com/ord/dicom/datatype_1_0"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xdb="http://xmlns.oracle.com/xdb"
    targetNamespace="http://xmlns.oracle.com/ord/dicom/datatype_1_0"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:annotation>
        <xs:documentation>
            Introduction
            This schema defines the DICOM standard data types that are used
            by all other DICOM XML schema definitions.

            Naming conventions:
            All DICOM value representation (VR) types are named with a
            2-character string, such as "AE" and "CS".
            All DICOM attribute type definitions are named as VR_ATTR_T,
            where VR is replaced by the attribute's 2-character VR.

            Note that each item of a sequence type (SQ) is of DATASET_T type.
            The DATASET_T type can recursively contain more attributes.
            The element name of an attribute is its value representation (VR)
            name. Oracle uses value representation names defined
            by the DICOM standard part 5. The element
            name to VR mappings are:
            APPLICATION_ENTITY    ---  AE
            AGE_STRING            ---  AS
            ATTRIBUTE_TAG         ---  AT
            CODE_STRING           ---  CS
            DATE                  ---  DA
            DECIMAL_STRING        ---  DS
            FLOAT_SINGLE          ---  FL
            FLOAT_DOUBLE          ---  FD
            INTEGER_STRING        ---  IS
            LONG_STRING           ---  LO
            LONG_TEXT             ---  LT
        </xs:documentation>
    </xs:annotation>

```

```

OTHER_BYTE      ---  OB
OTHER_FLOAT     ---  OF
OTHER_WORD      ---  OW
OTHER_WORD      ---  OWB
PERSON_NAME     ---  PN
SHORT_STRING    ---  SH
SIGNED_LONG     ---  SL
SEQUENCE        ---  SQ
SIGNED_SHORT    ---  SS
SHORT_TEXT      ---  ST
TIME            ---  TM
UNIQUE_ID       ---  UI
UNSIGNED_LONG   ---  UL
UNKNOWN         ---  UN
UNSIGNED_SHORT  ---  US
SIGNED_SHORT    ---  USS
UNLIMITED_TEXT ---  UT
EXTENDED_TYPE   ---  EXT
EXCEPTION_TYPE  ---  EXP
The VR types "OWB", "EXT", "EXP" and "USS" are
Oracle-defined extensions.
Please refer to the individual data type documentation for
more explanation.
</xs:documentation>
</xs:annotation>
<xs:simpleType name="AE">
  <xs:annotation>
    <xs:documentation>DICOM Value representation Application Entity</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:token">
    <xs:maxLength value="16"/>
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="AS">
  <xs:annotation>
    <xs:documentation>DICOM Value representation Age String.
    The age string can be expressed either in DICOM string
    format, or in number of days. When metadata is extracted
    from a DICOM object, both elements will be populated.
    XML documents can represent age by either format.
    Age in number of days is converted into an age string when
    XML metadata is encoded into a DICOM object.
    To convert from age string into the number of days:
      365 * number_of_year or 31 * number_of_month.
    Because AGE_STRING is mandatory, it is not necessary to
    convert from the number of days into an age string.
    </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="VALUE" nillable="true">
      <xs:simpleType>
        <xs:restriction base="xs:token">
          <xs:pattern value="[0-9]{3}(D|W|M|Y)"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="AGE_IN_DAYS" type="xs:unsignedInt" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:simpleType name="AT">
  <xs:annotation>
    <xs:documentation>
    DICOM VR type Attribute Tag. An attribute tag is expressed as two
    big-endian 2-byte hexadecimal number (group number followed by
    element number with no separator).
    </xs:documentation>
  </xs:annotation>

```

```
</xs:annotation>
<xs:restriction base="xs:hexBinary">
  <xs:minLength value="4"/>
  <xs:maxLength value="4"/>
</xs:restriction>
</xs:simpleType>
<xs:simpleType name="CS">
  <xs:annotation>
    <xs:documentation>DICOM VR type Code String</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:token">
    <xs:maxLength value="16"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="DA">
  <xs:annotation>
    <xs:documentation>DICOM VR type Date</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:date"/>
</xs:simpleType>
<xs:simpleType name="DS">
  <xs:annotation>
    <xs:documentation>DICOM VR type Decimal String</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:float"/>
</xs:simpleType>
<xs:simpleType name="DT">
  <xs:annotation>
    <xs:documentation>DICOM VR type Data Time</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:dateTime"/>
</xs:simpleType>
<xs:simpleType name="FL">
  <xs:annotation>
    <xs:documentation>DICOM VR type Floating-point single</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:float"/>
</xs:simpleType>
<xs:simpleType name="FD">
  <xs:annotation>
    <xs:documentation>DICOM VR type Floating-point Double</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:double"/>
</xs:simpleType>
<xs:simpleType name="IS">
  <xs:annotation>
    <xs:documentation>DICOM VR type Integer String</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:integer"/>
</xs:simpleType>
<xs:simpleType name="LO">
  <xs:annotation>
    <xs:documentation>DICOM VR type Long string</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:maxLength value="64"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="LT">
  <xs:annotation>
    <xs:documentation>DICOM VR type Long Text</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:maxLength value="10240"/>
  </xs:restriction>
</xs:simpleType>
```

```

<xs:simpleType name="OB">
  <xs:annotation>
    <xs:documentation>DICOM VR type Other Byte</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:base64Binary"/>
</xs:simpleType>
<xs:simpleType name="OF">
  <xs:annotation>
    <xs:documentation> VR type Other Float </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:float"/>
</xs:simpleType>
<xs:complexType name="OW">
  <xs:annotation>
    <xs:documentation>
      DICOM VR type Other Word in base64binary encoding.
      The mandatory attribute endian specifies the byte
      order of the binary value.
    </xs:documentation>
  </xs:annotation>
  <xs:simpleContent>
    <xs:extension base="xs:base64Binary">
      <xs:attribute name="endian" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:token">
            <xs:enumeration value="big"/>
            <xs:enumeration value="little"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="PN">
  <xs:annotation>
    <xs:documentation>
      DICOM VR type Person Name. Person Name can be
      expressed either in component format or as a single
      concatenated string. When metadata is extracted from a
      DICOM object, the person name type is encoded with
      both formats. Users can index and search DICOM
      metadata with either the component format or the
      concatenated string format.
      In component format, a name has an optional "type" attribute that
      indicates its encoding type. The value of the "type" attribute
      can be "unibyte", "ideographic" or "phonetic". A name may
      have up to five components: "FAMILY", "GIVEN", "MIDDLE",
      "PREFIX", and "SUFFIX".
    </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="NAME" minOccurs="0" maxOccurs="3" nillable="true">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="FAMILY" type="xs:string" minOccurs="0" nillable="true"/>
          <xs:element name="GIVEN" type="xs:string" minOccurs="0" nillable="true"/>
          <xs:element name="MIDDLE" type="xs:string" minOccurs="0" nillable="true"/>
          <xs:element name="PREFIX" type="xs:string" minOccurs="0" nillable="true"/>
          <xs:element name="SUFFIX" type="xs:string" minOccurs="0" nillable="true"/>
        </xs:sequence>
        <xs:attribute name="type" default="unibyte">
          <xs:simpleType>
            <xs:restriction base="xs:token">
              <xs:enumeration value="unibyte"/>
              <xs:enumeration value="ideographic"/>
              <xs:enumeration value="phonetic"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:complexType>
    </xs:element>
  </xs:sequence>

```

```

        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>
<xs:element name="VALUE" minOccurs="0" nillable="true">
  <xs:simpleType>
    <xs:restriction base="xs:token">
      <xs:maxLength value="64"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:simpleType name="SH">
  <xs:annotation>
    <xs:documentation>DICOM VR type SHort string</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:maxLength value="16"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="SL">
  <xs:annotation>
    <xs:documentation>DICOM VR type Signed Long</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:integer"/>
</xs:simpleType>
<xs:complexType name="SQ">
  <xs:annotation>
    <xs:documentation>DICOM VR type SeQuence.
      Note that item number can be explicitly encoded in XML.
      Number counts from 1 up.
      Each item is a DATASET_T type, which may contain
      any combination of DICOM attributes.
    </xs:documentation>
  </xs:annotation>
  <xs:sequence maxOccurs="unbounded">
    <xs:element name="ITEM" type="DATASET_T" minOccurs="0" nillable="true"/>
  </xs:sequence>
</xs:complexType>
<xs:simpleType name="SS">
  <xs:annotation>
    <xs:documentation>DICOM VR type Signed Short</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:integer"/>
</xs:simpleType>
<xs:simpleType name="ST">
  <xs:annotation>
    <xs:documentation>DICOM VR type Short Text</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:maxLength value="1024"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="TM">
  <xs:annotation>
    <xs:documentation>DICOM VR type TiMe</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:time"/>
</xs:simpleType>
<xs:simpleType name="UI">
  <xs:annotation>
    <xs:documentation>DICOM VR type Unique Identifier</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:token">

```



```

    <xs:maxLength value="128"/>
    <xs:pattern value="[0-9\.]+"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="UL">
  <xs:annotation>
    <xs:documentation>DICOM VR type Unsigned Long</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:unsignedInt"/>
</xs:simpleType>
<xs:complexType name="UN">
  <xs:annotation>
    <xs:documentation>
      DICOM VR type UNknown.
      This type contains a base64 dump of its binary content. The mandatory
      attribute "endian" specifies the byte order of this encoding.
    </xs:documentation>
  </xs:annotation>
  <xs:simpleContent>
    <xs:extension base="xs:base64Binary">
      <xs:attribute name="endian" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:token">
            <xs:enumeration value="big"/>
            <xs:enumeration value="little"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:simpleType name="US">
  <xs:annotation>
    <xs:documentation>DICOM VR type Unsigned Short</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:unsignedShort"/>
</xs:simpleType>
<xs:simpleType name="UT">
  <xs:annotation>
    <xs:documentation>DICOM VR type Unlimited Text</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string"/>
</xs:simpleType>
<xs:complexType name="EXT">
  <xs:annotation>
    <xs:documentation>DICOM Extension type
      This type does not have direct mapping to any value
      representation (VR) types defined in Part 5 of the
      DICOM standard.
      It can accommodate future extensions to DICOM VR
      types without modification to the XML schema definitions.
      The VR element specifies the value representation.
      The VALUE element specifies the XML value for the
      corresponding data element. The exact XML schema
      definition can be introduced in the future.
    </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="VR" type="xs:token"/>
    <xs:element name="VALUE" type="xs:anyType" nillable="true"/>
  </xs:sequence>
</xs:complexType>
<xs:simpleType name="EXP">
  <xs:annotation>
    <xs:documentation>DICOM Exception type.
      This type does not have direct mapping to any value

```

```

representation (VR) types defined in Part 5 of the
DICOM standard.
It indicates an error situation. It is equivalent to
an exception in the Java language.
The value of this data type is the original byte
array of the data type in the DICOM object.
</xs:documentation>
</xs:annotation>
<xs:restriction base="xs:base64Binary"/>
</xs:simpleType>
<xs:complexType name="DATASET_T">
  <xs:annotation>
    <xs:documentation>
      The dataset type maps the DICOM concept dataset
      into an XML schema type(See the DICOM standard P3-5) .
      A dataset may contain any number of DICOM attributes.
      Each type of attribute has a name that reflects
      the DICOM value representation of the attribute.
      Each attribute is strongly typed, and its type matches its DICOM
      VR. Note that DICOM attribute type SQ (sequence) may
      recursively contain items that are also of the dataset type.
    </xs:documentation>
  </xs:annotation>
  <xs:choice maxOccurs="unbounded" minOccurs="0">
    <xs:element name="APPLICATION_ENTITY" type="AE_ATTR_T" nillable="true"/>
    <xs:element name="AGE_STRING" type="AS_ATTR_T" nillable="true"/>
    <xs:element name="ATTRIBUTE_TAG" type="AT_ATTR_T" nillable="true"/>
    <xs:element name="CODE_STRING" type="CS_ATTR_T" nillable="true"/>
    <xs:element name="DATE" type="DA_ATTR_T" nillable="true"/>
    <xs:element name="DATE_TIME" type="DT_ATTR_T" nillable="true"/>
    <xs:element name="DECIMAL_STRING" type="DS_ATTR_T" nillable="true"/>
    <xs:element name="FLOAT_SINGLE" type="FL_ATTR_T" nillable="true"/>
    <xs:element name="FLOAT_DOUBLE" type="FD_ATTR_T" nillable="true"/>
    <xs:element name="INTEGER_STRING" type="IS_ATTR_T" nillable="true"/>
    <xs:element name="LONG_STRING" type="LO_ATTR_T" nillable="true"/>
    <xs:element name="LONG_TEXT" type="LT_ATTR_T" nillable="true"/>
    <xs:element name="OTHER_BYTE" type="OB_ATTR_T" nillable="true"/>
    <xs:element name="OTHER_FLOAT" type="OF_ATTR_T" nillable="true"/>
    <xs:element name="OTHER_WORD" type="OW_ATTR_T" nillable="true"/>
    <xs:element name="PERSON_NAME" type="PN_ATTR_T" nillable="true"/>
    <xs:element name="SHORT_STRING" type="SH_ATTR_T" nillable="true"/>
    <xs:element name="SIGNED_LONG" type="SL_ATTR_T" nillable="true"/>
    <xs:element name="SEQUENCE" type="SQ_ATTR_T" nillable="true" xdb:SQLType="CLOB"/>
    <xs:element name="SIGNED_SHORT" type="SS_ATTR_T" nillable="true"/>
    <xs:element name="SHORT_TEXT" type="ST_ATTR_T" nillable="true"/>
    <xs:element name="TIME" type="TM_ATTR_T" nillable="true"/>
    <xs:element name="UNIQUE_ID" type="UI_ATTR_T" nillable="true"/>
    <xs:element name="UNSIGNED_LONG" type="UL_ATTR_T" nillable="true"/>
    <xs:element name="UNKNOWN" type="UN_ATTR_T" nillable="true"/>
    <xs:element name="UNSIGNED_SHORT" type="US_ATTR_T" nillable="true"/>
    <xs:element name="UNLIMITED_TEXT" type="UT_ATTR_T" nillable="true"/>
    <xs:element name="EXTENDED_TYPE" type="EXT_ATTR_T" nillable="true"/>
    <xs:element name="EXCEPTION_TYPE" type="EXP_ATTR_T" nillable="true"/>
  </xs:choice>
  <xs:attribute name="number" type="xs:long" use="optional" default="1"/>
</xs:complexType>
<xs:complexType name="ATTR_VALUE_T">
  <xs:annotation>
    <xs:documentation>
      Attribute value type (ATTR_VALUE_T) maps to a single DICOM
      attribute value. Each type of attribute has a name that reflects
      the DICOM value representation of the attribute.
      Each attribute is strongly typed, and its type matches its DICOM
      VR. Certain DICOM configuration files, such as constraint
      documents, use ATTR_VALUE_T.
    </xs:documentation>
  </xs:annotation>

```

```

</xs:annotation>
<xs:choice>
  <xs:element name="APPLICATION_ENTITY" type="AE"/>
  <xs:element name="AGE_STRING" type="AS"/>
  <xs:element name="ATTRIBUTE_TAG" type="AT"/>
  <xs:element name="CODE_STRING" type="CS"/>
  <xs:element name="DATE" type="DA"/>
  <xs:element name="DATE_TIME" type="DT"/>
  <xs:element name="DECIMAL_STRING" type="DS"/>
  <xs:element name="FLOAT_SINGLE" type="FL"/>
  <xs:element name="FLOAT_DOUBLE" type="FD"/>
  <xs:element name="INTEGER_STRING" type="IS"/>
  <xs:element name="LONG_STRING" type="LO"/>
  <xs:element name="LONG_TEXT" type="LT"/>
  <xs:element name="OTHER_BYTE" type="OB"/>
  <xs:element name="OTHER_FLOAT" type="OF"/>
  <xs:element name="OTHER_WORD" type="OW"/>
  <xs:element name="PERSON_NAME" type="PN"/>
  <xs:element name="SHORT_STRING" type="SH"/>
  <xs:element name="SIGNED_LONG" type="SL"/>
  <xs:element name="SEQUENCE" type="SQ"/>
  <xs:element name="SIGNED_SHORT" type="SS"/>
  <xs:element name="SHORT_TEXT" type="ST"/>
  <xs:element name="TIME" type="TM"/>
  <xs:element name="UNIQUE_ID" type="UI"/>
  <xs:element name="UNSIGNED_LONG" type="UL"/>
  <xs:element name="UNKNOWN" type="UN"/>
  <xs:element name="UNSIGNED_SHORT" type="US"/>
  <xs:element name="UNLIMITED_TEXT" type="UT"/>
  <xs:element name="EXTENDED_TYPE" type="EXT"/>
  <xs:element name="EXCEPTION_TYPE" type="EXP"/>
</xs:choice>
</xs:complexType>
<xs:attributeGroup name="ATTR_GRP_T">
  <xs:annotation>
    <xs:documentation>
      Attribute group type (ATTR_GRP_T) is used by all DICOM attribute
      definitions. It defines XML attributes that are used by all DICOM
      attribute types.
      The "tag" attribute defines DICOM attributes in little-endian encoding.
      The "definer" attribute specifies the organization that has
      created the attribute. By default, all DICOM standard
      attributes have the definer name "DICOM".
      The "name" attribute specifies the canonical attribute name
      as defined by the data dictionary. For example, in
      an XML metadata schema definition, you can choose a tag
      PATIENT_DATE_OF_BIRTH or "DOB" for DICOM attribute
      (0010,0030), but its name attribute should match that of the
      DICOM standard: "Patient's Birth Date".
      The "number" attribute is an optional attribute to indicate the
      ordering of a multivalued attributes. Number counts from 1 up.
      The "truncated" attribute takes a Boolean value. If it is true,
      it indicates that the original length of the DICOM attribute
      exceeds the maximum length allowed for this XML value;therefore,
      it is truncated in XML. When this attribute is true,
      xsi:nil="true" for this attribute.
      Optionally, the "rawValue" attribute can be used to store
      values that do not conform to the DICOM standard. The
      associated attribute "byteOrderLE" specifies the byte order
      of the byte stream for the "rawValue" attribute.
      "offset" and "length" are Oracle-reserved attributes.
    </xs:documentation>
  </xs:annotation>
  <xs:attribute name="tag" type="AT" use="required"/>
  <xs:attribute name="definer" type="LO" default="DICOM"/>
  <xs:attribute name="name" type="SHORT_STRING_T"/>

```

```
<xs:attribute name="number" type="xs:long" use="optional" default="1"/>
<xs:attribute name="offset" type="xs:long"/>
<xs:attribute name="length" type="xs:long"/>
<xs:attribute name="truncated" type="xs:boolean" default="false"/>
<xs:attribute name="rawValue" type="xs:base64Binary"/>
<xs:attribute name="byteOrderLE" type="xs:boolean" default="true"/>
</xs:attributeGroup>
<xs:complexType name="AE_ATTR_T">
  <xs:simpleContent>
    <xs:extension base="AE">
      <xs:attributeGroup ref="ATTR_GRP_T" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="AS_ATTR_T">
  <xs:complexContent>
    <xs:extension base="AS">
      <xs:attributeGroup ref="ATTR_GRP_T" />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="AT_ATTR_T">
  <xs:simpleContent>
    <xs:extension base="AT">
      <xs:attributeGroup ref="ATTR_GRP_T" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="CS_ATTR_T">
  <xs:simpleContent>
    <xs:extension base="CS">
      <xs:attributeGroup ref="ATTR_GRP_T" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="DA_ATTR_T">
  <xs:simpleContent>
    <xs:extension base="DA">
      <xs:attributeGroup ref="ATTR_GRP_T" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="DS_ATTR_T">
  <xs:simpleContent>
    <xs:extension base="DS">
      <xs:attributeGroup ref="ATTR_GRP_T" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="DT_ATTR_T">
  <xs:simpleContent>
    <xs:extension base="DT">
      <xs:attributeGroup ref="ATTR_GRP_T" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="FD_ATTR_T">
  <xs:simpleContent>
    <xs:extension base="FD">
      <xs:attributeGroup ref="ATTR_GRP_T" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="FL_ATTR_T">
  <xs:simpleContent>
    <xs:extension base="FL">
```

```

        <xs:attributeGroup ref="ATTR_GRP_T" />
    </xs:extension>
</xs:simpleContent>
</xs:complexType>
<xs:complexType name="IS_ATTR_T">
    <xs:simpleContent>
        <xs:extension base="IS">
            <xs:attributeGroup ref="ATTR_GRP_T" />
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:complexType name="LO_ATTR_T">
    <xs:simpleContent>
        <xs:extension base="LO">
            <xs:attributeGroup ref="ATTR_GRP_T" />
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:complexType name="LT_ATTR_T">
    <xs:simpleContent>
        <xs:extension base="LT">
            <xs:attributeGroup ref="ATTR_GRP_T" />
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:complexType name="OB_ATTR_T">
    <xs:simpleContent>
        <xs:extension base="OB">
            <xs:attributeGroup ref="ATTR_GRP_T" />
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:complexType name="OF_ATTR_T">
    <xs:simpleContent>
        <xs:extension base="OF">
            <xs:attributeGroup ref="ATTR_GRP_T" />
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:complexType name="OW_ATTR_T">
    <xs:simpleContent>
        <xs:extension base="OW">
            <xs:attributeGroup ref="ATTR_GRP_T" />
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:complexType name="PN_ATTR_T">
    <xs:complexContent>
        <xs:extension base="PN">
            <xs:attributeGroup ref="ATTR_GRP_T" />
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:complexType name="SH_ATTR_T">
    <xs:simpleContent>
        <xs:extension base="SH">
            <xs:attributeGroup ref="ATTR_GRP_T" />
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:complexType name="SL_ATTR_T">
    <xs:simpleContent>
        <xs:extension base="SL">
            <xs:attributeGroup ref="ATTR_GRP_T" />
        </xs:extension>
    </xs:simpleContent>

```

```
</xs:complexType>
<xs:complexType name="SQ_ATTR_T">
  <xs:complexContent>
    <xs:extension base="SQ">
      <xs:attributeGroup ref="ATTR_GRP_T" />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="SS_ATTR_T">
  <xs:simpleContent>
    <xs:extension base="SS">
      <xs:attributeGroup ref="ATTR_GRP_T" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="ST_ATTR_T">
  <xs:simpleContent>
    <xs:extension base="ST">
      <xs:attributeGroup ref="ATTR_GRP_T" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="TM_ATTR_T">
  <xs:simpleContent>
    <xs:extension base="TM">
      <xs:attributeGroup ref="ATTR_GRP_T" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="UI_ATTR_T">
  <xs:simpleContent>
    <xs:extension base="UI">
      <xs:attributeGroup ref="ATTR_GRP_T" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="UL_ATTR_T">
  <xs:simpleContent>
    <xs:extension base="UL">
      <xs:attributeGroup ref="ATTR_GRP_T" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="UN_ATTR_T">
  <xs:simpleContent>
    <xs:extension base="UN">
      <xs:attributeGroup ref="ATTR_GRP_T" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="US_ATTR_T">
  <xs:simpleContent>
    <xs:extension base="US">
      <xs:attributeGroup ref="ATTR_GRP_T" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="UT_ATTR_T">
  <xs:simpleContent>
    <xs:extension base="UT">
      <xs:attributeGroup ref="ATTR_GRP_T" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="EXT_ATTR_T">
  <xs:annotation>
```

```

    <xs:documentation>
      This attribute is useful for representing attributes whose
      VR types are not supported natively by Oracle.
    </xs:documentation>
  </xs:annotation>
</xs:complexContent>
<xs:extension base="EXT">
  <xs:attributeGroup ref="ATTR_GRP_T" />
</xs:extension>
</xs:complexContent>
</xs:complexType>
<xs:complexType name="EXP_ATTR_T">
  <xs:annotation>
    <xs:documentation>
      This attribute type is useful for representing attributes that
      are present in a DICOM object, but whose definition cannot
      be found in the data dictionary. Such
      attributes cannot be parsed or interpreted.
    </xs:documentation>
  </xs:annotation>
  <xs:simpleContent>
    <xs:extension base="EXP">
      <xs:attributeGroup ref="ATTR_GRP_T" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="DOCUMENT_HEADER_T">
  <xs:annotation>
    <xs:documentation>
      Each time the XML configuration document is modified,
      a new element, DOCUMENT_CHANGE_LOG, is
      added to the DOCUMENT_HEADER.
      The change log describes who made what type of change to the
      XML document on which date. It also describes what DICOM
      standard document the modification is based upon, either
      a DICOM change proposal (CP) or a DICOM supplement.

      DOCUMENT_MODIFIER identifies the modifier of the present
      XML document. If it is generated by software, specify the name
      and version of the software.
      DOCUMENT_MODIFICATION_DATE specifies the date when
      this XML document is modified.
      DOCUMENT_VERSION specifies the version of the document after
      the modification.
      MODIFICATION_COMMENT briefly describes the modification.
      BASE_DOCUMENT describes the document or DICOM standard
      that the modification is based upon.
      BASE_DOCUMENT_RELEASE_DATE specifies the release date of
      the base document.
      BASE_DOCUMENT_DESCRIPTION briefly describes the base
      document.
    </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="DOCUMENT_CHANGE_LOG" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="DOCUMENT_MODIFIER" type="SHORT_STRING_T" />
          <xs:element name="DOCUMENT_MODIFICATION_DATE" type="SHORT_STRING_T" />
          <xs:element name="DOCUMENT_VERSION" type="SHORT_STRING_T" minOccurs="0" />
          <xs:element name="MODIFICATION_COMMENT" type="SHORT_TEXT_T" minOccurs="0" />
          <xs:element name="BASE_DOCUMENT" type="SHORT_STRING_T" minOccurs="0" />
          <xs:element name="BASE_DOCUMENT_RELEASE_DATE" type="xs:date" minOccurs="0" />
          <xs:element name="BASE_DOCUMENT_DESCRIPTION" type="SHORT_TEXT_T" minOccurs="0" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

```

```

    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="ATTR_DEFINERS_T">
  <xs:annotation>
    <xs:documentation>
      Attribute definer is identified by its name and UID.
      In Oracle's implementation, the DICOM standard is given the
      definer name "DICOM" and the UID "1.2.840.10008.1".
      All DICOM standard attributes are given the definer name "DICOM".
      Users can introduce private attributes of their own and encode them
      in an XML document. These private attributes are identified
      with the definer's name and UID. Oracle recommends that all DICOM
      private attributes be associated with a UID-qualified name.
    </xs:documentation>
  </xs:annotation>
  <xs:sequence maxOccurs="unbounded">
    <xs:element name="ATTR_DEFINER">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="NAME" type="LO" maxOccurs="unbounded"/>
          <xs:element name="UID" type="UI" minOccurs="0"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<!-- Attribute Tag (allowing x wildcard)-->
<xs:simpleType name="ATTR_TAG_T">
  <xs:annotation>
    <xs:documentation>
      The attribute tag type differs from DICOM VR
      type AT in that it allows the wildcard character 'x'.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:token">
    <xs:pattern value="([0-9a-fA-FxX]{8})"/>
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="ATTR_RANGE_T">
  <xs:annotation>
    <xs:documentation>
      The attribute range type defines a range of DICOM attributes.
      This data type is used in private attribute definitions.
      Certain private attributes accept a range of attribute tags.
    </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="STARTING_TAG" type="ATTR_TAG_T"/>
    <xs:element name="ENDING_TAG" type="ATTR_TAG_T"/>
  </xs:sequence>
</xs:complexType>
<xs:simpleType name="VALUE_LOCATOR_T">
  <xs:annotation>
    <xs:documentation>
      The DICOM value locator type identifies a particular
      DICOM attribute by "xxxxxxx(definer)", where
      "xxxxxxx" is the attribute tag and "definer" is the
      attribute definer, which can be the DICOM standard
      (DICOM) or other private sources.
      A locator path can also identify a particular
      descendent of a container type attribute (SQ).
      The n-th item of a sequence attribute is denoted by
      "xxxxxxx(definer)[n]".
      By default, the definer suffix "(definer)" can be
      omitted if the attribute is a DICOM standard tag.
    </xs:documentation>
  </xs:annotation>

```



```

    The index "n" of an item address "[n]" must be a
    positive integer. The item address suffix can be
    omitted if the item it pointed to is the first item
    of a sequence.
    For example, 00080096.00401101.00080100 is the code
    that identifies the first referring physician. The
    above value locator is equivalent to:
        00080096(DICOM)[1].00401101(DICOM)[1].00080100(DICOM)
</xs:documentation>
</xs:annotation>
<xs:restriction base="VALUE_LOCATOR_MACRO_T"/>
</xs:simpleType>
<xs:simpleType name="VALUE_LOCATOR_MACRO_T">
  <xs:annotation>
    <xs:documentation>
      VALUE_LOCATOR_MACRO_T is similar to the value locator
      type, except that it permits the use of a macro within
      the locator string.
      So, the macro locator string can be:
          ${TAG}(DICOM)[2].00080100
      This string indicates the code value (0008,0100) of the second
      item of a sequence attribute identified by ${TAG}.
      The macro parameter TAG can be replaced by a
      compatible attribute tag (code sequence attribute)
      later.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="SHORT_TEXT_T"/>
</xs:simpleType>
<xs:simpleType name="VM_T">
  <xs:annotation>
    <xs:documentation>
      DICOM value multiplicity (VM) specification.
      This type is used in DICOM dictionary documents.
      Patterns of valid specifications are:
      "k", "k-j", "k-n", "n", "k-kn".
      In these patterns, k and j are integers, k is less
      than j, and n is the letter n.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="SHORT_STRING_T">
    <xs:pattern value="([0-9]+-)?([0-9]*n|([0-9]+))"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="VR_T">
  <xs:annotation>
    <xs:documentation>
      DICOM value representation types.
      In the DICOM standard, VR for certain attributes
      is defined as "other word or byte", "US or SS", or
      "See Note". Oracle has extended the list of VR types and
      introduced OWB (for "other word or byte"),
      USS (for "US or SS"), and
      EXP (where VR definition does not apply).
      When an attribute of USS type is encoded into XML, it is
      automatically encoded as a signed short type.
      When an attribute of OWB type is encoded into XML, it is
      automatically encoded into other word type.
      An example of an attribute with VR type of EXP is
      the sequence item (FFFE, E000).
      For compatibility with future DICOM releases, if a new
      DICOM VR is introduced by the DICOM standard,
      users can mark such attributes as type "EXT??",
      where "??" should be replaced by the new VR name.
    </xs:documentation>
  </xs:annotation>

```

```
<xs:restriction base="xs:token">
  <xs:pattern value="AE"/>
  <xs:pattern value="AS"/>
  <xs:pattern value="AT"/>
  <xs:pattern value="CS"/>
  <xs:pattern value="DA"/>
  <xs:pattern value="DS"/>
  <xs:pattern value="DT"/>
  <xs:pattern value="FL"/>
  <xs:pattern value="FD"/>
  <xs:pattern value="IS"/>
  <xs:pattern value="LO"/>
  <xs:pattern value="LT"/>
  <xs:pattern value="OB"/>
  <xs:pattern value="OF"/>
  <xs:pattern value="OW"/>
  <xs:pattern value="PN"/>
  <xs:pattern value="SH"/>
  <xs:pattern value="SL"/>
  <xs:pattern value="SQ"/>
  <xs:pattern value="SS"/>
  <xs:pattern value="ST"/>
  <xs:pattern value="TM"/>
  <xs:pattern value="UI"/>
  <xs:pattern value="UL"/>
  <xs:pattern value="UN"/>
  <xs:pattern value="US"/>
  <xs:pattern value="UT"/>
  <xs:pattern value="USS"/>
  <xs:pattern value="OWB"/>
  <xs:pattern value="EXP"/>
  <xs:pattern value="EXT[A-Z]{2}"/>
</xs:restriction>
</xs:simpleType>
<xs:simpleType name="SHORT_TEXT_T">
  <xs:restriction base="xs:token">
    <xs:maxLength value="1999"/>
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="MIXED_TEXT_T" mixed="true">
  <xs:complexContent mixed="true">
    <xs:extension base="xs:anyType">
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
<xs:simpleType name="SHORT_STRING_T">
  <xs:restriction base="xs:token">
    <xs:maxLength value="128"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="SHORT_NAME_T">
  <xs:restriction base="xs:NCName">
    <xs:maxLength value="128"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="SHORT_ID_T">
  <xs:restriction base="xs:ID">
    <xs:maxLength value="64"/>
    <xs:pattern value="^[^\.]+"/>
  </xs:restriction>
</xs:simpleType>
</xs:schema>
```

B.4 Default DICOM Metadata Schema

The schema `ordcmmd.xsd`, shown in [Example B-4](#), defines the default DICOM metadata schema. The namespace for this schema is

`http://xmlns.oracle.com/ord/dicom/metadata_1_0`

Example B-4 Default DICOM Metadata Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
Copyright (c) 2007, Oracle. All rights reserved.

NAME
ordcmmd.xsd - XML schema for default DICOM metadata documents
-->

<xs:schema xmlns="http://xmlns.oracle.com/ord/dicom/metadata_1_0"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:dt="http://xmlns.oracle.com/ord/dicom/metadata_1_0"
targetNamespace="http://xmlns.oracle.com/ord/dicom/metadata_1_0" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:include schemaLocation="http://xmlns.oracle.com/ord/dicom/mddatatype_1_0"/>
  <xs:annotation>
    <xs:documentation>
      Introduction
      This schema defines the default DICOM metadata schema used
      by the ORDDicom object attribute (XMLType metadata).
    </xs:documentation>
  </xs:annotation>
  <xs:element name="DICOM_OBJECT" type="dt:DATASET_T"/>
</xs:schema>
```

B.5 Manifest Document Schema

The manifest document schema `ordcmfft.xsd`, shown in [Example B-5](#), defines the structure of the manifest documents, which are created when exporting a set of configuration documents from a DICOM repository. The manifest document specifies the name of each configuration document, its document type, and the load order to be used when importing the configuration document into a DICOM repository. The default manifest document is `ordcmfft.xml`. The namespace for this schema is

`http://xmlns.oracle.com/ord/dicom/manifest_1_0`

Example B-5 Manifest Document Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="http://xmlns.oracle.com/ord/dicom/manifest_1_0"
xmlns:dt="http://xmlns.oracle.com/ord/dicom/datatype_1_0"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://xmlns.oracle.com/ord/dicom/manifest_1_0" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:import namespace="http://xmlns.oracle.com/ord/dicom/datatype_1_0"
schemaLocation="http://xmlns.oracle.com/ord/dicom/datatype_1_0"/>

  <xs:annotation>
    <xs:documentation>
      This schema defines a DICOM manifest file. This file specifies the
      document name, type, and the order in which the configuration
      documents are loaded into the DICOM data model repository.
      This manifest file is created by the exportDataModel procedure
      and is used by the importDataModel procedure in the data model
      repository (ord_dicom_admin) API.
    </xs:documentation>
  </xs:annotation>
</xs:schema>
```

The DOCUMENT_VERSION in the DOCUMENT_HEADER represents the repository version. This value identifies a set of configuration documents in the repository. This value is for future use when we may maintain and export multiple versions of the data model repository.

```

DICOM_MANIFEST
  DOCUMENT_HEADER?
    DOCUMENT_CHANGE_LOG*
    DOCUMENT_MODIFIER
    DOCUMENT_MODIFICATION_DATE
    DOCUMENT_VERSION?
    MODIFICATION_COMMENT?
    BASE_DOCUMENT?
    BASE_DOCUMENT_RELEASE_DATE?
    BASE_DOCUMENT_DESCRIPTION?

  DOCUMENT_DEF+
    NAME
    TYPE
    LOAD_ORDER

</xs:documentation>
</xs:annotation>

<xs:simpleType name="DOCUMENT_NAME_T">
  <xs:annotation>
    <xs:documentation>
      Name of a document in the dicom data model repository.
      This is the file name of the XML document that has been
      exported from the repository or is being loaded into
      the repository.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:maxLength value="100" />
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="DOCUMENT_TYPE_T">
  <xs:annotation>
    <xs:documentation>
      The document types supported in the data model repository.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:pattern value="STANDARD_DICTIONARY" />
    <xs:pattern value="PRIVATE_DICTIONARY" />
    <xs:pattern value="MAPPING" />
    <xs:pattern value="ANONYMITY" />
    <xs:pattern value="PREFERENCE" />
    <xs:pattern value="CONSTRAINT" />
    <xs:pattern value="UID_DEFINITION" />
    <xs:pattern value="STORED_TAG_LIST" />
  </xs:restriction>
</xs:simpleType>

<xs:element name="DICOM_MANIFEST">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="DOCUMENT_HEADER" type="dt:DOCUMENT_HEADER_T" nillable="true" minOccurs="0"/>
      <xs:element name="DOCUMENT_DEF" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>
            Each document definition defines a document that to be

```

```

        inserted into or exported from the data model repository.
    </xs:documentation>
</xs:annotation>
<xs:complexType>
  <xs:sequence>
    <xs:element name="NAME" type="DOCUMENT_NAME_T"/>
    <xs:element name="TYPE" type="DOCUMENT_TYPE_T"/>
    <xs:element name="LOAD_ORDER" type="xs:integer"/>
  </xs:sequence>
</xs:complexType>
<xs:unique name="docName">
  <xs:selector xpath="NAME"/>
  <xs:field xpath="."/>
</xs:unique>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>

```

B.6 Mapping Document Schema

The mapping document schema `ordcmmmp.xsd`, shown in [Example B-6](#), defines the structure of the mapping documents. The namespace for this schema is

```
http://xmlns.oracle.com/ord/dicom/mapping_1_0
```

Example B-6 Mapping Document Schema

```

<?xml version="1.0" encoding="UTF-8"?>
<!--
Copyright (c) 2007, Oracle. All rights reserved.

NAME
ordcmmmp.xsd - XML schema for DICOM mapping documents
-->

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="http://xmlns.oracle.com/ord/dicom/mapping_1_0"
  xmlns:dt="http://xmlns.oracle.com/ord/dicom/datatype_1_0"
  targetNamespace="http://xmlns.oracle.com/ord/dicom/mapping_1_0" elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xs:import namespace="http://xmlns.oracle.com/ord/dicom/datatype_1_0"
    schemaLocation="http://xmlns.oracle.com/ord/dicom/datatype_1_0"/>
  <xs:annotation>
    <xs:documentation>
      This schema defines the DICOM (XML) mapping document.
      It defines how each DICOM attribute maps to an element of the
      DICOM metadata document.

      The mapping document is used by the metadata encoder to produce
      a DICOM metadata document. Each DICOM attribute is identified by
      a 4-byte hexadecimal attribute tag. Each DICOM attribute is mapped
      to an element of the XML metadata document designated by the PATH
      element. By default, a DICOM attribute can be null and is optional.

      XML_MAPPING_DOCUMENT
      Question mark "?" means optional items.
      Plus "+" means one or more items.
      Asterisk "*" means zero or more items.

      DOCUMENT_HEADER?
      DOCUMENT_CHANGE_LOG*
    </xs:documentation>
  </xs:annotation>

```

```

DOCUMENT_MODIFIER
DOCUMENT_MODIFICATION_DATE
DOCUMENT_VERSION?
MODIFICATION_COMMENT?
BASE_DOCUMENT?
BASE_DOCUMENT_RELEASE_DATE?
BASE_DOCUMENT_DESCRIPTION?
NAMESPACE?
ROOT_ELEM_TAG
UNMAPPED_ELEM
MAPPED_ELEM
MAPPED_PATH+ (occurs?, notEmpty?, writeTag?, writeDefiner?, writeName?, writeRawValue)
  {ATTRIBUTE_TAG(definer), PATH}+

</xs:documentation>
</xs:annotation>
<xs:element name="XML_MAPPING_DOCUMENT">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="DOCUMENT_HEADER" type="dt:DOCUMENT_HEADER_T" minOccurs="0"/>
      <xs:element name="NAMESPACE" type="dt:SHORT_TEXT_T">
        <xs:annotation>
          <xs:documentation>
            The namespace of the XML metadata schema on which a mapping
            document is based. Metadata from a DICOM object can be
            mapped into an XML document constrained by this XML
            metadata schema. If the value of this element is an empty string,
            the extracted XML metadata document is not
            associated with an XML schema.
            The order of the MAPPED_PATH elements
            MUST match the sequence of the corresponding XML
            elements in this namespace.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="ROOT_ELEM_TAG" type="dt:SHORT_STRING_T">
        <xs:annotation>
          <xs:documentation>
            This element specifies the root element tag of
            an XML metadata document.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="UNMAPPED_ELEM" type="dt:SHORT_STRING_T" nillable="true" minOccurs="0">
        <xs:annotation>
          <xs:documentation>
            This element specifies the XML path (appended to
            ROOT_ELEM_TAG) for unmapped attributes, that is, the set
            of DICOM attributes that are present in a DICOM object,
            but whose mappings have not been defined by the
            MAPPED_PATH elements of an XML mapping document.
            This element is optional. If this element is omitted or empty,
            the unmapped attributes are appended to ROOT_ELEM_TAG.
            If an XML schema is used to constrain the metadata document,
            the XML schema element pointed to by this element should
            be of type dt:DATASET_T. See the DICOM data type definition
            schema "http://xmlns.oracle.com/ord/dicom/datatype_1_0"
            and dt:DATASET_T for more information.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="MAPPED_ELEM" type="dt:SHORT_STRING_T" nillable="true" minOccurs="0">
        <xs:annotation>
          <xs:documentation>
            This element specifies the XML path for all mapped
            attributes, that is, the set of DICOM attributes that are

```

```

present in a DICOM object, and whose mappings
are defined by the MAPPED_XPATH elements
of an XML mapping document. This element
specifies a relative path from ROOT_ELEM_TAG.
For example, to map a DICOM attribute (0010,0010) to
the XML element at "/DICOM_METADATA/PATIENT/NAME",
specify the following
The ROOT_ELEM_TAG element value is "DICOM_METADATA".
The MAPPED_ELEM element value is "PATIENT" and
The MAPPED_PATH/PATH element value should be "NAME".
Alternatively,
if the value of element MAPPED_ELEM is an empty string,
then the value of the element MAPPED_PATH/PATH
should be "PATIENT/NAME".
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="MAPPED_PATH" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:annotation>
        <xs:documentation>
          A MAPPED_PATH element contains attribute tag and
          path pairs.
          An attribute tag uniquely identifies an attribute within the
          data dictionary. Wildcards are not allowed in an attribute
          tag specification in this release.
          The path consists of slash "/"-concatenated element names.
          A path specifies the destination of an attribute in the
          DICOM XML metadata document. The mapped path
          is the relative path from ROOT_ELEM_TAG and
          MAPPED_ELEM. The absolute path is:
          "${ROOT_ELEM_TAG} / ${MAPPED_ELEM} /
          ${MAPPED_PATH}".
          The optional attribute "occurs" specifies whether the
          attribute must exist in the original DICOM content.
          (The attribute tag must exist, but the attribute value
          can be an empty string, for example, a DICOM type 2
          attribute.)
          The optional attribute "notEmpty" specifies
          whether the attribute must have a value in
          the original DICOM content (type 1 in DICOM terms).
          Depending on the run-time preferences, if the
          above "occurs" or "notEmpty" condition is not
          met, an error may be thrown at run-time.
          The optional attribute "writeTag" specifies whether to
          add the attribute "tag" when writing the element.
          The tag attribute is of type "dt:AT". The value of this
          attribute is the DICOM attribute tag in little-endian
          encoding.
          The optional attribute "writeDefiner" specifies whether
          to add the attribute "definer" when writing the element.
          The definer attribute is of type "dt:LO". The value of
          this attribute is the same as the definer attribute of
          ATTRIBUTE_TAG element of the mapping document.
          The optional attribute "writeName" specifies whether
          to add the attribute "name" when writing the element.
          The name attribute is of type "dt:SHORT_STRING_T".
          The value of this element is the attribute name
          defined by the data dictionary.
          The optional attribute "writeRawValue" specifies whether
          to add the attribute "rawValue" when writing the element.
          The raw value attribute is of type "xs:hexBinary".
          This attribute only occurs when there is a parsing error
          for this attribute and no XML value can be extracted for
          the element. The value of this attribute is the

```

```

        hexadecimal dump of the original byte stream.
    </xs:documentation>
</xs:annotation>
<xs:element name="ATTRIBUTE_TAG" type="dt:VALUE_LOCATOR_T"/>
<xs:element name="PATH" type="dt:SHORT_TEXT_T"/>
</xs:sequence>
<xs:attribute name="occurs" type="xs:boolean" default="false"/>
<xs:attribute name="notEmpty" type="xs:boolean" default="false"/>
<xs:attribute name="writeTag" type="xs:boolean" default="false"/>
<xs:attribute name="writeDefiner" type="xs:boolean" default="false"/>
<xs:attribute name="writeName" type="xs:boolean" default="false"/>
<xs:attribute name="writeRawValue" type="xs:boolean" default="false"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>

```

B.7 Metadata Data Type Definition Schema

The schema `ordcmmddt.xsd`, shown in [Example B-7](#), defines the metadata data types that are used by DICOM metadata schemas. The namespace for this schema is

`http://xmlns.oracle.com/ord/dicom/metadata_1_0`

Example B-7 Data Type Definition Schema

```

<?xml version="1.0" encoding="UTF-8"?>
<!--
Copyright (c) 2007, 2009, Oracle and/or its affiliates.All rights reserved.

NAME
    ordcmmddt.xsd - XML schema for metadata data types
-->

<xs:schema xmlns="http://xmlns.oracle.com/ord/dicom/metadata_1_0"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xdb="http://xmlns.oracle.com/xdb"
  targetNamespace="http://xmlns.oracle.com/ord/dicom/metadata_1_0"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:annotation>
    <xs:documentation>
      Introduction
      This schema defines the data types that are used
      by DICOM metadata schemas.

      Naming conventions:
      All DICOM value representation (VR) types are named with a
      2-character string, such as "AE" and "CS".
      All DICOM attribute type definitions are named as VR_ATTR_T,
      where VR is replaced by the attribute's 2-character VR.

      Note that each item of a sequence type (SQ) is of DATASET_T type.
      The DATASET_T type can recursively contain more attributes.
      The element name of an attribute is its value representation (VR)
      name. Oracle uses value representation names defined
      by the DICOM standard part 5. The element
      name to VR mappings are:
      APPLICATION_ENTITY    --- AE
      AGE_STRING            --- AS
      ATTRIBUTE_TAG         --- AT
      CODE_STRING           --- CS
      DATE                  --- DA
      DECIMAL_STRING        --- DS
    </xs:documentation>
  </xs:annotation>

```



```

FLOAT_SINGLE    --- FL
FLOAT_DOUBLE    --- FD
INTEGER_STRING  --- IS
LONG_STRING     --- LO
LONG_TEXT       --- LT
OTHER_BYTE      --- OB
OTHER_FLOAT     --- OF
OTHER_WORD      --- OW
OTHER_WORD      --- OWB
PERSON_NAME     --- PN
SHORT_STRING    --- SH
SIGNED_LONG     --- SL
SEQUENCE        --- SQ
SIGNED_SHORT    --- SS
SHORT_TEXT      --- ST
TIME            --- TM
UNIQUE_ID       --- UI
UNSIGNED_LONG   --- UL
UNKNOWN         --- UN
UNSIGNED_SHORT  --- US
SIGNED_SHORT    --- USS
UNLIMITED_TEXT --- UT
EXTENDED_TYPE   --- EXT
EXCEPTION_TYPE  --- EXP
The VR types "OWB", "EXT", "EXP" and "USS" are
Oracle-defined extensions.
Please refer to the individual data type documentation for
more explanation.
</xs:documentation>
</xs:annotation>
<xs:simpleType name="AE">
  <xs:annotation>
    <xs:documentation>DICOM Value representation Application Entity</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:token">
    <xs:maxLength value="16"/>
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="AS">
  <xs:annotation>
    <xs:documentation>DICOM Value representation Age String.
    The age string can be expressed either in DICOM string
    format, or in number of days. When metadata is extracted
    from a DICOM object, both elements will be populated.
    XML documents can represent age by either format.
    Age in number of days is converted into an age string when
    XML metadata is encoded into a DICOM object.
    To convert from age string into the number of days:
    365 * number_of_year or 31 * number_of_month.
    Because AGE_STRING is mandatory, it is not necessary to
    convert from the number of days into an age string.
    </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="VALUE" nillable="true">
      <xs:simpleType>
        <xs:restriction base="xs:token">
          <xs:pattern value="[0-9]{3}(D|W|M|Y)"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="AGE_IN_DAYS" type="xs:unsignedInt" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:simpleType name="AT">
  <xs:annotation>

```

```

    <xs:documentation>
      DICOM VR type Attribute Tag. An attribute tag is expressed as two
      big-endian 2-byte hexadecimal number (group number followed by
      element number with no separator).
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:hexBinary">
    <xs:minLength value="4"/>
    <xs:maxLength value="4"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="CS">
  <xs:annotation>
    <xs:documentation>DICOM VR type Code String</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:token">
    <xs:maxLength value="16"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="DA">
  <xs:annotation>
    <xs:documentation>DICOM VR type DATE</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:date"/>
</xs:simpleType>
<xs:simpleType name="DS">
  <xs:annotation>
    <xs:documentation>DICOM VR type Decimal String</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:float"/>
</xs:simpleType>
<xs:simpleType name="DT">
  <xs:annotation>
    <xs:documentation>DICOM VR type Data Time</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:dateTime"/>
</xs:simpleType>
<xs:simpleType name="FL">
  <xs:annotation>
    <xs:documentation>DICOM VR type Floating-point single</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:float"/>
</xs:simpleType>
<xs:simpleType name="FD">
  <xs:annotation>
    <xs:documentation>DICOM VR type Floating-point Double</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:double"/>
</xs:simpleType>
<xs:simpleType name="IS">
  <xs:annotation>
    <xs:documentation>DICOM VR type Integer String</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:integer"/>
</xs:simpleType>
<xs:simpleType name="LO">
  <xs:annotation>
    <xs:documentation>DICOM VR type Long string</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:maxLength value="64"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="LT">
  <xs:annotation>
    <xs:documentation>DICOM VR type Long Text</xs:documentation>

```

```

</xs:annotation>
<xs:restriction base="xs:string">
  <xs:maxLength value="10240"/>
</xs:restriction>
</xs:simpleType>
<xs:simpleType name="OB">
  <xs:annotation>
    <xs:documentation>DICOM VR type Other Byte</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:base64Binary"/>
</xs:simpleType>
<xs:simpleType name="OF">
  <xs:annotation>
    <xs:documentation> VR type Other Float </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:float"/>
</xs:simpleType>
<xs:complexType name="OW">
  <xs:annotation>
    <xs:documentation>
      DICOM VR type Other Word in base64binary encoding.
      The mandatory attribute endian specifies the byte
      order of the binary value.
    </xs:documentation>
  </xs:annotation>
  <xs:simpleContent>
    <xs:extension base="xs:base64Binary">
      <xs:attribute name="endian" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:token">
            <xs:enumeration value="big"/>
            <xs:enumeration value="little"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="PN">
  <xs:annotation>
    <xs:documentation>
      DICOM VR type Person Name. Person Name can be
      expressed either in component format or as a single
      concatenated string. When metadata is extracted from a
      DICOM object, the person name type is encoded with
      both formats. Users can index and search DICOM
      metadata with either the component format or the
      concatenated string format.
      In component format, a name has an optional "type" attribute that
      indicates its encoding type. The value of the "type" attribute
      can be "unibyte", "ideographic" or "phonetic". A name may
      have up to five components: "FAMILY", "GIVEN", "MIDDLE",
      "PREFIX", and "SUFFIX".
    </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="NAME" minOccurs="0" maxOccurs="3" nillable="true">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="FAMILY" type="xs:string" minOccurs="0" nillable="true"/>
          <xs:element name="GIVEN" type="xs:string" minOccurs="0" nillable="true"/>
          <xs:element name="MIDDLE" type="xs:string" minOccurs="0" nillable="true"/>
          <xs:element name="PREFIX" type="xs:string" minOccurs="0" nillable="true"/>
          <xs:element name="SUFFIX" type="xs:string" minOccurs="0" nillable="true"/>
        </xs:sequence>
        <xs:attribute name="type" default="unibyte">

```

```

        <xs:simpleType>
            <xs:restriction base="xs:token">
                <xs:enumeration value="unibyte"/>
                <xs:enumeration value="ideographic"/>
                <xs:enumeration value="phonetic"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
</xs:complexType>
</xs:element>
<xs:element name="VALUE" minOccurs="0" nillable="true">
    <xs:simpleType>
        <xs:restriction base="xs:token">
            <xs:maxLength value="64"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:simpleType name="SH">
    <xs:annotation>
        <xs:documentation>DICOM VR type Short string</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
        <xs:maxLength value="16"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="SL">
    <xs:annotation>
        <xs:documentation>DICOM VR type Signed Long</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:integer"/>
</xs:simpleType>
<xs:complexType name="SQ">
    <xs:annotation>
        <xs:documentation>
            DICOM VR type Sequence.
            Note that item number can be explicitly encoded in XML.
            Number counts from 1 up.
            Each item is a DATASET_T type, which may contain
            any combination of DICOM attributes.
        </xs:documentation>
    </xs:annotation>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
        <xs:element name="ITEM" type="DATASET_T" minOccurs="0" nillable="true"/>
    </xs:sequence>
</xs:complexType>
<xs:simpleType name="SS">
    <xs:annotation>
        <xs:documentation>DICOM VR type Signed Short</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:integer"/>
</xs:simpleType>
<xs:simpleType name="ST">
    <xs:annotation>
        <xs:documentation>DICOM VR type Short Text</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
        <xs:maxLength value="1024"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="TM">
    <xs:annotation>
        <xs:documentation>DICOM VR type Time</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:time"/>

```

```

</xs:simpleType>
<xs:simpleType name="UI">
  <xs:annotation>
    <xs:documentation>DICOM VR type Unique Identifier</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:token">
    <xs:maxLength value="128"/>
    <xs:pattern value="[0-9\.\.]+"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="UL">
  <xs:annotation>
    <xs:documentation>DICOM VR type Unsigned Long</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:unsignedInt"/>
</xs:simpleType>
<xs:complexType name="UN">
  <xs:annotation>
    <xs:documentation>
      DICOM VR type UNknown.
      This type contains a base64 dump of its binary content. The mandatory
      attribute "endian" specifies the byte order of this encoding.
    </xs:documentation>
  </xs:annotation>
  <xs:simpleContent>
    <xs:extension base="xs:base64Binary">
      <xs:attribute name="endian" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:token">
            <xs:enumeration value="big"/>
            <xs:enumeration value="little"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:simpleType name="US">
  <xs:annotation>
    <xs:documentation>DICOM VR type Unsigned Short</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:unsignedShort"/>
</xs:simpleType>
<xs:simpleType name="UT">
  <xs:annotation>
    <xs:documentation>DICOM VR type Unlimited Text</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string"/>
</xs:simpleType>
<xs:complexType name="EXT">
  <xs:annotation>
    <xs:documentation>DICOM Extension type
      This type does not have direct mapping to any value
      representation (VR) types defined in Part 5 of the
      DICOM standard.
      It can accommodate future extensions to DICOM VR
      types without modification to the XML schema definitions.
      The VR element specifies the value representation.
      The VALUE element specifies the XML value for the
      corresponding data element. The exact XML schema
      definition can be introduced in the future.
    </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="VR" type="xs:token"/>
    <xs:element name="VALUE" type="xs:anyType" nillable="true"/>
  </xs:sequence>
</xs:complexType>

```

```

</xs:sequence>
</xs:complexType>
<xs:simpleType name="EXP">
  <xs:annotation>
    <xs:documentation>DICOM Exception type.
      This type does not have direct mapping to any value
      representation (VR) types defined in Part 5 of the
      DICOM standard.
      It indicates an error situation. It is equivalent to
      an exception in the Java language.
      The value of this data type is the original byte
      array of the data type in the DICOM object.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:base64Binary"/>
</xs:simpleType>
<xs:complexType name="DATASET_T">
  <xs:annotation>
    <xs:documentation>
      The dataset type maps the DICOM concept dataset
      into an XML schema type(See the DICOM standard P3-5) .
      A dataset may contain any number of DICOM attributes.
      Each type of attribute has a name that reflects
      the DICOM value representation of the attribute.
      Each attribute is strongly typed, and its type matches its DICOM
      VR. Note that DICOM attribute type SQ (sequence) may
      recursively contain items that are also of the dataset type.
    </xs:documentation>
  </xs:annotation>
  <xs:choice maxOccurs="unbounded" minOccurs="0">
    <xs:element name="APPLICATION_ENTITY" type="AE_ATTR_T" nillable="true"/>
    <xs:element name="AGE_STRING" type="AS_ATTR_T" nillable="true"/>
    <xs:element name="ATTRIBUTE_TAG" type="AT_ATTR_T" nillable="true"/>
    <xs:element name="CODE_STRING" type="CS_ATTR_T" nillable="true"/>
    <xs:element name="DATE" type="DA_ATTR_T" nillable="true"/>
    <xs:element name="DATE_TIME" type="DT_ATTR_T" nillable="true"/>
    <xs:element name="DECIMAL_STRING" type="DS_ATTR_T" nillable="true"/>
    <xs:element name="FLOAT_SINGLE" type="FL_ATTR_T" nillable="true"/>
    <xs:element name="FLOAT_DOUBLE" type="FD_ATTR_T" nillable="true"/>
    <xs:element name="INTEGER_STRING" type="IS_ATTR_T" nillable="true"/>
    <xs:element name="LONG_STRING" type="LO_ATTR_T" nillable="true"/>
    <xs:element name="LONG_TEXT" type="LT_ATTR_T" nillable="true"/>
    <xs:element name="OTHER_BYTE" type="OB_ATTR_T" nillable="true"/>
    <xs:element name="OTHER_FLOAT" type="OF_ATTR_T" nillable="true"/>
    <xs:element name="OTHER_WORD" type="OW_ATTR_T" nillable="true"/>
    <xs:element name="PERSON_NAME" type="PN_ATTR_T" nillable="true"/>
    <xs:element name="SHORT_STRING" type="SH_ATTR_T" nillable="true"/>
    <xs:element name="SIGNED_LONG" type="SL_ATTR_T" nillable="true"/>
    <xs:element name="SEQUENCE" type="SQ_ATTR_T" nillable="true"
      xdb:SQLType="CLOB" xdb:SQLName="SEQUENCE"/>
    <xs:element name="SIGNED_SHORT" type="SS_ATTR_T" nillable="true"/>
    <xs:element name="SHORT_TEXT" type="ST_ATTR_T" nillable="true"/>
    <xs:element name="TIME" type="TM_ATTR_T" nillable="true"/>
    <xs:element name="UNIQUE_ID" type="UI_ATTR_T" nillable="true"/>
    <xs:element name="UNSIGNED_LONG" type="UL_ATTR_T" nillable="true"/>
    <xs:element name="UNKNOWN" type="UN_ATTR_T" nillable="true"/>
    <xs:element name="UNSIGNED_SHORT" type="US_ATTR_T" nillable="true"/>
    <xs:element name="UNLIMITED_TEXT" type="UT_ATTR_T" nillable="true"/>
    <xs:element name="EXTENDED_TYPE" type="EXT_ATTR_T" nillable="true"/>
    <xs:element name="EXCEPTION_TYPE" type="EXP_ATTR_T" nillable="true"/>
  </xs:choice>
  <xs:attribute name="number" type="xs:long" use="optional" default="1"/>
</xs:complexType>
<xs:complexType name="ATTR_VALUE_T">
  <xs:annotation>
    <xs:documentation>

```

```

Attribute value type (ATTR_VALUE_T) maps to a single DICOM
attribute value. Each type of attribute has a name that reflects
the DICOM value representation of the attribute.
Each attribute is strongly typed, and its type matches its DICOM
VR. Certain DICOM configuration files, such as constraint
documents, use ATTR_VALUE_T.
</xs:documentation>
</xs:annotation>
<xs:choice>
  <xs:element name="APPLICATION_ENTITY" type="AE"/>
  <xs:element name="AGE_STRING" type="AS"/>
  <xs:element name="ATTRIBUTE_TAG" type="AT"/>
  <xs:element name="CODE_STRING" type="CS"/>
  <xs:element name="DATE" type="DA"/>
  <xs:element name="DATE_TIME" type="DT"/>
  <xs:element name="DECIMAL_STRING" type="DS"/>
  <xs:element name="FLOAT_SINGLE" type="FL"/>
  <xs:element name="FLOAT_DOUBLE" type="FD"/>
  <xs:element name="INTEGER_STRING" type="IS"/>
  <xs:element name="LONG_STRING" type="LO"/>
  <xs:element name="LONG_TEXT" type="LT"/>
  <xs:element name="OTHER_BYTE" type="OB"/>
  <xs:element name="OTHER_FLOAT" type="OF"/>
  <xs:element name="OTHER_WORD" type="OW"/>
  <xs:element name="PERSON_NAME" type="PN"/>
  <xs:element name="SHORT_STRING" type="SH"/>
  <xs:element name="SIGNED_LONG" type="SL"/>
  <xs:element name="SEQUENCE" type="SQ"/>
  <xs:element name="SIGNED_SHORT" type="SS"/>
  <xs:element name="SHORT_TEXT" type="ST"/>
  <xs:element name="TIME" type="TM"/>
  <xs:element name="UNIQUE_ID" type="UI"/>
  <xs:element name="UNSIGNED_LONG" type="UL"/>
  <xs:element name="UNKNOWN" type="UN"/>
  <xs:element name="UNSIGNED_SHORT" type="US"/>
  <xs:element name="UNLIMITED_TEXT" type="UT"/>
  <xs:element name="EXTENDED_TYPE" type="EXT"/>
  <xs:element name="EXCEPTION_TYPE" type="EXP"/>
</xs:choice>
</xs:complexType>
<xs:attributeGroup name="ATTR_GRP_T">
  <xs:annotation>
    <xs:documentation>
      Attribute group type (ATTR_GRP_T) is used by all DICOM attribute
      definitions. It defines XML attributes that are used by all DICOM
      attribute types.
      The "tag" attribute defines DICOM attributes in little-endian encoding.
      The "definer" attribute specifies the organization that has
      created the attribute. By default, all DICOM standard
      attributes have the definer name "DICOM".
      The "name" attribute specifies the canonical attribute name
      as defined by the data dictionary. For example, in
      an XML metadata schema definition, you can choose a tag
      PATIENT_DATE_OF_BIRTH or "DOB" for DICOM attribute
      (0010,0030), but its name attribute should match that of the
      DICOM standard: "Patient's Birth Date".
      The "number" attribute is an optional attribute to indicate the
      ordering of a multivalued attributes. Number counts from 1 up.
      The "truncated" attribute takes a Boolean value. If it is true,
      it indicates that the original length of the DICOM attribute
      exceeds the maximum length allowed for this XML value; therefore,
      it is truncated in XML. When this attribute is true,
      xsi:nil="true" for this attribute.
      Optionally, the "rawValue" attribute can be used to store
      values that do not conform to the DICOM standard. The
      associated attribute "byteOrderLE" specifies the byte order

```

```

        of the byte stream for the "rawValue" attribute.
        "offset" and "length" are Oracle-reserved attributes.
    </xs:documentation>
</xs:annotation>
<xs:attribute name="tag" type="AT" use="required"/>
<xs:attribute name="definer" type="LO" default="DICOM"/>
<xs:attribute name="name" type="SHORT_STRING_T"/>
<xs:attribute name="number" type="xs:long" use="optional" default="1"/>
<xs:attribute name="offset" type="xs:long"/>
<xs:attribute name="length" type="xs:long"/>
<xs:attribute name="truncated" type="xs:boolean" default="false"/>
<xs:attribute name="rawValue" type="xs:base64Binary"/>
<xs:attribute name="byteOrderLE" type="xs:boolean" default="true"/>
</xs:attributeGroup>
<xs:complexType name="AE_ATTR_T">
    <xs:simpleContent>
        <xs:extension base="AE">
            <xs:attributeGroup ref="ATTR_GRP_T" />
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:complexType name="AS_ATTR_T">
    <xs:complexContent>
        <xs:extension base="AS">
            <xs:attributeGroup ref="ATTR_GRP_T" />
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:complexType name="AT_ATTR_T">
    <xs:simpleContent>
        <xs:extension base="AT">
            <xs:attributeGroup ref="ATTR_GRP_T" />
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:complexType name="CS_ATTR_T">
    <xs:simpleContent>
        <xs:extension base="CS">
            <xs:attributeGroup ref="ATTR_GRP_T" />
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:complexType name="DA_ATTR_T">
    <xs:simpleContent>
        <xs:extension base="DA">
            <xs:attributeGroup ref="ATTR_GRP_T" />
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:complexType name="DS_ATTR_T">
    <xs:simpleContent>
        <xs:extension base="DS">
            <xs:attributeGroup ref="ATTR_GRP_T" />
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:complexType name="DT_ATTR_T">
    <xs:simpleContent>
        <xs:extension base="DT">
            <xs:attributeGroup ref="ATTR_GRP_T" />
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:complexType name="FD_ATTR_T">
    <xs:simpleContent>
        <xs:extension base="FD">

```



```

        <xs:attributeGroup ref="ATTR_GRP_T" />
    </xs:extension>
</xs:simpleContent>
</xs:complexType>
<xs:complexType name="FL_ATTR_T">
    <xs:simpleContent>
        <xs:extension base="FL">
            <xs:attributeGroup ref="ATTR_GRP_T" />
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:complexType name="IS_ATTR_T">
    <xs:simpleContent>
        <xs:extension base="IS">
            <xs:attributeGroup ref="ATTR_GRP_T" />
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:complexType name="LO_ATTR_T">
    <xs:simpleContent>
        <xs:extension base="LO">
            <xs:attributeGroup ref="ATTR_GRP_T" />
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:complexType name="LT_ATTR_T">
    <xs:simpleContent>
        <xs:extension base="LT">
            <xs:attributeGroup ref="ATTR_GRP_T" />
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:complexType name="OB_ATTR_T">
    <xs:simpleContent>
        <xs:extension base="OB">
            <xs:attributeGroup ref="ATTR_GRP_T" />
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:complexType name="OF_ATTR_T">
    <xs:simpleContent>
        <xs:extension base="OF">
            <xs:attributeGroup ref="ATTR_GRP_T" />
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:complexType name="OW_ATTR_T">
    <xs:simpleContent>
        <xs:extension base="OW">
            <xs:attributeGroup ref="ATTR_GRP_T" />
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:complexType name="PN_ATTR_T">
    <xs:complexContent>
        <xs:extension base="PN">
            <xs:attributeGroup ref="ATTR_GRP_T" />
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:complexType name="SH_ATTR_T">
    <xs:simpleContent>
        <xs:extension base="SH">
            <xs:attributeGroup ref="ATTR_GRP_T" />
        </xs:extension>
    </xs:simpleContent>

```

```

</xs:complexType>
<xs:complexType name="SL_ATTR_T">
  <xs:simpleContent>
    <xs:extension base="SL">
      <xs:attributeGroup ref="ATTR_GRP_T" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="SQ_ATTR_T">
  <xs:complexContent>
    <xs:extension base="SQ">
      <xs:attributeGroup ref="ATTR_GRP_T" />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="SS_ATTR_T">
  <xs:simpleContent>
    <xs:extension base="SS">
      <xs:attributeGroup ref="ATTR_GRP_T" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="ST_ATTR_T">
  <xs:simpleContent>
    <xs:extension base="ST">
      <xs:attributeGroup ref="ATTR_GRP_T" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="TM_ATTR_T">
  <xs:simpleContent>
    <xs:extension base="TM">
      <xs:attributeGroup ref="ATTR_GRP_T" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="UI_ATTR_T">
  <xs:simpleContent>
    <xs:extension base="UI">
      <xs:attributeGroup ref="ATTR_GRP_T" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="UL_ATTR_T">
  <xs:simpleContent>
    <xs:extension base="UL">
      <xs:attributeGroup ref="ATTR_GRP_T" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="UN_ATTR_T">
  <xs:simpleContent>
    <xs:extension base="UN">
      <xs:attributeGroup ref="ATTR_GRP_T" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="US_ATTR_T">
  <xs:simpleContent>
    <xs:extension base="US">
      <xs:attributeGroup ref="ATTR_GRP_T" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="UT_ATTR_T">
  <xs:simpleContent>

```

```

    <xs:extension base="UT">
      <xs:attributeGroup ref="ATTR_GRP_T" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="EXT_ATTR_T">
  <xs:annotation>
    <xs:documentation>
      This attribute is useful for representing attributes whose
      VR types are not supported natively by Oracle.
    </xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="EXT">
      <xs:attributeGroup ref="ATTR_GRP_T" />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="EXP_ATTR_T">
  <xs:annotation>
    <xs:documentation>
      This attribute type is useful for representing attributes that
      are present in a DICOM object, but whose definition cannot
      be found in the data dictionary. Such
      attributes cannot be parsed or interpreted.
    </xs:documentation>
  </xs:annotation>
  <xs:simpleContent>
    <xs:extension base="EXP">
      <xs:attributeGroup ref="ATTR_GRP_T" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="DOCUMENT_HEADER_T">
  <xs:annotation>
    <xs:documentation>
      Each time the XML configuration document is modified,
      a new element, DOCUMENT_CHANGE_LOG, is
      added to the DOCUMENT_HEADER.
      The change log describes who made what type of change to the
      XML document on which date. It also describes what DICOM
      standard document the modification is based upon, either
      a DICOM change proposal (CP) or a DICOM supplement.

      DOCUMENT_MODIFIER identifies the modifier of the present
      XML document. If it is generated by software, specify the name
      and version of the software.
      DOCUMENT_MODIFICATION_DATE specifies the date when
      this XML document is modified.
      DOCUMENT_VERSION specifies the version of the document after
      the modification.
      MODIFICATION_COMMENT briefly describes the modification.
      BASE_DOCUMENT describes the document or DICOM standard
      that the modification is based upon.
      BASE_DOCUMENT_RELEASE_DATE specifies the release date of
      the base document.
      BASE_DOCUMENT_DESCRIPTION briefly describes the base
      document.
    </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="DOCUMENT_CHANGE_LOG" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="DOCUMENT_MODIFIER" type="SHORT_STRING_T" />
          <xs:element name="DOCUMENT_MODIFICATION_DATE" type="SHORT_STRING_T" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>

```

```

    <xs:element name="DOCUMENT_VERSION" type="SHORT_STRING_T" minOccurs="0"/>
    <xs:element name="MODIFICATION_COMMENT" type="SHORT_TEXT_T" minOccurs="0"/>
    <xs:element name="BASE_DOCUMENT" type="SHORT_STRING_T" minOccurs="0"/>
    <xs:element name="BASE_DOCUMENT_RELEASE_DATE" type="xs:date" minOccurs="0"/>
    <xs:element name="BASE_DOCUMENT_DESCRIPTION" type="SHORT_TEXT_T" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:complexType name="ATTR_DEFINERS_T">
  <xs:annotation>
    <xs:documentation>
      Attribute definer is identified by its name and UID.
      In Oracle's implementation, the DICOM standard is given the
      definer name "DICOM" and the UID "1.2.840.10008.1".
      All DICOM standard attributes are given the definer name "DICOM".
      Users can introduce private attributes of their own and encode them
      in an XML document. These private attributes are identified
      with the definer's name and UID. Oracle recommends that all DICOM
      private attributes be associated with a UID-qualified name.
    </xs:documentation>
  </xs:annotation>
  <xs:sequence maxOccurs="unbounded">
    <xs:element name="ATTR_DEFINER">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="NAME" type="LO" maxOccurs="unbounded"/>
          <xs:element name="UID" type="UI" minOccurs="0"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<!-- Attribute Tag (allowing x wildcard)-->
<xs:simpleType name="ATTR_TAG_T">
  <xs:annotation>
    <xs:documentation>
      The attribute tag type differs from DICOM VR
      type AT in that it allows the wildcard character 'x'.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:token">
    <xs:pattern value="([0-9a-fA-FxX]{8})"/>
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="ATTR_RANGE_T">
  <xs:annotation>
    <xs:documentation>
      The attribute range type defines a range of DICOM attributes.
      This data type is used in private attribute definitions.
      Certain private attributes accept a range of attribute tags.
    </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="STARTING_TAG" type="ATTR_TAG_T"/>
    <xs:element name="ENDING_TAG" type="ATTR_TAG_T"/>
  </xs:sequence>
</xs:complexType>
<xs:simpleType name="VALUE_LOCATOR_T">
  <xs:annotation>
    <xs:documentation>
      The DICOM value locator type identifies a particular
      DICOM attribute by "xxxxxxx(definer)", where
      "xxxxxxx" is the attribute tag and "definer" is the
      attribute definer, which can be the DICOM standard
    </xs:documentation>
  </xs:annotation>

```


the sequence item (FFFE, E000).

For compatibility with future DICOM releases, if a new DICOM VR is introduced by the DICOM standard, users can mark such attributes as type "EXT??", where "??" should be replaced by the new VR name.

```

</xs:documentation>
</xs:annotation>
<xs:restriction base="xs:token">
  <xs:pattern value="AE"/>
  <xs:pattern value="AS"/>
  <xs:pattern value="AT"/>
  <xs:pattern value="CS"/>
  <xs:pattern value="DA"/>
  <xs:pattern value="DS"/>
  <xs:pattern value="DT"/>
  <xs:pattern value="FL"/>
  <xs:pattern value="FD"/>
  <xs:pattern value="IS"/>
  <xs:pattern value="LO"/>
  <xs:pattern value="LT"/>
  <xs:pattern value="OB"/>
  <xs:pattern value="OF"/>
  <xs:pattern value="OW"/>
  <xs:pattern value="PN"/>
  <xs:pattern value="SH"/>
  <xs:pattern value="SL"/>
  <xs:pattern value="SQ"/>
  <xs:pattern value="SS"/>
  <xs:pattern value="ST"/>
  <xs:pattern value="TM"/>
  <xs:pattern value="UI"/>
  <xs:pattern value="UL"/>
  <xs:pattern value="UN"/>
  <xs:pattern value="US"/>
  <xs:pattern value="UT"/>
  <xs:pattern value="USS"/>
  <xs:pattern value="OWB"/>
  <xs:pattern value="EXP"/>
  <xs:pattern value="EXT[A-Z]{2}"/>
</xs:restriction>
</xs:simpleType>
<xs:simpleType name="SHORT_TEXT_T">
  <xs:restriction base="xs:token">
    <xs:maxLength value="1999"/>
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="MIXED_TEXT_T" mixed="true">
  <xs:complexContent mixed="true">
    <xs:extension base="xs:anyType"/>
  </xs:complexContent>
</xs:complexType>
<xs:simpleType name="SHORT_STRING_T">
  <xs:restriction base="xs:token">
    <xs:maxLength value="128"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="SHORT_NAME_T">
  <xs:restriction base="xs:NCName">
    <xs:maxLength value="128"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="SHORT_ID_T">
  <xs:restriction base="xs:ID">
    <xs:maxLength value="64"/>
    <xs:pattern value="^[\\.]+"/>
  </xs:restriction>

```

```

    </xs:restriction>
  </xs:simpleType>
</xs:schema>

```

B.8 Preference Document Schema

The preference document schema `ordcmpf.xsd`, shown in [Example B-8](#), defines the structure of the preference documents. The namespace for this schema is

```
http://xmlns.oracle.com/ord/dicom/preference_1_0
```

Example B-8 Preference Document Schema

```

<?xml version="1.0" encoding="UTF-8"?>
<!--
  Copyright (c) 2007, 2009, Oracle and/or its affiliates. All rights reserved.
  NAME
    ordcmpf.xsd - XML Schema for DICOM preference documents.
-->

<xs:schema xmlns="http://xmlns.oracle.com/ord/dicom/preference_1_0"
  xmlns:dt="http://xmlns.oracle.com/ord/dicom/datatype_1_0"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  targetNamespace="http://xmlns.oracle.com/ord/dicom/preference_1_0" elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xs:import namespace="http://xmlns.oracle.com/ord/dicom/datatype_1_0"
    schemaLocation="http://xmlns.oracle.com/ord/dicom/datatype_1_0"/>
  <xs:annotation>
    <xs:documentation>
      Introduction
      This schema defines the run-time preference settings for
      Oracle Multimedia DICOM features.

      Structure Overview
      Question mark "?" means optional items.
      Plus "+" means one or more items.
      Asterisk "*" means zero or more items.

      DICOM_RUNTIME_PREFERENCES
      DOCUMENT_HEADER?
      DOCUMENT_CHANGE_LOG*
      DOCUMENT_MODIFIER
      DOCUMENT_MODIFICATION_DATE
      DOCUMENT_VERSION?
      MODIFICATION_COMMENT?
      BASE_DOCUMENT?
      BASE_DOCUMENT_RELEASE_DATE?
      BASE_DOCUMENT_DESCRIPTION?
      PREFERENCE_DEF+
      PARAMETER
      DESCRIPTION
      VALUE
      The allowed values for the PARAMETER element of a
      PREFERENCE_DEF entry and its corresponding
      VALUE element are as follows:

      PARAMETER: XML_SKIP_ATTR
      VALUE: an integer type (default 512, 128~ 32767)
      DESCRIPTION: When encoding a DICOM attribute into XML, skip
      attributes whose (child) XML element sizes (in bytes) are
      larger than XML_SKIP_ATTR.
      If an attribute is of simple type, this limit applies to the
      whole attribute.
      If the attribute type is SQ, this limit applies to its child

```

items.

For example, if an attribute is of type SQ and it contains child items of type OB, the limit applies to each child instance of type OB.

The smallest value allowed for this parameter is 128.

PARAMETER: AVG_ATTR_NUM

VALUE: an integer type (default 200, 20~2000)

DESCRIPTION: The average number of root-level attributes per DICOM object. This is a hint to the DICOM implementation.

Finding the optimal value for a database helps improve storage efficiency and performance. Too large a value may lead to wasted memory, and too small a value may lead to poor performance. An ideal value is one where most (suggested 95%) DICOM images have less than \$VALUE number of attributes.

The smallest value allowed for this parameter is 20.

The largest number allowed for this parameter is the total number of not retired standard attributes defined.

PARAMETER: CONFORMANCE_LEVEL

VALUE: enum { leastConform, ignoreException(default), mostConform}

DESCRIPTION:

The option "leastConform" means that all functions try to maximize the processing of a DICOM object and ignore any errors and exceptions.

"ignoreException" means that all functions ignore the types of exceptions given in the parameter "IGNORED_EXP_LIST". The default set of ignored exceptions includes: MISSING_ATTR, INVALID_LENGTH, MISSING_MAGIC, MISSING_HEADER, INVALID_VR, INVALID_VM, and PARSE_ERR.

"mostConform" means that all functions throw an exception if a DICOM object contains nonconformant content. This does not include backward compatibility cases allowed by the DICOM standard.

Note: By choosing an option other than "mostConform", you risk accepting invalid DICOM objects, possibly getting incorrect results. In this case, Oracle recommends setting the LOGGING_LEVEL parameter to "warning" or a more detailed level, and then examining the log file for possible errors.

PARAMETER: IGNORED_EXP_LIST

VALUE: EmptySpace-separated exception names from the following list:

```
{MISSING_MAGIC, MISSING_HEADER, MISSING_ATTR,
  FAULTY_VALUE, INVALID_LENGTH,
  INVALID_VM, INVALID_VR, UNSUPPORT_VALUE,
  UNDEFINED_VALUE, NOT_AN_IMAGE, PARSE_ERR}
```

Default: {MISSING_ATTR INVALID_LENGTH MISSING_MAGIC MISSING_HEADER INVALID_VR INVALID_VM PARSE_ERR}.

DESCRIPTION: This parameter is only effective when the value of the CONFORMANCE_LEVEL parameter is "ignoreException". If this is the case, the exceptions in the ignore exception list are ignored at run time. However, if the LOGGING_LEVEL parameter is set to "warning" or a more detailed level, the exception is logged. The program continues and skips the part of the DICOM object that has triggered an exception.

These exceptions are defined as follows:

MISSING_MAGIC: a DICOM object does not contain the file magic number "DICM".

MISSING_HEADER: a DICOM object does not have the file meta header (not conformant to the DICOM standard part 10).

MISSING_ATTR: a DICOM object does not have the mandatory attributes (type 1) required by the DICOM standard.

FAULTY_VALUE: a DICOM object has attribute values that lead to parsing errors.

INVALID_LENGTH: a DICOM object contains a length value that is not consistent with the DICOM encoding rules or a length that is not permitted by the DICOM data dictionary.

INVALID_VM: an attribute of a DICOM object has an invalid Value Multiplicity value (not consistent with the dictionary definition).

INVALID_VR: an attribute of a DICOM object has an invalid Value Representation value, which can either conflict with the data dictionary or has not been defined by the data dictionary.

UNSUPPORTED_VALUE: a DICOM object contains attribute values that are outside of the supported range; for example, an unsupported pixel representation value.

UNDEFINED_VALUE: a DICOM object contains attribute values that are not defined by the data model; for example, an undefined transfer syntax UID, an undefined SOP class UID, and so on.

NOT_AN_IMAGE: When an image content processing function is invoked on a DICOM object, if the object's SOP class UID is defined but its classification is not "storageClass", or its content type is not "image", an exception is thrown. It may mean that the UID definition document is not up-to-date. An administrator can update the document to define the SOP class UID as a "storageClass" of "image" type.

PARSE_ERR: When a DICOM object contains invalid data, a parse exception is thrown. This exception is ignored, and the parsing process continues.

PARAMETER: OUTPUT_RAW_VALUE

VALUE: an integer value (default to 0, no output) (-1 ~ 32767)

DESCRIPTION: What to output in an XML metadata document when the parsing of a DICOM object fails. The base64 encoding of the attribute's byte value can be returned in the rawValue attribute of a DICOM XML element. The VALUE element specifies the maximal length allowed for the rawValue attribute. If \$VALUE == -1, (not recommended), the entire attribute up to 32k is saved in the rawValue attribute in base64 encoding. If \$VALUE == 0, an empty string is saved in the rawValue attribute (recommended for production systems). If \$VALUE == N > 0, only the first N bytes of the attribute are saved in the rawValue attribute.

A nonzero value for this parameter is useful for debugging purposes. For a production system, do NOT pick a value larger than 64. The value -1 should never be used outside of a development environment.

PARAMETER: LOGGING_LEVEL

VALUE: enum {debug, conformance, warning(default), error, none }

DESCRIPTION: The logging level, if ordered by the level of detail from the most to the least is: "debug", "conformance", "warning", "error", and "none".

"debug" means extensive logging of all steps; it should only be used for debugging purposes.

"conformance" means to log all nonstandard conformance problems that are discovered. In general, nonconformance is very common for a DICOM object repository containing DICOM objects from different sources, for example, a hospital or an imaging center. This logging level may lead to large log files for most scenarios, and lower performance.

"warning" means to log all recoverable messages that require operator attention. For example, if a user invokes an image processing function on a DICOM object and Oracle does not recognize this DICOM object as an image, a warning message is logged stating that this DICOM object is not defined as an image. The processing of the image content may continue if the CONFORMANCE_LEVEL parameter is set to ignore "NOT_AN_IMAGE" exception.

"error" means to log only irrecoverable messages.

"none" means that logging is disabled.

Note: Do not use the "debug" option for a deployed system. It adds significant overhead and slows down all DICOM related functions.

PARAMETER: VALIDATE_METADATA

VALUE: Boolean{true, false(default)}

DESCRIPTION: The value of this parameter determines whether to validate the XML documents used in the DICOM functions and procedures. If the value of this parameter is false, the XML documents are not validated. If the value is set to true, the XML documents are validated against a specific XML schema that is registered with Oracle XML DB.

All XML documents used in the DICOM functions and procedures, except those that are generated by the method `extractMetadata()`, are validated against the Oracle default DICOM metadata schema. The XML documents generated by the method `extractMetadata()` are validated against the XML schema whose namespace is defined in the specified mapping document.

PARAMETER: EXP_IF_NULL_ATTR_IN_CONSTRAINT

VALUE: Boolean{true(default), false}

DESCRIPTION: A DICOM object may not contain certain attributes that are used in a constraint predicate. The object may contain an attribute, but its value is empty. Both cases result to a null value attribute. So a constraint predicate involving this attribute has a null parameter value such as `(null== MY_VALUE)`.

If this preference parameter is set to true, an exception is thrown if the first occurrence of a null-value attribute is not guarded by the "notEmpty" Boolean function. If this parameter is set to false, no exception is thrown and the predicate evaluates to false. To avoid confusion, it is always better to guard an attribute with "notEmpty" Boolean functions before using the attribute value in a predicate.

PARAMETER: MAX_RECURSION_DEPTH

VALUE: an integer type (default 16, 1 ~ 32767)

DESCRIPTION: This parameter restricts the number of levels of recursions when evaluating a recursive constraint on a DICOM object. If the recursion level exceeds this number, an exception is thrown.

PARAMETER: MANDATE_ATTR_TAGS_IN_STL

VALUE: Boolean{false(default), true}

DESCRIPTION: This parameter is used to enforce that all tags used by the constraint and mapping documents must be listed in the `STORED_TAG_LIST(STL)` document. This rule is not enforced by default. If this preference parameter is set to true, the rule is enforced. If an existing STL document does not satisfy this rule, the preference value cannot be set to true until the STL document is deleted.

PARAMETER: SQ_WRITE_LEN

VALUE: boolean{true(default), false}

DESCRIPTION: This parameter determines how the DICOM sequence (SQ) types are encoded by the DICOM function `writeMetadata()`. If the value of this parameter is true, the SQ types are encoded with explicit length and without item or sequence delimiters. This is the default behavior and allows DICOM viewers to skip the sequence attributes easily. If the value of this parameter is false, the SQ types are encoded with variable (or undefined) length and terminated with sequence delimiters. This allows backward compatibility with some

older DICOM viewers and Dicom applications as they only support undefined length for SQ types.

PARAMETER: SPECIFIC_CHARACTER_SET
 VALUE: enum {ASCII(default), ISO_IR 100, ISO_IR 101, ISO_IR 109, ISO_IR 110, ISO_IR 144, ISO_IR 127, ISO_IR 126, ISO_IR 138, ISO_IR 148, ISO_IR 13, ISO_IR 166, ISO_IR 192, GB18030}
 DESCRIPTION: This parameter determines how data elements with value representations of SH (Short String), LO (Long String), ST (Short Text) LT (Long Text), PN (Person Name) or UT (Unlimited Text) are decoded when the Specific Character Set (0008,0005) Attribute is missing. The DICOM standard states that the default character set (ISO-IR 6, or ASCII) shall be used for decoding when the Specific Character Set (0008,0005) Attribute is not specified. This parameter allows an application to specify a different character set to use in these cases.

PARAMETER: BINARY_SKIP_INVALID_ATTR
 VALUE: boolean{false(default), true}
 DESCRIPTION: It is possible that a DICOM object contains one or more attribute values that do not conform to the DICOM specification. In the event that such an attribute is present in a DICOM object, this parameter determines whether or not to "skip" (i.e. not include) the value of that attribute in the binary output when making a copy of the object in question. The default behavior, specified by setting this parameter to "false", is to not skip these values and to include them as-is in the corresponding output. When an attribute is skipped, its value is included in the output truncated to length 0.

```

</xs:documentation>
</xs:annotation>
<xs:element name="DICOM_RUNTIME_PREFERENCES">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="DOCUMENT_HEADER" type="dt:DOCUMENT_HEADER_T" minOccurs="0"/>
      <xs:element name="PREFERENCE_DEF" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>
            Each PREFERENCE_DEF entry describes one parameter
            that a repository administrator may modify to adjust the
            run-time behavior of the DICOM functionality.
          </xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:sequence>
            <xs:element name="PARAMETER" type="dt:SHORT_ID_T"/>
            <xs:element name="DESCRIPTION" type="dt:SHORT_TEXT_T" minOccurs="0"/>
            <xs:element name="VALUE">
              <xs:simpleType>
                <xs:restriction base="dt:SHORT_TEXT_T">
                  <xs:pattern value="[ 0-9_a-zA-Z]+"/>
                </xs:restriction>
              </xs:simpleType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>

```

B.9 Private Dictionary Document Schema

The private dictionary document schema `ordcmpv.xsd`, shown in [Example B-9](#), defines the structure of the private dictionary documents. The namespace for this schema is

```
http://xmlns.oracle.com/ord/dicom/privateDictionary_1_0
```

Example B-9 Private Dictionary Document Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
Copyright (c) 2007, Oracle. All rights reserved.

NAME
ordcmpv.xsd - XML schema for DICOM private dictionary documents
-->

<xs:schema xmlns="http://xmlns.oracle.com/ord/dicom/privateDictionary_1_0"
xmlns:dt="http://xmlns.oracle.com/ord/dicom/datatype_1_0"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://xmlns.oracle.com/ord/dicom/privateDictionary_1_0" elementFormDefault="qualified"
attributeFormDefault="unqualified">
<xs:import namespace="http://xmlns.oracle.com/ord/dicom/datatype_1_0"
schemaLocation="http://xmlns.oracle.com/ord/dicom/datatype_1_0"/>
<xs:annotation>
<xs:documentation>
Introduction
This schema defines the private attributes created by modality
manufacturers or organizations other than the DICOM
Standards Committee.

Structure Overview
Question mark "?" means optional items.
Plus "+" means one or more items.
Asterisk "*" means zero or more items.

DICOM_PRIVATE_ATTRIBUTES
DOCUMENT_HEADER?
DOCUMENT_CHANGE_LOG*
DOCUMENT_MODIFIER
DOCUMENT_MODIFICATION_DATE
DOCUMENT_VERSION?
MODIFICATION_COMMENT?
BASE_DOCUMENT?
BASE_DOCUMENT_RELEASE_DATE?
BASE_DOCUMENT_DESCRIPTION?
ATTRIBUTE_DEFINERS?
DEFINER+
NAME
ID?
PRIVATE_ATTRIBUTE_DEFINITION+
(TAG|TAG_RANGE)
NAME
DEFINER
VR?
VM?
RETIRED?
DOCUMENT_HEADER is an optional header to specify the
modification history. See dt:DOCUMENT_HEADER_T
for more information.
ATTRIBUTE_DEFINERS specify the owner of each
private attribute. See dt:ATTR_DEFINER_T for more
information.
```

A private dictionary contains one or more private attribute

definitions.

Each private attribute specification takes a tag specification, a name, a value representation type, a value multiplicity type, and a retired flag. See dt:VR_T dt:VM_T for the allowed values for the value representation and value multiplicity elements.

Note: Private attribute tags allow three specification types.

The tag can be a 4-byte hexadecimal number, a wildcard type such as "0039xx01", or a range type such as "0039xx01~0041xx01".

Multiple attribute definitions cannot be associated with the same definer-tag pair in a dictionary. For example, a simple attribute definition ("oracle", 60100010) matches a wildcard attribute ("oracle", 60xx0010), they cannot coexist in the private dictionary. Similarly, a range attribute definition ("Oracle", {6000-60FF}) overlaps a range attribute definition ("Oracle", {6010-6020}), they cannot coexist in the private dictionary. As a rule, an attribute must not match two entries in the dictionary.

```

</xs:documentation>
</xs:annotation>
<xs:element name="DICOM_PRIVATE_ATTRIBUTES">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="DOCUMENT_HEADER" type="dt:DOCUMENT_HEADER_T" minOccurs="0"/>
      <xs:element name="ATTRIBUTE_DEFINERS" type="dt:ATTR_DEFINERS_T" minOccurs="0"/>
      <xs:element name="PRIVATE_ATTRIBUTE_DEFINITION" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:choice>
              <xs:element name="TAG" type="dt:ATTR_TAG_T"/>
              <xs:element name="TAG_RANGE" type="dt:ATTR_RANGE_T"/>
            </xs:choice>
            <xs:element name="NAME" type="dt:SHORT_STRING_T"/>
            <xs:element name="DEFINER" type="dt:LO"/>
            <xs:element name="VR" type="dt:VR_T" minOccurs="0"/>
            <xs:element name="VM" type="dt:VM_T" minOccurs="0"/>
            <xs:element name="RETIRED" type="xs:boolean" default="false" minOccurs="0"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>

```

B.10 Standard Dictionary Document Schema

The standard dictionary document schema `ordcmsd.xsd`, shown in [Example B-10](#), defines the structure of the standard dictionary documents. The namespace for this schema is

`http://xmlns.oracle.com/ord/dicom/standardDictionary_1_0`

Example B-10 Standard Dictionary Document Schema

```

<?xml version="1.0" encoding="UTF-8"?>
<!--
Copyright (c) 2007, Oracle. All rights reserved.
NAME
  ordcmsd.xsd - XML Schema for DICOM standard dictionary document.
-->

<xs:schema xmlns="http://xmlns.oracle.com/ord/dicom/standardDictionary_1_0"

```

```

xmlns:dt="http://xmlns.oracle.com/ord/dicom/datatype_1_0"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
targetNamespace="http://xmlns.oracle.com/ord/dicom/standardDictionary_1_0"
elementFormDefault="qualified"
attributeFormDefault="unqualified">
<xs:import namespace="http://xmlns.oracle.com/ord/dicom/datatype_1_0"
  schemaLocation="http://xmlns.oracle.com/ord/dicom/datatype_1_0"/>
<xs:annotation>
  <xs:documentation>
    Introduction
    This schema defines the data dictionary that lists the DICOM
    standard attributes as published by the DICOM Standards Committee.
    No other attributes, such as those defined by a modality
    manufacturer or an organization other than NEMA,
    should be included in the standard data dictionary.
  
```

Structure Overview

Question mark "?" means optional items.
 Plus "+" means one or more items.
 Asterisk "*" means zero or more items.

```

DICOM_STANDARD_ATTRIBUTES
  DOCUMENT_HEADER?
  DOCUMENT_CHANGE_LOG*
  DOCUMENT_MODIFIER
  DOCUMENT_MODIFICATION_DATE
  DOCUMENT_VERSION?
  MODIFICATION_COMMENT?
    BASE_DOCUMENT?
    BASE_DOCUMENT_RELEASE_DATE?
  BASE_DOCUMENT_DESCRIPTION?
ATTRIBUTE_DEFINERS?
  DEFINER+
    NAME
    ID?
  STANDARD_ATTRIBUTE_DEFINITION+
  TAG
  NAME
  VR?
    VM?
    RETIRED?

```

DOCUMENT_HEADER is an optional header to specify the modification history. See dt:DOCUMENT_HEADER_T for more information.

ATTRIBUTE_DEFINERS specify the owner of each attribute. See dt:ATTR_DEFINER_T for more information. All DICOM standard attributes must have definer name "DICOM" and UID "1.2.840.10008.1".

A standard dictionary contains one or more standard attribute definitions.

Each standard attribute specification takes a tag specification, a name, a value representation type, a value multiplicity type, and a retired flag. See DICOM P3-6 2007 for a description of these elements. See dt:VR_T dt:VM_T for the allowed values of value representation and value multiplicity elements.

Note: Wildcard character "x" can be used to specify standard attribute tags (for example, 60xx0010 for overlay rows).

Multiple attribute definitions must not be associated with the same tag in a standard dictionary. For example, the

```

attribute definition 60100010 matches the wildcard
attribute 60xx0010, they cannot coexist in
the dictionary. As a rule, an attribute must not match two
entries in the dictionary.
</xs:documentation>
</xs:annotation>
<xs:element name="DICOM_STANDARD_ATTRIBUTES">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="DOCUMENT_HEADER" type="dt:DOCUMENT_HEADER_T" minOccurs="0"/>
      <xs:element name="ATTRIBUTE_DEFINERS" type="dt:ATTR_DEFINERS_T" minOccurs="0"/>
      <xs:element name="STANDARD_ATTRIBUTE_DEFINITION" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="TAG" type="dt:ATTR_TAG_T"/>
            <xs:element name="NAME" type="dt:SHORT_STRING_T"/>
            <xs:element name="VR" type="dt:VR_T" minOccurs="0"/>
            <xs:element name="VM" type="dt:VM_T" minOccurs="0"/>
            <xs:element name="RETIRED" type="xs:boolean" minOccurs="0"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>

```

B.11 Stored Tag List Document Schema

The stored tag list document schema `ordcmstl.xsd`, shown in [Example B-11](#), defines the structure of the DICOM stored tag list documents. The namespace for this schema is

```
http://xmlns.oracle.com/ord/dicom/attributeTag_1_0
```

Example B-11 Stored Tag List Document Schema

```

<?xml version="1.0" encoding="UTF-8"?>
<!--
Copyright (c) 2008, Oracle and/or its affiliates. All rights reserved.

NAME
ordcmstl.xsd - XML schema for the DICOM STORED_TAG_LIST document
-->

<xs:schema xmlns="http://xmlns.oracle.com/ord/dicom/attributeTag_1_0"
  xmlns:dt="http://xmlns.oracle.com/ord/dicom/datatype_1_0"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  targetNamespace="http://xmlns.oracle.com/ord/dicom/attributeTag_1_0" elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xs:import namespace="http://xmlns.oracle.com/ord/dicom/datatype_1_0"
    schemaLocation="http://xmlns.oracle.com/ord/dicom/datatype_1_0"/>
  <xs:annotation>
    <xs:documentation>
      <xs:documentation>
        Introduction
        This schema defines the STORED_TAG_LIST document.
        This document lists the attribute tags that are
        persistently stored in the metadata attribute of the ORDDICOM object.

        Structure Overview:
        Question mark "?" means optional items.
        Plus "+" means one or more items.
        Asterisk "*" means zero or more items.

```

```

    ATTRIBUTE_TAG_LIST
    DOCUMENT_HEADER?
    DOCUMENT_CHANGE_LOG*
    DOCUMENT_MODIFIER
    DOCUMENT_MODIFICATION_DATE
    DOCUMENT_VERSION?
    MODIFICATION_COMMENT?
    BASE_DOCUMENT?
    BASE_DOCUMENT_RELEASE_DATE?
    BASE_DOCUMENT_DESCRIPTION?
    ATTRIBUTE_TAG+
</xs:documentation>
</xs:annotation>
<xs:element name="ATTRIBUTE_TAG_LIST">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="DOCUMENT_HEADER" type="dt:DOCUMENT_HEADER_T" minOccurs="0"/>
      <xs:element name="ATTRIBUTE_TAG" type="dt:VALUE_LOCATOR_T" minOccurs="1" maxOccurs="unbounded" >
        <xs:annotation>
          <xs:documentation>
            Each ATTRIBUTE_TAG entry describes one locator path of a
            DICOM attribute in the list.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>

```

B.12 UID Definition Document Schema

The UID definition document schema `ordcmui.xsd`, shown in [Example B-12](#), defines the structure of the UID definition documents. The namespace for this schema is

`http://xmlns.oracle.com/ord/dicom/UIDdefinition_1_0`

Example B-12 UID Definition Document Schema

```

<?xml version="1.0" encoding="UTF-8"?>
<!--
  Copyright (c) 2007, Oracle. All rights reserved.

  NAME
    ordcmui.xsd - XML schema for DICOM UID definition documents.
-->

<xs:schema xmlns="http://xmlns.oracle.com/ord/dicom/UIDdefinition_1_0"
  xmlns:dt="http://xmlns.oracle.com/ord/dicom/datatype_1_0"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://xmlns.oracle.com/ord/dicom/UIDdefinition_1_0" elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xs:import namespace="http://xmlns.oracle.com/ord/dicom/datatype_1_0"
    schemaLocation="http://xmlns.oracle.com/ord/dicom/datatype_1_0"/>
  <xs:annotation>
    <xs:documentation>
      Introduction
      This schema defines DICOM UIDs.
      User can update this file to support new DICOM object types.

      Structure Overview
      Question mark "?" means optional items.
      Plus "+" means one or more items.
      Asterisk "*" means zero or more items.
    </xs:documentation>
  </xs:annotation>

```



```

DICOM_UID_DEFINITIONS
  DOCUMENT_HEADER?
    DOCUMENT_CHANGE_LOG*
    DOCUMENT_MODIFIER
    DOCUMENT_MODIFICATION_DATE
    DOCUMENT_VERSION?
    MODIFICATION_COMMENT?
      BASE_DOCUMENT?
      BASE_DOCUMENT_RELEASE_DATE?
      BASE_DOCUMENT_DESCRIPTION?
  UID_DEF (classification, isLE?, isEVR?, isCompressed?,retired?, contentType?)+
  UID
  NAME
  DESCRIPTION?

```

A UID_DEF entry describes a UID value.

The mandatory classification attribute specifies what a UID is. Its value can be "transferSyntax", "storageClass", "frameOfRef", "ldapOID", "entityID", or "other". "transferSyntax" means that the UID identifies transfer syntax. "storageClass" means that the UID identifies a storage class. "frameOfRef" means that the UID is a well-known frame of reference. "ldapOID" means that the UID is an LDAP OID. "entityID" means that the UID identifies an entity, which can be an organization or a device manufacturer. "other" means that the UID does not fall into any of the previous categories.

For entries that have a classification type of "transferSyntax", the attributes "isLE", "isEVR", and "isCompressed" further define the transfer syntax. These attributes are ignored for all other classification types. The "isLE" attribute specifies whether the binary stream will be encoded with little-endian byte order (defaults to true). The "isEVR" attribute specifies whether the binary stream will use the explicit VR encoding rule (defaults to true). The "isCompressed" attribute specifies whether the transfer syntax means that the data content is compressed (defaults to true).

If an entry has a classification type of "storageClass", the "contentType" attribute further specifies the primary content of a DICOM object belonging to this class. The value of this attribute can be "image", "waveform", "report" or "other". "image" can be single-frame, multi-frame images, or video. "waveform" can be ECG, EEG, or any other 1D signal. "report" means a structured report. "other" means overlay, GSPS, KO, or any other object types that do not belong to the previous categories. For example "Ultrasound Multi-frame Image Storage" SOP class has a UID of "1.2.840.10008.5.1.4.1.1.3.1". Its primary content is image.

```

</xs:documentation>
</xs:annotation>
<xs:element name="DICOM_UID_DEFINITIONS">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="DOCUMENT_HEADER" type="dt:DOCUMENT_HEADER_T" nillable="true" minOccurs="0"/>
      <xs:element name="UID_DEF" maxOccurs="unbounded">
        <xs:complexType>
          <xs:complexContent>
            <xs:extension base="UID_ENTRY_T">
              <xs:attribute name="classification" use="required">
                <xs:simpleType>

```

```
<xs:restriction base="xs:token">
  <xs:enumeration value="transferSyntax"/>
  <xs:enumeration value="storageClass"/>
  <xs:enumeration value="frameOfRef"/>
  <xs:enumeration value="ldapOID"/>
  <xs:enumeration value="entityID"/>
  <xs:enumeration value="other"/>
</xs:restriction>
</xs:simpleType>
</xs:attribute>
<xs:attribute name="isLE" type="xs:boolean" default="true"/>
<xs:attribute name="isEVR" type="xs:boolean" default="true"/>
<xs:attribute name="isCompressed" type="xs:boolean" default="true"/>
<xs:attribute name="retired" type="xs:boolean" default="false"/>
<xs:attribute name="contentType" default="image">
  <xs:simpleType>
    <xs:restriction base="xs:token">
      <xs:enumeration value="image"/>
      <xs:enumeration value="waveform"/>
      <xs:enumeration value="report"/>
      <xs:enumeration value="other"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
</xs:extension>
</xs:complexContent>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:complexType name="UID_ENTRY_T">
  <xs:sequence>
    <xs:element name="UID" type="dt:UI"/>
    <xs:element name="NAME" type="dt:SHORT_STRING_T"/>
    <xs:element name="DESCRIPTION" type="dt:SHORT_TEXT_T" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
</xs:schema>
```

DICOM Encoding Rules

Digital Imaging and Communications in Medicine (DICOM) includes [transfer syntax](#). The transfer syntax is a complete set of [DICOM encoding rules](#) for medical images.

Note: In Oracle Database 10g Release 2 (10.2), Oracle Multimedia DICOM introduced these encoding rules to support metadata extraction and image content processing only. In Oracle Database 11g Release 1 (11.1), Oracle Multimedia DICOM provided full support for medical imaging.

This appendix includes these sections:

- [Transfer Syntax for Medical Imaging](#) on page C-1
- [Definitions for Transfer Syntax Abbreviations](#) on page C-2

See Also:

- *Oracle Multimedia Reference* for more information about image processing
- <http://medical.nema.org/> for more information about DICOM encoding rules (DICOM standard, Part 5 and Part 6)

C.1 Transfer Syntax for Medical Imaging

[Table C-1](#) lists the DICOM encoding rules that support medical imaging in Oracle Multimedia DICOM. See [Section C.2](#) for descriptions of the abbreviations used in the Support column for [Table C-1](#).

Table C-1 *Encoding Rules for Transfer Syntax*

Value	Name	Support
1.2.840.10008.1.2	Implicit VR Little Endian Default Transfer Syntax	PR
1.2.840.10008.1.2.1	Explicit VR Little Endian Transfer Syntax	PR
1.2.840.10008.1.2.1.99	Deflated Explicit VR Little Endian	PR
1.2.840.10008.1.2.2	Explicit VR Big Endian	PR
1.2.840.10008.1.2.4.50	JPEG Baseline (Process 1)	PR
1.2.840.10008.1.2.4.51	JPEG Extended (Process 2 & 4)	R
1.2.840.10008.1.2.4.52	JPEG Extended (Process 3 & 5) (Retired)	R

Table C-1 (Cont.) Encoding Rules for Transfer Syntax

Value	Name	Support
1.2.840.10008.1.2.4.53	JPEG Spectral Selection, Non-Hierarchical (Process 6 & 8) (Retired)	R
1.2.840.10008.1.2.4.54	JPEG Spectral Selection, Non-Hierarchical (Process 7 & 9) (Retired)	R
1.2.840.10008.1.2.4.55	JPEG Full Progression, Non-Hierarchical (Process 10 & 12) (Retired)	R
1.2.840.10008.1.2.4.56	JPEG Full Progression, Non-Hierarchical (Process 11 & 13) (Retired)	R
1.2.840.10008.1.2.4.57	JPEG Lossless, Non-Hierarchical (Process 14)	R
1.2.840.10008.1.2.4.58	JPEG Lossless, Non-Hierarchical (Process 15) (Retired)	R
1.2.840.10008.1.2.4.59	JPEG Extended, Hierarchical (Process 16 & 18) (Retired)	R
1.2.840.10008.1.2.4.60	JPEG Extended, Hierarchical (Process 17 & 19) (Retired)	R
1.2.840.10008.1.2.4.61	JPEG Spectral Selection, Hierarchical (Process 20 & 22) (Retired)	R
1.2.840.10008.1.2.4.62	JPEG Spectral Selection, Hierarchical (Process 21 & 23) (Retired)	R
1.2.840.10008.1.2.4.63	JPEG Full Progression, Hierarchical (Process 24 & 26) (Retired)	R
1.2.840.10008.1.2.4.64	JPEG Full Progression, Hierarchical (Process 25 & 27) (Retired)	R
1.2.840.10008.1.2.4.65	JPEG Lossless, Hierarchical (Process 28) (Retired)	R
1.2.840.10008.1.2.4.66	JPEG Lossless, Hierarchical (Process 29) (Retired)	R
1.2.840.10008.1.2.4.70	JPEG Lossless, Non-Hierarchical, First-Order Prediction (Process 14 [Selection Value 1])	R
1.2.840.10008.1.2.4.80	JPEG-LS Lossless Image Compression	R
1.2.840.10008.1.2.4.81	JPEG-LS Lossy (Near-Lossless) Image Compression	R
1.2.840.10008.1.2.4.90	JPEG 2000 Image Compression (Lossless Only)	PR*
1.2.840.10008.1.2.4.91	JPEG 2000 Image Compression	PR*
1.2.840.10008.1.2.4.100	MPEG2 Main Profile @ Main Level	PR
1.2.840.10008.1.2.5	RLE Lossless	PR

C.2 Definitions for Transfer Syntax Abbreviations

These abbreviations are used in [Table C-1](#):

P

The `processCopy()`, `createDicomImage()`, and `writeMetadata()` procedures and methods are able to produce DICOM content of this transfer syntax.

R

The `extractMetadata()`, `setProperties()`, and `isConformanceValid()` methods are able to read DICOM content of this transfer syntax. If the encoding rule is *not* changing between the source and destination DICOM content, then the `processCopy()`, `createDicomImage()`, and `writeMetadata()` procedures and methods are also supported for this transfer syntax.

* (the asterisk character)

This encoding rule requires the Java Advanced Imaging (JAI) Image I/O Tools provided by Oracle.

See the following script, which is available in the Oracle home directory after installation:

On Linux and UNIX:

```
<ORACLE_HOME>/ord/im/admin/initjp2.sql
```

On Windows:

```
<ORACLE_HOME>\ord\im\admin\initjp2.sql
```

See Also:

<http://java.sun.com/> for more information about JAI Image I/O Tools

- (the hyphen character)

No support (for P or R) is provided.

DICOM Processing and Supported Formats

This appendix describes processing operations and supported formats for DICOM content.

This appendix includes these sections:

- [DICOM Image Content and Compression Formats](#) on page D-1
- [The frame Image Processing Operator](#) on page D-3
- [Other Image Processing Operators](#) on page D-3
- [Multiframe Image Processing and Creation](#) on page D-4
- [Multiframe DICOM Content Processing to AVI Format](#) on page D-4
- [Order of Precedence with processCopy\(\) Method Arguments](#) on page D-5

See [Chapter 7](#) and [Chapter 8](#) for reference information about the processCopy() method.

See Also:

Oracle Multimedia Reference for a complete list of image processing operators and details about each operator

D.1 DICOM Image Content and Compression Formats

Photometric interpretation of DICOM image content is specified by the tag <00280004>. [Table D-1](#) shows the supported photometric interpretations for the processCopy() method with DICOM images.

Table D-1 DICOM Content Photometric Interpretations

Action	Supported Photometric Interpretation Values
READING	MONOCHROME1, MONOCHROME2, RGB, PALETTECOLOR, YBR_FULL, YBR_FULL_422, YBR_RCT, and YBR_ICT
WRITING	MONOCHROME1, MONOCHROME2, RGB, YBR_FULL, YBR_FULL_422, YBR_RCT, and YBR_ICT

For the action `READING`, DICOM images with the listed photometric interpretation values can be decoded and the pixel data can be extracted. For the action `WRITING`, DICOM images with the listed photometric interpretation values can be encoded and the pixel data can be written with the corresponding photometric interpretations.

Compression format type in DICOM images is specified by the transfer syntax UID value or the tag <00020010>. [Table D-2](#) shows the supported compression formats for the `processCopy()` method with DICOM images.

Table D-2 DICOM Content Compression Formats

Action	Supported Compression Format Values
READING	DEFLATE, JPEG, JPEG 2000 ¹ , MPEG, ² RAW ³ , and RLE
WRITING	DEFLATE, JPEG, JPEG 2000 ¹ , MPEG ⁴ , RAW ³ , and RLE ⁵

¹ With JAI image I/O installations (see [Appendix C](#))

² Supports extracting MPEG content from DICOM format to MPEG format (with `processCopy()` method only). No other processing operations are supported.

³ With multibit from 8-bit to 16-bit, inclusive

⁴ Supports encapsulating MPEG content into DICOM format (with `createDicomImage()` method only)

⁵ When writing DICOM images with RLE compression, specify DICOMRLE as the compression format value (see [Section 3.2.6](#)).

For the action `READING`, DICOM images with the listed compression format values can be decoded and the pixel data can be decompressed. For the action `WRITING`, DICOM images with the compression format values can be encoded and the pixel data can be compressed with the corresponding compression format types.

To specify the compression format for the `processCopy()` method, use the `compressionFormat` operator. If the image is not compressed (the compression format value is `RAW`), the encoding bits of each pixel are specified in the user's metadata, which is passed as one argument of the `processCopy()` method.

See [Chapter 7](#) and [Chapter 8](#) for reference information about the `processCopy()` method, and for more information about the `compressionFormat` operator.

The following subsections describe some content compression formats in more detail:

- [DEFLATE Compression Format](#)
- [MPEG Compression Format](#)
- [RLE Compression Format](#)

D.1.1 DEFLATE Compression Format

For the DEFLATE compression format, you can specify the speed or the amount of compression with the `deflateLevel` operator. This operator accepts either a string or an integer from 1 to 9. The valid values are further defined in the following table:

Value	Type	Description
<code>bestspeed</code>	String	Fastest compression rate
<code>bestcompression</code>	String	Highest compression level
<code>defaultcompression</code>	String	Default value
1 through 9	Integer	A range of values, representing these results: <ul style="list-style-type: none"> ■ The lower the value, the faster the compression rate ■ The higher the value, the higher the compression level

To specify the highest DEFLATE compression level using a string value for the `deflateLevel` operator (highlighted in bold), use the following syntax:

```
dicom.processcopy('fileformat=dicom, compressionFormat=deflate,
deflateLevel=bestcompression', sop_instance_uid, dest);
```

This processing operation, which is performed at a relatively slow rate, results in an image with a high amount of compression. Specifying the string value `bestspeed`, by contrast, would cause the image to be processed at a fast rate and result in an image with a low amount of compression.

To specify the highest DEFLATE compression level using an integer value for the `deflateLevel` operator (highlighted in bold), use the following syntax:

```
dicom.processcopy('fileformat=dicom, compressionFormat=deflate,
deflateLevel=9', sop_instance_uid, dest);
```

This processing operation is also performed at a slow rate. Similar to the previous example, it results in an image with a high amount of compression. In general, using an integer value enables you to fine-tune the speed and the level of compression. In this example, specifying the highest integer value (9) results in the same level of compression as with the string value `bestcompression`.

D.1.2 MPEG Compression Format

With the `processCopy()` method, you can read MPEG content from DICOM format and extract it to MPEG format.

With the `createDicomImage()` method, you can write MPEG content into DICOM format.

D.1.3 RLE Compression Format

You can write DICOM images with RLE compression using the following syntax:

```
'fileFormat=dicom, compressionFormat=dicomrle'
```

D.2 The frame Image Processing Operator

Oracle Multimedia provides the frame operator for DICOM image processing. You can use the frame operator to extract a specific frame image from a multiframe DICOM image. You can also combine the frame operator with other image processing operators, such as `scale` and `rotate`, to perform multiple operations on a single frame extracted from a multiframe DICOM image.

To extract a specific frame by the number of the frame, use the following syntax:

```
frame = <frame number>
```

If the specified frame number is less than 1 or greater than the maximum number of frames in the DICOM image, the attempted operation returns an invalid frame exception. The frame operator can be used for DICOM images with only one frame. However, in this case, the value for the frame number must always be 1.

D.3 Other Image Processing Operators

Other image processing operators include `fileFormat`, `contentFormat`, `compressionFormat`, `cut`, `scale`, and `rotate`.

See Also:

Oracle Multimedia Reference for a complete list of image processing operators and details about each operator

D.4 Multiframe Image Processing and Creation

Oracle Multimedia supports the creation and processing of multiframe DICOM images.

To process a single frame in a multiframe DICOM image, specify the frame within the frame operator statement in the processCopy command (see [Section D.2](#) for an example of this syntax). You can also combine the frame operator with other operators, such as scale and rotate, to perform multiple operations on a single frame extracted from a multiframe DICOM image.

To create a multiframe DICOM image, use the following syntax:

```
frame = ALL
```

Alternatively, you can combine the frame operator with other operators, such as scale and rotate, to perform multiple operations on all the frames in a multiframe DICOM image.

You can also use the `frame = ALL` syntax to specify processing operations to be performed on single-frame DICOM images.

See Also:

Oracle Multimedia Reference for a complete list of image processing operators and details about each operator

D.5 Multiframe DICOM Content Processing to AVI Format

Oracle Multimedia supports the processing of multiframe DICOM content to the Microsoft Video for Windows Audio Video Interleave (AVI) format. You can generate output in AVI format from multiframe DICOM content (such as MRIs, CTs, and Ultrasound videos).

To process multiframe DICOM content into AVI output, invoke the processCopy() method (or relational procedure) on the DICOM content and convert it into AVI format. Use the following syntax to specify the processCopy() method (or procedure) with the value of the fileFormat operator set to `avi`, as shown in the example:

```
'fileFormat=avi'
```

The preceding example shows the fileFormat operator set to `avi`, with no other parameters. Optionally, in addition to specifying `avi` as the value of the fileFormat operator, you can also specify *either* the framerate *or* the frametime parameter. The framerate parameter represents the number of frames or images of the resulting AVI video that are displayed each second. The frametime parameter represents the duration or time, in number of seconds, that each frame or image in the resulting AVI video is displayed. The default value for both of these optional parameters is 1.

The following code segment shows how to specify the framerate parameter for a display of ten frames of AVI video in each second:

```
'fileFormat=avi, framerate=10'
```

The following code segment shows how to specify the framerate parameter for a display of one frame of AVI video for every one-tenth of a second:

```
'fileFormat=avi, framerate=0.1'
```

Note: Specify at most one optional parameter each time you invoke the processCopy() method (or procedure) with the fileFormat operator set to `avi`. Specifying both optional parameters at the same time causes an exception to be thrown.

When the value of the fileFormat operator is set to `avi`, you can also combine this operator with other operators, such as `scale` and `rotate`, to perform multiple operations on multiframe DICOM content.

See [processCopy\(\) to BLOBs](#) for reference information about the processCopy() method that supports this feature. See [processCopy\(\) for BFILES](#) and [processCopy\(\) for BLOBs](#) for reference information about the processCopy() relational procedures that support this feature.

See Also:

Oracle Multimedia Reference for more information about the AVI data format

D.6 Order of Precedence with processCopy() Method Arguments

When you use the processCopy() method to process a DICOM image to another DICOM image, you can specify the optional metadata argument, `SYS.XMLTYPE`. The processCopy() method uses this metadata argument to specify the encoding rules for the output DICOM image. If a conflict exists between the command argument and the metadata argument, the command argument takes precedence over the metadata argument. The encoding information in the output DICOM image is updated with the result of the processing operation, as specified by the command argument and the metadata argument (in that order).

See Also:

Oracle XML DB Developer's Guide for more information about XMLType operations

DICOM Sample Applications

This appendix briefly describes some sample SQL and PL/SQL scripts and an Oracle Application Express sample application that you can use with Oracle Multimedia DICOM. [Table E-1](#) lists the name and the URL path of these sample programs, starting from this URL for the Oracle Technology Network (OTN) Products Web site

<http://www.oracle.com/technology/products/>

Table E-1 Oracle Multimedia DICOM Sample Programs

Name	URL Path from OTN Products Web Site
Managing DICOM Format Data in Oracle Database 11g	multimedia/
DICOM Image Archive Demonstration	database/application_express/packaged_apps/ packaged_apps.html#DICOM

This appendix includes these sections:

- [Oracle Multimedia DICOM Tutorial](#) on page E-1
- [Oracle Multimedia DICOM Image Archive Demonstration](#) on page E-2

E.1 Oracle Multimedia DICOM Tutorial

The sample scripts in [Chapter 5](#) are provided to help you learn to develop applications using Oracle Multimedia DICOM to upload to the database, retrieve from it, and manipulate DICOM content.

Some sample scripts were extracted from the tutorial **Managing DICOM Format Data in Oracle Database 11g**, and adapted for this manual. You can find this tutorial, and other examples, on the Oracle Multimedia Web page of the Oracle Technology Network Web site as follows:

Start from the Oracle Technology Network Products Web site at

<http://www.oracle.com/technology/products/>

Then, enter the URL path for the DICOM tutorial, as listed in [Table E-1](#).

Select **Oracle Multimedia DICOM Tutorial** under **Quick Picks** to go to the Web page **Managing DICOM Format Data in Oracle Database 11g**.

See [Chapter 5](#) for more information about using these sample scripts with Oracle Multimedia DICOM.

E.2 Oracle Multimedia DICOM Image Archive Demonstration

The Oracle Multimedia DICOM Image Archive Demonstration sample application in [Chapter 6](#) demonstrates the features of Oracle Multimedia DICOM to help you build custom applications. This sample application includes two user interfaces. The interface **DICOM Image Archive** is for clinicians and researchers; the interface **DICOM Image Archive Administration** is for administrators only.

This sample application, along with other packaged applications and sample code, is available on the Oracle Application Express Web page of the Oracle Technology Network Web site as follows:

Start from the Oracle Technology Network Products Web site at

<http://www.oracle.com/technology/products/>

Then, enter the URL path for the DICOM Image Archive Demonstration, as listed in [Table E-1](#).

Select **DICOM Image Archive Demonstration** under **Packaged Applications** to download the Oracle Multimedia DICOM Image Archive Demonstration sample application.

See [Chapter 6](#) for more information about using this sample application.

Migrating from Release 10.2 DICOM Support

This appendix describes two options for migrating data and application code from DICOM support (in `ORDImage` objects) in Oracle Database 10g Release 2 (10.2) to the new DICOM support in Oracle Database 11g Release 1 (11.1). The first migration option has less impact on existing applications. The second migration option takes full advantage of the new DICOM features.

The examples in this appendix are based on the table `imageTable`, which is defined as follows:

```
imageTable( id          integer,
            image       ordsys.ordimage,
            thumb       ordsys.ordimage
```

where:

- `image`: the name of the `ORDImage` column.
- `thumb`: the name of the thumbnail image column.

This appendix includes these sections:

- [Using the DICOM Relational Interface to Migrate Applications](#) on page F-1
- [Copying Data and Rewriting Applications for DICOM](#) on page F-2
- [Choosing a Migration Option](#) on page F-3

F.1 Using the DICOM Relational Interface to Migrate Applications

You can leave the data in the original `ORDImage` objects and use the new DICOM relational interface to access the new DICOM functions.

The advantages of using this migration option are as follows:

- You can migrate your applications quickly, and with minimal impact to your existing Oracle Database 10g Release 2 application.
- You do not have to copy your existing DICOM content.

The only disadvantage of using this migration option is that the DICOM relational interface does not take full advantage of the new DICOM features. Use the migration option explained in [Section F.2](#) to take full advantage of the new DICOM features.

Use this migration option as follows:

1. Leave your data in `ORDImage` columns.
2. Use `ORDImage` methods to access `ORDImage` functions, as you did in Release 10.2.

3. Use the new ORD_DICOM relational package to access new DICOM features.

The following code segment shows this process, using ORDImage objects to store DICOM content.

```

alter table imageTable add (meta sys.xmltype);
declare
  img ordsys.ordimage; thb ordsys.ordimage;
  ctx raw(64) := null; meta sys.xmltype;
begin
  insert into imageTable (id,image, thumb)
    values (1, ordsys.ordimage.init(),
           ordsys.ordimage.init())
    returning image, thumb into img, thb;
  img.importFrom(ctx, 'file', 'DICOMDIR',
                'example.dcm');
  img.processCopy('fileFormat = jfif
                 maxScale=100 100', thb);
  meta :=
ORD_DICOM.extractMetadata(img.getContent,
  'ALL');
.
.
.

```

In this segment, the first two lines of code highlighted in bold show how to modify the ORDImage table (imageTable) by adding a column to store DICOM metadata as an XML document. The last line of highlighted code shows the use of the extractMetadata() method in the ORD_DICOM package (relational interface) to extract all attributes of the embedded DICOM content.

F.2 Copying Data and Rewriting Applications for DICOM

You can copy data into the ORDDicom object type and rewrite your application to use ORDDicom methods. Using this migration option enables you to take full advantage of the features of the new ORDDicom object type.

The disadvantages of using this migration option are as follows:

- You must copy your existing DICOM content into an ORDDicom object type.
- You must rewrite the parts of your application that access Oracle Multimedia.

Use this migration option as follows:

1. Copy your data into ORDDicom objects, as shown in the following code example:

```

alter table imageTable add (dicomImage
  ORDSYS.ORDDicom, metadata sys.XMLTYPE);
-- For each row
update imageTable t set t.dicomImage = new
  orddicom(t.image);
alter table imageTable drop column image;

```

Assuming that you have an existing table (imageTable) with an ORDImage column (image), this example adds a DICOM image column (dicomImage) to store DICOM content as type ORDDicom, adds a metadata column (metadata) to store extracted attributes as an XML document, updates the table by copying the new DICOM content from the image column into the dicomImage column, and then removes the image column from the table.

2. Modify your application to use the new ORDDicom methods, as shown in the following code segment:

```

declare
    img ordsys.orddicom; thb ordsys.ordimage;
                                meta sys.xmltype;
begin
    insert into imageTable (id, dicomImage, thumb,
        metadata)
        values (1, ordsys.orddicom('file', 'DICOMDIR',
                'example.dcm', 1),
        ordsys.ordimage.init(), null))
        returning dicomimage, thumb into
            img, thb;
    img.processCopy('fileFormat=jfif maxScale=100
100', thb);
    meta := img.extractMetadata('ALL');
.
.
.

```

This segment shows an existing application that has been changed to use the new ORDDicom object type. The first two lines of code highlighted in bold show variable declarations for the ORDDicom object (*img*) and the XMLType metadata (*meta*). The next line of highlighted code shows how to insert a row into a table with an ORDDicom object pointing to a DICOM file in the local file system. The next line of highlighted code is similar to the syntax for Oracle Database 10g Release 2, and shows how to generate a JPEG thumbnail image from a DICOM image. The last line of highlighted code shows how to extract all attributes in the embedded DICOM content into an XML document.

See Also:

Oracle XML DB Developer's Guide for more information about XMLType operations

F.3 Choosing a Migration Option

Determining whether to migrate your application by using either the new DICOM relational interface or the new ORDDicom object type depends on your situation and the kinds of applications that exist in your environment. For example, if you have extensive data and limited resources, the migration option of copying your existing DICOM content into an ORDDicom object type might prove too costly and inconvenient. Instead, you might choose to forego the new DICOM features in favor of speedily migrating your application, with limited impact on the existing data. In contrast, if you have limited data in ORDImage objects, you might choose to rewrite your applications and copy your existing data into the ORDDicom object type to enable access to all the new DICOM features.

Glossary

anonymity document

An XML document that specifies the set of attributes to be made anonymous, and defines the actions required to make those attributes anonymous.

application conformance

The semantic consistency of DICOM content with regard to application-specific constraint rules, which can be stronger or weaker than rules in the DICOM standard. Administrators can define constraint documents to include user-defined rules that are specific to a particular organization.

configuration document

A unique document for each database instance that is applicable to all DICOM content stored in the database. Configuration documents are managed by the repository. Examples include private and standard dictionary documents, mapping documents, constraint documents, and preference documents.

conformance validation

The process of checking the conformance of DICOM content against a set of constraint documents that define rules with which to validate conformance. Oracle extends conformance validation to DICOM metadata documents as well.

constraint document

An XML document that defines a collection of rules to validate the conformance of DICOM content, according to the DICOM standard and other organization-wide guidelines. Constraint documents specify the relationships and semantic constraints of attributes that are not expressed by the DICOM metadata schema. Constraint documents are based on the SOP class specification defined in Part 3 of the DICOM standard. Administrators can define constraint documents to include user-defined rules that are specific to a particular organization.

constraint macro

A simplified definition of a complex constraint rule. Constraint macros can be defined as global macros, and follow the same predicate definition grammar as constraint rules. Constraint macros can include predicate operands, which can contain macro parameters that are replaced with parameter values when the macro is invoked. Constraint macros can also be recursive, which is useful when specifying validation requirements for hierarchical or recursive structured DICOM content.

constraint predicate

A definition of a condition that operates on individual attributes or sets of attributes in the DICOM content. A constraint predicate can be a logical statement, a relational statement that compares values, a function call evaluation that returns a Boolean type, or a reference to other predicate definitions. Predicate definitions are recursive.

constraint rule

A collection of rules to validate the conformance of DICOM content according to the DICOM standard. These rules are defined in constraint documents, which specify the relationships and semantic rules of attributes that are not expressed by the DICOM metadata schema. Administrators can define constraint documents to include user-defined rules that are specific to a particular organization.

data model repository

A set of collectively managed, user-configurable documents that defines the run-time behavior of Oracle Multimedia DICOM. Administrators can update the repository to configure Oracle Multimedia DICOM for a particular database instance.

DICOM

See **Digital Imaging and Communications in Medicine (DICOM)**.

DICOM attribute tag

An encoded representation of the metadata within DICOM content that can be used with XML schemas. DICOM attribute tags include standard tags, which are supported by the DICOM standard, and user-defined private tags. DICOM attribute tags contain a group number and an element number.

DICOM conformance validator

A utility that checks the syntactical and semantic consistency of DICOM content in accordance with constraint rules specified in the data model repository.

DICOM content

Multiple standalone DICOM Information Objects that are structured and encoded as defined in Part 10 of the DICOM standard (commonly referred to as DICOM Part 10 files). For more information, see the DICOM standard on the NEMA Web site at

<http://medical.nema.org/>

DICOM data type

See **value representation**.

DICOM data type definition schema

An XML schema that defines the value representations (DICOM data types) provided in Part 5 of the DICOM standard. This data type definition schema is strongly coupled with the DICOM parser. It is designed by Oracle, the content is fixed, and it ships with Oracle Multimedia.

DICOM encoding rules

See **transfer syntax**.

DICOM image

Any DICOM content that can be decomposed into 2-D pixel arrays. Examples include a video segment, a volume scan, and a single-frame dental image.

DICOM metadata document

An XML document that contains the metadata, as encoded attributes, extracted from DICOM content. Oracle provides methods to build a metadata document from DICOM content. Each metadata schema requires a mapping document that defines how attributes from the DICOM content are to be mapped into an XML document that conforms to the schema. In addition, each metadata schema references the DICOM data type definition schema.

DICOM parser

A utility that extracts metadata from DICOM content into a structure in memory.

DICOM Part 10 file

A standalone DICOM Information Object, which is constructed in accordance with the data structure and encoding definitions specified in Part 10 of the DICOM standard. For more information, see the DICOM standard on the NEMA Web site at

<http://medical.nema.org/>

DICOM standard

The dominant standard for radiology imaging and communication to which all major manufacturers conform. The DICOM standard is primarily developed and maintained by working groups of domain experts. A new version is usually published once a year. The standard is available worldwide from the National Electrical Manufacturers Association (NEMA) Web site at

<http://medical.nema.org/>

DICOM value locator

A DICOM type that identifies an attribute in the DICOM content, either at the root level or from the root level down. Each level in the tree hierarchy is represented by a sublocator.

DICOM volume scan

A set of images gathered in a single imaging operation. These images can be stored as separate slices in multiple ORDDicom objects. Or, they can be stored as single ORDDicom multiframe objects.

DICOM XML encoder

A utility that converts the in-memory structure of the extracted DICOM attributes into an XML document.

Digital Imaging and Communications in Medicine (DICOM)

A medical imaging standard initiated by the American College of Radiology (ACR) and the National Electrical Manufacturers Association (NEMA) to enhance the connectivity of radiological devices.

image processor

A utility that includes Java Advanced Imaging (JAI). This utility provides support for operations such as producing thumbnail-size images and converting between DICOM and other supported image formats. When used with Oracle Multimedia DICOM methods, this utility supports import and export operations between the database and operating system files (external file storage).

information object

An object-oriented representation of a real world entity in DICOM. Examples include images and waveforms captured by imaging devices.

manifest document

An XML file that is created when exporting a set of configuration documents from a DICOM repository, and used when importing a set of configuration documents into a DICOM repository. For each configuration document within the set to be imported, the manifest document specifies the name of the configuration document, the document type, and the load order. The default manifest document is `ordcmmft.xml`. This document conforms to the registered XML schema `ordcmmft.xsd`.

mapped attribute

A part of a DICOM data element whose XML path is explicitly defined in a mapping document.

unmapped attribute

A part of a DICOM data element whose XML path is not explicitly defined in a mapping document.

mapping document

An XML document that defines how each attribute maps to a particular element of a DICOM XML metadata document. This document determines the structure of the extracted XML representation of the DICOM metadata.

metadata encoding

The process of encoding the extracted DICOM attributes into a DICOM metadata document.

metadata extraction

The process of extracting attributes from DICOM content.

metadata schema

The XML schema document that constrains the DICOM metadata document. This schema references the DICOM data type definition schema. Oracle supports customization of the metadata schema for each instance of the database. Oracle ships a default metadata schema with Oracle Multimedia. Administrators can update the default metadata schema and the corresponding mapping document to define a schema that is specific to the database instance.

nonscalar attribute

A property of DICOM value locators that exists when the last attribute tag in an array is specified as DICOM type SQ, OW, OB, OF, or UN.

Oracle *interMedia*

In Oracle Database 11g Release 1 (11.1), the name Oracle *interMedia* was changed to Oracle Multimedia.

ORDDicom

Object relational type for DICOM format medical images and other data.

ORDDicom database object

A database object used to encapsulate DICOM content and extracted attributes. The database object has public member methods that permit the querying and processing of ORDDicom objects.

preference document

An XML document that defines a set of run-time parameters, such as specifying whether Oracle Multimedia DICOM skips invalid attributes and their values in the binary output of DICOM content when making a copy of DICOM content. Oracle ships a default preference document with Oracle Multimedia. Administrators can update the default preference document to change the run-time behavior. For example, administrators can specify whether to validate XML documents used in DICOM functions and procedures or cause attributes larger than a specified size to be omitted when encoding attributes into XML.

private attributes

A set of attributes defined by an organization and encoded in DICOM content according to the encoding rules set by that organization for those attributes. Private attributes can add modality-specific, manufacturer-specific, or site-specific information to DICOM content. Private attributes are not administered by the DICOM standard, and they are generally not known or used by any organization other than the one that defined the private attributes.

private dictionary document

A document that lists the private attributes in the data dictionary. A data dictionary can contain one or more private dictionary documents. Each private dictionary document contains the definitions for a set of private attributes and the UID of the organization that defined the private attributes. Private dictionary documents can be published by Oracle or a third party. DICOM administrators can add new private dictionary documents to the data dictionary as they become available. Private dictionary documents enable users to extend the standard dictionary document definitions by adding manufacturer-specific or enterprise-specific attributes to DICOM content. Private dictionary documents are constrained by private dictionary schemas.

repository

A library used to store documents that are applicable to all ORDDicom objects stored in a database instance. Administrators can access the repository and update the documents stored in it. Changes made to these documents change the outcome of the affected Oracle Multimedia DICOM methods and procedures. Examples of documents stored in the repository include mapping documents, data dictionary documents, and constraint documents. At run time, a specified method is used to query the repository to obtain the latest copy of these documents. For example, a DICOM administrator can update an attribute definition in a data dictionary document and change its data type from DA (date) to DT (date time). From that point forward, after setting to the new data model the parser interprets the attribute as an instance of the DT data type, and the metadata encoder encodes the attribute as DT in all future metadata documents.

scalar attribute

A property of DICOM value locators in an array of attribute tags for supported DICOM types.

schema validation

The process of using an XML schema definition to validate an XML document that is constrained by the schema. Schema validation enables users to confirm the correctness

of data types, data formats, and data hierarchies. Schema validation applies to XML documents only. It does not apply to DICOM content.

service object pair (SOP)

The combination of services and information objects.

SOP class

A service object pair (SOP) class. This class is used to model a category of information object and a set of operations associated with that information object.

standard attributes

The set of attributes defined by the DICOM Standards Committee, which is published in Part 6 of the DICOM standard. Standard attributes can be modified or deprecated by the DICOM Standards Committee in the future. The number of standard attributes increases each year as the DICOM standard expands to include new areas of technology.

standard dictionary document

A document that lists the attributes defined by the DICOM standard. The DICOM standard dictionary document is converted from Part 6 of the DICOM standard. The DICOM Standards Committee expects to publish this document in XML format in the future. Oracle software releases include a copy of the standard dictionary document tied to a particular release of the DICOM standard. Administrators can update this standard dictionary document to reflect the most recent changes made by the DICOM Standards Committee.

standards conformance

The syntactical and semantic consistency of DICOM content with regard to the DICOM standard. The default constraint document that Oracle ships with Oracle Multimedia defines rules that enforce conformance with parts of the DICOM standard.

stored tag list document

An optional XML document that specifies the DICOM attributes to be extracted from the embedded DICOM content and stored in the XML metadata attribute of the ORDDicom object when the setProperties() method is called. Generally, stored tag list documents contain the attribute tags used in mapping and constraint documents.

transfer syntax

Encoding rules for medical imaging that specify how to encode DICOM content into a binary stream. These rules also specify how to compress the data content. Supported DICOM compression schemes (codecs) include RLE, JPEG, and JPEG2000.

UID definition document

An XML document that lists unique identifiers (UIDs) defined by the DICOM standard or a private organization. Rather than defining the DICOM content UIDs, UID definition documents contain a registry of standard UIDs that classify the DICOM content and express standard semantics.

unique identifier (UID)

A 64-byte, dot-concatenated, numeric string (similar to an IP address). The UID is based on an ISO object identifier (OID). It is commonly constructed with a root that uniquely identifies the organization producing the DICOM content, and a suffix that uniquely identifies the DICOM content within that organization.

value multiplicity

A constraint rule that specifies how many times the value of an attribute can be repeated. It is part of the standard attribute specification (defined in Part 6 of the DICOM standard).

value representation

The data type, as defined by the DICOM standard. The DICOM standard defines standard data types in Part 5. See the XML schema `ordcmrdt.xsd` (available in the `ord/dicom/xml/xsd` directory under `<ORACLE_HOME>`) for more information about the data types supported by Oracle Multimedia DICOM.

XML mapping document

An XML document used to define how to map DICOM attributes to elements in the DICOM metadata document. The XML mapping document is used by the metadata encoder to produce a DICOM metadata document. Because the metadata document is constrained by the metadata schema, the XML mapping document must match the metadata schema. Oracle Multimedia defines a default mapping document that coincides with the DICOM metadata schema. Administrators can update the mapping document and the corresponding metadata schema to define a schema that is specific to each database instance.

Index

A

- administrator views, 3-2, 9-4, 12-15
- anonymity documents
 - characteristics, 10-2
 - defined, 2-7
 - examples, 2-17
 - writing custom, 10-5
 - XML schemas, B-2
- APIs
 - DICOM relational, 8-1
 - ORD_DICOM data model utility, 4-1
 - ORD_DICOM_ADMIN data model repository API, 12-1
 - ORDDicom object, 7-1

C

- compression format types, D-2
- configuration documents
 - anonymity documents, 2-7
 - characteristics, 10-2
 - constraint documents, 2-7
 - creating, 10-5
 - default, 3-1
 - defined, 2-6
 - deleting from the repository, 9-8
 - exporting from the repository, 9-5
 - inserting into the repository, 9-5
 - mapping documents, 2-7
 - preference documents, 2-7
 - private dictionary documents, 2-7
 - standard dictionary documents, 2-7
 - stored tag list documents, 2-8
 - UID definition documents, 2-8
 - updating in the repository, 9-7
 - writing, 10-5
- conformance validation, 2-14, 3-8
- constraint documents
 - characteristics, 10-2
 - defined, 2-7
 - examples, 2-14, 3-9
 - writing custom, 10-11
 - XML schemas, B-4
- constructors
 - ORDDicom, 7-4

- ORDDicom() for BLOBs, 7-5
- ORDDicom() for ORDImage, 7-6
- ORDDicom() for other sources, 7-7
- converting images
 - illustrated, 2-15
- createDicomImage() for BFILES procedure, 8-20
- createDicomImage() for BLOBs procedure, 8-22
- createDicomImage() for ORDImage procedure, 8-24

D

- data type definitions
 - XML schemas, B-32
- default DICOM metadata
 - XML schema, B-27
- deleteDocument() procedure, 12-7
- deleting constraint documents, 11-4, 11-5
- deleting mapping documents, 11-3
- DICOM administrator
 - creating configuration documents, 10-1
 - managing configuration documents, 11-1
 - ORDADMIN role, 9-2
- DICOM attributes
 - accessing, 5-8
 - examples, 5-8
 - retrieving, 3-5
 - searching, 3-5
- DICOM conformance validator, 2-2
- DICOM content, 1-2, 2-1
 - creating tables, 5-2
 - loading into tables, 3-4, 5-3
 - storing, 5-3, 6-3
- DICOM data model
 - defined, 2-5
- DICOM data model functions
 - getDictionaryTag(), 4-3
 - getMappingXPath(), 4-5
- DICOM data model procedures
 - setDataModel(), 4-9
- DICOM data model repository, 2-2
- DICOM data model utility interface
 - ORD_DICOM package, 4-1
- DICOM Image Archive Administration interface
 - administrator tasks, 6-4
- DICOM Image Archive interface
 - clinician tasks, 6-3

- researcher tasks, 6-3
- search methods, 6-4
- DICOM image processor, 2-2
- DICOM images
 - compressing, 3-7
 - converting, 3-7
 - creating from other formats, 3-7
 - multiframe, D-4
 - processing, 3-7, D-1
 - supported formats, D-1
- DICOM metadata
 - editing, 3-6
 - writing, 3-6
- DICOM metadata documents
 - defined, 2-10
- DICOM parser, 2-1
- DICOM relational API, 8-1
- DICOM relational interface, 8-1
 - ORD_DICOM package, 8-1
- DICOM relational interface functions
 - extractMetadata() for BFILES, 8-4
 - extractMetadata() for BLOBs, 8-6
 - extractMetadata() for ORDImage, 8-8
 - isAnonymous() for BFILES, 8-10
 - isAnonymous() for BLOBs, 8-11
 - isAnonymous() for ORDImage, 8-12
 - isConformanceValid() for BFILES, 8-13
 - isConformanceValid() for BLOBs, 8-15
 - isConformanceValid() for ORDImage, 8-17
- DICOM relational interface procedures
 - createDicomImage() for BFILES, 8-20
 - createDicomImage() for BLOBs, 8-22
 - createDicomImage() for ORDImage, 8-24
 - export(), 8-26
 - importFrom(), 8-28
 - makeAnonymous() for BFILES, 8-30
 - makeAnonymous() for BLOBs, 8-32
 - makeAnonymous() for ORDImage, 8-34
 - processCopy() for BFILES, 8-36
 - processCopy() for BFILES with SOP instance UID, 8-37
 - processCopy() for BLOBs, 8-39
 - processCopy() for BLOBs with SOP instance UID, 8-40
 - processCopy() for ORDImage, 8-42
 - processCopy() for ORDImage with SOP instance UID, 8-44
 - writeMetadata() for BFILES, 8-46
 - writeMetadata() for BLOBs, 8-48
 - writeMetadata() for ORDImage, 8-50
- DICOM value locator type, 10-1
- DICOM XML encoder, 2-1

E

- editDataModel() procedure, 12-8
- encoding rules, C-1
- exception handling
 - Java, 5-14
 - PL/SQL, 5-13

- export() method, 7-10
- export() procedure, 8-26
- exportDocument() procedure, 12-9
- exporting constraint documents, 11-5
- exporting mapping documents, 11-3
- extracting DICOM metadata
 - administrator tasks, 3-5
 - developer tasks, 3-5
- extractMetadata() for BFILES function, 8-4
- extractMetadata() for BLOBs function, 8-6
- extractMetadata() for ORDImage function, 8-8
- extractMetadata() method, 7-12

F

- frame operator
 - image processing, D-3
- frames
 - multiple in DICOM images, D-4
- functions
 - extractMetadata() for BFILES, 8-4
 - extractMetadata() for BLOBs, 8-6
 - extractMetadata() for ORDImage, 8-8
 - generateTagListDocument(), 12-4
 - getDictionaryTag(), 4-3
 - getDocumentContent(), 12-5
 - getMappingXPath(), 4-5
 - isAnonymous() for BFILES, 8-10
 - isAnonymous() for BLOBs, 8-11
 - isAnonymous() for ORDImage, 8-12
 - isConformanceValid() for BFILES, 8-13
 - isConformanceValid() for BLOBs, 8-15
 - isConformanceValid() for ORDImage, 8-17

G

- generateTagListDocument() function, 12-4
- generating stored tag list documents, 11-8
- getAttributeByName() method, 7-14
- getAttributeByTag() method, 7-15
- getContent() method, 7-16
- getContentLength() method, 7-17
- getDictionaryTag() function, 4-3
- getDocumentContent() function, 12-5
- getMappingXPath() function, 4-5
- getSeriesInstanceUID() method, 7-18
- getSOPClassUID() method, 7-19
- getSOPInstanceUID() method, 7-20
- getSourceInformation() method, 7-21
- getSourceLocation() method, 7-22
- getSourceName() method, 7-23
- getSourceType() method, 7-24
- getStudyInstanceUID() method, 7-25

H

- handling exceptions
 - Java, 5-14
 - PL/SQL, 5-13

I

IHE

- Integrating the Healthcare Enterprise, 1-1
- image processing, 5-9
- image processing operators
 - frame, D-3
 - other, D-3
- import() method, 7-26
- importFrom() procedure, 8-28
- importing JDBC classes, 5-14
- insertDocument() procedure, 12-11
- inserting constraint documents, 11-2
- inserting generated stored tag list documents, 11-9
- inserting mapping documents, 11-2
- inserting stored tag list documents, 11-7
- interfaces
 - DICOM relational, 8-1
 - ORD_DICOM_ADMIN data model repository, 12-1
 - ORDDicom object, 7-1
- isAnonymous() for BFILEs function, 8-10
- isAnonymous() for BLOBs function, 8-11
- isAnonymous() for ORDImage function, 8-12
- isAnonymous() method, 7-28
- isConformanceValid() for BFILEs function, 8-13
- isConformanceValid() for BLOBs function, 8-15
- isConformanceValid() for ORDImage function, 8-17
- isConformanceValid() method, 7-29
- isLocal() method, 7-31

J

Java

- configuring your environment, 5-14
 - errors, 5-14
 - examples, 5-13
 - exceptions, 5-14
 - importing classes, 5-14
- ### JDBC
- importing classes, 5-14

L

- loading the data model, 3-3, 9-4

M

- makeAnonymous() for BFILEs procedure, 8-30
- makeAnonymous() for BLOBs procedure, 8-32
- makeAnonymous() for ORDImage procedure, 8-34
- makeAnonymous() method, 7-32
- Managing DICOM Format Data in Oracle Database
 - 11g sample application, 5-1, E-1
- manifest documents
 - XML schemas, B-29
- mapping documents
 - characteristics, 10-3
 - defined, 2-7
 - examples, 2-13
 - writing custom, 10-25

- XML schemas, B-29
- MEDICAL_IMAGE_OBJ table, 7-2
- MEDICAL_IMAGE_REL table, 8-2
- metadata XML schemas, A-1, B-1

- methods
 - export(), 7-10
 - extractMetadata(), 7-12
 - getAttributeByName(), 7-14
 - getAttributeByTag(), 7-15
 - getContent(), 7-16
 - getContentLength(), 7-17
 - getSeriesInstanceUID(), 7-18
 - getSOPClassUID(), 7-19
 - getSOPInstanceUID(), 7-20
 - getSourceInformation(), 7-21
 - getSourceLocation(), 7-22
 - getSourceName(), 7-23
 - getSourceType(), 7-24
 - getStudyInstanceUID(), 7-25
 - import(), 7-26
 - isAnonymous(), 7-28
 - isConformanceValid(), 7-29
 - isLocal(), 7-31
 - makeAnonymous(), 7-32
 - ORDDicom, 7-9
 - processCopy() to BLOBs, 7-34
 - processCopy() to ORDDicom, 7-35
 - processCopy() to ORDImage, 7-37
 - setProperties(), 7-39
 - writeMetadata(), 7-40
- migrating DICOM applications
 - DICOM relational interface, F-1
 - ORDDicom object type interface, F-2
- migration options
 - DICOM applications, F-3

N

NEMA

- National Electrical Manufacturers Association, 1-1

O

- object types
 - ORDDicom, 7-3
- Oracle Data Pump utilities
 - ORDDATA schema, 9-9
- Oracle *interMedia* DICOM *See* Oracle Multimedia DICOM
- Oracle Multimedia DICOM, 1-1
- Oracle Multimedia DICOM Image Archive
 - Demonstration sample application, 6-1, E-2
- ORD_DICOM data model utility API, 4-1
- ORD_DICOM package, 4-2, 4-8, 8-3, 8-19
 - DICOM data model utility interface, 4-1
 - DICOM relational interface, 8-1
 - ordcpksp.sql file, 4-1, 8-1
- ORD_DICOM_ADMIN data model repository API, 12-1

- ORD_DICOM_ADMIN data model repository
 - interface, 12-1
- ORD_DICOM_ADMIN functions
 - generateTagListDocument(), 12-4
 - getDocumentContent(), 12-5
- ORD_DICOM_ADMIN package, 12-1, 12-3, 12-6
 - ordcrpsp.sql file, 12-1
- ORD_DICOM_ADMIN procedures
 - deleteDocument(), 12-7
 - editDataModel(), 12-8
 - exportDocument(), 12-9
 - insertDocument(), 12-11
 - publishDataModel(), 12-13
 - rollbackDataModel(), 12-14
- ORDADMIN role, 9-2
- ORDDATA schema
 - Oracle Data Pump utilities, 9-9
- orddcm_conformance_vld_msgs view, 4-11
- orddcm_constraint_names view, 4-12
- orddcm_document_refs view, 12-16
- orddcm_document_types view, 4-14
- orddcm_documents view, 4-13
- ORDDicom constructors, 7-4
 - ORDDicom() for BLOBs, 7-5
 - ORDDicom() for ORDImage, 7-6
 - ORDDicom() for other sources, 7-7
- ORDDicom methods, 7-9
 - export(), 7-10
 - extractMetadata(), 7-12
 - getAttributeByName(), 7-14
 - getAttributeByTag(), 7-15
 - getContent(), 7-16
 - getContentLength(), 7-17
 - getSeriesInstanceUID(), 7-18
 - getSOPClassUID(), 7-19
 - getSOPInstanceUID(), 7-20
 - getSourceInformation(), 7-21
 - getSourceLocation(), 7-22
 - getSourceName(), 7-23
 - getSourceType(), 7-24
 - getStudyInstanceUID(), 7-25
 - import(), 7-26
 - isAnonymous(), 7-28
 - isConformanceValid(), 7-29
 - isLocal(), 7-31
 - makeAnonymous(), 7-32
 - processCopy() to BLOBs, 7-34
 - processCopy() to ORDDicom, 7-35
 - processCopy() to ORDImage, 7-37
 - setProperty(), 7-39
 - writeMetadata(), 7-40
- ORDDicom object
 - illustrated, 2-4
 - illustrated within a table, 2-4
- ORDDicom object API, 7-1
- ORDDicom object type, 7-3
 - ordcspec.sql file, 7-1
- ORDDicom object type interface, 7-1
- ORDDicom objects
 - anonymity, 5-10

- ORDDicom() for BLOBs constructor, 7-5
- ORDDicom() for ORDImage constructor, 7-6
- ORDDicom() for other sources constructor, 7-7

P

- packages
 - ORD_DICOM, 4-2, 4-8, 8-3, 8-19
 - ORD_DICOM_ADMIN, 12-1, 12-3, 12-6
- photometric interpretation, D-1
- PL/SQL
 - configuring your environment, 5-1
 - errors, 5-13
 - examples, 5-7, 5-9, 5-10, 5-11, 11-2, 11-3, 11-4, 11-5, 11-7, 11-8, 11-9
 - exceptions, 5-13
- preference documents
 - characteristics, 10-4
 - defined, 2-7
 - writing custom, 10-45
 - XML schemas, B-47
- private dictionary documents
 - characteristics, 10-4
 - defined, 2-7
 - writing custom, 10-42
 - XML schemas, B-52
- procedures
 - createDicomImage() for BFILES, 8-20
 - createDicomImage() for BLOBs, 8-22
 - createDicomImage() for ORDImage, 8-24
 - deleteDocument(), 12-7
 - editDataModel(), 12-8
 - export(), 8-26
 - exportDocument(), 12-9
 - importFrom(), 8-28
 - insertDocument(), 12-11
 - makeAnonymous() for BFILES, 8-30
 - makeAnonymous() for BLOBs, 8-32
 - makeAnonymous() for ORDImage, 8-34
 - processCopy() for BFILES, 8-36
 - processCopy() for BFILES with SOP instance UID, 8-37
 - processCopy() for BLOBs, 8-39
 - processCopy() for BLOBs with SOP instance UID, 8-40
 - processCopy() for ORDImage, 8-42
 - processCopy() for ORDImage with SOP instance UID, 8-44
 - publishDataModel(), 12-13
 - rollbackDataModel(), 12-14
 - setDataModel(), 4-9
 - writeMetadata() for BFILES, 8-46
 - writeMetadata() for BLOBs, 8-48
 - writeMetadata() for ORDImage, 8-50
 - processCopy() for BFILES procedure, 8-36
 - processCopy() for BFILES with SOP instance UID procedure, 8-37
 - processCopy() for BLOBs procedure, 8-39
 - processCopy() for BLOBs with SOP instance UID procedure, 8-40

- processCopy() for ORDIImage procedure, 8-42
- processCopy() for ORDIImage with SOP instance UID procedure, 8-44
- processCopy() method, D-1
 - order of operations for DICOM images, D-5
- processCopy() to BLOBs method, 7-34
- processCopy() to ORDDicom method, 7-35
- processCopy() to ORDIImage method, 7-37
- protecting patient privacy, 5-10
 - administrator tasks, 3-11
 - developer tasks, 3-11
- public views, 3-2, 4-10
- publishDataModel() procedure, 12-13

R

- rollbackDataModel() procedure, 12-14

S

- sample applications
 - Managing DICOM Format Data in Oracle Database 11g, 5-1, E-1
 - Oracle Multimedia DICOM Image Archive Demonstration, 6-1, E-2
- service object pair
 - defined, 1-2
- setDataModel() procedure, 4-9
- setProperty() method, 7-39
- SOP
 - service object pair, 1-2
- standard dictionary documents
 - characteristics, 10-3
 - defined, 2-7
 - writing custom, 10-40
 - XML schemas, B-53
- stored tag list documents
 - characteristics, 10-4
 - defined, 2-8
 - inserting into the repository, 9-7
 - writing custom, 10-53
 - XML schemas, B-55

T

- transfer syntax, C-1
 - defined, 1-2

U

- UID definition documents
 - characteristics, 10-4
 - defined, 2-8
 - writing custom, 10-51
 - XML schemas, B-56
- updating mapping documents, 11-3
- updating the data model, 3-3, 9-4

V

- validating conformance, 5-11

- administrator tasks, 3-9
- developer tasks, 3-10
- views
 - administrator, 3-2, 9-4, 12-15
 - orrdcm_conformance_vld_msgs, 4-11
 - orrdcm_constraint_names, 4-12
 - orrdcm_document_refs, 12-16
 - orrdcm_document_types, 4-14
 - orrdcm_documents, 4-13
 - public, 3-2, 4-10

W

- writeMetadata() for BFILEs procedure, 8-46
- writeMetadata() for BLOBs procedure, 8-48
- writeMetadata() for ORDIImage procedure, 8-50
- writeMetadata() method, 7-40

X

- XML metadata documents
 - examples, 2-13
- XML schemas
 - finding user-defined, 9-3
 - metadata, A-1, B-1
 - registering with Oracle XML DB, 9-3

