Oracle® Multimedia

DICOM Developer's Guide

11*g* Release 1 (11.1)

B28416-01

July 2007

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®

Oracle Multimedia DICOM Developer's Guide, 11*g* Release 1 (11.1)

B28416-01

Primary Author: Sue Pelski

Contributors: Rob Abbott, Janet Blowney, Fengting Chen, Bill Gettys, Dongbai Guo, Dong Lin, Susan Mavris, Valarie Moore, Prajna Parida, James Steiner, Yingmei Sun, Simon Watt, Manjari Yalavarthy, Jie Zhang

Contents

6 DICOM Relational Interface Reference

List of Examples

List of Figures

List of Tables

Preface

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

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

In Oracle Database 11g Release 1 (11.1), the name Oracle *inter*Media has been changed to Oracle Multimedia. The feature remains the same, only the name has changed. References to Oracle *inter*Media will be replaced with Oracle Multimedia, however some references to Oracle *inter*Media or *inter*Media may still appear in graphical user interfaces, code examples, and related documents in the Documentation Library for Oracle Database 11g Release 1 (11.1).

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.

The sample code in this guide will not necessarily match the code shipped with the Oracle installation. If you want to run examples that are shipped with the Oracle installation on your system, use the files provided with the installation. Do not attempt to compile and run the code in this guide.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. 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.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Related Documents

Note: For information added after the release of this guide, refer to 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 software documentation set:

- *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 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 on 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*

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

Many of the examples in this book are based on the database user PM and the tables MEDICAL_IMAGE_OBJ and MEDICAL_IMAGE_REL, which will be 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 yourself.

Conventions

In this guide, Oracle *inter*Media (now known as Oracle Multimedia) was sometimes referred to as *inter*Media.

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

Also in examples, vertical ellipsis points indicate that information not directly related to the example has been omitted.

In statements or commands, horizontal ellipsis points indicate that parts of the statement or command not directly related to the example have been omitted.

Also in statements or commands, angle brackets enclose user-supplied names and brackets enclose optional clauses from which you can choose one or none.

Although Boolean is a proper noun, it is presented as boolean in this guide when its use in Java code requires case-sensitivity.

The following text conventions are also 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.
italic	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.

Part I

Common Usage and Reference

This part includes introductory and conceptual information, as well as user and reference information that is common to administrators and developers of DICOM applications.

Part I contains the following chapters:

1

Introduction to Oracle Multimedia DICOM

This chapter contains background information about medical imaging as well as a general introduction to the Oracle Multimedia DICOM (formerly Oracle *inter*Media DICOM) feature.

This chapter includes the following sections:

1.1 Medical Imaging and Communication

Digital Imaging and Communications in Medicine (DICOM) is a medical imaging standard. This standard 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. Nor was it possible to connect hardware devices from different manufacturers.

In 1985, the American College of Radiology and the National Electrical Manufacturers Association jointly published a medical imaging and communication standard, named the ACR-NEMA standard, to address this problem. In 1993, the ACR-NEMA standard was revised and renamed as DICOM (Version 3.0). Since then, the DICOM standard has become the dominant standard for radiology imaging and communication. All major manufacturers conform to this 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 3.10-2007 of the DICOM standard (commonly referred to as DICOM Part 10 files). DICOM content can include different types of data, such as patient administration information, waveforms, images, slices of 3-D volumes, video segments, diagnostic reports, graphics, or text annotations. DICOM content contains a number of standard attributes. Optionally, DICOM content can also contain private attributes. (In the DICOM standard, the phrase DICOM objects refers to DICOM content.)

Like other standards, DICOM is mostly 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, NEMA publishes a new version of the standard each year. This standard is available worldwide from the NEMA Web site at

`http://medical.nema.org/`

The Integrating the Healthcare Enterprise (IHE) initiative also provides information about issues related to DICOM content and communication. This information is available from the IHE Web site at

http://www.ihe.net/

The DICOM standard has two major areas of focuses: the **data model** (or file format) and the communication protocol. The data model is defined using object-oriented programming principles. Content such as images and waveforms captured by medical devices are represented as information objects. Services such as get, find, and store operations can be defined on these objects. The services and the information objects are combined into a **service object pair (SOP)**.

DICOM 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. The binary data can also be exchanged over a network with the DICOM communication protocol, which covers the upper layers (application, presentation, and session layer) of the Open Systems Interconnection (OSI) seven-layer model. The DICOM communication protocol is typically implemented on top of TCP/IP. Recently, the DICOM standard introduced Web access to DICOM objects (WADO). WADO deals primarily with HTTP access to DICOM objects. Messages exchanged between a DICOM server and a DICOM object involve operations such as radiology workflow, gray-scale image rendering, image printing, and storage and retrieval.

The Oracle Multimedia DICOM feature is concerned 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.

1.2 Oracle Multimedia and DICOM

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

Oracle Database 11*g* Release 1 (11.1) continues to provide the same DICOM support in ORDImage. In addition, this release provides more complete DICOM support in a new ORDDicom object type.

1.2.1 Oracle Multimedia DICOM Format Support

With Oracle Database 11*g* Release 1 (11.1), Oracle Multimedia provides full support for DICOM, the format 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, while 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 already 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 10*g* 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 11*g* Release 1 (11.1) extends 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 will be able to create highly specialized indexes to DICOM content based on standard and private DICOM attribute tags.

1.2.4 DICOM Conformance Validation

Oracle Multimedia can verify that DICOM content adheres to a set of user-specified conformance 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 conformance 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 11*g* Release 1 (11.1) adds methods and functions to copy and convert images from DICOM into other image formats (for example: JPEG, GIF, PNG, and TIFF), and to copy and generate scaled versions and thumbnail images. In addition, this release provides a set of optional methods and functions to copy and process (for example: compress, scale, rotate, and crop) image content, optionally during the conversion process.

To view medical images stored in the DICOM format in Web applications, a copy of the images must be created in formats that are compatible with the browsers that are currently used in the industry. Oracle Database 11*g* Release 1 (11.1) enables 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 Private DICOM Content Anonymous

Oracle Database 11*g* Release 1 (11.1) adds 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 that should be made anonymous and the actions to take to make them anonymous.

This method can be used to generate new, anonymous ORDDicom objects, assuring that users of a DICOM medical archive see only the DICOM content and metadata that they are authorized to see. For example, clinicians need 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, on the other hand, need 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 allows appropriate access for different classes of users of a DICOM medical archive, regardless of the application used to access the ORDDicom objects in the archive.

1.2.7 Creating ORDDicom Objects from Images and Metadata

Oracle Database 11*g* Release 1 (11.1) includes 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. 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.

Storing and retrieving film-based medical images is expensive and prone to error. Clinical and research purposes require that some DICOM content be retained for extended periods of time. Replacing film-based medical images with DICOM images reduces storage and retrieval costs. Storing scanned images with their metadata in DICOM format can make non-DICOM images more useful. Using the same technique, 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 in Oracle Database 11*g* 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 need to be up and running at all times. 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.

2

Oracle Multimedia DICOM Concepts

This chapter describes Oracle Multimedia DICOM at a conceptual level.

This chapter includes the following sections:

See Part II, DICOM Development, for specific information about developing DICOM applications.

See Part III, DICOM Administration, for more information about managing the DICOM data model repository.

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: database tier -- Oracle Database; and client tier -- thick clients.

In the database tier, through the use of 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 as well as 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 with respect to 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, import and export operations between the database and external file storage systems are possible. 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.

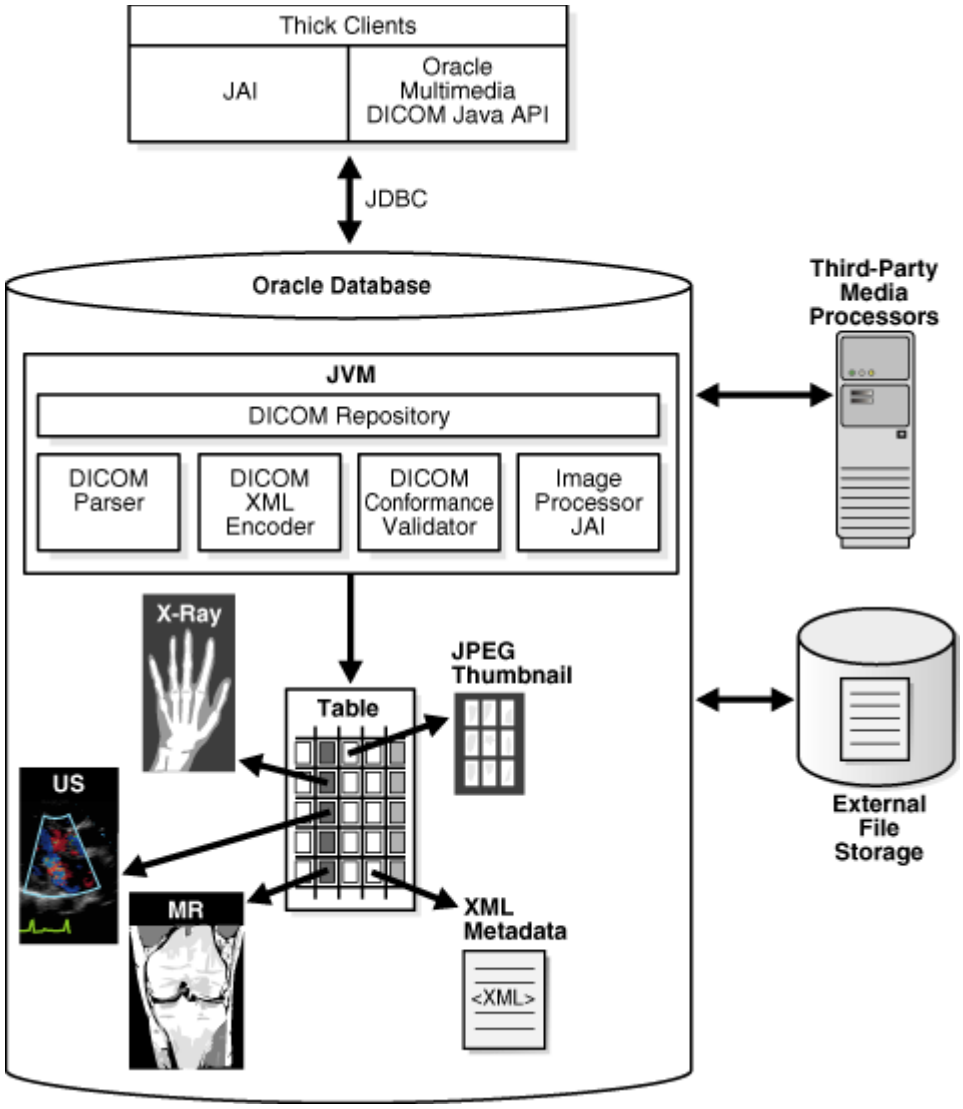
For more information about JAI, see the Sun Microsystems Java Web site at

`http://java.sun.com/`

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.

See *Oracle Multimedia DICOM Java API Reference* for information about using Oracle Multimedia DICOM with Java.

Figure 2–1 Oracle Multimedia DICOM Architecture



For a view of the complete architecture for Oracle Multimedia, see Figure 1-1 in *Oracle Multimedia User's Guide*.

2.2 Oracle Multimedia DICOM Storage

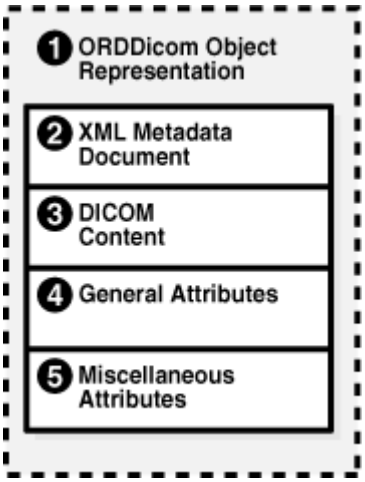
When using the object interface, you must create an ORDDicom object in a table before you can perform Oracle Multimedia DICOM operations on DICOM content. Oracle Multimedia defines the ORDDicom object type, which is similar to a Java or C++ class, to contain DICOM content.

Figure 2–2 shows an ORDDicom object at a very high level. Items in Figure 2–2 are numbered as a means of identifying 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

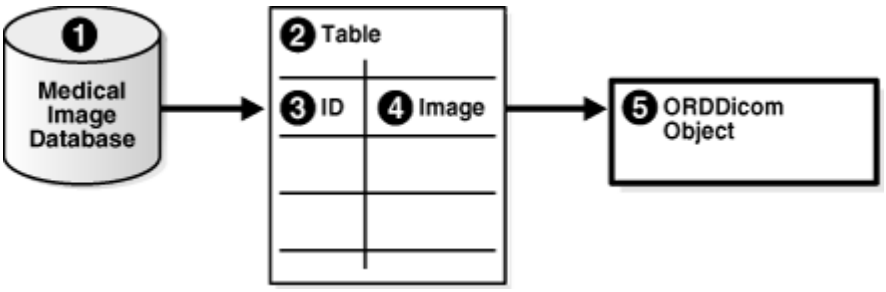
Figure 2–2 ORDDicom Object



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 as a means of identifying 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 make up the data model can be inserted and deleted at run time when there are multiple user sessions 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.

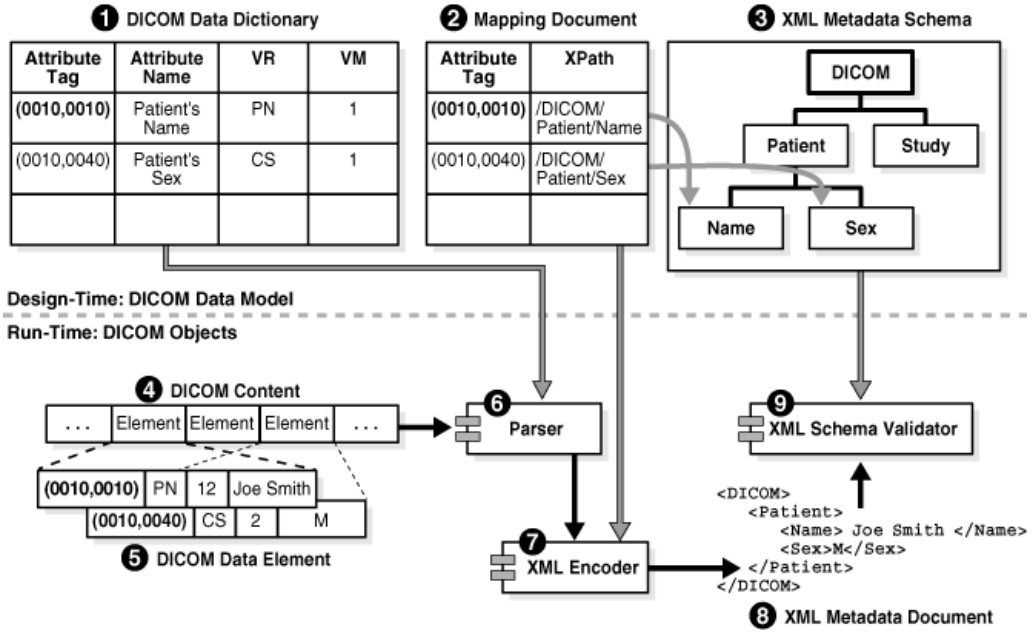
At design time, DICOM administrators can change the DICOM data model. Publishing these changes affects the run-time behavior of the extract metadata feature, and other DICOM operations.

Figure 2–4 shows three components of the data model. Items in Figure 2–4 are numbered as a means of identifying 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 should 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 is not necessarily 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.

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, 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`

Table 2–1 (Cont.) Configuration Documents and Their XML Schemas

Document Type	Default XML Document	XML Schema Definition
Constraint	ordcmct.xml	ordcmct.xsd
	ordcmcmd.xml	
	ordcmcmc.xml	
Mapping	ordcmmp.xml	ordcmmp.xsd
Preference	ordcmpf.xml	ordcmpf.xsd
Private Dictionary	ordcmpv.xml	ordcmpv.xsd
Standard Dictionary	ordcmsd.xml	ordcmsd.xsd
UID Definition	ordcmui.xml	ordcmui.xsd

An **anonymity document** is an XML document that specifies the set of attributes to be made anonymous, and the actions to be taken 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 the DICOM content with 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 should map to a particular element in an XML metadata document tree. This document determines the structure of the extracted XML representation of the DICOM metadata. The default mapping document, `ordcmmp.xml`, defines the mapping from DICOM content to XML representation as a flat list with all XML encoded DICOM attributes included under the root element `<DICOM_OBJECT>`.

A **preference document** is an XML document that defines run-time parameters, such as how to log warning messages when processing DICOM content. The default preference document is `ordcmpf.xml`.

A **private dictionary document** is an XML document that enables users to define manufacturer-specific or enterprise-specific attributes of 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. The default standard dictionary document is `ordcmsd.xml`.

A **UID definition document** is an XML document that lists the unique identifiers (UIDs) for each DICOM data type. The UID is based on an ISO object identifier (OID) that uniquely identifies the DICOM content worldwide. 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. 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 as well as 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** as well as updates from the **Admin Session 2** editing session **XG2**.

Administrator Sessions:

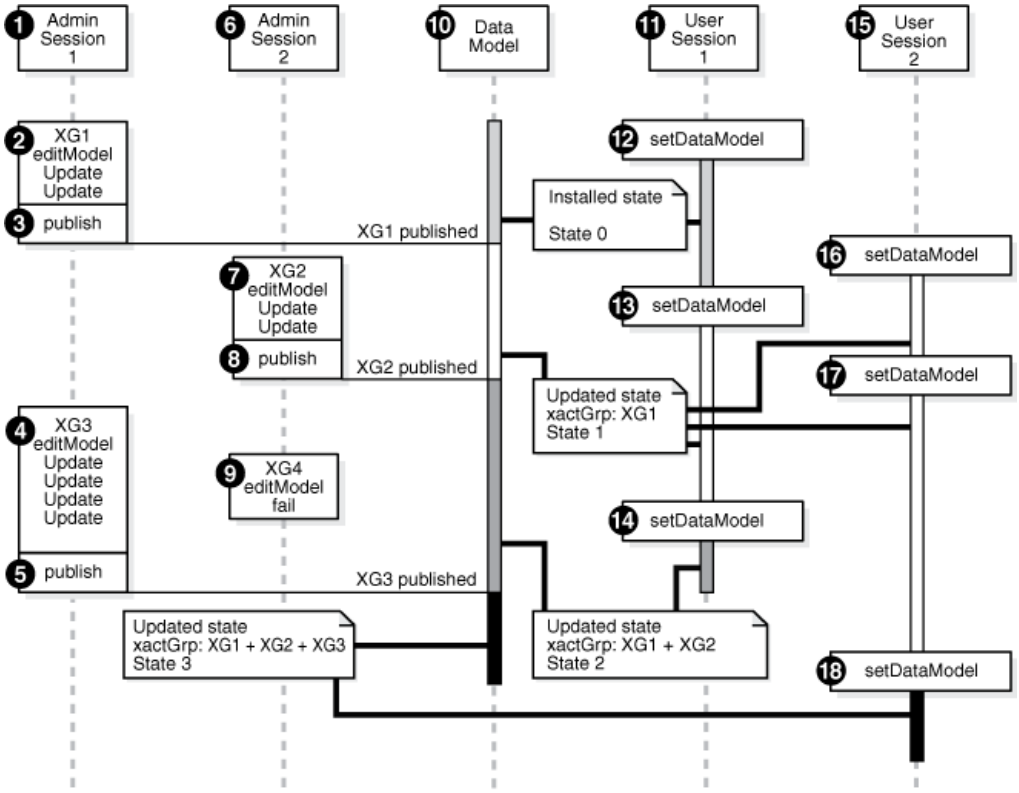
- Item 1- All boxes in this column represent tasks performed by **Admin Session 1**.
- Item 6 - All boxes in this column represent tasks performed by **Admin Session 2**.
- Item 2 - **Admin Session 1** calls the editDataModel( ) procedure to begin editing session **XG1**. This locks the installed version 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 already locked the data model, **Admin Session 2** is unable to 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 once 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 once 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 frequently during user sessions. All users 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 needs to see new data model changes. This procedure is available to users through the DICOM data model utility in the DICOM package interface.

See Part III, DICOM Administration, for more information about the data model repository administration interface. See Chapter 4 for more information about the DICOM data model utility in the 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.

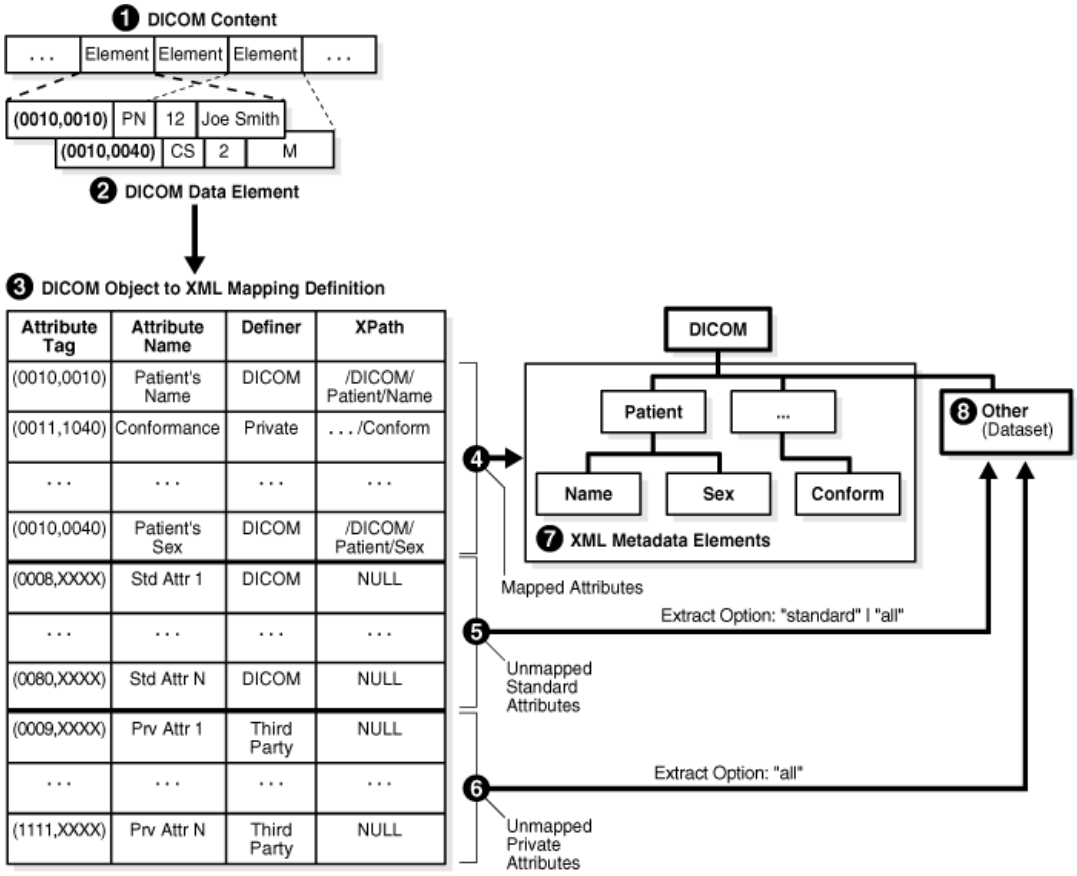
The mapping of DICOM content to the DICOM metadata document is defined by a mapping document. The mapping document defines how attributes from the DICOM content should be mapped into an XML document that conforms to the schema. Like other configuration documents, mapping documents are managed by the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

Oracle provides a default XML metadata schema (`ordcmmd.xsd`) and a matching XML mapping document (`ordcmmp.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 (`ordcmmddt.xsd`).

Figure 2–6 shows the components involved in the metadata extraction and XML mapping process. Each numbered item inFigure 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 inFigure 2–6 show the flow of data. 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, attributes are sorted by their attribute tags and listed by their value representations. 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 11.2.3 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), 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), 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 the standard attributes that are included in the DICOM content.

If the extract option is `all` (Item 6), 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 11.2.6 for more information about specifying these limits within a preference document.

If the mapping option is all or standard, 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 11.2.3.

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 this example, 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&apos;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 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 with respect to the DICOM standard or the guidelines for a particular organization or enterprise. Example 2–3 shows one of the installed constraint documents (`ordcmct.xml`).

Example 2–3 Sample Constraint Document

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Copyright (c) 2007, Oracle. All rights reserved.
  NAME
  ordcmct.xml - Oracle Multimedia DICOM default 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">
  <DOCUMENT_HEADER>
    <dt:DOCUMENT_CHANGE_LOG>
      <dt:DOCUMENT_MODIFIER>Dongbai Guo</dt:DOCUMENT_MODIFIER>
      <dt:DOCUMENT_MODIFICATION_DATE>2007-04-09</dt:DOCUMENT_MODIFICATION_DATE>
      <dt:DOCUMENT_VERSION>1.0</dt:DOCUMENT_VERSION>
      <dt:MODIFICATION_COMMENT>Oracle default constraint rules</dt:MODIFICATION_COMMENT>
    </dt:DOCUMENT_CHANGE_LOG>
```

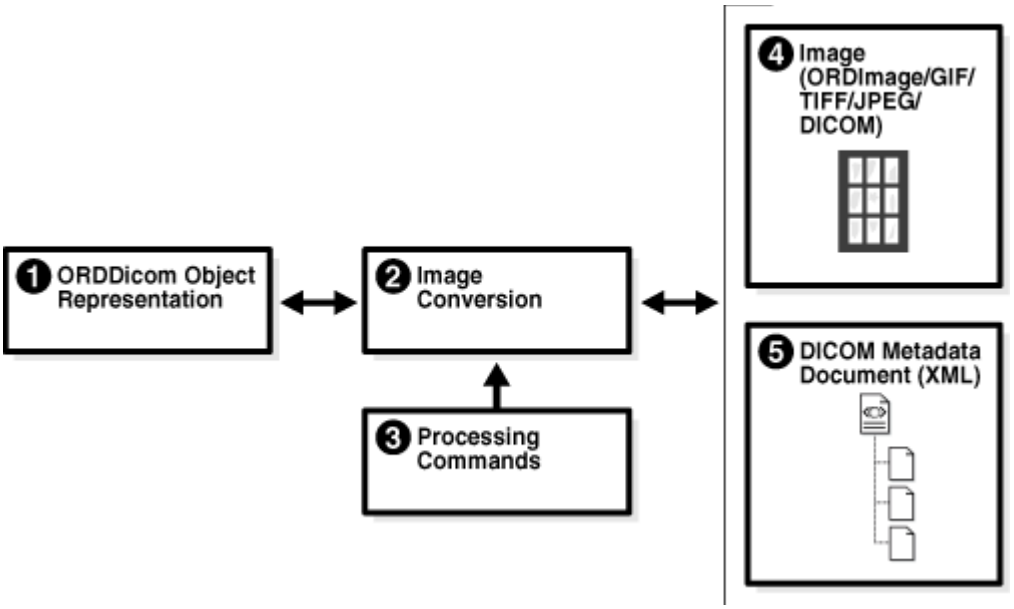
```
 </DOCUMENT_HEADER>
<EXTERNAL_RULE_INCLUDE name="ImagePixelMacro">
  A subset of Image Pixel Macro defined in DICOM standard,
  PS 3.3-2007, Table C.7-11b
</EXTERNAL_RULE_INCLUDE>
<EXTERNAL_RULE_INCLUDE name="GeneralStudyModule">
  A subset of General Study Module defined in DICOM standard,
  PS 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>

<GLOBAL_RULE name="OracleOrdDicomImage">
  <PREDICATE>
    <GLOBAL_RULE_REF>ImagePixelMacro</GLOBAL_RULE_REF>
  </PREDICATE>
  <ACTION action="warning" when="false">missing mandatory image attribute</ACTION>
</GLOBAL_RULE>

<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>
</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 installed global constraint rule named `OracleOrdDicomImage`.

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 then 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 JSP tag libraries or other tools.

2.8 Making DICOM Content Anonymous

Government regulations, such as the Health Insurance Portability and Accountability Act (HIPAA) in the United States, mandate the protection of private data about patients. Sharing DICOM content with external resources often requires making patient private data anonymous. Making DICOM content anonymous in the database avoids exposing patient private data outside of the database, simplifying the protection of that information.

For more information about the HIPAA regulations in the U. S., see the HIPAA Web site at

http://www.hhs.gov/ocr/hipaa/

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, as well as 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 as well as 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 as well as 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>
```

3

Overview of DICOM Development

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

Because Oracle Multimedia DICOM is fully functional after installing Oracle Multimedia, developers can begin writing applications immediately using the following application programming interfaces (APIs):

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

Administrators can assist developers by inserting or deleting configuration documents from the data model repository. To accomplish these tasks, administrators can use the ORD_DICOM_ADMIN data model repository API.

Oracle Multimedia DICOM provides capabilities for a number of 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 your environment. Using information views or invoking data model utility functions, administrators can obtain attributes and other detailed information about these configuration documents. Users can also work directly with the DICOM content, metadata attributes, and other objects to perform various operations.

This chapter includes the following sections:

Table 3–1 provides cross-references to other locations within the Oracle Multimedia documentation set where you can access additional information about topics mentioned in this chapter.

Table 3–1 Additional References for Users

Topic	More Information
Reference information for public information views	Chapter 4
Reference information for administrator information views	Chapter 9
Reference information for the DICOM data model utility API	Chapter 4
Reference information for the ORDDicom object API	Chapter 5
Reference information for the DICOM relational API	Chapter 6
Reference information for the ORD_DICOM_ADMIN data model repository API	Chapter 9
Reference information for the DICOM Java API	*Oracle Multimedia DICOM Java API Reference*
Examples of operations on DICOM content	Chapter 7
Examples of administrative operations in the data model repository	Chapter 10
Information about writing configuration documents	Chapter 11
Listings of the DICOM XML schemas	Appendix B

After installation, each database includes a set of default configuration documents in the Oracle Multimedia DICOM data model repository. See Table 2–1 for a list of these documents.

After installation, administrators can add configuration documents that are specific to a particular organization for the following types of documents:

- Anonymity documents - XML documents that can be used to specify the set of attributes to be made anonymous as well as the actions to be taken to make those attributes anonymous.
- Constraint documents - XML documents that define a collection of rules, including the relationships and semantic constraints of attributes not expressed by the DICOM metadata schema, to validate the conformance of DICOM content with the DICOM standard.
- Mapping documents - XML documents that define how each attribute maps to a particular element in an XML metadata document, and determines the structure of DICOM metadata documents.
- Preference documents - XML documents that define run-time parameters, such as turning the logging of warning messages on or off or specifying categories of error messages.
- Private dictionary documents - XML documents that can be used to extend the standard dictionary document definitions.

- Standard dictionary documents - XML documents that can be used to reflect updates to the DICOM standard.
- UID definition documents - XML documents that list the unique identifiers (UIDs) defined by the DICOM standard.

See Appendix A and Chapter 11 for more information about these documents.

See Part III, DICOM Administration for more information about managing configuration documents in the DICOM repository.

3.1 Loading 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 needs to see 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, call the setDataModel( ) procedure as follows:

```
exec ord_dicom.setdatamodel;
```

See setDataModel( ) Procedure in Chapter 4 (and editDataModel( ) Procedure in Chapter 9) for reference information.

3.2 Accessing Information about Documents in the Repository

A number of information views are available to users of the DICOM repository. These information views provide details about the documents in the repository, including names of documents, types of documents, names of constraints, and constraint validation messages.

Table 3–2 lists the public information views that are available to users (and administrators).

Table 3–2 Public Information Views

Name	Access Category
orddcm_conformance_vld_msgs	Public (messages for user's schema only)
orddcm_constraint_names	Public
orddcm_document_types	Public
orddcm_documents	Public

Users (and administrators) commonly use the orddcm_documents and orddcm_document_types views to view details for the documents in the repository. The orddcm_documents view lists details of the documents stored in the repository.

The orddcm_document_types view identifies the supported Oracle Multimedia DICOM document types with a list of codes.

Two other public information views are also available that provide information about constraints. The orddcm_constraint_names view lists the names of the constraints installed in the repository. The orddcm_conformance_vld_msgs view lists the constraint messages that are generated for a set of constraints during a validation operation. This view lists only the constraint messages for the current user.

See Chapter 4 for details about public information views.

For administrators only, Oracle Multimedia DICOM provides the orddcm_document_refs information view. This view lists the documents in the repository that are referenced by other documents in the repository. See orddcm_document_refs in Chapter 9 for details about this administrator view.

3.3 Loading DICOM Content

You can use the SQL*Loader utility to load DICOM content into existing tables in Oracle Database.

To load DICOM content, first you must create a table with the appropriate columns and initialize those columns. Then, call the SQL*Loader utility and load the DICOM content from your data files into the table columns as SecureFile LOBs (see *Oracle Multimedia User's Guide*). Before performing any operations on the DICOM content, you must call the setDataModel( ) procedure to load the DICOM data model.

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 private DICOM content anonymous.

See Section 7.3 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 *Oracle Database Utilities* for more information about using the SQL*Loader utility to load objects and LOBs into Oracle Database.

3.4 Extracting DICOM Metadata

Oracle Multimedia DICOM provides support for extracting metadata from DICOM content. By searching the extracted metadata, applications can search the DICOM content.

To extract all the DICOM attributes into an XML document that conforms to the default metadata XML schema, first call the setProperties( ) method to store the extracted 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 a column of the same table where the ORDDicom object is stored. This metadata column can be bound to the application-specific XML schema.

3.4.1 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 with Oracle XML DB as a global XML schema. (See Example 10–1 for a sample of this task.)

   See *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 11.2.3 for detailed information about creating mapping documents.

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

   See Section 8.5 for information about inserting mapping documents into the repository.

3.4.2 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 orddcm_documents in Chapter 4 for reference information about this view.

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 extractMetadata( ) in Chapter 5 for reference information about this method.

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

3.5 Searching and Retrieving 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 setProperties( ) in Chapter 5 for reference information.)

To search and retrieve attributes within the metadata XML document, call either of these SQL functions: extractValue( ) or extract( ) by specifying the XPath expression for the attributes. (See *Oracle Database SQL Language Reference* for reference information.) To make searching faster, the attributes in the metadata XML document can also be indexed. (See *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 getAttributeByName( ) method. (See getAttributeByTag( ) and getAttributeByName( ) in Chapter 5 for reference information.) This process is not recommended for large tables, or for frequent retrievals of attributes. However, if you do use this process, Oracle recommends building function-based indexes on these methods to make searching faster.

Note: Before you can retrieve DICOM attributes, you must call the setProperties( ) method to populate the attributes of the ORDDicom object.

3.6 Writing and Editing DICOM Metadata

Oracle Multimedia DICOM provides support for creating new ORDDicom objects from existing ORDDicom objects with metadata embedded within the DICOM content that has been modified or overwritten. The writeMetadata( ) method creates a new copy of the ORDDicom object from the original ORDDicom object and the modified metadata. The original ORDDicom object is preserved.

Perform the following tasks to write and edit DICOM metadata:

1. Extract the DICOM metadata from DICOM content by calling the extractMetadata( ) method, using the Oracle default mapping document.

   See extractMetadata( ) in Chapter 5 for reference information about this method.

2. Add or modify DICOM attributes in the extracted metadata XML document.

   See Chapter 8 for 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 in Chapter 5 for reference information about constructors.

4. Call the writeMetadata( ) method to write the modified 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 the metadata in the DICOM content and object attributes of the new ORDDicom object.

   See writeMetadata( ) in Chapter 5 for reference information about this method.

3.7 Processing, Converting, and Compressing DICOM Image Data

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

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:

  ```
  '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'
  ```

See Chapter 5 for reference information about the processCopy( ) methods and supported commands.

3.8 Creating DICOM Images from Secondary Capture Images

Oracle Multimedia DICOM provides the relational procedure createDICOMImage( ) to create DICOM format images from images in formats such as JPEG or TIFF in addition to DICOM metadata in data type XMLType.

Perform the following tasks to create DICOM format images from secondary images and DICOM metadata:

1. Load the image into a BLOB, or define the image as a BFILE type.
2. Create an XMLType object from the DICOM metadata of the corresponding image.
3. Create an empty BLOB object as the placeholder for the new DICOM image.
4. Create the DICOM format image using the createDICOMImage( ) procedure.

See Chapter 6 for reference information about the createDICOMImage( ) procedure.

3.9 Validating Conformance with DICOM Constraints

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

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

Conformance constraints are a collection of rules to use 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.

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 information view orddcm_constraint_names.

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

See Section 3.2 for more information about public information views.

Regardless of the interface you use, validating the conformance of DICOM content requires a combination of administrator and developer tasks. The following subsections describe these tasks. Administrator tasks must always be performed first.

3.9.1 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. This is the document where you 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 11.2.2 for information about how to create constraint documents.

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

   See Section 8.5.1 for information about inserting constraint documents into the repository.

3.9.2 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 orddcm_constraint_names in Chapter 4 for reference information about this view.

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

   ```
   select t.dicom_src.isConformanceValid('PatientModule')
    from medical_image_obj t;
   ```

   See isConformanceValid( ) in Chapter 5 for reference information about this method.

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 orddcm_conformance_vld_msgs in Chapter 4 for reference information about this view.

If your DICOM content does not conform to the constraint rules defined for your organization, you have the option of writing another ORDDicom object with corrected DICOM metadata. See Section 3.6 for information about writing DICOM metadata.

3.10 Protecting Private Patient Data

Oracle Multimedia DICOM provides support to protect the confidentiality of private patient data with the following methods:

- makeAnonymous( ) - Use this method to remove or replace private patient data. This method creates a new ORDDicom object, and preserves the original ORDDicom object.
- isAnonymous( ) - Use this method to check if the private patient data for a specified ORDDicom object has been removed or replaced, according to a specified anonymity document. In general, call this method before calling the makeAnonymous( ) method.

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 patient identifying information, as necessary.

3.10.1 Protecting Privacy - Administrator Tasks

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

1. Create an anonymity document that defines the DICOM attributes to be removed or replaced.

   See Section 11.2.1 for information about how to create anonymity documents.

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

   See Section 8.5.1 for information about inserting anonymity documents into the repository.

3.10.2 Protecting Privacy - Developer Tasks

As a developer, perform the following tasks to complete the process of making private 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 orddcm_documents in Chapter 4 for reference information about this view.

2. Create an empty ORDDicom object using the empty ORDDicom constructor if you plan to call the makeAnonymous( ) method. The empty ORDDicom object will be 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());
   ```

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

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

See isAnonymous( ) in Chapter 5 for reference information about this method.

4. Call the makeAnonymous( ) method to copy and make the original ORDDicom object anonymous, and then write the new DICOM content into the 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 makeAnonymous( ) in Chapter 5 for reference information about this method.

4

DICOM Data Model Utility Reference

Oracle Multimedia describes the DICOM data model utility in the ORD_DICOM package. This 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)

Oracle Multimedia contains the following information about the DICOM data model utility:

This chapter describes the functions, procedures, and information views in the DICOM data model utility interface, which operate on the DICOM data model repository. For information about other DICOM application programming interfaces (APIs), see the following chapters:

- Chapter 5 - ORDDicom object API
- Chapter 6 - DICOM relational API
- Chapter 9 - ORD_DICOM_ADMIN data model repository API

See *Oracle Multimedia DICOM Java API Reference* for information about the DICOM Java API.

DICOM Data Model Utility Functions and Procedures

The ORD_DICOM package defines the following DICOM data model utility functions and procedures:

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 is `'DICOM'`, which 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.

Note: Call the setDataModel( ) procedure at the following times:

- At the beginning of each database session
- Whenever the application needs to see new data model changes

---

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 'ordcmmp.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 `'ordcmmp.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.

Note: Call the setDataModel( ) procedure at the following times:

- At the beginning of each database session
- Whenever the application needs to see new data model changes

---

To see a list of all the mapping documents in the data model repository, query the public information view orddcm_documents.

Examples

Example 1:

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

```
select ord_dicom.getMappingXPath('00100010') as 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 extract(t.metadata, ord_dicom.getMappingXPath('00100010'),
'xmlns="http://xmlns.oracle.com/ord/dicom/metadata_1_0"') 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:

```
select ord_dicom.getMappingXPath('00082218') as 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 extract(t.metadata, ord_dicom.getMappingXPath('00082218'),
'xmlns="http://xmlns.oracle.com/ord/dicom/metadata_1_0"') 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><
SHORT_STRING tag="00080100" definer="DICOM" name="Code Value" offset="606" lengt
h="8">T-11170</SHORT_STRING><SHORT_STRING tag="00080102" definer="DICOM" name="C
oding Scheme Designator" offset="622" length="4">SNM3</SHORT_STRING><LONG_STRING
 tag="00080104" definer="DICOM" name="Code Meaning" offset="634" length="8">Maxi
lla</LONG_STRING></ITEM></SEQUENCE>

1 row selected.
```

where:

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

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 needs to see new data model changes.

Parameters

modelName
The model name of the data model repository. The default name is `'DEFAULT'`, which is the only value supported in Oracle Database 11g Release 1 (11.1).

Pragmas

None.

Exceptions

None.

Usage Notes

You may want to call the setDataModel( ) procedure only once during a database session. Subsequent calls to this procedure may result in changed behavior if the data model has changed since you made the original call to the setDataModel( ) procedure.

To use the same DICOM data model throughout a session, Oracle recommends following the call to the setDataModel( ) 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 will be deleted during a rollback operation if the call to the setDataModel( ) procedure is made within the transaction that is being rolled 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.

Examples

Load the repository from the database into memory:

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

DICOM Repository Public Information Views

This section describes the Oracle Multimedia DICOM repository public information views, which are the following:

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 information view lists the constraint messages generated during constraint validation. The public read and delete access privileges are granted for this information view.

Usage Notes

This information 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.          PatientModule        Validation error:       log                 01-MAR-07
3384726461.899958945.                         missing mandatory                           01.40.21.158337 PM
2180235641.3197827030                         attribute for
                                              patient module


1.2.840.114346.          PatientModule        Warning:                warning             01-MAR-07
3384726461.899958945.                         validation failure                          01.40.21.168322 PM
2180235641.3197827030


2 rows selected.
```

orddcm_constraint_names

Format

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

Description

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

Usage Notes

Before querying this information view, call the setDataModel( ) procedure at least once during the database session. Call the procedure again whenever the application needs to see new data model changes.

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 information view lists details of the documents stored in the repository. The public read access privilege is granted for this information view.

Usage Notes

Before querying this information view, call the setDataModel( ) procedure at least once during the database session. Call the procedure again whenever the application needs to see new data model changes.

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, which are available upon installation.

```
--------------------------------------------------------------------------------
DOC_NAME          DOC_TYPE               CREATE_DA        INSTALLED_BY_ORACLE
--------------------------------------------------------------------------------
ordcmpv.xml       PRIVATE_DICTIONARY     04-OCT-06        1
ordcmmp.xml       MAPPING                04-OCT-06        1
ordcman.xml       ANONYMITY              04-OCT-06        1
ordcmpf.xml       PREFERENCE             04-OCT-06        1
ordcmui.xml       UID_DEFINITION         04-OCT-06        1
ordcmcmc.xml      CONSTRAINT             04-OCT-06        1
ordcmcmd.xml      CONSTRAINT             04-OCT-06        1
ordcmct.xml       CONSTRAINT             04-OCT-06        1
ordcmsd.xml       STANDARD_DICTIONARY    04-OCT-06        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 the following values:
		`STANDARD_DICTIONARY`
		`PRIVATE_DICTIONARY`
		`CONSTRAINT`
		`MAPPING`
		`ANONYMITY`
		`PREFERENCE`
		`UID_DEFINITION`

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 the following 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`

DOC_TYPE_DESC	VARCHAR2(4000)	Document type description
		This column contains the following 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`

Description

This read-only information view identifies the supported Oracle Multimedia DICOM document types. Use this information 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 information 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

7 rows selected.
```

Part II

DICOM Development

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

Part II contains the following chapters:

5

ORDDicom Reference

Oracle Multimedia describes 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)

Oracle Multimedia contains the following information about the ORDDicom object type:

- ORDDicom Object Type on on page 5-3
- ORDDicom Constructors on page 5-4
- ORDDicom Methods on page 5-9

This chapter describes the attributes, constructors, and methods in the ORDDicom object interface. For information about other DICOM application programming interfaces (APIs), see the following chapters:

- Chapter 4 - DICOM data model utility API
- Chapter 6 - DICOM relational API
- Chapter 9 - ORD_DICOM_ADMIN data model repository API

See *Oracle Multimedia DICOM Java API Reference* for information about the DICOM Java API.

5.1 ORDDicom Object Example Media Table and Directory Definition

The methods described in this reference chapter show examples based on the MEDICAL_IMAGE_OBJ table, which these examples create in the Product Media (PM) sample schema. Refer to the MEDICAL_IMAGE_OBJ table definition that follows when reading through the examples. See *Oracle Database Sample Schemas* for information about the sample schemas.

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 DICOM files you have for the ones shown 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.

5.1.1 Directory Definition

The following statements must be issued before executing the examples, where "/mydir/work" is the directory where the user (pm) can find the DICOM files:

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

5.1.2 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/<pm-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

Oracle Multimedia creates the ORDDicom object type, which 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 `ordcspec.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

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

- ORDDicom( ) for BLOBs
- ORDDicom( ) for ORDImage
- ORDDicom( ) for other sources

The ORDDicom object has embedded BLOB attributes. BLOB locators must be initialized before they can be accessed. Thus, newly constructed ORDDicom objects (except when constructed from a temporary BLOB) must be inserted into a table before calling object member methods on these ORDDicom objects. This section describes the ORDDicom constructors.

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 or not 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 or not 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 for migrating 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'`.

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 or not 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', 'DICOMDIR', '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', 'DICOMDIR', '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 on the ORDDicom methods, which are the following:

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.

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.

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', 'ordcmmp.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, in a VARCHAR2 string for DICOM attributes other than SQ type attributes. For SQ type attributes, this method returns a segment of XML code in a VARCHAR2 string. This method is a helper method only.

Parameters

attributeName
The name of a specified attribute or item.

definerName
The name of the attribute definer. The default name is `'DICOM'`.

Pragmas

None.

Exceptions

None.

Usage Notes

Use this method to retrieve any single attribute in the embedded DICOM content. For performance reasons, do not use this method to retrieve more than two or three attributes.

Before calling this method for the first time, call the setProperties( ) method.

This method is not recommended for use in applications that require maximum performance.

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, in a VARCHAR2 string for DICOM attributes other than SQ type attributes. For SQ type attributes, this method returns a segment of XML code in a VARCHAR2 string. This method is a helper method only.

Parameters

tag
The code value used to specify a DICOM attribute or item tag, as a hexadecimal string. To access child attributes of the sequence type (SQ), use the `"."` notation. For example: `"00082218.00080100"` returns the code value (tag `"00080100"`) of anatomic region sequence (tag `"00082218"`). And `"00080005[2]"` returns the second item value of the specific character set attribute (tag `"00080005"`).

definerName
The name of the attribute definer. The default name is `'DICOM'`.

Pragmas

None.

Exceptions

None.

Usage Notes

Use this method to retrieve any single attribute in the embedded DICOM content. For performance reasons, do not use this method to retrieve more than two or three attributes.

Before calling this method for the first time, call the setProperties( ) method.

This method is not recommended for use in applications that require maximum performance.

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;
```

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;
```

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;
```

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 already 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 or not 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.

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 or not 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 or not patient identifying information has been removed from the embedded DICOM content.

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 or not 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 information view orddcm_conformance_vld_msgs. The public information 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

None.

Examples

Check if the ORDDicom object is conformance valid. Then, show any conformance validation messages that are generated.

```
select t.dicom_src.isConformanceValid('PatientModule')
 from medical_image_obj t;

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:

```
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;
/
```

where:

- <*unique-UID*>: 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.

processCopy( ) to BLOBs

Format

```
processCopy (SELF   IN ORDDicom,
             command IN VARCHAR2,
             dest IN OUT NOCOPY BLOB)
```

Description

Copies the ORDDicom image 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 an image processing operator as input. Valid values include: `frame`, `contentFormat`, `fileFormat`, `compressionFormat`, `cut`, `scale`, and `rotate`. See the description for the process( ) method in *Oracle Multimedia Reference* for a complete list of image processing operators and details on each operator.

dest
The destination BLOB that contains the output of the process command on the ORDDicom image.

Pragmas

None.

Exceptions

None.

Usage Notes

Use this method to process an image into a BLOB after copying it from the ORDDicom object. In this case, the output in the BLOB is a raster image.

In addition, you can use this method to process waveform or video DICOM content. In this case, the output in the BLOB is a video file in AVI format.

See Appendix C for information about the encoding rules that support image content processing.

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 image object that is input into a destination ORDDicom image object, and then performs the specified processing operations on the destination ORDDicom image object. Only the DICOM attributes of the destination ORDDicom image are updated with image information. 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: `compressionFormat`, `frame`, `contentFormat`, `cut`, `scale`, and `rotate`. See the description for the process( ) method in *Oracle Multimedia Reference* for a complete list of image processing operators and details on each operator.

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 image with the new metadata.

metadata
The new metadata to be written into the new DICOM image.

Pragmas

None.

Exceptions

None.

Usage Notes

Use this method to modify or fix the image data in the current embedded DICOM content.

In addition, you can use this method to create an ORDDicom object with the image content compressed using JPEG or JPEG 2000.

You can also use this method to extract a single frame into an ORDDicom object from a multiframe ORDDicom object.

See Appendix C for information about the encoding rules that support image content processing.

Examples

Copy the ORDDicom object into an ORDDicom image object and then process it:

```
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;
/
```

where:

- *<unique-UID>*: 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.

processCopy( ) to ORDImage

Format

```
processCopy (SELF   IN ORDDicom,
             command IN VARCHAR2,
             dest IN OUT NOCOPY ORDImage)
```

Description

Copies the ORDDicom image 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 the description for the process( ) method in *Oracle Multimedia Reference* for a complete list of image processing operators and details on each operator.

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 an 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.

Examples

Copy the ORDDicom object 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 all the embedded DICOM content attributes are extracted into the metadata attribute of ORDDicom object. The XML metadata conforms to the default metadata schema namespace `http://xmlns.oracle.com/ord/dicom/metadata_1_0`.

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.

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.

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;
/
```

6

DICOM Relational Interface Reference

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, and who do not want to migrate their existing medical image applications to use Oracle Multimedia ORDDicom objects, can use the Oracle Multimedia DICOM relational interface for managing their medical image data.

The DICOM relational interface defines 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.

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)

Oracle Multimedia contains the following information about the DICOM relational interface:

- DICOM Relational Functions on page 6-3
- DICOM Relational Procedures on page 6-13

This chapter describes the functions and procedures in the DICOM relational interface. For information about other DICOM application programming interfaces (APIs), see the following chapters:

- Chapter 4 - DICOM data model utility API
- Chapter 5 - ORDDicom object API
- Chapter 9 - ORD_DICOM_ADMIN data model repository API

See *Oracle Multimedia DICOM Java API Reference* for information about the DICOM Java API.

6.1 DICOM Relational Example Media Table and Directory Definition

The functions and procedures described in this reference chapter show examples based on the MEDICAL_IMAGE_REL table, which these examples create in the Product Media (PM) sample schema. Refer to the MEDICAL_IMAGE_REL table definition that follows when reading through the examples. See *Oracle Database Sample Schemas* for information about the sample schemas.

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( ) method. Substitute DICOM files you have for the ones shown in the examples.

6.1.1 Directory Definition

The following statements must be issued before executing the examples, where "/mydir/work" is the directory where the user (pm) can find the DICOM files:

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

6.1.2 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/<pm-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 the following DICOM relational functions:

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 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`.

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 the 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. This document refers to the default metadata namespace `http://xmlns.oracle.com/ord/dicom/metadata_1_0`.

Pragmas

None.

Exceptions

None.

Usage Notes

None.

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', 'ordcmmp.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 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`.

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 the 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. This document refers to the default metadata namespace `http://xmlns.oracle.com/ord/dicom/metadata_1_0`.

Pragmas

None.

Exceptions

None.

Usage Notes

None.

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', 'ordcmmp.xml');
end;
```

extractMetadata( ) for ORDImage

Format

```
extractMetadata (
  data IN ORDSYS.ORDImage,
  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`.

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 the 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. This document refers to the default metadata namespace `http://xmlns.oracle.com/ord/dicom/metadata_1_0`.

Pragmas

None.

Exceptions

None.

Usage Notes

None.

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', 'ordcmmp.xml');
end;
/
```

isAnonymous( ) for BFILEs

Format

```
isAnonymous(
  src IN BFILE,
  anonymityDocName IN VARCHAR2 DEFAULT 'ordcman.xml')
RETURN INTEGER
```

Description

Determines whether or not 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 or not 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 or not 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 or not 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 information view orddcm_conformance_vld_msgs (see Chapter 4).

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

None.

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
select ord_dicom.isConformanceValid(t.bfile_src, 'PatientModule')
  from medical_image_rel t where id = 1;

-- 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 or not 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 information view orddcm_conformance_vld_msgs (see Chapter 4).

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

None.

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
select ord_dicom.isConformanceValid(t.blob_src, 'PatientModule')
  from medical_image_rel t where id = 1;

-- 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 or not 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 information view orddcm_conformance_vld_msgs (see Chapter 4).

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

None.

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
select ord_dicom.isConformanceValid(t.image_src, 'PatientModule')
  from medical_image_rel t where id = 1;

-- 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 the following DICOM relational procedures:

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 the 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.

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 the 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.

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 the 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.

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

None.

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

None.

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:

```
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;
/
```

where:

- <*unique-UID*>: 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.

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:

```
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;
/
```

where:

- <*unique-UID*>: 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.

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:

```
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;
/
```

where:

- <*unique-UID*>: 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.

processCopy( ) for BFILEs

Format

```
processCopy (
   src IN BFILE,
   command IN VARCHAR2,
   dest IN OUT NOCOPY BLOB)
```

Description

Processes and copies the input DICOM image into a new raster image. The input DICOM image remains unchanged.

Parameters

src
The input DICOM image stored in the source BFILE.

command
A command string that accepts an image processing operator as input. Valid values include: `fileFormat`, `frame`, `contentFormat`, `compressionFormat`, `cut`, `scale`, and `rotate`. See the description for the process( ) method in *Oracle Multimedia Reference* for a complete list of image processing operators and details on each operator.

dest
An empty BLOB in which to store the destination image.

Pragmas

None.

Exceptions

None.

Usage Notes

See Appendix C for information about the encoding rules that support image content processing.

Examples

Copy the DICOM image 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 BLOBs

Format

```
processCopy (
   src IN BLOB,
   command IN VARCHAR2,
   dest IN OUT NOCOPY BLOB)
```

Description

Processes and copies the input DICOM image into a new raster image. The input DICOM image remains unchanged.

Parameters

src
The input DICOM image stored in the source BLOB.

command
A command string that accepts an image processing operator as input. Valid values include: `fileFormat`, `frame`, `contentFormat`, `compressionFormat`, `cut`, `scale`, and `rotate`. See the description for the process( ) method in *Oracle Multimedia Reference* for a complete list of image processing operators and details on each operator.

dest
An empty BLOB in which to store the destination image.

Pragmas

None.

Exceptions

None.

Usage Notes

See Appendix C for information about the encoding rules that support image content processing.

Examples

Copy the DICOM image 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 ORDImage

Format

```
processCopy (
   src IN ORDSYS.ORDImage,
   command IN VARCHAR2,
   dest IN OUT NOCOPY BLOB)
```

Description

Processes and copies the input DICOM image into a new raster image. The input DICOM image remains unchanged.

Parameters

src
The input DICOM image 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 the description for the process( ) method in *Oracle Multimedia Reference* for a complete list of image processing operators and details on each operator.

dest
An empty BLOB in which to store the destination image.

Pragmas

None.

Exceptions

None.

Usage Notes

See Appendix C for information about the encoding rules that support image content processing.

Examples

Copy the DICOM image 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 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

Processes and copies the input DICOM image into a new DICOM image or raster image. The input DICOM image remains unchanged.

Parameters

src
The input DICOM image stored in the source BFILE.

command
A command string that accepts an image processing operator as input. Valid values include: `frame`, `contentFormat`, `compressionFormat`, `cut`, `scale`, and `rotate`. See the description for the process( ) method in *Oracle Multimedia Reference* for a complete list of image processing operators and details on each operator.

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 image.

metadata
The new metadata to be written into the new DICOM image.

Pragmas

None.

Exceptions

None.

Usage Notes

See Appendix C for information about the encoding rules that support image content processing.

Examples

Copy the DICOM image from a BFILE into a BLOB with a specified SOP instance UID and then process it:

```
  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;
/
```

where:

- <*unique-UID*>: 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.

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

Processes and copies the input DICOM image into a new DICOM image or raster image. The input DICOM image remains unchanged.

Parameters

src
The input DICOM image stored in the source BLOB.

command
A command string that accepts an image processing operator as input. Valid values include: `frame`, `contentFormat`, `compressionFormat`, `cut`, `scale`, and `rotate`. See the description for the process( ) method in *Oracle Multimedia Reference* for a complete list of image processing operators and details on each operator.

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 image.

metadata
The new metadata to be written into the new DICOM image.

Pragmas

None.

Exceptions

None.

Usage Notes

See Appendix C for information about the encoding rules that support image content processing.

Examples

Copy the DICOM image from a BLOB into another BLOB with a specified SOP instance UID and then process it:

```
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;
/
```

where:

- <*unique-UID*>: 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.

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

Processes and copies the input DICOM image into a new DICOM image or raster image. The input DICOM image remains unchanged.

Parameters

src
The input DICOM image 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 the description for the process( ) method in *Oracle Multimedia Reference* for a complete list of image processing operators and details on each operator.

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 image.

metadata
The new metadata to be written into the new DICOM image.

Pragmas

None.

Exceptions

None.

Usage Notes

See Appendix C for information about the encoding rules that support image content processing.

Examples

Copy the DICOM image from an ORDImage object into a BLOB with a specified SOP instance UID and then process it:

```
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;
/
```

where:

- <*unique-UID*>: 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.

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.

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.

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.

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;
/
```

7

DICOM Application Development

This chapter describes how to develop applications using Oracle Multimedia DICOM. Using either the ORDDicom object type interface, the relational interface, or the DICOM Java interface Oracle Multimedia DICOM provides support for Oracle Database so 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.

This chapter includes the following sections:

Oracle Multimedia provides more features than those that are described in this chapter. For additional information, see the following documents in the Oracle Multimedia software documentation set:

- *Oracle Multimedia Reference*
- *Oracle Multimedia User's Guide*
- *Oracle Multimedia DICOM Java API Reference*
- *Oracle Multimedia Java API Reference*
- *Oracle Multimedia Servlets and JSP Java API Reference*

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/`

7.1 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

For a user scott to use these examples, the following statements must be issued before scott executes the examples, where /mydir/work is the directory where scott will find the DICOM files:

```
CONNECT /as sysdba
CREATE OR REPLACE DIRECTORY FILE_DIR as '/mydir/work';
GRANT READ ON DIRECTORY FILE_DIR TO 'scott';
```

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

7.2 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 snippet shown in Example 7–1 creates the table medical_image_table, with the four columns id, dicom, imageThumb, and anonDicom.

Example 7–1 Create a Table for DICOM Content

```
CONNECT scott/tiger

create table medical_image_table
             (id          integer primary key,
              dicom       ordsys.orddicom,
              imageThumb  ordsys.ordimage,
              anonDicom   ordsys.orddicom)
--
-- Use SecureFile LOBS for media content
--
lob(dicom.source.localData)      store as SecureFile,
lob(imageThumb.source.localData) store as SecureFile,
lob(anonDicom.source.localData)  store as SecureFile;
```

Example 7–1 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 *Oracle Database SecureFiles and Large Objects Developer's Guide* for more information about SecureFile LOBs. See *Oracle Multimedia User's Guide* for tuning tips with SecureFile LOBs.

7.3 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 machine that differs from the machine that is running the server for Oracle Database. The data can also be loaded locally on the same machine as the database server. For more information on SQL*Loader, see *Oracle Database Utilities*.

A typical SQL*Loader session accepts a control file and one or more data files as input. The control file defines how the data will be loaded 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 7–2 shows a control file for loading DICOM data into the table `medical_image_table`, which you created in Section 7.2. 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 7–2 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.
--
--    sqlldr userid=USER/PASSWORD  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 initalized 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. For more information about LOBs and logging, see *Oracle Database SecureFiles and Large Objects Developer's Guide*.

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:

```
sqlldr userid=USER/PASSWORD  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 7–3. 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.

Example 7–3 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.

Example 7–3 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 7–3 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. For more information about LOBs and logging, see *Oracle Database SecureFiles and Large Objects Developer's Guide*.

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);
```

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

7.4 Developing DICOM Applications Using the PL/SQL API

This section shows basic PL/SQL code examples that store and manipulate DICOM content inside a database using Oracle Multimedia DICOM.

Oracle Multimedia DICOM allows you to store DICOM content in database tables with columns of type ORDDicom. Table 7–1 shows some of the attributes that are contained within an ORDDicom object in a database table.

Table 7–1 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

7.4.1 Selecting DICOM Attributes

This section shows how to access DICOM attributes from the DICOM content you loaded in Section 7.3.

After the table `medical_image_table` is populated and metadata has been extracted, you can access metadata using SQL queries. Example 7–4 demonstrates how to select extracted DICOM metadata from the DICOM content. The code statements where these operations are performed are highlighted in bold.

Example 7–4 Selected 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 id,
        extractValue(t.dicom.metadata,
        '/DICOM_OBJECT/*[@name="Patient''s Name"]/VALUE',
        'xmlns=http://xmlns.oracle.com/ord/dicom/metadata_1_0') as "PATIENT_NAME",
        extractValue(t.dicom.metadata,
        '/DICOM_OBJECT/*[@name="Patient ID"]',
        'xmlns=http://xmlns.oracle.com/ord/dicom/metadata_1_0') as "PATIENT_ID",
        extractValue(t.dicom.metadata,
        '/DICOM_OBJECT/*[@name="Modality"]',
        'xmlns=http://xmlns.oracle.com/ord/dicom/metadata_1_0') as "MODALITY"
from medical_image_table t;
```

Running Example 7–4 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
```

7.4.2 Creating Thumbnail Images and Changing Image Formats

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

As an example, to create a JPEG thumbnail image from a DICOM image, a new ORDImage object is generated 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 snippet shown in Example 7–5 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 7–5 where these tasks are performed are highlighted in bold.

Example 7–5 Generate and Process 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 7–5 generates the following output:

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

7.4.3 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 should be removed or replaced and what action should be taken to make each attribute anonymous.

The default anonymity document, `ordcman.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 snippet in Example 7–6 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 snippet, 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 7–6 where the makeAnonymous( ) method is called is highlighted in bold.

Example 7–6 Populate the Column and Generate 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);

-- look at our handiwork
select id,
        extractValue(t.anonDicom.metadata,
        '/DICOM_OBJECT/*[@name="Patient''s Name"]/VALUE',
        'xmlns=http://xmlns.oracle.com/ord/dicom/metadata_1_0') as "PATIENT_NAME",
        extractValue(t.anonDicom.metadata,
        '/DICOM_OBJECT/*[@name="Patient ID"]',
        'xmlns=http://xmlns.oracle.com/ord/dicom/metadata_1_0') as "PATIENT_ID"
from medical_image_table t;
```

Running Example 7–6 generates the following output:

```
ID PATIENT_NAME                   PATIENT_ID
--- ------------------------------ ----------
 1  Joe^Smith                      anonymous
```

7.4.4 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.

Example 7–7 checks the conformance of the column `dicom` of the table `medical_image_table`.

The code statement in Example 7–7 where this task is performed is highlighted in bold.

Example 7–7 Check DICOM Conformance

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

select id, t.dicom.isconformanceValid('OracleOrdObject') as conformant
from medical_image_table t;
```

Running Example 7–7 generates the following output:

```
ID  CONFORMANT
--- ----------
1     1
```

If the DICOM content used in this example had not conformed to the Oracle default constraint definitions, a message or messages would have been inserted into a table

that you can see by querying the information view orddcm_conformance_vld_msgs. This view lists the constraint messages generated during constraint validation.

The following code snippet 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)
```

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 3.9 for information about what to do if your DICOM content does not conform to the constraint rules defined for your organization.

7.4.5 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 users to know what went wrong during program operation. Proper error handling practices ensure that, whenever possible, you will always be able to recover from an error while running an application. In addition, proper error handling provides you with the information you need so that you will always know what went wrong.

When handling exceptions, PL/SQL uses exception blocks. For example, in PL/SQL, the exception may 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. For more information about handling PL/SQL exceptions, see *Oracle Database PL/SQL Language Reference*.

7.5 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 users 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.

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

7.5.1 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 it includes the appropriate Oracle Java libraries for the Oracle Multimedia and other features you intend to use. For each Java library, the following table lists the name of the Java library, the Oracle Multimedia or other features that require that library, details about the JDK version that supports the library, the platform, and the path name under the *<ORACLE_HOME>* directory where you can obtain the library JAR file.

Name of Oracle Java Library	Related Feature	JDK Version, Platform, and Location
Oracle JDBC library	All Oracle Multimedia features	JDK 5 or later, on Linux and UNIX: *<ORACLE_HOME>*`/jdbc/lib/ojdbc5.jar` JDK 5 or later, on Windows: *<ORACLE_HOME>*`\jdbc\lib\ojdbc5.jar`
Oracle Multimedia Java classes library	All Oracle Multimedia features	JDK 5 or later, on Linux and UNIX: *<ORACLE_HOME>*`/ord/jlib/ordim.jar` JDK 5 or later, on Windows: *<ORACLE_HOME>*`\ord\jlib\ordim.jar`
Oracle Multimedia DICOM Java classes library	DICOM feature	JDK 5 or later, on Linux and UNIX: *<ORACLE_HOME>*`/ord/jlib/orddicom.jar` JDK 5 or later, on Windows: *<ORACLE_HOME>*`\ord\jlib\orddicom.jar`
Oracle XDB Java classes library	DICOM feature Oracle Multimedia metadata extraction	JDK 5 or later, on Linux and UNIX: *<ORACLE_HOME>*`/rdbms/jlib/xdb.jar` JDK 5 or later, on Windows: *<ORACLE_HOME>*`\rdbms\jlib\xdb.jar`
Oracle Multimedia Java classes for servlets and JSP library	Java servlets and JSP applications	JDK 5 or later, on Linux and UNIX: *<ORACLE_HOME>*`/ord/jlib/ordhttp.jar` JDK 5 or later, on Windows: *<ORACLE_HOME>*`\ord\jlib\ordhttp.jar`

Name of Oracle Java Library	Related Feature	JDK Version, Platform, and Location
NLS Character Set Conversion library(Optional) *<ORACLEHOME>*/jlib/orai18n.jar	NLS character set conversion required[1]	JDK 5 or later, on Linux and UNIX: *<ORACLE_HOME>*/jlib/orai18n.jar JDK 5 or later, on Windows: *<ORACLE_HOME>*\jlib\orai18n.jar

[1] If NLS character set conversion is required between the client application and the database, you must include the orai18n.jar file in the CLASSPATH variable. If NLS character set conversion is required, but the appropriate library is not specified, character-based attributes of Oracle Multimedia object types may be returned as hex-encoded strings. See *Oracle Database JDBC Developer's Guide and Reference* for more information about NLS character set conversion.

Note: If you are using the JDBC OCI driver, you must specify the location of the JDBC OCI shared library in one of the following variables: LD_LIBRARY_PATH (for Linux or UNIX) or PATH (for Windows).

Depending on your platform, store the JDBC OCI shared library at one of the following locations under the *<ORACLE_HOME>* directory:

<ORACLE_HOME>/lib (for libocijdbc11.so on Linux and UNIX)
<ORACLE_HOME>\bin (for ocijdbc11.dll on Windows)

Because this library path is shared, it may have been specified previously to enable the use of other client applications such as SQL*Plus.

7.5.2 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 into 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;
```

7.5.3 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 users to know what went wrong during program operation. Proper error handling practices ensure that, whenever possible, you will always be able to recover from an error while running an application. In addition, proper error handling provides you with the information you need so that you will always know what went wrong.

When handling exceptions, Java uses the try/catch block. For example, in Java, the exception may 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. For more information about handling Java exceptions, see *Oracle Database Java Developer's Guide* and *Oracle Database JDBC Developer's Guide and Reference*.

Part III

DICOM Administration

This part includes user and reference information for administrators of the DICOM data model repository.

Part III contains the following chapters:

8

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. 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 allowed to make changes in the data model repository. Administrators of the DICOM data model repository are assigned the ORDADMIN role.

Administrators can perform a number of repository management tasks. This chapter provides guidelines for the following tasks: loading the repository, and retrieving, adding, and removing content from the repository. See Chapter 11 for detailed 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 9 for detailed reference information about the ORD_DICOM_ADMIN data model repository API.

In addition, administrators can use the following APIs, which are also provided by Oracle Multimedia DICOM:

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

Note: Keep these administrative guidelines in mind:

- 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 publishDataModel( ) Procedure in Chapter 9 for more information about this procedure.
- The editDataModel( ) procedure enables administrators to lock the data model while making changes.

This chapter includes the following sections:

Table 8–1 provides cross-references for administrators to other locations within the Oracle Multimedia software documentation set where you can access additional information about topics mentioned in this chapter.

Table 8–1 Additional References for Administrators

Topic	More Information
Reference information for public information views	Chapter 4
Reference information for administrator information views	Chapter 9
Reference information for the DICOM data model utility API	Chapter 4
Reference information for the ORDDicom object API	Chapter 5
Reference information for the DICOM relational API	Chapter 6
Reference information for the ORD_DICOM_ADMIN data model repository API	Chapter 9
Reference information for the DICOM Java API	*Oracle Multimedia DICOM Java API Reference*
Examples of operations on DICOM content	Chapter 7
Examples of administrative operations in the data model repository	Chapter 10
Information about writing configuration documents	Chapter 11
Listings of the DICOM XML schemas	Appendix B

8.1 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 a number of public information views. Only administrators can export, insert, or delete configuration documents from the data model repository. And, only administrators can query administrator only information views.

8.2 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 needs to see new data model changes.

Note: 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.)

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

```
exec ord_dicom.setdatamodel;
```

See setDataModel( ) Procedure in Chapter 4 for reference information.

Using the ORD_DICOM_ADMIN package, call the editDataModel( ) procedure as follows:

```
exec ord_dicom_admin.editDataModel();
```

See editDataModel( ) Procedure in Chapter 9 for reference information.

8.3 Browsing the Repository with Information Views

A number of public information views are available for browsing the DICOM repository. An administrator-only information view is also available to assist with DICOM repository administration. Information views provide details about documents in the repository, including names of documents, types of documents, references to other documents, names of constraints, and constraint validation messages.

Table 8–2 lists the information views and indicates which views are available to all users (public) or only to administrators.

Table 8–2 Administrator and Public Information Views

Name	Access Category
orddcm_conformance_vld_msgs	Public (messages for user's schema only)
orddcm_constraint_names	Public
orddcm_document_refs	Administrators only
orddcm_document_types	Public
orddcm_documents	Public

Administrators can use the orddcm_document_refs information view to see the list of documents that are referenced by other documents in the repository. This read-only information view is available to administrators only. See Chapter 9 for details about this view.

In addition, administrators (and other users) can use the orddcm_documents view to see the list of details about the documents stored in the repository. They can also use the orddcm_document_types view, which identifies the supported Oracle Multimedia DICOM document types by providing the list of codes for the document types. These public information views are read-only.

Two other public information views are available. The orddcm_constraint_names view lists the names of the constraints installed in the repository. The orddcm_conformance_vld_msgs view lists the constraint messages generated during constraint validation only for the schema that belongs to the user.

See Chapter 4 for details about public information views.

Administrators commonly use the orddcm_documents, orddcm_document_types, and orddcm_document_refs views when inserting, updating, and deleting documents from the repository.

8.4 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 needs to see new data model changes.

   See Chapter 4 for reference information about the setDataModel( ) procedure.

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 Chapter 9 for reference information about the getDocumentContent( ) function and the exportDocument( ) procedure.

8.5 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 9 for reference information about these procedures. See Chapter 10 for an example of the insertion process.

The following subsections briefly describe the insertion process for different types of documents.

8.5.1 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 with dependencies first. Then, insert the remaining documents.

8.5.2 Inserting Dictionary Documents

Inserting standard or private dictionary documents requires merging all the 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.

8.5.3 Inserting Preference and UID Definition Documents

To insert preference documents, first export the installed, Oracle-defined preference document, changing parameter values as required. Then, insert the updated user-defined preference document into the repository.

Administrators can insert user-defined UID definition documents to add new private UID values or to reflect updates in the DICOM standard. 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 allowed in the repository.

8.6 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 9 for reference information about these procedures. See Chapter 10 for an example of the update process.

The following subsections briefly describe the update process for different types of documents.

8.6.1 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.

8.6.2 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.

8.6.3 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.

8.7 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 9 for reference information about these procedures. See Chapter 10 for an example of the deletion process.

The following subsections briefly describe the deletion process for different types of documents.

8.7.1 Deleting Anonymity, Mapping, and Constraint Documents

For anonymity documents and mapping documents, the order of deletion is irrelevant. For constraint documents, however, the order of deletion is important. Constraint documents are deleted in reverse order from their insertion order. In addition, if there are dependencies among any of the constraint documents to be deleted, delete the documents with no dependencies first. Then, delete the remaining documents.

8.7.2 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.

8.7.3 Deleting Preference and UID Definition Documents

When a user-defined preference document is deleted, the values of the preference parameters revert back 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 back to the installed values from the default Oracle-defined UID definition document (`ordcmui.xml`).

9

ORD_DICOM_ADMIN Package Reference

Oracle Multimedia DICOM describes the ORD_DICOM_ADMIN package. This package lists 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 information views for the DICOM repository to be used by DICOM administrators.

> **Note:** Administrators of the DICOM data model repository must be assigned the ORDADMIN role. See Section 8.1 for more information about administrator privileges.

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)

Oracle Multimedia contains the following information:

- ORD_DICOM_ADMIN Data Model Repository Functions and Procedures on page 9-2
- DICOM Repository Administrator Information Views on page 9-13

This chapter describes the functions, procedures, and information views in the ORD_DICOM_ADMIN data model repository interface. For information about other DICOM APIs, see the following chapters:

- Chapter 4 - DICOM data model utility API
- Chapter 5 - ORDDicom object API
- Chapter 6 - DICOM relational API

See *Oracle Multimedia DICOM Java API Reference* for information about the DICOM Java API.

ORD_DICOM_ADMIN Data Model Repository Functions and Procedures

The ORD_DICOM_ADMIN package defines the following ORD_DICOM_ADMIN data model repository functions and procedures:

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 contents of the original documents.

Parameters

docName
The name of the document whose contents are 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.

> **Note:** 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.

Examples

Get the contents of a specified document (`ordcmpf.xml`) in the repository:

```
CONNECT pm/<pm-password>
-- user pm has been assigned the ORDADMIN role

exec ord_dicom.setDataModel;
 set long 5000;
 -- to see the contents on screen
 select ord_dicom_admin.getDocumentContent('ordcmpf.xml') from dual;
 -- to store in a table
 create table repos_exp_docs( name varchar2(100),  doc xmltype);
 insert into repos_exp_docs (name, doc) select 'ordcmpf.xml',
 ord_dicom_admin.getDocumentContent('ordcmpf.xml') from dual;
```

where:

- repos_exp_docs: a table in which to store the contents of the specified preference document.

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 8.7 for detailed 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.

In addition, Oracle recommends calling the exportDocument( ) procedure to make a copy of the specified document before deleting it from the repository.

Examples

Copy and then delete a document (`sample_pref.xml`) from the repository:

```
CONNECT pm/<pm-password>
-- user pm has been assigned the ORDADMIN role

exec ord_dicom_admin.editDataModel();
 -- list most recent docs first
 select doc_name from orddcm_documents order by create_date desc;
 -- export doc
 exec ord_dicom_admin.exportDocument('sample_pref.xml', 'EXPORT_DIR',
  'sample_pref_exp.xml');
 -- delete doc
 exec ord_dicom_admin.deleteDocument('sample_pref.xml');
 -- list remaining docs
 select doc_name from orddcm_documents order by create_date desc;
 -- publish changes and release locks
 exec ord_dicom_admin.publishDataModel();
```

where:

- EXPORT_DIR: an Oracle directory object for which the administrator has been granted the write privilege, and to which the contents of the specified file will 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.

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.

> **Note:** 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.

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', 'EXPORT_DIR',
     'ordcmpf_exp.xml');
```

where:

- EXPORT_DIR: an Oracle directory object for which the administrator has been granted the write privilege that contains the file where the contents of the specified file will be copied. (See Section 5.1.1 for an example of creating an Oracle directory object.)

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 information view orddcm_document_types (see Chapter 4). 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, except when 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 information view orddcm_documents (see Chapter 4) 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 information view orddcm_document_types (see Chapter 4). 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.

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. See *Oracle XML DB Developer's Guide* for information about registering XML schemas. (The metadata schema associated with the default mapping document is registered with Oracle XML DB upon installation of Oracle Multimedia DICOM. See Appendix B and Example 10–1 for more information.)

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;
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.

Parameters

None.

Pragmas

None.

Exceptions

None.

Usage Notes

Before calling this procedure, call the editDataModel( ) procedure.

Depending on when the administrator publishes changes to the data model repository, users who are currently connected to the repository may need 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 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.

Examples

Publish the changes to the repository:

```
exec ord_dicom_admin.editDataModel();
exec ord_dicom_admin.deleteDocument('sample_pref.xml');
select doc_name from orddcm_documents;
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.

Parameters

None.

Pragmas

None.

Exceptions

None.

Usage Notes

Before calling this procedure, call the editDataModel( ) procedure.

At end of the database session if you want to roll back the data model, call this procedure. Because the changes to the data model were terminated rather than published, the information 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();
exec ord_dicom_admin.deleteDocument('sample_pref.xml');
select doc_name from orddcm_documents;
exec ord_dicom_admin.rollbackDataModel();
select doc_name from orddcm_documents;
```

DICOM Repository Administrator Information Views

This section describes the Oracle Multimedia DICOM repository information view for administrators. The information view is called orddcm_document_refs.

See Chapter 4 for information about the Oracle Multimedia DICOM repository public information 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 information view lists the documents that are referenced by other documents. The select privilege is granted to the ORDADMIN role for this information view.

Usage Notes

Before querying this information view, call either the setDataModel( ) or the editDataModel( ) procedure. If you call the setDataModel( ) procedure, call it again whenever the application needs to see new data model changes.

> **Note:** 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.

Examples

Show the references between a set of Oracle-installed configuration documents:

```
SQL> select doc_name, ref_by_doc_name from ordsys.orddcm_doc_refs;

--------------------------------------------------------------------------------
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
```

10

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 a number of 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. Using information 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 the following sections:

See Chapter 9 for detailed 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/`

10.1 Sample Session: 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.

  Example 10–1 shows a sample of how to register a schema as global with Oracle XML DB.

Example 10–1 Registering a Global XML Schema

```
GRANT XDBADMIN TO QUINE;

Grant succeeded.

CONNECT quine/curry

Connected.

BEGIN
  DBMS_XMLSCHEMA.registerSchema(
    SCHEMAURL => 'http://xmlns.example.com/example.xsd',
    SCHEMADOC => bfilename('XMLDIR','example.xsd'),
    LOCAL => FALSE,
    GENTYPES => TRUE,
    GENTABLES => FALSE,
    CSID => nls_charset_id('AL32UTF8'));
END;
/
```

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

Perform the following steps:

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;
```

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;
```

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_types view:

```
select * from orddcm_document_types;
```

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.

10.2 Sample Session: 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 Example 10–1.)

  See *Oracle XML DB Developer's Guide* for information about registering XML schemas.

Perform the following steps:

Step 1 Edit the Data Model

As in Section 10.1, 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;
```

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');
```

The 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;
```

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;
```

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 10.1, publish the changes and unlock the data model, as follows:

```
exec ord_dicom_admin.publishDataModel();
```

The data model includes the updated configuration document.

10.3 Sample Session: 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:

Step 1 Edit the Data Model

As in Section 10.1, 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.

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');
```

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;
```

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;
```

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

Creating Configuration Documents

Administrators can perform a number of 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. See Chapter 8 for information about other repository management tasks.

This chapter includes the following sections:

> **Note:** In this chapter, names of attributes for XML elements appear in monospace type.

11.1 Characteristics of Configuration Documents

After installation, the Oracle Multimedia DICOM data model repository contains the following set of default configuration documents:

- Anonymity (`ordcman.xml`) - This document defines the actions required to make specific DICOM attributes anonymous.
- Constraint (`ordcmct.xml`, `ordcmcmd.xml`, `ordcmcmc.xml`) - These documents define the rules to check the conformance of DICOM content, with respect to the DICOM standard and other organization-wide guidelines.
- Mapping (`ordcmmp.xml`) - This document defines how each DICOM attribute maps to an element of the DICOM metadata document.
- Standard Dictionary (`ordcmsd.xml`) - This document lists the registry of DICOM data elements, as defined by the DICOM standard.
- Private Dictionary (`ordcmpv.xml`) - This document includes a registry of private data elements, as defined by equipment manufacturers or enterprises.
- Preference (`ordcmpf.xml`) - This document includes the run-time preference values for Oracle Multimedia DICOM features.
- UID Definition (`ordcmui.xml`) - This document lists and categorizes the registry of DICOM unique identifiers (UIDs), as defined by the DICOM standard.

Each type of configuration document has a specific set of characteristics, which are described briefly in the following sections. See Chapter 8 for more information about inserting, updating, and deleting configuration documents. See Chapter 10 for examples of these processes.

11.1.1 Characteristics of Anonymity Documents

Anonymity documents have the following 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.

11.1.2 Characteristics of Constraint Documents

Constraint documents have the following 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 as well as 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 EXP_IF_NULL_ATTR_IN_CONSTRAINT parameter in the preference document is used to specify whether to raise an exception when it encounters a null value for the attributes specified in the constraint rules. If the parameter is set to `true`, the isConformanceValid( ) method raises an exception; otherwise, the predicate that was encountered is evaluated to `false`.

11.1.3 Characteristics of Mapping Documents

Mapping documents have the following 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 as well as 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 XML namespace, the metadata XML schema that corresponds to that mapping document must be registered and consistent with the mapping document to enable the extractMetadata( ) method to function correctly.

- Extracted metadata is schema validated only if both of the following conditions exist:
  - The mapping document provides an XML schema namespace.
  - The value of the VALIDATE_METADATA parameter in the preference document is `true` (the default).

11.1.4 Characteristics of Standard Dictionary Documents

Standard dictionary documents have the following 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.

11.1.5 Characteristics of Private Dictionary Documents

Private dictionary documents have the following 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.

11.1.6 Characteristics of Preference Documents

Preference documents have the following characteristics:

- A maximum of two (one Oracle-defined and one user-defined) preference documents are allowed in the repository. The installed, Oracle-defined preference document (`ordcmpf.xml`) includes Oracle-fixed parameter names and lists of values. The user-defined preference document can update the default preference document values that were set in the Oracle-defined preference document.
- Changes to values in preference documents change the behavior of DICOM methods, functions, and procedures. Specifically, preference values that are defined in the user-defined preference document override the default values defined in the Oracle-defined preference document.

11.1.7 Characteristics of UID Definition Documents

UID definition documents have the following characteristics:

- A maximum of two (one Oracle-defined and one user-defined) UID definition documents are allowed 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.

11.2 Writing Configuration Documents

Administrators can create one or more configuration documents to support specific applications or organizations. The following subsections describe how to create each type of configuration document.

11.2.1 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.

The following table lists the values that are allowed for the `action` attribute:

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: `SOP_INSTANCE_UID` or `SOP_CLASS_UID`). In these cases, use the `replace` action with an appropriate value in place of the `remove` action.
`replace`	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. **Note:** This action value is not allowed for global actions. For example: 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 `ordcmrdt.xsd` as `xs:token of length 16`. The replacement value must be a string representation that conforms to this definition. 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 `ordcmrdt.xsd` as `xs:integer`. The replacement value must be a string representation that conforms to this definition.

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.

The following table describes the elements that define the global actions in the anonymity document:

Element	Description
<PRIVATE_ATTRIBUTES>	Specifies the action performed on all defined and undefined private attributes. **Note:** The action for a private attribute defined by the <INDIVIDUAL_ ATTRIBUTE> element always overrides the global action defined by the <PRIVATE_ ATTRIBUTES> element.
<UNDEFINED_STANDARD_ATTRIBUTES>	Specifies the action performed on all standard attributes that are not defined in the standard dictionaries in the data model repository. A DICOM attribute tag contains a group number and an element number. A standard attribute tag is identified by an even-numbered group number.
<UNDEFINED_PRIVATE_ATRIBUTES>	Specifies the action performed on all private attributes that are not defined in the private dictionaries in the data model repository. Note: The action value defined by this element takes precedence over the action value defined by the <PRIVATE_ ATTRIBUTES > element.

Examples of valid values for an <ATTRIBUTE_TAG> element are as follows: `00100010,00100010(DICOM),10871100(PRIVATE ORG).`

Note: Currently, only the following values are allowed for the definer name in a private attribute:

- The uppercase and lowercase letters `A-Z`
- The numbers `0-9`
- The characters: `'.',' '` (space), and `'/'`

The following subsections contain examples that show how to create anonymity documents.

11.2.1.1 Making a Standard Attribute Anonymous - Example 1

This example shows how to replace the standard attribute `Patient's Name` with specified values that make that attribute anonymous in the resulting DICOM content. The XML statements where these actions are defined are highlighted in bold.

```
<INDIVIDUAL_ATTRIBUTE>
   <ATTRIBUTE_TAG>00100010</ATTRIBUTE_TAG>
   <DESCRIPTION>Patient's Name </DESCRIPTION>
   <ANONYMITY_ACTION action="replace">anonymous</ANONYMITY_ACTION>
</INDIVIDUAL_ATTRIBUTE>
```

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 following example defines the standard attribute tag `00100010` in the standard dictionary:

```
<STANDARD_ATTRIBUTE_DEFINITION>
   <TAG>00100010</TAG>
   <NAME>Patient's Name</NAME>
   <VR>PN</VR>
   <VM>1</VM>
</STANDARD_ATTRIBUTE_DEFINITION>
```

This tag definition 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 by the DICOM standard (in Part 5). 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.

11.2.1.2 Making a Private Attribute Anonymous - Example 2

This example shows how to replace the private attribute `10871100(PRIVATE ORG)` with specified values that make that attribute anonymous in the resulting DICOM content. The XML statements where these actions are defined are highlighted in bold.

```
<INDIVIDUAL_ATTRIBUTE>
   <ATTRIBUTE_TAG>10871100(PRIVATE ORG)</ATTRIBUTE_TAG>
   <DESCRIPTION>Media Type </DESCRIPTION>
   <ANONYMITY_ACTION action="replace"> replaced</ANONYMITY_ACTION>
</INDIVIDUAL_ATTRIBUTE>
```

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.

In the following example, assume that the private dictionary tag `10871100` is defined as follows:

```
<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 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.

11.2.1.3 Making All Private Attributes Anonymous - Example 3

These examples show how to make a number of private attributes anonymous by removing or replacing those attributes with specified values that make them anonymous in the resulting DICOM content.

Note: Keep the following 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.

The following example uses the action value remove to remove all the private attributes in the resulting DICOM content.

```
<PRIVATE_ATTRIBUTES action="remove"></PRIVATE_ATTRIBUTES>
<UNDEFINED_PRIVATE_ATTRIBUTES action="remove" />
```

The following example uses the action value remove to remove all the undefined private attributes from the resulting DICOM content. And, this example 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.

```
<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>
```

The following example uses the action value none to include all defined private attributes in the resulting DICOM content. And, this example uses the action value remove to remove all undefined private attributes from the resulting DICOM content.

```
<PRIVATE_ATTRIBUTES action="none"></PRIVATE_ATTRIBUTES>
<UNDEFINED_PRIVATE_ATTRIBUTES action="remove" />
```

The following example uses the action value remove to remove all the defined private attributes from the resulting DICOM content. And, this example uses the action value none to include all undefined private attributes in the resulting DICOM content.

```
<PRIVATE_ATTRIBUTES action="remove"></PRIVATE_ATTRIBUTES>
<UNDEFINED_PRIVATE_ATTRIBUTES action="none" />
```

11.2.1.4 Making Undefined Standard Attributes Anonymous - Example 4

This example uses the action value remove to remove all the undefined standard attributes from the resulting DICOM content.

```
<UNDEFINED_STANDARD_ATTRIBUTES action="remove" />
```

11.2.2 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, with respect 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 with respect to one or more invokable constraint rules. Each invokable 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 predicates. Predicates define 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.

Constraint macros can be used to simplify the definition of complex constraint rules. 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 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 information view orddcm_conformance_vld_msgs. (See orddcm_conformance_vld_msgs in Chapter 4 for reference information about this information view.)

The following subsections contain examples that show how to create constraint documents.

11.2.2.1 Defining a Simple Constraint Rule - Example 1

This example 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 3.3-2007, 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 following predicate checks whether the attribute `SOP Class UID (0008,0016)` is not empty:

```
<PREDICATE>
  <BOOLEAN_FUNC operator="notEmpty">
    <ATTRIBUTE_TAG>00080016</ATTRIBUTE_TAG>
  </BOOLEAN_FUNC>
</PREDICATE>
```

The following predicate checks whether the attribute `SOP Instance UID (0008,0018)` is not empty:

```
<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 preceding predicates. The prescribed way to undertake this operation is to define another predicate that performs a logical AND operation on the preceding two predicates, as follows:

```
<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. Thus, taking the complete constraint rule from the constraint document `ordcmcmd.xml`, the global rule `SOPCommonModule` can be defined as follows:

```
<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 preceding example shows how to define predicates that represent logical relations or Boolean functions. Predicates that represent relational relations can be defined similarly.

11.2.2.2 Defining Constraint Rules by Importing Other Constraint Rules - Example 2

This example shows how to construct constraint rules hierarchically by referencing other external constraint rules, as well as 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 of three constraint rules: `SOPCommonModule`, `GeneralSeriesModule`, and `GeneralStudyModule`, as shown in the following example:

```
<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` by the following external rule statement:

```
<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>
```

In Example 2, the global rule `OracleOrdObject` references the global rule `SOPCommonModule` that is defined in Example 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.

11.2.2.3 Defining and Referencing Constraint Macros - Example 3

This example shows how to construct a constraint macro that checks whether or not a DICOM attribute is a code sequence attribute by following the first two conditions required by the Code Sequence Macro, which is defined in the DICOM standard, in PS 3.3-2007, 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.

The following global macro definition (see the constraint document `ordcmcmc.xml`) checks whether the attributes `Code Value (0008,0100)` and `Coding Scheme Designator (0008,0102)` are not empty:

```
<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: `${ }`. In the preceding example, the parameter `CodeAttr` represents the code sequence to be examined. Thus, checking whether the code value of the parameter `CodeAttr` is not empty is equivalent to checking whether the parameter `${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 the parameters for that macro.

The following example shows the definition for the global rule `GeneralSeriesModule` (in the constraint document `ordcmcmd.xml`), which invokes the constraint macro `CodeSequenceMacro`. The code segment where this macro is invoked is highlighted in bold.

```
<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 information 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.

11.2.3 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 `ordcmmp.xsd` defines the XML schema that constrains mapping documents. (See Section B.5 for a listing of the mapping document schema `ordcmmp.xsd`.)

The following subsections contain examples that show how to create mapping documents and corresponding metadata schemas.

11.2.3.1 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 details about the data type definition schema `ordcmrdt.xsd`.

- <NAMESPACE>: This mandatory element specifies the namespace of the metadata XML document. This namespace should 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 the 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 information about unmapped attributes.
- <MAPPED_ELEM>: This element specifies the XML path to the element that is the parent element of all the 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 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 the 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.5 in Appendix B of this guide for details about the mapping document schema.

11.2.3.2 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 `ordcmmddt.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 `ordcmmddt.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 allow an empty value.

See Section B.6 for details about the data type definition schema `ordcmmddt.xsd`.

11.2.3.3 Mapping Document for Metadata with No Schema Constraints - Example 1

This section 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 does not need to conform to any XML schema.

This section includes examples for a mapping document, followed by the resulting metadata XML document. Appropriate XML statements are highlighted in bold to show where the main actions take place in both the mapping document and the related XML document. Descriptions of each bolded XML statement follow the examples.

The following example of a mapping document shows 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.

```
<?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>
```

The following example 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'`:

```
<?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>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>
  </KEY_ATTRIBUTES>
</DICOM_OBJECT>
```

In the preceding examples, these actions take place:

- 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( ) in Chapter 5 for reference information about this method and the supported values for the extractOption parameter.

11.2.3.4 Mapping Document for Metadata with Schema Constraints and a Mapped Section Only - Example 2

This section 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 as well as in metadata XML documents with XML schema constraints.

This section includes examples for 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 take place 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( ) in Chapter 5).

The following example of a mapping document shows how to specify the namespace of the metadata XML document in a mapping document so that the metadata document can be constrained by an XML schema.

```
<?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>
```

This mapping document differs from the mapping document in Section 11.2.3.3 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 following example shows the XML schema definition for this mapping document:

```
<?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: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 the preceding examples, 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 `nillable` 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 `ordcmmddt.xsd`. In Example 2, the `"PERSON_NAME"` data type definition is copied from the schema `ordcmmddt.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 will return the following error:

  ```
  ORA-53259: cannot extract metadata that conforms to the
  schema definition
  ```

The following example 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'`:

```
<?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>
```

11.2.3.5 Mapping Document for Metadata with Schema Constraints - Example 3

This section describes how to create a mapping document and a metadata XML document with XML schema constraints, including the unmapped section.

This section includes examples for 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 take place in both the mapping document and the related XML document. Descriptions of each bolded XML statement follow the examples.

The following example of a mapping document shows how to include a sequence type attribute in the mapped section.

```
<?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>
```

This mapping document differs from the mapping document in Section 11.2.3.4 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 following example shows how to define the unmapped root element in the XML schema to include the unmapped section in the extracted metadata XML document. This example also shows how to use Oracle-defined data types from the schema `ordcmmddt.xsd`.

```
<?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 the preceding examples, the following actions take place:

- Instead of defining its own data types as in Section 11.2.3.4, this XML schema uses the Oracle-defined data types by copying the schema `ordcmmddt.xsd` into an

application-specific XSD file and then changing the namespace declaration, as in this XML 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 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.

- 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.

The following example 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'`:

```
<?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>
```

The following example 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'`:

```
<?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>
    <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>
```

11.2.4 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.9 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 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 the 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 NEMA.

Through the Oracle Technology Network, Oracle may release new standard dictionary documents along with installation instructions for DICOM administrators. These new standard dictionary documents would reflect new releases of, or addendum to, the DICOM standard.

The following subsections contain examples that show how to create standard dictionary documents.

11.2.4.1 Defining Standard Attributes - Examples 1 and 2

The standard attribute definition in the standard dictionary allows 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 which 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 the 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.

Definition Using a Simple Attribute Tag - Example 1

This example shows how to define a standard attribute with a simple attribute tag. The XML statements where this action is defined are highlighted in bold.

```
<STANDARD_ATTRIBUTE_DEFINITION>
   <TAG>00080008</TAG>
   <NAME>Image Type</NAME>
   <VR>CS</VR>
   <VM>1-n</VM>
</STANDARD_ATTRIBUTE_DEFINITION>
```

In Example 1, 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 `ordcmrdt.xsd` for more information about the value representation and value multiplicity data types supported by Oracle Multimedia DICOM.)

Definition Using a Wildcard Attribute Tag - Example 2

This example shows how to define a standard attribute with a wildcard attribute tag. The XML statements where this action is defined are highlighted in bold.

```
<STANDARD_ATTRIBUTE_DEFINITION>
   <TAG>60xx0010</TAG>
   <NAME>Overlay Rows</NAME>
   <VR>US</VR>
   <VM>1</VM>
</STANDARD_ATTRIBUTE_DEFINITION>
```

In Example 2, the standard attribute definition contains the wildcard tag `60xx0010`.

11.2.4.2 Retiring a Standard Attribute - Example 3

Standard attributes included in the standard dictionary document are retired in the DICOM standard when they are no longer needed.

This example shows the definition of a standard attribute as retired. The XML statement where this action is defined is highlighted in bold.

```
<STANDARD_ATTRIBUTE_DEFINITION>
   <TAG>00080010</TAG>
   <NAME>Recognition Code</NAME>
   <VR>CS</VR>
   <VM>1</VM>
   <RETIRED>true</RETIRED>
</STANDARD_ATTRIBUTE_DEFINITION>
```

In Example 3, 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.

11.2.5 Creating Private Dictionary Documents

Private dictionary documents are XML documents that list the attributes defined by modality manufacturers or organizations other than NEMA.

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.8 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 the 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.

11.2.5.1 Defining Private Attributes - Examples 1 Through 3

The private attribute definition in the private dictionary allows 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 which 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.

Definition Using a Simple Attribute Tag - Example 1

This example shows how to define a private attribute with a simple attribute tag. The XML statements where this action is defined are highlighted in bold.

```
<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 1, 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.)

Definition Using a Wildcard Attribute Tag - Example 2

This example shows how to define a private attribute with a wildcard attribute tag. The XML statements where this action is defined are highlighted in bold.

```
<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 2, the private attribute definition contains the wildcard tag `0119XX00` and the definer name `PRIVATE_ORG`.

Definition Using a Range Attribute Tag - Example 3

This example shows how to define a private attribute with a range attribute tag. The XML statements where this action is defined are highlighted in bold.

```
<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 3, 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 segment shows an example of an attribute definition that is *not* allowed 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 XML segment, the tag-definer pair (`01191100`, `PRIVATE_ORG`) matches the tag-definer pair (`0119XX00`, `PRIVATE_ORG`).

11.2.5.2 Defining Attribute Definers - Example 4

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 the 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`.

The following example shows how to define attribute definers:

```
<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 4, 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.

11.2.5.3 Retiring a Private Attribute - Example 5

Private attributes included in private dictionary documents can be retired when they are no longer needed.

The following example shows how to define a private attribute as retired. The XML statement where this action is defined is highlighted in bold.

```
<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>
```

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.

11.2.6 Creating Preference Documents

Preference documents are XML documents that list the set of preferences that define the run-time behavior of Oracle Multimedia DICOM.

The XML schema `ordcmpf.xsd` defines the XML schema that constrains preference documents. (See Section B.7 for a listing of the preference document schema `ordcmpf.xsd`. See the text included within the <xs: annotation> element for a detailed description of the preference parameters and their valid values.)

The default preference document `ordcmpf.xml` includes the default values for the run-time preference parameters for Oracle Multimedia DICOM.

Each preference value in a preference document is defined by a < PREFERENCE_ DEF> element.

Note: Keep these additional guidelines in mind:

- A maximum of two (one Oracle-defined and one user-defined) preference documents are allowed in the repository.
- Changes to values in preference documents change the behavior of DICOM methods, functions, and procedures. Specifically, preference 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.

The following subsection contains an example that shows how to create preference documents.

11.2.6.1 Defining Preferences - Example 1

This example shows how to define the VALIDATE_METADATA preference parameter in a preference document. The XML statements where this action is defined are highlighted in bold.

```
<PREFERENCE_DEF>
  <PARAMETER>VALIDATE_METADATA</PARAMETER>
  <DESCRIPTION>Validate XML Metadata document</DESCRIPTION>
  <VALUE>true</VALUE>
</PREFERENCE_DEF>
```

In Example 1, the <PARAMETER> element specifies the Oracle-defined parameter VALIDATE_METADATA.

The <VALUE> element represents the value of the parameter VALIDATE_METADATA, which is passed to the extractMetadata( ) method. The extractMetadata( ) method accepts a mapping document as input. The mapping document contains a namespace parameter (which can be empty). If an XML schema is registered at the namespace specified in the mapping document, and the value of the VALIDATE_METADATA parameter is true, the extractMetadata( ) method validates the resulting XML document against the designated XML schema. If the value of the VALIDATE_METADATA parameter is false, the resulting XML document is not validated against the schema.

11.2.7 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.10 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.

Each UID definition in a UID definition document is defined by a <UID_DEF> element.

Note: Keep these additional guidelines in mind:

- A maximum of two (one Oracle-defined and one user-defined) UID definition documents are allowed 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 on 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.

11.2.7.1 Defining a UID Definition - Example 1

This example shows how to define a storage class UID definition. The XML statements where this action is defined are highlighted in bold.

```
<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 1, 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.

11.2.7.2 Retiring a UID Definition - Example 2

This example shows how to define a UID definition for transfer syntax as retired. The XML statements where this action is defined are highlighted in bold.

```
<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 2, 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.

Part IV

Appendixes

This part includes additional information related to the Oracle Multimedia DICOM feature.

Part IV contains the following appendixes:

- Appendix A, "Configuration Documents"
- Appendix B, "XML Schemas"
- Appendix C, "Encoding Rules"
- Appendix D, "DICOM Image Processing"
- Appendix E, "Migrating from Release 10.2 DICOM Support"

A

Configuration Documents

This appendix lists the names of the DICOM configuration documents, which are defined as XML schemas. The schema file name, including the file extension, is used as the configuration document name in the DICOM repository.

The DICOM configuration documents are associated with one or more DICOM XML schemas. For information about the XML schemas, see Appendix B.

The DICOM configuration documents are as follows:

- Anonymity document (`ordcman.xml`)
- Constraint document (`ordcmct.xml`)
- Mapping document (`ordcmmp.xml`)
- Preference document (`ordcmpf.xml`)
- Private Dictionary document (`ordcmpv.xml`)
- Standard Dictionary document (`ordcmsd.xml`)
- UID Definition document (`ordcmui.xml`)

These configuration documents are available as XML files that are located in the `ord/dicom/xml` directory under *<ORACLE_HOME>* upon installation of Oracle Multimedia DICOM.

B

XML Schemas

This appendix lists the DICOM XML schemas. These global schemas, which are used by the methods of the ORDDicom object type, are loaded into the DICOM repository data model and registered as global XML schemas in Oracle Database with Oracle XML DB when Oracle Multimedia DICOM is installed. (See *Oracle XML DB Developer's Guide* for information about registering XML schemas.)

> **Note:** The sample code in this appendix will not necessarily match the code shipped with the Oracle installation. If you want to run examples that are shipped with the Oracle installation on your system, use the files provided with the installation. Do not attempt to compile and run the code in this appendix.

The DICOM XML schemas are associated with one or more DICOM configuration documents. For information about the configuration documents, see Appendix A.

The DICOM XML schemas are listed in this chapter as follows:

The latest versions of these schemas are available as XML files in the `ord/dicom/xml/xsd` directory under *<ORACLE_HOME>*. To examine the schemas, query the dictionary view as follows: `ALL_XML_SCHEMAS`]. In addition, read the documentation embedded within each schema file for more information.

For additional information about XML schemas, see the World Wide Web Consortium Web site at

`http://www.w3.org/XML/Schema`

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: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").
        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:
               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:simpleType>
      <xs:restriction base="xs:token">
        <xs:maxLength value="64"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</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 `ordcmrdt.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, Oracle. All rights reserved.

   NAME
     ordcmrdt.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
      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
      concatentated 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 is not explicitly encoded in XML.
      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">
    <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: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 "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:nill="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="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: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 "xxxxxxxx(definer)", where
      "xxxxxxxx" 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
      "xxxxxxxxx(definer)[n]".
      By default, the definer suffix "(definer)" can be
      omitted if the attribute is a DICOM standard tag.
      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:pattern value="([0-9a-z /A-Z\(\)\[\]\.#]+)"/>
  </xs:restriction>
</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:pattern value="[$0-9_a-z /A-Z\(\)\[\]\.\{\}#]*"/>
  </xs:restriction>
</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"?>
<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 Mapping Document Schema

The mapping document schema `ordcmmp.xsd`, shown in Example B–5, defines the structure of the mapping documents. The namespace for this schema is

`http://xmlns.oracle.com/ord/dicom/mapping_1_0`

Example B–5 Mapping Document Schema

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

   NAME
     ordcmmp.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*
         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.6 Metadata Data Type Definition Schema

The schema `ordcmmddt.xsd`, shown in Example B–6, 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–6 Data Type Definition Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
 Copyright (c) 2007, Oracle. 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 DICOM standard 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
      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
      concatentated 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 is not explicitly encoded in XML.
      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">
    <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: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 "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:nill="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="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: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 "xxxxxxxx(definer)", where
        "xxxxxxxx" 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
        "xxxxxxxxx(definer)[n]".
        By default, the definer suffix "(definer)" can be
        omitted if the attribute is a DICOM standard tag.
        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:pattern value="([0-9a-z /A-Z\(\)\[\]\.#]+)"/>
    </xs:restriction>
  </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:pattern value="[$0-9_a-z /A-Z\(\)\[\]\.\{\}#]*"/>
  </xs:restriction>
</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.7 Preference Document Schema

The preference document schema `ordcmpf.xsd`, shown in Example B–7, defines the structure of the preference documents. The namespace for this schema is

`http://xmlns.oracle.com/ord/dicom/preference_1_0`

Example B–7 Preference Document Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
 Copyright (c) 2007, Oracle. 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, and
INVALID_VM.
"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}

Default: {MISSING_ATTR INVALID_LENGTH MISSING_MAGIC
         MISSING_HEADER INVALID_VR INVALID_VM}.
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.

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 DICOM function extractMetadata takes a
mapping document as an input parameter. A mapping document
contains a namespace parameter (which can be empty). If an
XML schema is registered at this namespace, and the value of
the VALIDATE_METADATA parameter is true, the extractMetadata
function validates the resulting XML document against the
designated schema.
If the value of this parameter is false, the resulting XML document
is not validated.

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
        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.
  </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.8 Private Dictionary Document Schema

The private dictionary document schema `ordcmpv.xsd`, shown in Example B–8, 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–8 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
     standard 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.9 Standard Dictionary Document Schema

The standard dictionary document schema `ordcmsd.xsd`, shown in Example B–9, 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–9 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 standard 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.10 UID Definition Document Schema

The UID definition document schema `ordcmui.xsd`, shown in Example B–10, 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–10 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.

      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>
```

C

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. In Oracle Database 10*g* Release 2, Oracle Multimedia DICOM introduced these encoding rules to support metadata extraction and image content processing only (see *Oracle Multimedia Reference*). With Oracle Database 11*g* Release 1 (11.1), Oracle Multimedia DICOM provides full support for medical imaging.

Table C–1 lists the DICOM encoding rules that support medical imaging in Oracle Multimedia DICOM.

The following abbreviations are used in Table C–1:

- P = The processCopy( ), createDICOMImage( ), and writeMetadata( ) 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, the processCopy( ), createDICOMImage( ), and writeMetadata( ) methods are also supported for this transfer syntax.
- * = This encoding rule requires the Java Advanced Imaging (JAI) Image I/O Tools provided by Sun Microsystems. For more information about JAI Image I/O Tools, see the Sun Microsystems Web site at

  `http://java.sun.com/`

  See also the following script, which is available in the Oracle home directory after installation:

  ```
  <ORACLE_HOME>/ord/im/admin/initjp2.sql (on Linux and UNIX)
  <ORACLE_HOME>\ord\im\admin\initjp2.sql (on Windows)
  ```

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	-
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	R
1.2.840.10008.1.2.5	RLE Lossless	PR

For more information about DICOM encoding rules, see the DICOM standard (Part 5 and Part 6). This standard is available on the Web site for the National Electrical Manufacturers Association (NEMA) at

http://medical.nema.org/

D

DICOM Image Processing

This appendix describes image processing operations for DICOM content.

See *Oracle Multimedia Reference* for a complete list of image processing operators and details on each operator. See Chapter 5 and Chapter 6 for reference information about the processCopy( ) method.

D.1 The frame Image Processing Operator

The frame operator is new 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.2 Other Image Processing Operators

Other image processing operators include fileFormat, contentFormat, compressionFormat, cut, scale, and rotate. See *Oracle Multimedia Reference* for a complete list of image processing operators and details on each operator.

D.3 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, and PALETTECOLOR
WRITING	MONOCHROME1, MONOCHROME2, and RGB

For the Action `READING`, DICOM images with the listed photometric interpretation values can be decoded and the pixel data can be correctly extracted. For the Action `WRITING`, DICOM images with the listed photometric interpretation values can be encoded and the pixel data can be correctly 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	JPEG, JPEG 2000[1], RLE, and RAW[2]
WRITING	JPEG, JPEG 2000[3], and RAW[4]

[1] With JAI image IO installations (see Appendix C)
[2] With 8-bit, 12-bit, 16-bit, and other bits
[3] With JAI image IO installations (see Appendix C)
[4] With 8-bit, 12-bit, and 16-bit

For the Action `READING`, DICOM images with the listed compression format values can be decoded and the pixel data can be correctly decompressed. For the Action `WRITING`, DICOM images with the compression format values can be encoded and the pixel data can be correctly compressed with the correspondent 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 5 and Chapter 6 for reference information about the processCopy( ) method and for details about the compressionFormat 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 the example of this syntax in Section D.1). 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. See *Oracle Multimedia Reference* for a complete list of image processing operators and details on each operator.

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.

D.5 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).

E

Migrating from Release 10.2 DICOM Support

This appendix describes two options for migrating data and application code from DICOM support (in ORDImage objects) from Oracle Database 10*g* Release 2 (10.2) to the new DICOM support in Oracle Database 11*g* Release 1 (11.1). The first option has less impact on existing applications. The second 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.

E.1 Using the DICOM Relational Interface to Migrate Applications

Leaving the data in the original ORDImage objects and using the new DICOM relational interface to access the new DICOM functions will have less of an impact on your Release 10.2 applications. In addition, you need not move your existing DICOM content.

Use this 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 the attributes of the embedded DICOM content.

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 10*g* Release 2 application.
- You do not need to copy your existing DICOM content.

The only disadvantage of using this option is that the DICOM relational interface does not take full advantage of the new DICOM features.

E.2 Copying Data and Rewriting Applications for DICOM

Copying data into the ORDDicom object type and rewriting your application to use ORDDicom methods will take full advantage of the new DICOM features.

Use this 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 10*g* 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.

The advantage of using this migration option is that it takes full advantage of the features of the new ORDDicom object type.

The disadvantages of using this 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.

E.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 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. On the other hand, 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 the actions to be taken to make the attributes anonymous.

application conformance

The semantic consistency of DICOM content with respect 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 the 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 with the DICOM standard. 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 rule

A collection of rules to validate the conformance of DICOM content with respect 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

A set of user-configurable documents that includes an object-oriented representation of the file format in DICOM. These documents combine information objects and services, in pairs.

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 conformance validator

A utility that checks the syntactical and semantic consistency of DICOM content with respect to constraint rules specified in the data model repository.

DICOM content

Standalone DICOM Information Objects that are encoded according to the data structure and encoding definitions of PS 3.10-2007 of the DICOM standard (commonly referred to as DICOM Part 10 files). For more information about DICOM Information Objects, see the DICOM standard, which is available worldwide from the NEMA Web site at

`http://medical.nema.org/`

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 should 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 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.

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. For example: images and waveforms captured by imaging devices.

mapping document

An XML document that defines how each attribute should map to a particular element in an XML metadata document tree. This document determines what the DICOM metadata document looks like.

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. Administrators can update the default metadata schema and the corresponding mapping document to define a schema that is specific to the database instance.

Oracle *inter*Media

In Oracle Database 11g Release 1 (11.1), the name Oracle *inter*Media has been changed to Oracle Multimedia.

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 how to log warning messages when processing DICOM content. Oracle ships a default preference document. Administrators can update the default preference document to change the run-time behavior. For example, administrators can change the logging destination. They can turn the logging of warning messages on or off. Or, they can specify the categories of errors to be ignored.

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 the ORDDicom objects stored in a database instance. Administrators can access the repository and update the documents stored in it. Changes made to these documents will 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 will interpret the attribute as an instance of the DT data type, and the metadata encoder will encode the attribute as DT in all future metadata documents.

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.

service object pair (SOP) class

A class 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 standard committee, which is published in Part 6 of the DICOM standard. Standard attributes can be modified or deprecated by the DICOM standard 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 standard 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 standard committee.

standards conformance

The syntactical and semantic consistency of DICOM content with respect 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.

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 the unique identifiers (UIDs) for each DICOM data type.

unique identifier (UID)

A 64-byte, dot-concatenated, numeric string (similar to an IP address). The UID is a DICOM data type that is based on an ISO object identifier (OID) that uniquely identifies DICOM content worldwide. 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

C

D

E

F

G

H

I

J

L

M

O

P

R

S

T

U

V

W

X