Oracle® Healthcare Data Repository

Programmer's Guide

Release 8.0

E90932-01

February 2019

ORACLE®

Oracle Healthcare Data Repository Programmer's Guide, Release 8.0

E90932-01

Contents

10 HDR Messaging Services

11 HDR Exception Handling

12 Integrating the Healthcare Enterprise

13 HDR Clinical Document Architecture (CDA) Persistence

Preface

This document helps programmers work with Oracle Healthcare Repository and its features.

Finding More Information

Oracle Help Center

The latest user documentation for Oracle Healthcare Data Repository is available at https://docs.oracle.com/en/industries/health-hdr-80/index.html.

My Oracle Support

The latest release notes, patches and white papers are on My Oracle Support (MOS) at https://support.oracle.com. For help with using MOS, see https://docs.oracle.com/cd/E74665_01/MOSHP/toc.htm.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Where to Find Oracle Documentation

The Oracle website contains links to all Oracle user and reference documentation. You can view or download a single document or an entire product library.

Note: Check oracle.com to ensure you have the latest updates to the documentation.

Oracle Health Sciences Data Management Workbench Documentation

To get user documentation for HDR Release 8.0, go to:

https://docs.oracle.com/en/industries/health-hdr-80/index.html

Oracle Health Sciences Documentation

To get user documentation for Oracle Health Sciences applications, go to the Oracle Health Sciences documentation page on oracle.com at:

https://docs.oracle.com/en/industries/health-sciences/

Other Oracle Documentation

To get user documentation for other Oracle products, go to https://www.oracle.com/technology/documentation/index.html.

Conventions

The following text conventions are used in this document:

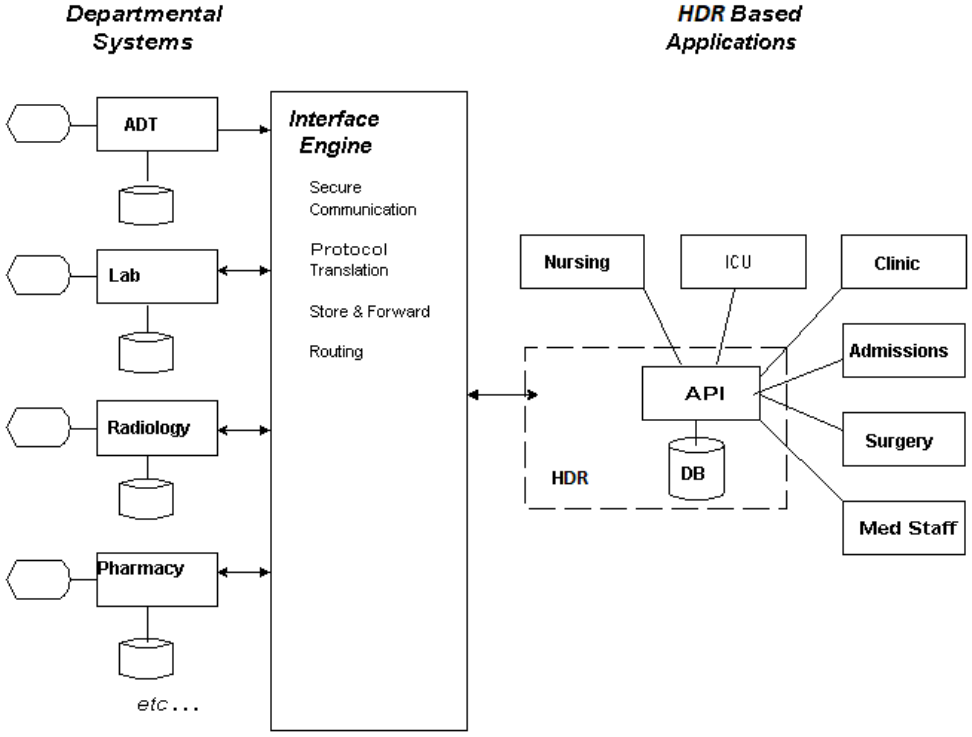
Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates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

1

HDR Components

- Services
- Java APIs
- Integrating the Healthcare Enterprise (IHE)
- Messaging Interface
- Concurrent Program Service for Scheduling Jobs
- Enterprise Terminology Services
- Profile Option Services
- Audit Services
- RIM Service

Figure 1–1 Example of a typical HDR solution:



Oracle Healthcare Data Repository (HDR) includes solution components critical to the creation of applications that support the management and provision of care in diverse healthcare settings. The associated functionality includes content from administrative, clinical and financial healthcare domain areas. This *Programmer's Guide* complements Oracle Healthcare Data Repository API documentation.

HL7 Reference Information Model (RIM) Version 3 provides a model of the healthcare domain, including compatibility with XML messaging. By focusing on standard interfaces, HDR makes it easy to build applications using industry standard tools (thanks to J2EE) and to interoperate with other healthcare systems (thanks to HL7).

The Java API is used by programmers to develop healthcare applications. The messaging interface is used to exchange data with other computing systems. HDR-based applications typically use both systems.

At an architectural level, HDR has three key interfaces with other software:

- **The Java Application Programming Interface (API):** The Java API lets customers write applications using HDR services. Because services are exposed as Remote and Local EJBs, client code can either be co-located on the middle-tier with HDR, or reside on its own tier, communicating via Java RMI. See "Java APIs" on page 1-3.
- **HL7 version 3 Based Messages:** HDR supports the transmission of well formed HL7 version 3 messages that conform with the Oracle Healthcare Data Repository HL7 Version 3 Messaging Conformance Specification. This interface supports interoperability between HDR based applications and third party applications. See "Messaging Interface" on page 1-4.
- **Services:** The service interfaces are a standards based collection of Enterprise Javabeans (EJBs), each of which provides a functional *service*. Its architecture is driven by two complementary standards: *Java 2 Platform, Enterprise Edition (J2EE) and HL7 Version 3*. J2EE provides industry standard interfaces and protocols for the plumbing of the system. See "Services" on page 1-2.

Services

Service Categories

- **Client-side Classes**: Represent the HDR public Java Applications Programming Interface, which includes an implementation of the HL7v3 Reference Information Model (RIM) and abstract data types specification.
- **Foundation Services**: Provide repositories for domain information. Enterprise Terminology Services (ETS) provide a centralized repository for storing coded medical terminologies. RIM Services provide a repository for storing any element of medical information that can be modeled in accordance with the RIM.
- **Application Services**: Provide higher level services based on the foundation services. HDR supplies the following application services:
  - Inbound Message Processor (IMP): Provides for the receipt of HL7v3 messages.
- **Configuration Services**: Provide Java APIs for configuring various HDR aspects.
- **Audit Services**: Provide auditing of critical HDR functions.

Service interfaces are not EJB interfaces by themselves; they are implemented by *bean wrapper* classes that wrap EJB Local and Remote bean interfaces. Similarly, the server-side EJB implementations delegate to Application Modules (AMs) within the Oracle Application Development Framework (ADF, formerly BC4J). The EJBs themselves are stateless with container managed transactions.

This architecture provides several benefits:

- Bean Wrapper classes hide the differences between Remote and Local mode EJBs. This lets application developers switch deployment models with minimal impact on their own code.
- Container Managed Transactions (with some exceptions) let application developers use the Java Transaction API (JTA) to manage distributed transactions involving HDR and their own services.
- Stateless EJBs provide superior middle-tier scalability to stateful approaches.

Java APIs

HDR services are exposed to programmers through Java APIs, which include the following elements:

Client-side Service Classes

The classes described below are called *client classes* because in remote deployments they reside on the client application side of the network, while the service implementations themselves reside within an application server on a remote computer. All client classes required to use any HDR services are packaged together into a zip file hdr-client-1.0.0-SNAPSHOT.zip in the client directory of hdr-1.0.0-SNAPSHOT.zip. This file must exist in the class path of any application that uses HDR, along with supporting EJB classes.

- **Data Transfer or Value Objects (DTOs)**: These objects represent snapshots of HDR's internal data model and are used to exchange data between HDR and client applications. Note that for the HL7 package these objects are called RIM objects and reside in the oracle.hsgbu.hdr.hl7.rim package.
- **Factories**: Factory classes are used to create data transfer objects for submission to HDR services. Different services use different factories, and some may use multiple factories.
- **Service Interface**: Each service has at least one corresponding Java interface. These interfaces contain methods that provide the functionality of the service. Services typically provide CRUD-style operations for some part of the HDR data model, but may also define higher-level business logic. The service interfaces do not provide business logic; business logic is defined by client-developed applications based on HDR.
- **Fetches and Criteria Objects**: These objects support the complex search and retrieval operations supported by many services. A criteria object defines a filter that specifies which objects to retrieve. Criteria are analogous to a SQL WHERE clause. A fetchmap defines the kind of data to be returned. It is analogous to a SQL SELECT clause. Criteria and fetches can refer to different (albeit related) objects. For example, a query may be defined to retrieve all encounters (the fetch) for a given patient (the criteria).

Client applications use the *ServiceLocator* class to retrieve a handle to a service, rather than instantiating services directly. The *ServiceLocator* provides a single point from

which clients establish connections to HDR and access its services. Instantiating the *ServiceLocator* and establishing a session with HDR is described in "The Service Locator" on page 4-1.

Integrating the Healthcare Enterprise (IHE)

HDR supports sharing documents that conform to IHE standards across healthcare enterprises. The documents are stored in an HDR repository, and metadata identifying the documents is stored in a Document Registry.

See also:

"Integrating the Healthcare Enterprise" on page 12-1

Messaging Interface

HDR can send and receive HL7 version 3 messages that conform with the *Oracle Healthcare Data Repository HL7 Version 3 Messaging Conformance Specification*, and HDR provides services that can parse and construct XML messages. These services are exposed in essentially the same way as normal HDR services and have a Java API that can be called programmatically by ISV (Independent Software Vendor) code.

HDR messaging services (IMP) can also be accessed through an interface engine and an adapter such as HDR Gateway, which is in turn responsible for the transport-layer issues of sending and receiving messages over the network. This is the typical way that messaging is used by an HDR application. The *Implementation Guide* describes how to configure the HDR Gateway.

Inbound Messaging Services

Healthcare enterprises typically operate a number of departmental systems such as ADT, diagnostic departments, pharmacy, and others that may be acquired from multiple vendors. Such systems require messaging services to communicate events and request actions from applications throughout the enterprise.

To route the message from the source system, an external interface engine that handles HL7 message translation and routing must be implemented for IMP (Inbound Message Processor) to function. Although a single interface engine is typically required, multiple interface engines can be implemented. An interface engine is not included with HDR. However, Oracle B2B/BPEL can be used as an Interface Engine.

The Inbound Message Processor (IMP) provides message interpretation, persistence, and acknowledgement services to HDR applications for processing inbound messages from external systems. IMP supports XML formatted inbound messages that conform to the HL7 version 3 messaging standard and compliant with messaging schemas of HDR supported message types.

See also:"HDR Messaging Services" on page 10-1

Concurrent Program Service for Scheduling Jobs

HDR provides a JMS queue based job scheduler service called ConcurrentProgramService, which is exposed on ServiceLocator as a stateless session bean. ConcurrentProgramService is used to schedule the following jobs.

Loading Messaging Metadata

Messaging metadata load can be scheduled as a background job using the ConcurrentProgramService.loadMessagingMetadata() API call. The API returns a request ID that can be used to monitor the job status using the ConcurrentProgramService.getJobStatus() API.

Loading MTK Schema to HDR Server

Messaging Toolkit (MTK) schemas from a local directory specified in the profile option value for profile code CTB_MTK_SCHEMA_DIR_PATH, can be loaded to HDR server location using the ConcurrentProgramService.loadMTKCustomSchema() API.

Loading MTK Interaction Schema to HDR Server

Composite message schema can be created using the Messaging Toolkit (MTK) custom interaction schemas using the ConcurrentProgramService.loadMTKCustomInteractionSchema() API.

Enterprise Terminology Services

Enterprise Terminology Services (ETS) is a core component of HDR that incorporates a range of terminology systems and provides extensive terminology services to HDR applications, including the following principal features:

- Consistent and real-time access to all terminology content - whether standards-based or user-defined.
- Support for standards-based terminologies:
  - SNOMED, LOINC, and FDB terminologies through a specialized model (referred as Core terminologies in ETS).
  - Other terminologies like ICD9, CPT4, and HCPCS through a generic model.
- Support for user-defined terminologies.
- High level of terminology integration - between different terminologies and between different versions of the same terminology.
- Support for user-defined containers of terminology content such as Concept Lists and Classifications. These containers can be used for building application interfaces, constraining and validating attribute values, and generating context-sensitive reports.
- Multi language support (MLS) on concept descriptions (except Classifications and editable terminologies).

See Also:

The *Implementation Guide* for the prerequisites and procedures necessary for the implementation of ETS Services.

Profile Option Services

Profile options are configurable preferences that affect the way an Oracle application looks and behaves. System administrators can control HDR behavior by setting profile option values. Application developers can control application behavior by programming their applications to perform in accordance with customized profile option values.

Examples of typical profile options include the following:

- Language: Determines the language in which the application is displayed to users.
- Date Format: Determines the format (mmddyyyy, ddmmyy...) for date displays.

System administrators can set profile options at the following levels:

- User (highest level)
- Org
- Site (lowest level)

The profile option values set at each level define runtime values for each user's profile options. An option's runtime value is the highest level setting for that option.

A profile option can be set at more than one level. When a profile option is set to more than one level, an order of precedence applies: Site has the lower priority, superseded by Org, which is further superseded by User. A profile option value entered at the Site level can thus be overridden by value entered at the Org level, and value entered at the Org level can thus be overridden by the value entered at the User level.

See Also:

The *Implementation Guide* for the prerequisites and procedures necessary for the implementation of Profile Option Services.

Create, Set, and Update Profile Options

The HDR ProfileOptionService enables users to manager profile options and its associated values at various profile levels.

Creating Profile Options

Create new profile options using the ProfileOptionService.createProfileOption API.

For example, a physician order entry application could define a profile option that defines the default sort order of patient problem lists (by date, severity,...).

Updating Profile Options

Update profile options using the ProfileOptionService.updateProfileOption API.

Create and Update New Profile Option Values

Create new profile option values for the already created profile options using the ProfileOptionService.setProfileOptionValue API.

You can update existing profile option values for the already created profile options using the ProfileOptionService.setProfileOptionValue API.

Audit Services

HDR auditing services let you log and monitor all HDR activities, to monitor security policy and regulation compliance by recording actions taken by users during sessions. Such actions could include invoking an API, performing a custom function, or other defined events.

HDR Configuration Manager, a GUI tool, lets security administrators define auditing policies. Implementation of HDR Audit Services includes the following steps:

- Enabling HDR Audit Services
- Initializing existing audit event types

- Creating new audit event types
- Invoking HDR Audit Services

Enabling Audit Services

The HDR Audit Service [AuditService] is a core HDR interface that lets you log and monitor HDR activities, to monitor security policy and regulation compliance—by recording actions taken during user sessions. Such event records can help detect actual or attempted violations of policy and operation procedures.

HDR Audit Services can be enabled (turned on) or disabled (turned off) globally. When enabled, audit events of all seeded and user-defined audit event types can be audited. When disabled, Audit Services is not operative.

Auditing is turned on or off by setting the profile option CTB: Auditing ON to Y or N respectively. By default, CTB: Auditing ON is set to Y on install. Use the ProfileOptionService to update this value. The profile option service API to update this profile option is:

```
ProfileOptionService.setProfileOptionValue
```

RIM Service

HL7 Reference Information Model (RIM) Version 3 provides a model of the Healthcare Domain, including compatibility with XML messaging. By focusing on standard interfaces, HDR makes it easy to build applications using industry standard tools (thanks to J2EE) and to interoperate with other Healthcare systems, thanks to HL7.

2

Design Recommendations

Use Version-Aware References

Oracle recommends the use of version aware references wherever feasible to avoid issues associated with version agnostic references.

Oracle Healthcare Data Repository lets you update existing objects without prior retrieval if the object id is known. HDR can also create an object if it does not exist, or do nothing or update it if it does. While useful, careless use of these features may result in corrupt data, and technology limitations may cause further unexpected errors. These concepts also apply to messaging side effects because they use version agnostic references internally.

References are created by the makeReference method of the Act, Role, and Entity factories. References may be version aware or version agnostic. Version aware references are created by specifying the version number of the object being referenced when creating the reference, and only apply the updates to the version of the object specified. If that version is not the latest one an error occurs. Version agnostic references have no version specified and thus always apply to the latest version of the object, but note that there are problems in HDR with the implementation of version agnostic references.

My Oracle Support article XXXXXXXXX.1 identifies strategies for properly handling the concurrent update of information as well as strategies for dealing with errors that may sometime occur.

Follow J2EE Best Practices

HDR is based on Java 2 Enterprise Edition (J2EE). J2EE is an established family of technologies with a substantial amount of published reference information. You should leverage this information when designing your applications.

This provides two important benefits. First, it lets programmers easily compare the performance implications for using remote or local mode in their applications. Second, it can speed development in situations where programmers share an HDR instance. With a shared instance, programmers can develop and test their components in remote mode, letting each work within separate containers, while sharing a common HDR instance running in its own container. It would be better for each developer to have a separate HDR instance, but this is not always possible.

Note: ■Oracle Technology Network, OTN, includes information describing good use of the Application Server, particularly the J2EE Design Pattern Samples

- Sun Microsystems Enterprise BluePrints also provide valuable information

It is particularly important that you minimize the number of round-trips required by your application. HDR provides a number of mechanisms to assist with this, such as the bulk retrieval capabilities of the Rim Service.

Choose an Appropriate Deployment Model

HDR supports access to its services through local and remote EJBs. In remote mode, HDR runs in its own application server instance and client code communicates with it through RMI. In local mode, HDR and the client code are co-located on the same application server. Selection of the appropriate deployment model depends on the requirements of your application.

HDR hides the differences between the two deployment modes—thus permitting the same code to be executed largely without change in either mode (although some minor changes are required to set the mode to local.

For production purposes, the choice comes down to a balance between performance and flexibility. Accessing HDR in local mode performs better than remote mode in terms of throughput and latency. However, local deployment requires your application code to be co-located on the same application server. For many Web based applications, this may be acceptable, but there are other applications (such as rich clients) where this is not possible. In such cases, remote deployment may be preferable.

However, the choice between local and remote access is not absolute. HDR exposes all of its services as remote and local EJBs. In this way, it is possible to interact with HDR in different modes from within the same application. For example, you could split your business logic between a rich client-tier application in remote mode, and a customer middle-tier session bean in local mode, both of which could access HDR.

Package Custom Server-side Code in its Own EAR File

Client code that accesses HDR in local mode should be packaged into its own EAR file, and appropriate deployment options configured to connect the client code's EAR file

to the HDR EAR file. You should never extract and add your own code to the HDR EAR file.

Design Your Own Business Level Services on Top of HDR

HDR supports a wide variety of applications, from message processing to patient admissions GUIs. It is therefore necessarily generic. For applications that are more specific, we recommend that you develop your own interfaces that encapsulate the business logic of your application. In doing so, you should consider the following:

- HL7's RIM Objects (and by extension HDR's objects) are very generic and may provide fields and cardinalities that are of no relevance to your application. In such cases, defining your own simplified versions of RIM Objects may significantly reduce the complexity and increase the usability of your API.
- Be careful of API granularity. If it is too fine-grained, it may require your code to make an excessive number of calls, eroding performance; if it is too coarse, it may be difficult to build on.
- Use *stateless* rather than *stateful* designs. Stateless designs (where all necessary information is supplied to a service) are typically more scalable than stateful designs.

Design Your Own Domain Model to Take Full Advantage of GUI Frameworks

Current generation GUI frameworks such as Oracle's ADF provide excellent support for connecting GUI views and controllers with simple models based on Java Beans, and standard Java types such as *String* and *BigDecimal*.

HDR's RIM Objects use classes that implement the HL7 abstract data type specification. These data types are not JavaBeans and cannot be used directly by GUI frameworks without writing conversion routines. Because HDR RIM Objects are defined in terms of RIM Datatypes this also precludes their direct use by many GUI frameworks.

An obvious solution to this problem is to write the appropriate conversion utilities. The details describing how to do this vary from one GUI framework to another. This approach therefore has the potential to tie your application to a particular GUI framework, although you avoid the overhead of creating your own model.

An alternative approach is to convert HDR Objects into JavaBeans that are defined in terms of other JavaBeans and standard Java types. This framework-friendly model could then be used with any GUI framework that supports JavaBean components for its model. This model does not require a 1:1 mapping with the RIM; it might well represent a simplified model used by your business interfaces.

All things being equal, we recommend taking the second approach. Defining your own domain model using JavaBeans lets your business logic layer be more easily ported from one GUI framework to another, and you will have a ready made, simplified set of objects to pass to your business interfaces.

Consider Threading Issues

HDR's client-side components are not synchronized and are intended for use by one thread at a time. For most web applications this is not an issue, as each request receives a single thread and the programming model forbids spawning additional

threads. We recommend that you take advantage of your development framework to avoid having to write your own thread handling code. If this is not possible, you should take the usual precautions that you would for an unsynchronized *java.util.Collection*, or any other regular, unsynchronized Java object.

Avoid Treatment of Your ServiceLocator as a Handle to a Session

Do not pool or otherwise cache ServiceLocators to maintain HDR sessions. Pooling or caching ServiceLocators to maintain HDR sessions can give rise to complicated, unnecessary schemes to cache ServiceLocators.

Avoid Deprecated Methods

HDR uses Java standard syntax to indicate deprecated methods. You should never use deprecated methods for new code because they may be dangerous, obsolete, or unsupported. Existing code should be migrated as soon as possible, as Oracle reserves the right to remove deprecated methods and classes completely from the HDR interface. Deprecated methods are identified in the following publications; refer to these documents for more details including alternative methods. Many development environments highlight the use of deprecated methods. We recommend that you make use of such features.

See Also: Deprecated HDR methods are identified in the following Oracle publications:

- *Oracle Healthcare Data Repository API Documenation.*

Use Oracle Identity Manager for Single Sign-On

HDR is deployed on WebLogic application server. HDR uses the standard J2EE application security mechanisms. Users, accessing the DR services, can configure security providers such as Oracle Identity Manager, LDAP, Single Sign-On, or custom security implementation to manage WebLogic security principals and credentials.

See Also: ■*Oracle Healthcare Data Repository Implementation Guide for more information about configuring user accounts.*

- *Understanding WebLogic Security section in WebLogic Server Documentation.*

Use HTTP Form Authentication

Single sign-on authentication is the preferred mechanism for deploying HDR and HDR-based web applications.

When single sign-on is not configured, we recommend using HTTP form authentication rather than HTTP basic authentication. With HTTP basic authentication, each client call to the web application performs login validation at least two times: once to determine if the client has permissions to access the web resource, and again during HDR API calls. With HTTP form authentication, the HTTP login form displays upon the first web access, and an HTTP cookie transparently authenticates the user.

3

Set Up Client-side Libraries

Note: Oracle Healthcare Data Repository Implementation Guide for further information about HDR implementation and setup of an HDR instance.

Client-side Libraries

You must install the following client-side libraries to develop and run HDR applications:

Client-side Libraries
<HDR product install home>/weblogic/jars/hdr-cfg-client-8.0.0.jar
<HDR product install home>/weblogic/jars/hdr-client-1.0.0-8.0.0.jar
<HDR product install home>/weblogic/jars/hdr-commons-1.0.0-8.0.0.jar
<HDR product install home>/weblogic/jars/hdr-ets-common-1.0.0-8.0.0.jar
<HDR product install home>/weblogic/jars/hdr-ets-core-1.0.0-8.0.0.jar
<HDR product install home>/weblogic/jars/hdr-ihe-client-1.0.0-8.0.0.jar
<HDR product install home>/weblogic/jars/hdr-rim-client-1.0.0-8.0.0.jar
<HDR product install home>/weblogic/jars/wlfullclient.jar
<HDR product install home>/hdr_exploded_app/lib/xmlparserv2.jar
<HDR product install home>/hdr_exploded_app/lib/adfm.jar
<HDR product install home>/hdr_exploded_app/lib/adf-share-ca.jar
<HDR product install home>/hdr_exploded_app/lib/adf-share-base.jar
<Weblogic home>/oracle_common/modules/javax.ejb_3.2.0.jar
<Weblogic home>/oracle_common/modules/javax.mail_1.1.0.0_1-4-4.jar
<Weblogic home>/oracle_common/modules/oracle.jdbc_11.2.0/ojdbc6.jar

4

Use the Service Locator to Create HDR Sessions

The Service Locator

ServiceLocator is the primary entry point into HDR and helps create HDR sessions. Although ServiceLocator is used in the same way for remote or local access to HDR, its usage varies depending on whether it is used inside or outside of a container. Use it for the following functions:

- Establishing HDR sessions
- Configuring the access mode (local or remote)
- Retrieving handles to services through JNDI (Java Naming and Directory Interface)

Each ServiceLocator is associated with a single HDR session. Each HDR session defines an application-level context in which processing occurs. Services retrieved through a particular ServiceLocator instance receive that ServiceLocator session as their own. A ServiceLocator is used to establish a new HDR session.

Configure Service Locator

ServiceLocator is configured by properties passed to the getInstance factory method. Three types of properties can be configured: JNDI properties, client mode, and authentication properties. The following table lists typical configuration parameters and their possible values. JNDI parameter values are used with an Oracle Application Service JNDI client library:

Service Locator Configuration Properties

Property Name	Description	Values

javax.naming.Context.INITIAL_CONTEXT_FACTORY	Specifies the factory class used to instantiate context factory that is required by JNDI to establish initial connection.	For remote mode: "weblogic.jndi.WLInitialContextFactory"; not required in local mode.
javax.naming.Context.PROVIDER_URL	Specifies the location of the JNDI context containing HDR bean bindings.	For remote mode: "t3://*hostname:port*" where "hostname" is the host running HDR and port is the WebLogic admin port configured for the Application Service; not required in local mode.
javax.naming.Context.SECURITY_PRINCIPAL	Specifies the JNDI principal; is set by ServiceLocator.login when a session is created.	When creating a ServiceLocator for an existing session, the value must be identical to the username passed to ServiceLocator.login when the session was established.
javax.naming.Context.SECURITY_CREDENTIALS	Specifies the JNDI credential; is set by ServiceLocator.login when a session is created.	When creating a ServiceLocator for an existing session, the value must be identical to the password passed to ServiceLocator.login when the session was established.
ServiceLocator.CLIENT_MODE	Specifies whether the ServiceLocator should attempt to connect to Remote or Locale EJBs	ServiceLocator.REMOTE or ServiceLocator.LOCAL. The default is ServiceLocator.REMOTE.

JNDI properties (named in table, with names starting with javax.naming.Context) can be configured by Java System properties, a jndi.properties file in the classpath, or by setting the values programmatically on the Properties passed to ServiceLocator.getInstance. The order of precedence is that programmatic properties override jndi file properties that in turn override Java System properties. We recommend that you always configure javax.naming.Context.INTIAL_CONTEXT_FACTORY and javax.naming.Context.PROVIDER_URL using Java System properties or jndi.properties to avoid unexpected behavior, and the need for extra configuration in your application code. You should configure the remaining configuration options on the Properties object that you pass to ServiceLocator.getInstance.

Note: Always configure javax.naming.Context.INTIAL_CONTEXT_FACTORY and javax.naming.Context.PROVIDER_URL implicitly using jndi.properties or other Java System properties. Please refer to your Application Service documentation for more information.

Create a New Session

Example 4-1 illustrates typical usage of the ServiceLocator. The ServiceLocator is configured by creating a Properties class and configuring the JNDI and HDR client mode properties. The application then calls the login method that establishes a session between client and server. The application retrieves a service from the service locator by calling getRimService(), and calls logout to delete the session. Once logout has been called, the session is terminated and any services retrieved within that session cease to function.

Example 4–1 Outline of Service Locator Usage for a New Session

```
Properties properties = ...;
ServiceLocator serviceLocator = ServiceLocator.getInstance(properties);
serviceLocator.login(username, password);
```

```
RimService rimService = serviceLocator.getRimService();
...
serviceLocator.logout();
```

Set the Client Mode

This mode specifies which kind of EJB the ServiceLocator should attempt to retrieve from JNDI. In local mode, the ServiceLocator attempts to retrieve EJB local home interfaces and beans. In remote mode, the ServiceLocator attempts to retrieve EJB remote home interfaces and Beans. The two supported modes are defined the constants LOCAL and REMOTE on the ServiceLocator class. REMOTE is used by default if no other mode is specified. Each instance of ServiceLocator can function in only one mode.

Example 4–2 Setting the Client Mode

```
// Configure an Instance of ServiceLocator
// Configure an Instance of ServiceLocator
Properties properties = new Properties();
properties.setProperty(ServiceLocator.CLIENT_MODE, ServiceLocator.LOCAL);
ServiceLocator serviceLocator = ServiceLocator.getInstance(properties);
serviceLocator.login(username, password);
// Retrieve a service
RimService rimService = serviceLocator.getRimService();
```

Authorization Considerations

This section describes the different behaviors of the ServiceLocator.login method in local and remote mode, and the effect of the Java Authentication and Authorization Service (JAAS) on HDR authentication.

ServiceLocator.login

The behavior of ServiceLocator.login depends on whether or not the calling code is protected by the application server:

- If ServiceLocator.login is called from code running outside of a container, the identity and credentials passed to login are presented as the user name and password for the user when authenticating.
- If ServiceLocator.login is called from code running *outside* of a container, the identity and credentials passed to login are presented as the user name and password for the user when authenticating.
- If ServiceLocator.login is called from code running *inside* of a container (always the case in local mode), the behavior depends upon whether or not the application server has been configured to manage access to the component. If the component is protected by the container, the user name and password values are **ignored**; the ServiceLocator uses the user name and password of the current EJB session context (javax.ejb.SessionContext), which is established by the application server before it permits access to the component. The behavior for unprotected components is more complicated and is described below.

Note: ServiceLocator.login *ignores* the user name and password passed to it when executing within a protected component within an application server, and uses the user name associated with the current EJB SessionContext. That user has to be authorized to access HDR functionality by both JAAS and the HDR Security Service. Session initiation otherwise fails and a *Session could not be created* authorization exception is thrown.

When calling ServiceLocator.login from code running inside of a container, the ServiceLocator.login call only succeeds if made from within a component whose EJB SessionContext includes details of an authenticated user. Users are associated with an EJB SessionContext when they first attempt access to a *protected* component. The deployment configuration of your application and WebLogic determine which components are protected. If a component is protected, the application server authenticates any user before permitting access to the component. For web applications, this is typically achieved through a login screen.

Once authenticated, the user identity can be propagated between components. This essentially means that a call to ServiceLocator.login may fail if it is called from an unprotected component and no authentication has occurred. You should thus ensure that you protect the entry points to your applications with an appropriate security constraint in your WebLogic deployment descriptors.

Example 4–3 Use ServiceLocator by Calling Login From a Remote Client

This code sample uses ServiceLocator by calling Login from an RMI Client outside of an WebLogic:

```
Properties properties = new Properties();
properties.setProperty(ServiceLocator.CLIENT_MODE, ServiceLocator.REMOTE);
ServiceLocator serviceLocator = ServiceLocator.getInstance(properties);
serviceLocator.login("userName", "password");
RimService rimService = serviceLocator.getRimService();
IMPService impService = serviceLocator.getIMPService();
serviceLocator.logout();
```

Note that services are not usable after ServiceLocator.logout().

JAAS Authentication in WebLogic Server

Before accessing HDR Services you must connect to and authenticate with HDR. The authentication process is carried out through JAAS by the *WebLogic*, the underlying application server. JAAS in turn delegates these requests to a provider that authenticates users against a repository, and determines authorization based on EJB deployment descriptors and other application server configuration files. JAAS also provides access *authorization* to particular EJBs and EJB methods.

JAAS and the ServiceLocator

The following key fundamentals relate to the behavior and use of the ServiceLocator:

- JAAS authentication is a function of establishing a session with the application server. If authentication is successful, the authenticated user details are used to authenticate with JAAS when attempting to access protected EJB components.

- JAAS authorization occurs whenever a user attempts access to a protected resource (for example, an HDR Service EJB). JAAS authorization is in addition to that provided by HDR Authorization mechanisms.
- JAAS is configured with a separate repository of user names as passwords (such as an LDAP server), outside of HDR. External user repositories such as Oracle Identity Management Suite can be integrated with WebLogic server.

See Also: The WebLogic application server documentation. For more information about available JAAS providers, JAAS configuration, and use by WebLogic, follow this link: https://docs.oracle.com/middleware/12213/wls/SCPRG/fat_client.htm#SCPRG224.

5

Audit Events and Log Errors

HDR Audit Services

The HDR Audit Service [AuditService] is a core HDR interface that lets you log and monitor HDR activities, to monitor security policy and regulation compliance—by recording actions taken during user sessions. Such event records can help detect actual or attempted violations of policy and operation procedures.

The AuditService.createEventLog method lets you record an audit event record in the table `OHF_HDR_AU_ACCESS_LOG`. You can use a reporting tool of your choice to generate auditing reports by querying this table for audit record details of a relevant event.

The EventLog value object has two attributes that are mandatory and validated; the rest are optional and not validated:

EventLog Value Object Attributes

Value Object	Attribute	Mandatory / Optional	Validated
`EventLog`	`EventOutcome`	Mandatory	Not Validated
`EventLog`	`EventType`	Mandatory	As a valid HDR profile option

When createEventLog is called, the following checks are made to determine whether to record an event or to ignore the request:

1. If the CTB profile option CTB: Auditing On (CTB_AU_AUDIT_FLAG) is not set to Y, the audit event is not recorded.
2. If the Break-The-Glass profile option is turned on and the CTB profile option CTB: Audit All When in Break-The-Glass (CTB_AU_ALL_WHEN_BTG) is not set to Y, the audit event is not recorded.
3. If the attribute EventType of value object EventLog is a valid CTB profile option and the profile option is set to Y, the audit event is recorded.

Example 5–1 Create an Audit Event Record

The following code sample creates an audit event record:

```
AuditingHelper
auditHelper = new AuditingHelper();
```

```
AuditService = mServiceLocator.getAuditService();
EventLog eventLog = auditHelper.newEventLog();

// "MyEventType" is a valid CTB profile option
eventLog.setEventType("MyEventType");
// mEventOutcome is a valid membership code in the CTB_AU_EVENT_OUTCOME
conceptlisteventLog.setEventOutcome(mEventOutcome);

DataTypeFactory dataTypeFac
= DataTypeFactory.getInstance(mServiceLocator);
II ii = dataTypeFac.newII(mUID), dataTypeFac.newST(mST);
SET_II iis = dataTypeFac.newSET_II(ii);

// finally
mAuditService.createEventLog(eventLog);
```

Initializing Existing Audit Event Types

Audit event types can selectively be turned on or off. When both the global auditing flag and a particular audit event type are turned on, events of this particular type are audited by HDR Audit Service.

Following is the list of HDR audit event types is seeded for HDR use. By default, these event types are turned on.

1. CTB: Audit Receive Message
2. CTB: Audit Resending of Message
3. CTB: Audit Update OID
4. CTB: Audit Skipping of Message
5. CTB: Audit Generation of Message
6. CTB: Audit Query on Personal Health Information
7. CTB: Audit Insert/Update of Personal Health Information

Creating New Audit Event Types

Applications developed on the HDR Platform can define business audit event types in addition to the seeded event types.

For example, an Admitting application might define an audit event type asAdmit Patient, and monitor events of this type.

Note: Although HDR provides the mechanism to audit business events, it is your responsibility to implement the appropriate audit calls to log such events.

To create a new audit event type, use ProfileOptionService.createProfileOption to create a new profile option with the new audit event type as the profile option code.

Invoking HDR Audit Services

After defining new audit event types, applications can log audit events of these types by calling the Audit Services interface.

Reference

Oracle Healthcare Data Repository Javadoc

Table 5–1 Service and Methods: Audit Services

Level	Detail
Package	oracle.hsgbu.hdr.auditing
Class	AuditService
Methods	createEventLog

Prerequisite

Creating New Audit Event Types

Login

This is an API-based implementation procedure.

Responsibility

Any responsibility.

Navigation

This is an API-based implementation procedure.

Steps

1. Turn on HDR Audit Services and the audit event type.
   - Enabling Audit Services
   - Initializing Existing Audit Event Types
2. In the application code, call the createEventLog method with the new event type as the value of the EventType attribute.

   Note:*Oracle Healthcare Data Repository Javadoc*

Attribute Values in Audit Events

Every entry in the audit trail has the attributes listed by the attributes table included in oracle.hsgbu.hdr.auditing.EventLog. This can be found in the *Oracle Healthcare Data Repository Javadoc*.

Log Error Messages

Oracle HDR uses the JDK Logging Framework to provide the ability to log error messages for debugging, error reporting and alerting purposes, as discussed in the following sections:

- Teminology
- Log Configuration Parameters

Teminology

Terminologies used by JDK Logging Framework:

Log Message

A log message contains the application messages in different formats supported by the JDK Logging Framework. The log message format is configurable in the JDK logging properties file logging.properties.

Level

A logging level is a threshold set by the system administrator to control message logging. The logging level can be set to any level supported by the JDK Logging Framework. Once set, only messages having a severity greater than or equal to the defined level are logged.

Example: Setting the level to SEVERE results in the logging of only error messages; setting the level to FINEST results in the logging of all messages.

Log4j Logging Configuration

HDR logging can be configured to use either JDK Logging or Log4J Logging Framework. Th user has to specify which logging framework he would want to use through a system property: HDR_LOG_PROVIDER. The possible list of values for this property are JDK, LOG4J, jdk or log4j. This property should be included in the WebLogic Server startup script as a JVM argument. In case the user doesn't supply this property then, the application falls back on JDK Logging.

To use Log4j Logging, the following steps are required:

1. Download log4j-api-2.10.0.jar and log4j-core-2.10.0.jar files from the Apache website and copy them to `${WL_DOMAIN_HOME}/lib directory`.
2. Create the log4j2.properties configuration file and copy it to the `${WL_DOMAIN_HOME}` directory. A sample configuration file can be found here.
3. Add the -Dlog4j.configurationFile=log4j2.properties JVM argument to WebLogic Server startup script.
4. Add the JVM argument -DHDR_LOG_PROVIDER=LOG4J.
5. Restart the WebLogic Server.

Log Configuration Parameters

By default, the HDR installer creates the JDK logging properties file logging.properties in the hdr_domain directory. The logging.properties sets the default logging to file logging with log file name hdr.log. The default log level *WARNING* is configured in the log module CTBAppsLogger.

HDR Terminology jobs use ETSJobLogger for generating job log files. The default log level is *FINEST* (configurable using logging.properties) and the default handler is file handler which takes name of log file from HDR Terminology PROGRAM_ARGUMENT (this is to maintain specific log file names for specific types of jobs.

HDR Terminology jobs use an internal logger for generating Execution report. This logger is internally configured and its attributes are not configurable.

6

ID and Profile Option Services

HDR OID Service

HL7 defines OID as a globally unique string representing an ISO Object Identifier (OID), in a form that consists only of numbers and dots (Example: 2.16.840.1.113883.3.1). According to ISO, OIDs are paths in a tree structure, with the left-most number representing the root and the right-most number representing a leaf. OIDs are created in HDR through the DataTypeFactory.newOID method.

HDR Object Identifiers

All HDR objects are uniquely identifiable, based on internal identifiers that are system generated at the time of creation. Each such identifier takes the form of an Instance Identifier DataType, which consists of a root and an extension that together uniquely identify an HDR object.The root uniquely identifies the implementing organization represented by this instance of HDR. The extension uniquely identifies this specific instance of the HDR object (e.g. an act, role or entity). Note, user defined or externally supplied instance identifiers may also be persisted for an object. These are in addition to the system-generated identifier. The root of these IIs is modeled as InternalOID RootId. InternalOID has two mandatory attributes, namely RootName and RootId. RootName can be one of the string constants defined in OIDService. RootId is of type OID. ConfigurationFactory methods can be used to create a blank instance of InternalOID.

A set of InternalOIDs must be configured during implementation to enable HDR to generate identifiers. The OIDService interface is used to configure the InternalOID values. HDR will suffix the root with the appropriate extension, to provide the unique identifier for the object.

See Also: ■The, `HL7 web site` current version-3 ballot documentation for details about OID and II data types, and `http://www.hl7.org/oid/` for more information about OIDs.

- `ISO/IEC 8824` standard standard for the ISO standard for further details on OIDs.
- *Oracle Healthcare Data Repository Implementation Guide* for information about HDR Internal OIDs.

Examples: This section contains the following code samples:

- Set up OID Service and Required Factories.
- Create an Internal OID.
- Query an Internal OID
- Query All Internal OIDs
- Update an Internal OID

Example 6–1 Set up OID Service and Required Factories

The following code sample shows how to set up the service and factories required:

```
// Get the OID service
OIDService oidService = mServiceLocator.getOIDService();
// Create the configuration factory instance
ConfigurationFactory configFactory = ConfigurationFactory.getInstance();
// Create the datatype factory instance
DataTypeFactory.getInstance();
```

Example 6–2 Create an Internal OID

The following code sample shows how to configure an Internal OID with a new root id. Use the appropriate factory methods to create the instance of OID and InternalOID:

```
// Create an OID datatype instance using the datatype factory
OID rootOID = dataTypeFactory.newOID("9.989898.5");
// Create the Internal OID object using the configuration factory
InternalOID internalRootOID = configFactory.newInternalRootOID();
// set the root for the Internal OID
internalRootOID.setRootId(rootOID);
// create the Internal OID using the service
oidService.registerOID(internalRootOID);
```

Example 6–3 Query an Internal OID

The following code sample shows how to query an Internal OID. Pass the appropriate Root Name constant as the parameter to this service method. All Root Name constants are defined in OIDService:

```
// Find the InternalOID using the get method
InternalOID internalOID = oidService.getOID(OIDService.INTERNAL_ROOT);
```

Example 6–4 Query All Internal OIDs

The following code sample shows how to query all Internal OIDs. This method returns all InternalOIDs configured in the system:

```
// Find all the InternalOIDs using the get all method
InternalOID[] new line/Enter internalOIDs = oidService.getAllOIDs();
```

Example 6–5 Update an Internal OID

The following code sample shows how to configure an Internal OID by updating its root value. This same method is used for update and to create operations. Update is not allowed if some RIM objects are already using the root:

```
// Create an OID datatype instance using the datatype factory
OID rootOID = dataTypeFactory.newOID("9.989898.9");
// Create the Internal OID object using the configuration factory
InternalOID internalRootOID = configFactory.newInternalRootOID();
```

```
// set the root for the Internal OID
internalRootOID.setRootId(rootOID);
// update the Internal OID using the service
oidService.registerOID(internalRootOID);
```

HDR Profile Option Service

Profile options are configurable preferences that affect the way an application behaves. Application developers can control how their HDR based applications operate by defining new profile options using Oracle Self Service Web Applications and programming the applications to inspect their values at run time to determine behavior. System Administrators can then use Oracle Self Service Web Applications to control operational aspects by setting the values of relevant profile options for specific organizations and users.

The primary use of the Profile Option service ProfileOptionService is to retrieve profile option values at run time. The Profile Option Service also provides methods to let you define profile options and set their values at multiple levels. Typically you define and set profile options using Oracle Self Service Web Applications rather than defining them programmatically.

There are three profile option levels:

- Site
- User

These sections help you to:

- Define Profile Options
- Create Profile Options
- Set Profile Option Values
- Retrieve Profile Option Values

> **See Also:** *Oracle Healthcare Data Repository Implementation Guide* for information about implementing Profile Option Services using Oracle Self Service Web Applications.

Examples: The following code samples help you to:

- Create a Profile Option
- Set a Profile Option Value at Site Level
- Set a Profile Option Value at User Level
- Retrieve a Profile Option Value for Current User
- Retrieve a Profile Option Value at any Level

Define Profile Options

When you define a new profile option, you specify constraints to describe valid values for that profile option, and you can also specify whether your end users can change the value of a profile option. The ProfileOption value object is constructed using the ConfigurationHelper factory class.

If a profile option value type is LOOKUP, you should specify an active concept list that belongs to the LOOKUPS_GROUP group. No validations are performed against the profile option value if the value type is not LOOKUP.

Profile options have system administrator access settings and user access settings to control who can view and change values at certain levels. System administrator settings control which values are visible to and updateable by a user in a system administrator role. A system administrator has default access to all values at all levels. Access settings specify updateable and visible flags for each level for each profile option. When updateable flags are set to Y, visible flags must be set to Y. User access settings are similar but are restricted to the user level. Therefore there is only one set of user access updateable and visible flags per profile option and they default to Y. These flags control whether a user can view or update personal values for this profile option. You can use the getProfileOptions method to retrieve a list of profile options that are accessible by system administrator or other user.

Table 6–1 Profile Options for HDR:

PROFILE_ OPTION_CODE	PROFILE_OPTION_ NAME	DESCRIPTION	Permitted Values
ETS_MLS_ LANGUAGE_ CODE	ETS MLS Language Code	Default language code.	ISO 639 alpha-2 Language Code
ETS_MLS_ COUNTRY_CODE	ETS MLS Country Code	Default country code.	ISO 3166 alpha-2 Country Code
CTB_AU_AUDIT_ FLAG	CTB: Auditing On	Flag to switch ON/OFF Auditing.	Y/N
CTB_AU_ RECEIVE_MSG	CTB: Audit Receive Message	Flag to specify if IMP should create an audit log for message received. Set to 'Y' to audit; set to 'N' otherwise.	Y/N
CTB_MS_STORE_ MESSAGE	CTB: Store Incoming Message	Flag to specify if IMP should store incoming message. Set to 'Y' to store; set to 'N' otherwise.	Y/N
CTB_AU_UPDATE_ OID	CTB: Audit Update OID	Flag to specify if any change in HDR OID configuration should be audited. Set to 'Y' to audit; set to 'N' otherwise.	Y/N
CTB_MTK_ SCHEMA_DIR_ PATH	CTB: MTK schema upload path	Location where the custom schema and MIF files will be uploaded in the HDR server.	Valid middle tier server location to upload MTK custom schema.
CTB_XDS_B_ REGISTRY_ ASYNC_URL	CTB: XDSb Registry Asynchronous Endpoint URL	XDSb Document Registry's Asynchronous URL.	URL
CTB_XDS_B_ REGISTRY_URL	CTB: XDSb Registry URL	XDSb Document Registry's URL.	URL
CTB_XDS_ DOCUMENT_ IMPORT	CTB: XDSb Document Import	Flag to specify if IHE XDS.b document import mode should be turned on. Set to 'Y' to enable document import; set to 'N' otherwise.	Y/N

Table 6–1 (Cont.) Profile Options for HDR:

PROFILE_ OPTION_CODE	PROFILE_OPTION_ NAME	DESCRIPTION	Permitted Values
CTB_XDS_AUDIT_ SERVER_PORT	CTB: XDSb Audit Server Port	ATNA audit logging server port number. Either UDP or TLS port based on CTB_XDS_ AUDIT_SERVER_ TRANSPORT_PROTOCOL value.	Port number
CTB_XDS_AUDIT_ SERVERNAME	CTB: XDSb Audit Server Name	ATNA audit logging server hostname or IP address.	Hostname or IP address
CTB_XDS_AUDIT_ SERVER_ TRANSPORT_ PROTOCOL	CTB: XDSb Audit Server Transport Protocol	The protocol to be used for IHE ATNA audit logging.	UDP or TLS
CTB_XDS_ REPOSITORY_ UNIQUE_ID	CTB: XDSb Repository Unique Id	IHE XDS.b document repository unique id.	Valid OID
CTB_XDS_WS_ ATOMIC_ TRANSACTION	CTB: XDSb WS-Atomic Transaction	Flag to specify if XDS.b Document Registry is invoked using WS-Atomic Transaction Protocol. Set to 'Y' to enable; set to 'N' otherwise.	Y/N
CTB_EN_USE_ VAL_CRITERIA	CTB: Validation Criteria for EN_USE	Depending on this criteria, the concept list for validating EN will be picked. 0 for CL_EN_ USE, 1 for CL_EN_USE_EXT and 2 for CL_EN_USE union CL_EN_USE_EXT.	0,1 or 2

Create Profile Options

Example 6–6 Create a Profile Option

This code sample creates a profile option within the following constraints. Its purpose is to create a profile option for use in the subsequent examples in this section. You should use Oracle Self Service Web Applications and avoid programmatic creation of a new profile option. Note that the example can be executed only once for any given ProfileOptionCode.

- *Value type is LOOKUP; use the concept list CTB_YES_NO.*
- *System Administrator access only.*
- *Site level and User level are updateable and visible.*

```
 ConfigurationHelper configHelper = new ConfigurationHelper();

ProfileOptionService profileOptionService =
serviceLocator.getProfileOptionService();

//Construct profile option value object
ProfileOption profileOption = configHelper.newProfileOption();
//Set the profile option code.  scenarioProfileOptionCode is a String variable,
and must be unique.
profileOption.setProfileOptionCode(scenarioProfileOptionCode);
profileOption.setProfileOptionName("Scenario Profile Option");
profileOption.setDescription("Profile option created by programmers guide.");
//Setting value type constraint. Valid value is null or LOOKUP
profileOption.setValueTypeCode("LOOKUP");
```

```
//Lookup type is any concept list in the LOOKUP_GROUP group
profileOption.setConstraint1("CTB_YES_NO");
//By default, all levels are Y.  So you must turn off other levels.
profileOption.setOrgVisibleFlag("N");
//This profile option only allows system administrator role to view and change
//the profile option value.  The user access flag must be turned off.
profileOption.setUserAccessVisibleFlag("N");
profileOption.setUserAccessUpdateableFlag("N");

//Create the profile option now
profileOptionService.createProfileOption(profileOption);
```

Set Profile Option Values

After the profile option is created, a system administrator or other user can set profile option values at each profile level by calling the setProfileOptionValue(ProfileOptionValue) method. Whenever you set profile option values, the prior values are overwritten. An exception is thrown when constraints set in the profile option are not met. Refer to setProfileOptionValue(ProfileOptionValue) (in the Oracle Healthcare Data Repository API Documenation) for more information about exceptions thrown by this method. The ProfileOptionValue value object is constructed using the ConfigurationHelper factory class.

Example 6–7 Set a Profile Option Value at Site Level

The following code sample sets a profile option value at the Site level for the profile option [scenarioProfileOptionCode] created in Example 6-7. Its purpose is to set the Site level option used in the later examples in this section. The Site level option values affect the way all applications run at a given installation and should be set using Oracle Self Service Web Applications rather than setting the values programmatically.

```
ProfileOptionValue siteLevelProfOptionVal = configHelper.newProfileOptionValue();
siteLevelProfOptionVal.setProfileOptionCode(scenarioProfileOptionCode);
//Set the level value.  The valid values are SITE and USER.
siteLevelProfOptionVal.setProfileOptionLevelCode("SITE");
//The profile option value must be a valid value in the CTB_YES_NO concept list.
siteLevelProfOptionVal.setProfileOptionValue("N");
profileOptionService.setProfileOptionValue(siteLevelProfOptionVal);
```

Example 6–8 Set a Profile Option Value at User Level

The following code sample sets a profile option value at user-level (sysadmin) for the profile option [scenarioProfileOptionCode] created in Example 6-7. Its purpose is to set the User level option used in the later examples in this section. The User level option values affect the way the application runs for a given user and should be set using Oracle Self Service Web Applications rather than setting the values programmatically.

```
ProfileOptionValue userLevelProfOptionVal = configHelper.newProfileOptionValue();
userLevelProfOptionVal.setProfileOptionCode(scenarioProfileOptionCode);
//Set the level value.  The valid values are SITE and USER.
userLevelProfOptionVal.setProfileOptionLevelCode("USER");
//The profile option level value must be a valid user account login identity.
userLevelProfOptionVal.setProfileOptionLevelValue("SYSADMIN");
//The profile option value must be a valid value in the CTB_YES_NO concept list.
userLevelProfOptionVal.setProfileOptionValue("Y");
profileOptionService.setProfileOptionValue(userLevelProfOptionVal);
```

Retrieve Profile Option Values

Your HDR-based application can retrieve profile option values in two ways:

- By retrieving a profile option value at a level such as User for a particular instance of that level, such as *user name(login identity)*.
- By retrieving a profile option value without specifying any level. In this case, the system automatically accounts for level precedence, and returns the most appropriate value. *User* level has higher precedence than *Site* level.

Example 6–9 Retrieve a Profile Option Value for Current User

The following code sample retrieves the value of profile option [scenarioProfileOptionCode] created in Example 6-8 for the current login user. It retrieves the profile option value of Y set in Example 6-9.

```
//Get the current login user's login identity
SessionService sessionService = serviceLocator.getSessionService();
String value =
profileOptionService.getProfileOptionValueForLevel(scenarioProfileOptionCode,"USER
" 'SYSADMIN');
//Retrieve the value set in Example 6-9
String value =
profileOptionService.getProfileOptionValueForLevel(scenarioProfileOptionCode,"USER
",loginId);
```

Example 6–10 Retrieve a Profile Option Value at any Level

The following code sample retrieves the value of profile option [scenarioProfileOptionCode] created in Example 6-7 without specifying a level:

```
String value =
profileOptionService.getProfileOptionValue(scenarioProfileOptionCode);
```

The Site and User level option values were set for [scenarioProfileOptionCode] in Example 6-8 and Example 6-9. This method returns the current login User's value of Y because the *User* level has higher precedence than the *Site* level.

7

RIM Services

The oracle.hsgbu.hdr.hl7 package contains a single interface [RIM Service] that provides the main entry point into HDR's query functionality and RIM object persistence.

Submit a Query

Example 7–1 Submit a Query

Use the following RimService APIs to query Acts, ActRelationships, Participations, Roles and Entities directly:

- `queryActs(ServiceLocator s1, ActFetch fetch)` throws HDRRimException
- `queryParticipations(ServiceLocator s1, ParticipationFetch fetch)` throws HDRRimException
- `queryRoles(ServiceLocator s1, RoleFetch fetch)` throws HDRRimException
- `queryEntities(ServiceLocator s1, EntityFetch fetch)` throws HDRRimException
- `queryActRelationships(ServiceLocator s1, ActRelationshipFetch fetch)` throws HDRRimException

The results of a query are an Iterator of Acts, ActRelationships, Participations, Roles and Entities respectively.

HDR HL7 Domain Constants

Domain constants provide consistent definitions for structural codes. The names of the classes correspond to HL7 vocabulary domains that are coded no exceptions (CNE). Domain constants should be used wherever possible.

- Import the ActStatus Domain Class
- Use the ActStatus Domain Class to Set Status Code

Example 7–2 Import the ActStatus Domain Class

Import the ActStatus Domain Class

```
import oracle.hsgbu.hdr.hl7.rim.domain.ActStatus;
```

Example 7–3 Use the ActStatus Domain Class to Set Status Code

Use the ActStatus domain class to set the status code on an act that is being created or updated.

```
act.setStatusCode(ActStatus.ACTIVE);
```

The following domain classes are available:

- AcknowledgementCondition
- AcknowledgementType
- ActClass
- ActMood
- ActRelationshipCheckpoint
- ActRelationshipJoin
- ActRelationshipSplit
- ActRelationshipType
- ActStatus
- ActUncertainty
- BinaryDataEncoding
- CommunicationFunctionType
- ContextControl
- EntityClass
- EntityDeterminer
- EntityStatus
- LocalMarkupIgnore
- ManagedParticipationStatus
- ModifyIndicator
- NullFlavor
- ParticipationSignature
- ParticipationType
- ProcessingID

- ProcessingMode
- QueryPriority
- QueryResponse
- QueryStatusCode
- RelationalOperator
- RelationshipConjunction
- ResponseLevel
- ResponseModality
- RoleClass
- RoleLinkType
- RoleStatus
- Sequencing

HDR Factories

This section contains the following topics:

Factories

Classes in this package provide factory methods for creating instances of the following data types and classes:

- Data Type Factory
- RIM Object Factories
- Query Component Factory

A reference to ServiceLocator is required to get an instance of each factory. Factories are thus tied to a specific HDR session, and should not be cached and reused across different login sessions.

> **Note:** *Do not use Java's new operator to construct instances of HDR classes*. Use factories instead— to ensure that RIM objects or data types that require a service locator for their operation will have access to one.

Data Type Factory

Because the Code System UID is mandatory on coded datatypes (CD, CE, CV), factory methods that do not include the code system UID (Example: methods that take the code system name instead) incur a performance penalty while the factory method requests the UID from ETS.

To achieve optimal performance, provide the code system UID explicitly. You can retrieve the complete mapping of code system names to UIDs by querying the corresponding ETS coding system.

Note: You can also retrieve the complete mapping of code system names to UIDs in the following way:

- For coding systems registered with HL7, an online OID Registry is provided as a free service by the HL7 website.

RIM Object Factories

You can create RIM objects through one of the three available factories: ActFactory, EntityFactory, or RoleFactory.

Each factory contains one generic factory method, for creating Act, Entity, and Role objects respectively:

- ActFactory.newAct
- EntityFactory.newEntity
- RoleFactory.newRole

In addition, there are numerous convenience methods within each class for creating various subtypes of Acts, Entities and Roles.

Examples:

- Create RIM Observations Using the Generic Factory Method.
- Create RIM Observations Using the Convenience Factory Method.

Example 7–4 Create RIM Observations Using the Generic Factory Method

```
DataTypeFactory dtf = DataTypeFactory.getInstance();
ActFactory af = ActFactory.getInstance(serviceLocator);
Observation observation = (Observation) af.newAct(ActClass.OBS,
ActMood.EVN, dtf.nullCD(NullFlavor.NI), dtf.nullSET_II(NullFlavor.NI));
```

Example 7–5 Create RIM Observations Using the Convenience Factory Method

The following example illustrates the preferred method for creating RIM objects:

```
DataTypeFactory dtf = DataTypeFactory.getInstance();
ActFactory af = ActFactory.getInstance(serviceLocator);
Observation observation = af.newObservation(ActMood.EVN,
dtf.nullCD(NullFlavor.NI),
dtf.nullSET_II(NullFlavor.NI));
```

Query Component Factory

See Also: HDR Query for information about creating criteria and fetch objects using this factory, submitting them to RIM Service, and interpreting results returned.

Reference Modifiers

In addition to factory classes, the `oracle.hsgbu.hdr.hl7.rim.factories` package contains the ReferenceModifier class, a typed enumeration of all possible values for RIM class reference modifiers.

> **See Also:** Use RIM Services for information about how RIM service persistence uses ReferenceModifier values.

HDR Query

- Fetches
- Criteria
- Navigate the Result Graph
- HDR RIM Extensions
- Original Coded Attributes

The query framework provides a RIM-like Java interface to retrieve data from the HDR repository. This document introduces constructs and features included with the query framework to retrieve information.

At the high level, a query is defined in terms of fetch and criteria objects. A fetch corresponds to a request for a certain type of information (for example, an ActFetch is a request for act information). Fetches are used in combination with criteria, which specify conditions objects must meet if they are to be retrieved by the query. In SQL query terms, a fetch corresponds to the *SELECT/FROM* clause and the criteria to the *WHERE* clause.

The code samples below help you do the following:

- Retrieve the Patient Id
- Construct Patient Role
- Associate Patient Role Criteria with Role Fetch
- Incremental Fetch / Ordering
- Using QBE API to Set Class Code Criteria
- Using QBE API to Set Instance Identifier Criteria
- Implicit AND Behavior: Restricting Retrieved Results
- Submit a Query
- Retrieve Ultimate Survivor of an Entity
- Patient Query by ID Code Sample
- Query for Identified Patient Roles and Player Entities

> **Query Submission Diagram Conventions :**
> - *Fetches* are represented by red boxes.
> - *Criteria* are represented by boxes that correspond to standard RIM diagram colors for their object type.
> - Text adjacent to links distinguishes between link types where necessary (such as Player vs Scoper).

Scenario

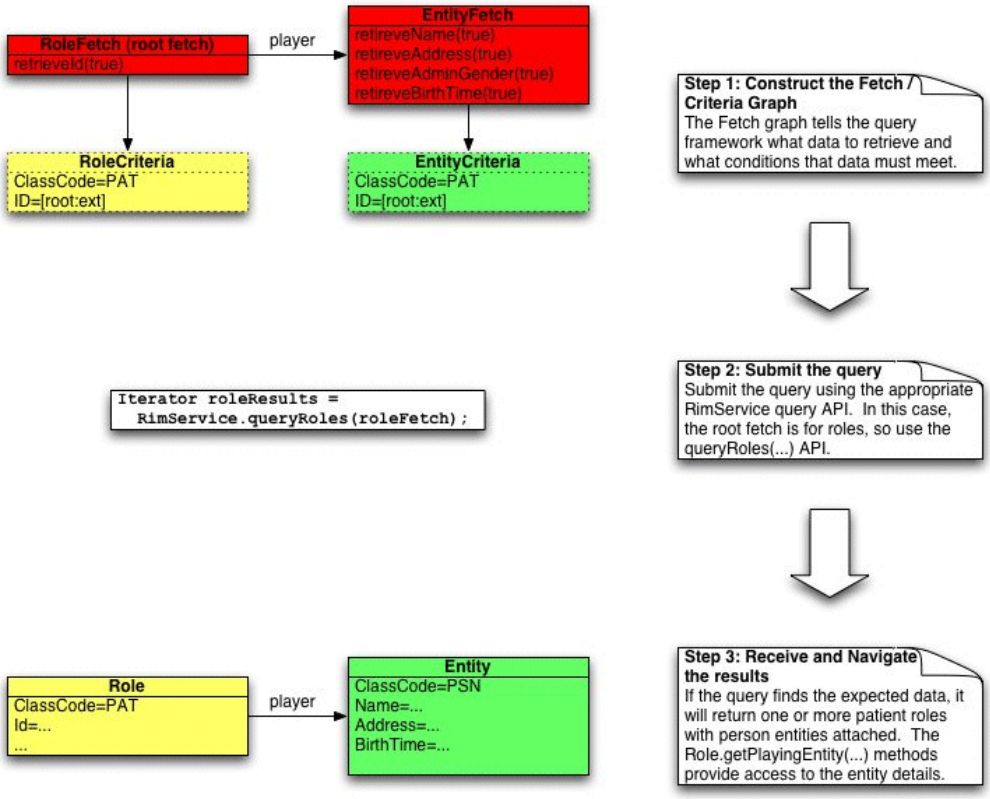
This scenario illustrates the following query fundamentals:

- Submitting a query.
- Combining fetches and criteria to retrieve data.
- Resulting object graph and the fetch graph symmetry.
- Navigating object graph results.

Figure 7.5.1 describes how to retrieve a patient (including the playing entity) based on the patient id. The following information must be retrieved:

- The patient role with the id attribute.
- The playing person entity with the name, address, birth time and administrative gender code attributes.

Figure 7–1 Scenario Description



Fetches

A *fetch* is a request for information. Queries consist of one or more connected fetches that request some combination of Act, ActRelationship, Participation, Entity and Role data. These sections describe important fetch functions used in querying:

- Flexible Retrieval
- Set Criteria on Fetches
- Add Detail Fetches

- Incremental Fetches
- Order Fetch Results
- Cyclic (Recursive) Fetches

Flexible Retrieval

Flexible retrieval is the ability to specify a subset of attributes to retrieve in a query. This lets you optimize performance by retrieving only the data necessary to satisfy application requirements. For each fetch there are several retrieve<AttributeName>(boolean bool) methods that can be used to specify which attributes should be retrieved.

Example 7–6 Retrieve the Patient Id

The following code line requests the retrieval of the patient id:

```
patRoleFetch.retrieveId(true);
```

By default, no attributes are retrieved so you must call the appropriate retrieve method for all of the attributes you require. For convenience, fetches include a retrieveAll() method that specifies that all attributes of the class should be retrieved. There is also a retrieveNone() method that acts in reverse to the retrieveAll() method. The retrieve methods used by a fetch specify the *minimum* set of data that the system must retrieve to fulfill the query; the platform may retrieve additional data beyond that requested.

Set Criteria on Fetches

Criteria are used in combination with fetches to constrain the data retrieved by a query. You can associate criteria with a fetch in two ways:

- Pass the criteria as a parameter to the factory method for the fetch. Each fetch factory method is overloaded to accept a criteria object of the corresponding HL7 type (Example 7-16).
- Call setTopLevel<HL7Class>Criteria, which associates a criteria with a preexisting fetch.

> **Note:** Use of either method (Examples 7-16, 7-17) is discretionary; there is no functional difference between the two methods.

Example 7–7 Construct Patient Role

In the introductory scenario the patient role fetch was constructed with the role criteria using the following code line:

```
RoleFetch patRoleFetch = queryComponentFactory.newRoleFetch(patRoleCriteria);
```

Associating Patient Role Criteria with Role Fetch: Each fetch has a setTopLevel<HL7Class>Criteria(...) method that associates criteria with a pre-constructed fetch.

Example 7–8 Associate Patient Role Criteria with Role Fetch

To associate the patient role criteria with a pre-constructed role fetch, use the following code:

```
RoleFetch patRoleFetch = queryComponentFactory.newRoleFetch();
patRoleFetch.setTopLevelRoleCriteria(patRoleCriteria);
```

Add Detail Fetches

Fetches can be linked together to retrieve related Act, ActRelationship, Participation, Role and Entity data in a single query. The added fetches are called detail fetches. Detail fetches behave like normal fetches allowing specific attributes to be retrieved and criteria to be specified, but they only retrieve data associated with their master.

Detail fetches are added using the add<RelatedObject>Fetch(...) APIs on the fetch class. These APIs are specific to each fetch to restrict the query results to a RIM consistent structure. As the API name suggests, it is possible to add multiple detail fetches of the same type to a master, each having their own set of attributes to retrieve, with their own criteria. This lets you group detail objects by criteria. For example, you can retrieve the subject participations and attending physician participations of an act separately by adding two participation detail fetches (one with criteria for subject participations and the other with criteria for the attending physician participations) to the main act fetch.

Detail fetches also play an important role in navigating the result graph, as every result object is related to the fetch that caused it to be retrieved.

Note: Navigate the Result Graph for more information about detail fetches.

Incremental Fetches

The query framework has a pseudo-incremental fetch mechanism that lets you retrieve blocks of results by a fetch. This is useful to improve query response time and to page through results when processing large result sets.

Developers specify the size of the result set (window size) required using the setRetrievalWindow(int first, int last) API on the top level fetch. To implement a paging solution, submit the same fetch graph to the RimService multiple times modifying the retrieval window accordingly. To ensure that subsequent queries return consistent results, you should apply an ordering directive on the fetch (described in the next section). Sample code demonstrating a paging solution follows the ordering fetch results discussion (see Example 7-18).

Note: Read consistency is not guaranteed between calls. If data that matches the query criteria is persisted between subsequent fetches the newly persisted data is retrieved.

Order Fetch Results

To facilitate incremental fetching, the query framework lets you order results in ascending or descending order of a subset of attributes. This is to ensure that the result set is ordered, so that subsequent incremental fetch results are consistent.

Specify the set of ordering attributes by using the addOrderBy(String attribute, int order API on the top level fetch. Results can be ordered by a restricted set of attributes represented by constants in QueryComponentFactory (look for ORDER_BY_ <ATTRIBUTE_NAME> constants).

Example 7–9 Incremental Fetch / Ordering

The following example illustrates how to use the incremental fetch and ordering features of the query framework to retrieve ten acts at a time:

```
// create Act fetch and criteria
```

```
// order the fetch by 1 or more orderable act attributes
actFetch.addOrderBy(QueryComponentFactory.ORDER_BY_ACT_EXISTENCE_TIME_LOW,
 QueryComponentFactory.ORDER_BY_ASCENDING);
// set the intial window size
actFetch.setRetrievalWindow(1, 10);
// execute the query
Iterator firstTenResults = rimService.queryActs(serviceLocator, actFetch);
// display first 10 results
...
// set the next window size
actFetch.setRetrievalWindow(11, 20);
// execute the results
Iterator secondTenResults = rimService.queryActs(serviceLocator, actFetch);
// display second 10 results
...
```

Cyclic (Recursive) Fetches

Fetches can be linked in cycles to retrieve data that is recursively related. This lets you avoid coding individual fetches for each component in the recursive relationship. This is particularly useful for retrieving a series of related acts.

Figure 7–2 Cyclic Recursive Fetches

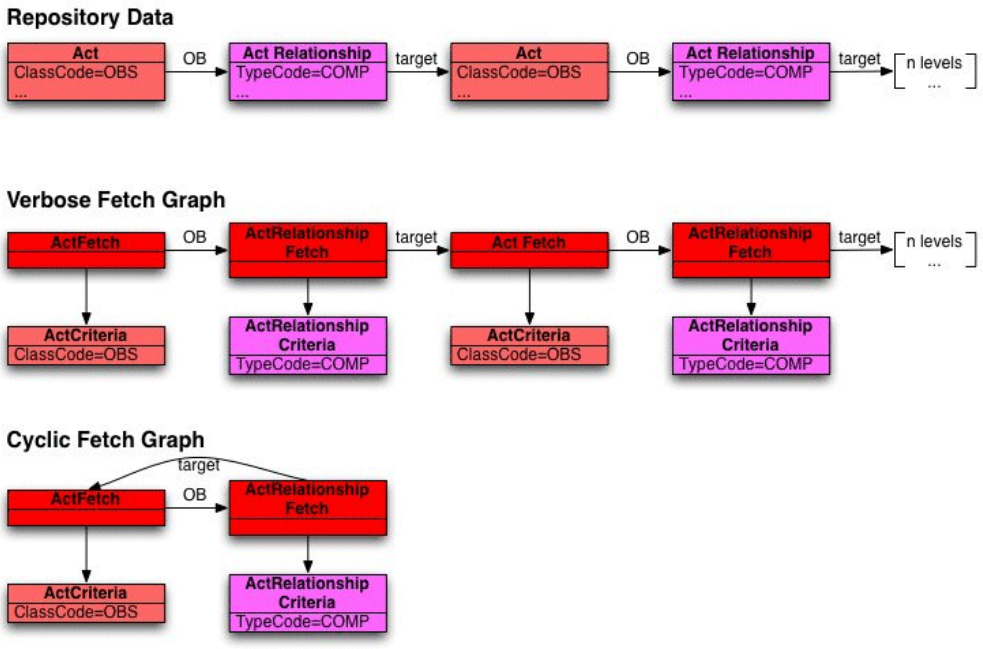


Figure 7–2 illustrates how cyclic fetches can be used to retrieve an entire chain of acts, avoiding the need to code an entire chain of fetch/criteria objects:

Note: *Exercise care when using this feature*; the query response time and memory consumption is relative to the depth of recursion and there is no mechanism to restrict the level of results retrieved. If you have data that exists in a deeply recursive structure and you only want to recurse to a certain level it may be better to explicitly code the expanded fetch graph rather that use the cyclic fetch functionality.

Criteria

Query criteria are used to specify conditions HL7 objects must satisfy to be retrieved by a particular query–a criteria graph is analogous to the WHERE clause in a SQL Select statement.

Criteria exist for the main HL7 objects (Acts, ActRelationships, Participations, Roles and Entities) as well as the more complex datatypes that provide a high degree control in restricting the data being retrieved.

There are two fundamental types of criteria:

- *Attribute criteria*, which let you set restrictions on attributes such as an Act class code.
- *Connective criteria*, which allow attribute criteria to be *ANDed* and *ORed* together to construct more complex conditions.

Criteria described in further detail:

- Attribute Criteria
- Query-by-Example (QBE) APIs
- Query-by-Criteria (QBC) APIs
- CodedTypeCriteria APIs
- Implicit AND Behavior for Attribute Criteria
- Connective Criteria

Attribute Criteria

AttributeCriteria are used to specify conditions that object attributes must satisfy to be retrieved by a query. Specific criteria classes exist for the core HL7 objects and for datatypes where necessary. The criteria classes provide a combination of *query-by-criteria (QBC)* and *query-by-example (QBE)* APIs depending upon the type of attribute. The following table lists all of the attribute types and indicates whether a QBE or QBC interface is exposed for each type:

Attribute Types

Query-by-Example Types	Query-by-Criteria Types
BL	AD
CS	ADXP
CD/CE	CD/CE
II	EN
INT	ENXP
REAL	GTS
ST	IVL<DATATYPE>
TS	MO
URL	PQ
RTO<DATATYPE, DATATYPE>	
SC	
TEL	

ED

Query-by-Example (QBE) APIs

Query-by-example interfaces are provided for attributes where constraints can be adequately specified using an example datatype. The following attribute types fall into this category:

- *BL*
- *CS*
- *CD*
- *CE*
- *II*
- *INT*
- *REAL*
- *ST*
- *TS*
- *URL*

Where the attribute is represented by a single datatype the interface has the following form:

public void set<AttributeName>(SearchOperator op, <DataType> value);

where SearchOperator is one of the following:

- EQUALS
- NOT_EQUALS
- GREATER_THAN
- LESS_THAN
- GREATER_THAN_OR_EQUAL
- LESS_THAN_OR_EQUAL
- IS_NULL
- IS_NULL
- IS_NOT_NULL
- LIKE
- NOT_LIKE

Example 7–10 Using QBE API to Set Class Code Criteria

The following code sample illustrates the usage of this style of API for setting a class code criteria for an entity:

```
entityAttributeCriteria.setClassCode(SearchOperator.EQUALS, EntityClass.PSN);
```

The following interface is exposed for set based attributes *(SET<CS>, SET<CD>, SET_II etc):*

```
public void set<AttributeName>(SetSearchOperator op, <DataType>[] values);
```

The following special operators are provided to support set based searching: ALL and ANY. The ALL operator is used to specify that the target search object must have *all* of

the values passed in the array type. The ANY operator means that the object must have *at least one* of the values passed in the array to match.

Example 7–11 Using QBE API to Set Instance Identifier Criteria

The following code sample illustrates typical usage of this form of API when setting Instance Identifier criteria for an object:

```
II[] patientIIs = new II[1];
...

patientIIs[0] = dataTypeFactory.newII(dataTypeFactory.newUID("ROOT"),
                                      dataTypeFactory.newST("EXTENSION"),
                                      dataTypeFactory.newBL(true));
   patRoleCriteria.setId(SetSearchOperator.ANY, patientIIs);
```

Query-by-Criteria (QBC) APIs

Query-by-Criteria interfaces are provided for attributes represented by complex datatypes-those where using an example datatype to specify a constraint is not flexible enough to meet common uses. For example, there is a criteria interface for EN attributes because it consists of a set of use codes (SET<CS>), a list of name parts (LIST<ENXP>), a formatted string (ST), and a valid time (IVL<TS>, all of which require individual constraints to be set in various use cases. This cannot be specified with a QBE interface. The attribute sets that fit into this category are represented by the following datatypes:

- AD
- ADXP
- ED
- EN
- ENXP
- GTS
- IVL<INT>
- IVL<MO>
- IVL<PQ>
- IVL<REAL>
- IVL<TS>
- MO
- PQ
- RTO<MO,PQ>
- RTO<PQ>
- SC
- TEL

...where the attribute is represented by a complex datatype the interface has the following form:

public void set<AttributeName>Criteria(<Datatype>Criteria criteria);

The API differs from the QBC interface for complex datatypes by accepting a versioning type parameter. This is used to specify whether the neighboring criteria

should be applied to the version that is directly related to the master object (version dependent), or to any version of the detail object (version independent). Versioning and query are discussed in more detail in Versioning and Query.

CodedTypeCriteria APIs

You can use CodedTypeCriteria expose functionality, in the API, to query by qualifiers, classifications, and equivalence. Classifications and equivalence are defined in ETS (Enterprise Terminology Services). You can combine three types of queries using ConnectiveCriteria. You can apply only one type of query on any single Criteria object.

- Querying-by-Qualifiers
- Querying-by-Classifications
- Querying-by-Equivalence

Querying-by-Qualifiers Queries on CD.qualifier follow the Query-by-Criteria pattern described in the Criteria section. The following operators can be used for qualifier queries:

- SearchOperator.IS_NULL
- SetSearchOperator.ANY
- SetSearchOperator.NONE
- SetSearchOperator.ALL

CodedTypeAttributeCriteria.includeEquivalentCodes(boolean, String, boolean) cannot be used with the CodedTypeAttributeCriteria.setQualifier(SearchOperator, CR[]) method.

```
CodedTypeAttributeCriteria codedTypeCriteria =
queryComponentFactory.newCodedTypeAttributeCriteria();

// use the DataTypeFactory to instantiate the data types

codedTypeCriteria.setCode(parentCode);
codedTypeCriteria.setCodeSystem(parentCodeSystem);
codedTypeCriteria.setQualifier(SearchOperator.IS_NULL, new CR[] { } );

// null values are passed in for the II, the StatusCode and the CurrentVersionFlag,
because we are not
// querying using these attributes

RoleAttributeCriteria roleCriteria =
queryComponentFactory.newRoleAttributeCriteria(RoleClass.ACCESS, roleCode,
null, null, null);

roleCriteria.setTargetSiteCode(codedTypeCriteria);
RoleFetch roleFetch = queryComponentFactory.newRoleFetch(roleCriteria);
roleFetch.retrieveAll();

Iterator fetchedRoles = rimService.queryRoles(localServiceLocator, roleFetch);
```

Querying-by-Classifications Classification queries are constructed using ClassificationCriteria, which is a subclass of CodedTypeCriteria. Only ClassificationAttributeCriteria.setCode(ST) applies to a query by classification. Other methods inherited from CodedTypeAttributeCriteria are not applicable. This is

because the value is predetermined to be one appropriate to the code system that contains classifications or because the method does not apply to this type of query.

ClassificationAttributeCriteria classificationCriteria = qcf.newClassificationAttributeCriteria();

// Set the classification code; no code system is needed because classifications are all

// stored in a specific code system that is known to the system.

classificationCriteria.setCode(classificationCode);

ActAttributeCriteria actCriteria = this.queryComponentFactory.newActAttributeCriteria();

actCriteria.setCode(codedTypeAttrCriteria);

actCriteria.setId(SetSearchOperator.ALL, ii);

ActFetch actFetch = this.queryComponentFactory.newActFetch(actCriteria);

actFetch.retrieveAll();

Iterator fetchedActs = rimService.queryActs(localServiceLocator, actFetch);

Querying-by-Equivalence Equivalence queries must use CodedTypeAttributeCriteria.includeEquivalentCodes(boolean, String, boolean) method. These queries must specify a code and code system. The setQualifierMethod(SearchOperator, CR[]) method cannot be invoked on a CodedTypeAttributeCriteria instance where includeEquivalentCodes(..) has been called.

```
CodedTypeAttributeCriteria codedTypeAttrCriteria =
this.queryComponentFactory.newCodedTypeAttributeCriteria();
codedTypeAttrCriteria.setCode(actCode);
codedTypeAttrCriteria.setCodeSystem(actCodeSystem);
codedTypeAttrCriteria.includeEquivalentCodes(true, null, true);
ActAttributeCriteria actCriteria =
this.queryComponentFactory.newActAttributeCriteria();
actCriteria.setCode(codedTypeAttrCriteria);
actCriteria.setId(SetSearchOperator.ALL, ii);

ActFetch actFetch = this.queryComponentFactory.newActFetch(actCriteria);
actFetch.retrieveAll(); Iterator fetchedActs =
rimService.queryActs(localServiceLocator, actFetch);
```

Implicit AND Behavior for Attribute Criteria

When multiple criteria are set on an instance of an AttributeCriteria (regardless of the type), they are implicitly ANDed together. For example, see Example 7-21:

Example 7–12 Implicit AND Behavior: Restricting Retrieved Results

In this code sample, the retrieved results are restricted to those Acts that have a ...

```
actAttributeCriteria.setMoodCode(SearchOperator.EQUALS, ActMood.EVN);
...
ctAttributeCriteria.setClassCode(SearchOperator.EQUALS, ActClass.OBS);
```

ConnectiveCriteria can be used to build more complex criteria where ANDs and ORs can be used.

Connective Criteria

Connective criteria are used to AND and OR attribute criteria together, permitting more complex queries to be constructed. You must use connective criteria with attribute criteria of the same type. For example, you can use an ActConnectiveCriteria to AND/OR multiple Act criteria (attribute or connective) together. You should exercise care with regard to the level the connective is applied; otherwise the results may not be what you expected.

Figure 7–3 Retrieve active encounter event and observation event Acts: Example 1:

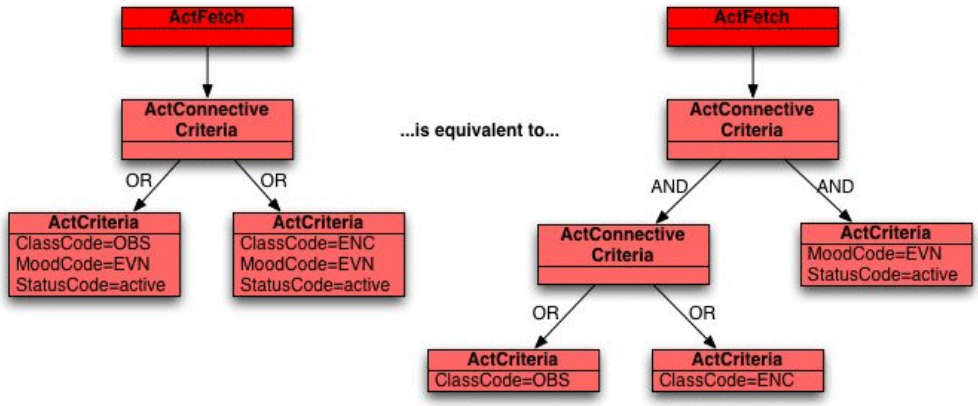
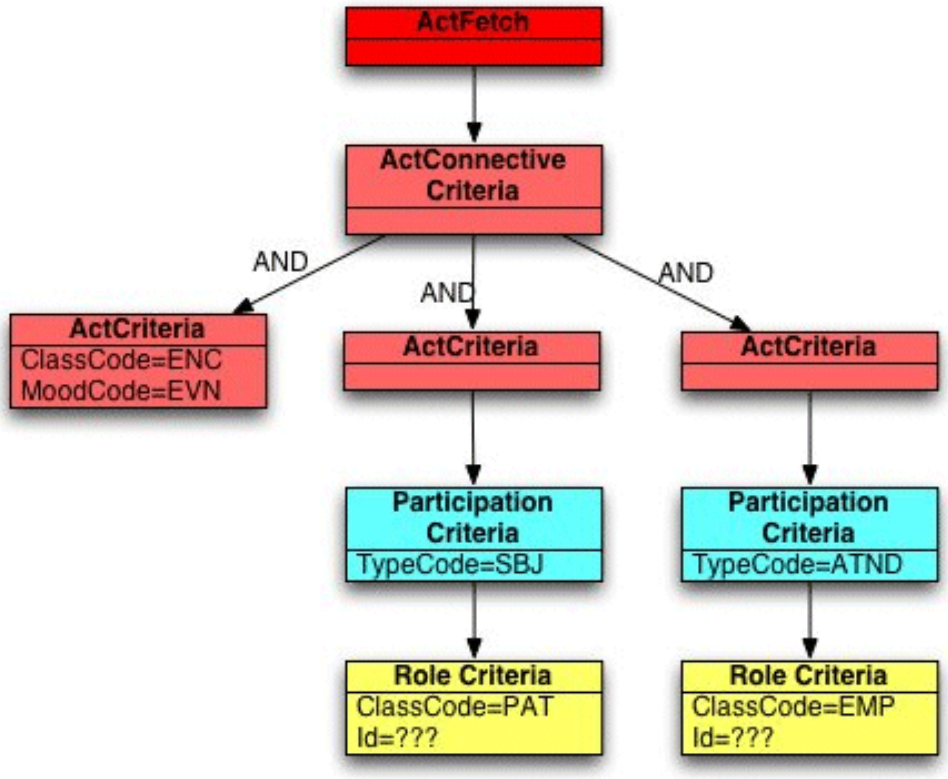


Figure 7–3 illustrates the use of connective criteria. In this example, two equivalent fetch diagrams show that the same result can be achieved in different ways. In the first diagram, an OR connective is used between two act attribute criteria to fetch the required results. The result of the query will be the union of the results returned by the execution of the two attribute criteria independently. The criteria on the left retrieves all active encounter event acts, and the criteria on the right retrieves all active observation events. The resulting set of retrieved acts will be the union of the two sets.

The second diagram is semantically equivalent but achieves the result using a combination of AND and OR connectives. The left side of the AND connective retrieves all active acts in EVN mood and the right side retrieves all acts of class ENC or OBS. The end result is the intersection of the two individual sets-all active encounter and observation events.

Figure 7–4 Retrieve encounter events that have a subject participation to an identified patient role with an attending participation to an identified employee role: Example 2:



As in Example 1, you must exercise care to ensure that the connective is made at the correct level.

Figure 7–4 shows two branches of criteria connected at the top level by an AND connective. The result of the query will be the intersection of the results returned by the execution of the two branched independently. The left branch retrieves all encounter events that have an SBJ participation to a PAT role. The right branch retrieves all encounter events that have an ATND participation to an EMP role.

Figure 7–5

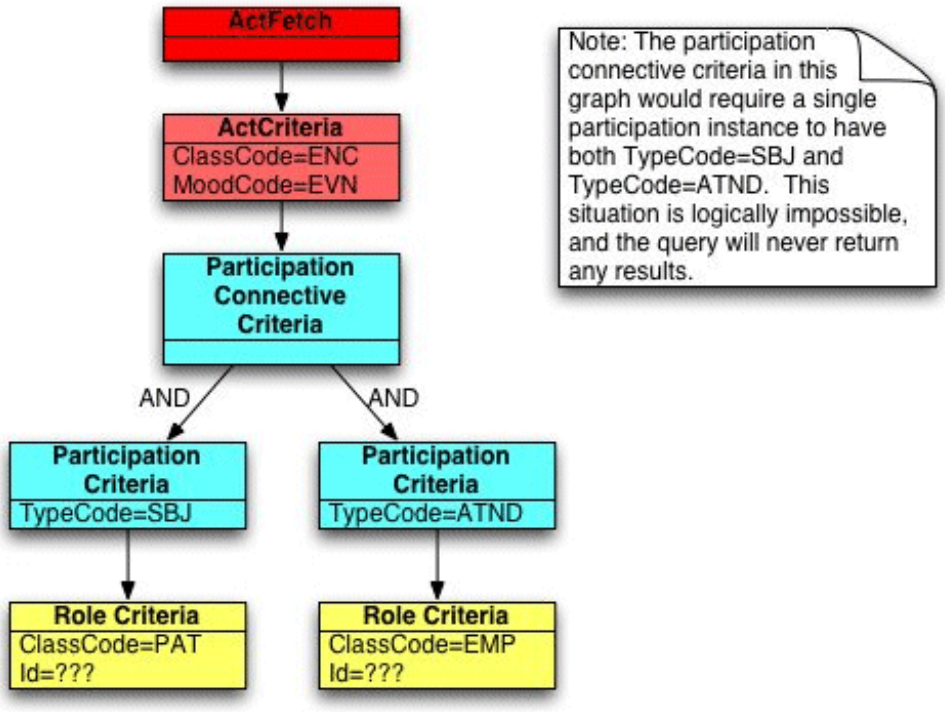


Figure 7–5 shows a typical mistake of specifying the above criteria with the connective at the participation level. Logically this can be thought of as asking for encounter events that have a participation with a class code of SBJ and ATND-an impossible construct that returns no results.

Navigate the Result Graph

Complex queries that consist of several fetches return a RIM object structure that replicates the fetch graph that was submitted (assuming that existing data matches the criteria specified). The core RIM classes contain specific APIs that let you navigate from the master object to its related details. The following table lists the complete set of core RIM navigational APIs:

Core RIM Navigational APIs

Class	Navigational API	Description
Act	getParticipations(...)	Returns act participations fetched by a detail participation fetch.
	getIBActRelationships(...)	Returns inbound act relationships fetched by a detail inbound act relationship fetch.
	getOBActRelationships(...)	Returns outbound act relationships fetched by a detail outbound act relationship fetch.

ActRelationship	getSource(...)	Returns the source act that was fetched by a detail act fetch.
	getTarget(...)	Returns the target act that was fetched by a detail act fetch.
Participation	getAct(...)	Returns the act that was fetched by a detail act fetch.
	getRole(...)	Returns the role that was fetched by a detail role fetch.
Role	getParticipations(...)	Returns participations fetched by a detail participation fetch
	getPlayerEntity(...)	Returns entities fetched by a detail player entity fetch
	getScoperEntity(...)	Returns entities fetched by a detail scoper entity fetch
Entity	getPlayedRoles(...)	Returns roles fetched by a detail played role fetch
	getScopedRoles(...)	Returns roles fetched by a detail scoped role fetch
	getOwnedPlayedRoles(...)	Returns roles fetched by a detail owned played role fetch
	getOwnedScopedRoles(...)	Returns roles fetched by a detail owned scoped role fetch

Note: Fetches have additional navigation methods for HDR extensions such as control act and merge navigations, discussed in HDR RIM Extensions.

Two types of navigational APIs provide flexibility when navigating the result graph. The simplest form accepts no parameters and returns all details retrieved regardless of which particular fetch was responsible for the result being retrieved.

The overloaded form accepts a fetch parameter and returns results directly related to the execution of the detail fetch passed in. This API is used when multiple detail fetches of the same type are added to a fetch.

For example, to retrieve an act, subject and attending physician participations you could have an act fetch and add separate subject and attending physician participation fetches. To navigate from the act to the participations, you can retrieve both concurrently by calling the no parameter getParticipations method, or you can access each participation individually by calling the navigation method that takes a participation fetch parameter passing in the particular fetch.

Figure 7–6

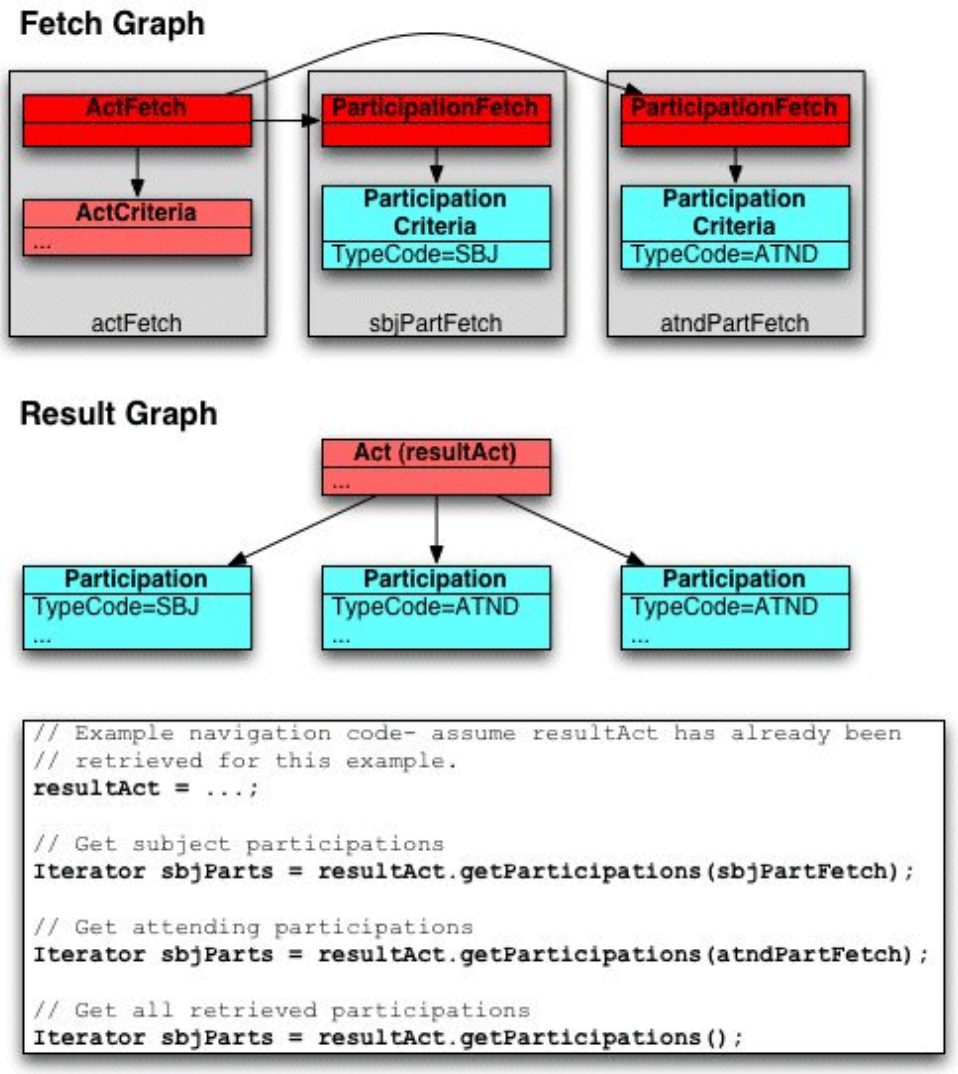


Figure 7–6 illustrate Fetch and Result Graphs.

HDR RIM Extensions

- Versioning and Query
- Versioning and Fetches
- Versioning and Criteria
- Retrieving the Current Version
- Retrieving a Specific Version
- Detail Criteria Versioning Behavior
- Detail Fetches
- Detail Criteria
- ControlAct Querying

- Person Merge Querying
- Owned Roles

Versioning and Query

The query framework provides several APIs on the core fetch and criteria classes that let you fetch particular versions of objects and specify criteria on the different object versions. These APIs are described below.

Versioning and Fetches

In the context of fetches, the versioning behavior determines whether *all versions* or the *referenced version* of a detail object are retrieved by a detail fetch. The version of a detail object to retrieve is specified on the detail fetch by using the fetch factory methods (on QueryComponentFactory) that accept a versioning parameter. The QueryComponentFactory defines two versioning constants to be used with the following factory methods:

- `QueryComponentFactory.VERSION_DEPENDENT`
- `QueryComponentFactory.VERSION_INDEPENDENT`

It is possible to use the same factory methods to construct the top level fetch. However, the versioning constant supplied is ignored as it only applies to detail fetches. To specify particular versions for the top level fetch you must set the version on the criteria for that fetch.

Note: By default, navigations are version dependent. Detail navigations that retrieve a single object, such as Participation to Role, return an undefined version of the detail object unless a specific version is requested (via a criteria with setCurrentVersion(...) or setVersionNum(...) methods).

Versioning and Criteria

The core attribute criteria classes provide APIs for specifying particular versions of objects as well as versions of detail objects. These APIs are described in the following sections.

Retrieving the Current Version

To restrict retrieval to the most current version of an object you can use the following AttributeCriteria method:

```
setCurrentVersion(boolean currentVersionFlag);
```

This method, combined with a version independent detail navigation, effectively retrieves the most current version of a related object. Alternatively, this method can be used to retrieve all but the current version of an object by passing false as the argument.

Retrieving a Specific Version

To retrieve a specific version of an object, use the following API:

```
setVersionNum(int versionNumber);
```

As with any other criteria specified, if the version specified does not exist no results are returned.

Detail Criteria Versioning Behavior

The criteria setters for neighboring core objects accept a versioning parameter that specifies whether the detail criteria is applied to the referenced version or any version of the detail object.

Detail Fetches

Assuming patient role version 1 and both versions of the player entity match the criteria, to retrieve patient and person information a detail player entity fetch is attached to the role fetch. When constructing the entity fetch you can specify either the VERSION_DEPENDENT or VERSION_INDEPENDENT constant for the versioning parameter (provided you use the EntityFetch newEntityFetch(EntityCriteria criteria, int versioningType) API on QueryComponentFactory). If you construct the entity fetch with the VERSION_DEPENDENT constant the person directly linked to the role is returned–in this case, version 1 of the person. If you construct the entity fetch with the VERSION_INDEPENDENT constant the current version (version 2) of the person is returned.

Detail Criteria

Assuming you have a role criteria that matches version 1 of the patient role, you can add a detail player entity criteria and specify either the VERSION_DEPENDENT orVERSION_INDEPENDENT constant for the versioning parameter. If you added a detail player entity criteria that matched person entities with name John Smith, the role would only be fetched if its playing entity has that name.

ControlAct Querying

You can retrieve or base queries on the ControlAct associated with an object using methods in the fetch and criteria interfaces. To fetch the creating/updating ControlAct, add a detail fetch using the following API:

```
addControlActFetch(ActFetch actFetch);
```

To specify criteria based on the creating or updating ControlAct, add a dependent criteria using the following API:

```
setControlActCriteria(ActCriteria actCriteria);
```

Both of these APIs work in the same way as other dependent fetch/criteria setter methods except that they exclude any versioning behavior; there is only one version of ControlActs associated with a particular version of an object.

Person Merge Querying

Person merge is the ability to collate data from a number of person entities (typically entered in different systems) into a single person that contains the superset of data. The single entity at the end of a potential chain of merges is typically called the ultimate survivor and the group of entities involved in a set of merge operations (including the ultimate survivor) is called the merge peers group.

The query framework exposes methods on the entity fetch and criteria classes that let you fetch and specify criteria for the merge group. To fetch members of the merge group, add a detail fetch using the following API:

```
addMergePeersEntityFetch(EntityFetch fetch);
```

To specify criteria for the merge group, add a detail criteria using the following API:

```
setMergePeersCriteria(EntityCriteria entityCriteria);
```

The EntityAttributeCriteria class also contains the following method that lets you specify whether or not the entity is the ultimate survivor of a merge:

```
setUltimateSurvivor(boolean ultimateSurvivor);
```

The ultimate surviving person of a set of potential merges can be found efficiently by combining the merge peers navigation with the setUltimateSurvivor(...) criteria specification.

Example 7–13 Retrieve Ultimate Survivor of an Entity

The following example shows how to retrieve the ultimate survivor of an entity:

```
// create the ultimate survivor fetch and criteria
   EntityAttributeCriteria ultimateSurvivorCriteria =
mQueryComponentFactory.newEntityAttributeCriteria();
   ultimateSurvivorCriteria.setUltimateSurvivor(true);
   EntityFetch survivorFetch =
mQueryComponentFactory.newEntityFetch(ultimateSurvivorCriteria,
    QueryComponentFactory.VERSION_INDEPENDENT);
   survivorFetch.retrieveId(true);

   // create the fetch and criteria for an entity that has been merged
   EntityAttributeCriteria entityACriteria =
mQueryComponentFactory.newEntityAttributeCriteria();
   entityACriteria.setCurrentVersion(true);
   entityACriteria.setId(SetSearchOperator.ANY, new II[]{sEntityAII});
   EntityFetch entityAFetch =
mQueryComponentFactory.newEntityFetch(entityACriteria);
   entityAFetch.retrieveId(true);

   // add the ultimate survivor fetch
   entityAFetch.addMergePeersEntityFetch(survivorFetch);
```

Owned Roles

Owned roles are a HDR extension to the RIM that lets your create roles that do not directly participate in an act (the role is linked to or owned by an entity). The query framework supports the retrieval of owned roles specifically through detail fetch and criteria methods in the entity fetch and criteria classes respectively. To fetch an owned role, add a detail fetch to an EntityFetch using the following API:

```
addOwned<Played/Scoped>RoleFetch(RoleFetch ownedplayedroleFetch);
```

To specify criteria for an owned role, add a detail criteria using the following API:

```
setOwned<Played/Scoped>RoleCriteria(RoleCriteria roleCriteria, int
versioningType);
```

> **Note:** The query framework does not distinguish between typical roles and owned roles when the standard fetch and criteria APIs are used. In a role fetch where both normal and owned roles match the criteria they are all returned by the query. To return only owned roles you must use the detail fetch and criteria APIs on the entity fetch and criteria interfaces illustrated above.

Example 7–14 Patient Query by ID Code Sample

Find a person associated with a known patient Id, retrieving basic demographic information:

```
/**
```

```
 * Finds roles with the following attributes:
 * - class code = PAT
 * - id = id of the particular patient
 *
 * and must meet the following detail criteria:
 *
 * 1. be played by a PSN entity
 *
 * The query should return the patient, the player entity.
 *
 */
package oracle.hsgbu.hdr.sample.scenarios;


import java.io.IOException;
import oracle.hsgbu.hdr.exception.HDRRimException;
import oracle.hsgbu.hdr.hl7.query.EntityAttributeCriteria;
import oracle.hsgbu.hdr.hl7.query.SearchOperator;
import oracle.hsgbu.hdr.hl7.query.EntityFetch;
import oracle.hsgbu.hdr.hl7.query.RoleAttributeCriteria;
import oracle.hsgbu.hdr.hl7.query.SetSearchOperator;
import oracle.hsgbu.hdr.hl7.query.RoleFetch;
import oracle.hsgbu.hdr.hl7.rim.domain.EntityClass;
import oracle.hsgbu.hdr.hl7.rim.domain.RoleClass;
import oracle.hsgbu.hdr.hl7.rim.factories.QueryComponentFactory;
import oracle.hsgbu.hdr.hl7.rim.factories.DataTypeFactory;
import oracle.hsgbu.hdr.hl7.rim.types.II;
import oracle.hsgbu.hdr.hl7.rim.Role;
import oracle.hsgbu.hdr.hl7.rim.Entity;


import java.util.Iterator;
import java.util.Properties;
import oracle.hsgbu.hdr.exception.CommonException;
import oracle.hsgbu.hdr.fwk.servicelocator.common.ServiceLocator;

public class PatientSearchById
{
  public void executeScenario() throws HDRRimException, CommonException,
IOException
  {
    ServiceLocator serviceLocator = getServiceLocator();
    QueryComponentFactory queryComponentFactory =
QueryComponentFactory.getInstance();
    DataTypeFactory dataTypeFactory = DataTypeFactory.getInstance();

    // create an EntityAttributeCriteria and specify that the class code should
equal PSN
    EntityAttributeCriteria eCriteria =
queryComponentFactory.newEntityAttributeCriteria();
    eCriteria.setClassCode(SearchOperator.EQUALS, EntityClass.PSN);

    // create a person Entity fetch with the PSN entity criteria and specify that
the name, administrative gender
    // code, birth time, code and addr attributes should be retrieved.
    EntityFetch entityFetch = queryComponentFactory.newEntityFetch(eCriteria,
    QueryComponentFactory.VERSION_DEPENDENT);
    entityFetch.retrieveId(true);
    entityFetch.retrieveName(true);
    entityFetch.retrieveAdministrativeGenderCode(true);
```

```
    entityFetch.retrieveBirthTime(true);
    entityFetch.retrieveCode(true);
    entityFetch.retrieveAddr(true);

     // create the patient II which will be used in the role criteria
    II[] patientIIs = new II[1];
    patientIIs[0] = dataTypeFactory.newII(dataTypeFactory.newUID("1.2.3.4"),
                                          dataTypeFactory.newST("PAT_ROLE_1_II_
EXT"),
                                          dataTypeFactory.newBL(true));

    // create an EntityAttributeCriteria and specify that the class code should
equal PAT and that the id
    // should be one of the id's in the II array
    RoleAttributeCriteria patRoleCriteria =
queryComponentFactory.newRoleAttributeCriteria();
    patRoleCriteria.setClassCode(SearchOperator.EQUALS, RoleClass.PAT);
    patRoleCriteria.setId(SetSearchOperator.ANY, patientIIs);

    // create the patient Role fetch with the PAT role criteria and specify that
the id should be retrieved
    RoleFetch patRoleFetch = queryComponentFactory.newRoleFetch(patRoleCriteria);
    patRoleFetch.retrieveId(true);
    patRoleFetch.retrieveClassCode(true);

    // link the person fetch to the patient fetch
    patRoleFetch.addPlayerEntityFetch(entityFetch);

    // submit the query
    Iterator patientRoles =
serviceLocator.getRimService().queryRoles(serviceLocator, patRoleFetch);

    System.err.println("PatientSearchById Query Results:");
    System.err.println("*********************************");
    for(;patientRoles.hasNext();)
    {
      Role patientRole = (Role)patientRoles.next();
      System.err.println("Role Id: " + patientRole.getId());
      System.err.println("Role Class: " + patientRole.getClassCode());

      Entity playerEntity = patientRole.getPlayerEntity(entityFetch);
      System.err.println("Entity Id: " + playerEntity.getId());
      System.err.println("Entity Version Number: " +
playerEntity.getVersionNum());
     System.err.println("Entity Class Code: " + playerEntity.getClassCode());
    }
  }

   private  ServiceLocator getServiceLocator() throws IOException, CommonException
{
        Properties props = new Properties();

props.load(this.getClass().getClassLoader().getResourceAsStream("jndi.properties")
);
        return ServiceLocator.getInstance(props);
    }
}
```

Example 7–15 Query for Identified Patient Roles and Player Entities

This code sample queries for new or active encounters admitted by a known staff member; retrieves the associated subject patient with basic demographics, as well as the location of the encounter:

```
/**
 * The roles must have the following attributes:
 * - class code = ENC
 * - mood code = EVN
 * - status code = NEW or ACTIVE
 * - code = ???
 * - current version = true
 *
 * and must meet the following criteria:
 *
 * 1. have an ADM participation to either a EMP or ASSIGNED role where the role id
is that of the staff practitioner
 * 2. have an SUBJ participation to a PAT role played by a PSN entity
 * 3. have a LOC participation to a SDLOC role
 *
 * The query should return the encounters along with the ADM, SUBJ and LOC
participations
 * along with their corresponding roles and player entities.
 *
 */

package oracle.hsgbu.hdr.sample.scenarios;

import java.io.IOException;
import oracle.hsgbu.hdr.exception.HDRRimException;
import oracle.hsgbu.hdr.hl7.rim.types.II;
import oracle.hsgbu.hdr.hl7.rim.types.CD;
import oracle.hsgbu.hdr.hl7.query.RoleAttributeCriteria;
import oracle.hsgbu.hdr.hl7.query.SetSearchOperator;
import oracle.hsgbu.hdr.hl7.query.SearchOperator;
import oracle.hsgbu.hdr.hl7.query.ParticipationAttributeCriteria;
import oracle.hsgbu.hdr.hl7.query.ActAttributeCriteria;
import oracle.hsgbu.hdr.hl7.query.ActFetch;
import oracle.hsgbu.hdr.hl7.query.RoleFetch;
import oracle.hsgbu.hdr.hl7.query.ParticipationFetch;
import oracle.hsgbu.hdr.hl7.query.EntityAttributeCriteria;
import oracle.hsgbu.hdr.hl7.query.EntityFetch;
import oracle.hsgbu.hdr.hl7.query.ActCriteria;
import oracle.hsgbu.hdr.hl7.rim.domain.RoleClass;
import oracle.hsgbu.hdr.hl7.rim.domain.ParticipationType;
import oracle.hsgbu.hdr.hl7.rim.domain.ActClass;
import oracle.hsgbu.hdr.hl7.rim.domain.ActMood;
import oracle.hsgbu.hdr.hl7.rim.domain.ActStatus;
import oracle.hsgbu.hdr.hl7.rim.domain.EntityClass;
import oracle.hsgbu.hdr.hl7.rim.factories.QueryComponentFactory;
import oracle.hsgbu.hdr.hl7.rim.factories.DataTypeFactory;
import oracle.hsgbu.hdr.hl7.rim.Act;
import oracle.hsgbu.hdr.hl7.rim.Participation;
import oracle.hsgbu.hdr.hl7.rim.Role;
import oracle.hsgbu.hdr.hl7.rim.Entity;

import java.util.Iterator;
import java.util.Properties;
import oracle.hsgbu.hdr.exception.CommonException;
import oracle.hsgbu.hdr.fwk.servicelocator.common.ServiceLocator;
```

```
import oracle.hsgbu.hdr.hl7.rim.domain.NullFlavor;

public class EncountersByStaffPractitioner {

    public void executeScenario() throws HDRRimException, CommonException,
IOException {
        ServiceLocator serviceLocator = getServiceLocator();
        QueryComponentFactory queryComponentFactory =
QueryComponentFactory.getInstance();
        DataTypeFactory dataTypeFactory = DataTypeFactory.getInstance();

        // create the staff II which will be used in the role criteria
        II[] practitionerIIs = new II[1];
        practitionerIIs[0] =
dataTypeFactory.newII(dataTypeFactory.newUID("1.2.3.4"),
                dataTypeFactory.newST("STAFF_ROLE_1_II_EXT"),
                dataTypeFactory.newBL(true));

        // practitioner role
        RoleAttributeCriteria rAssignedCriteria =
queryComponentFactory.newRoleAttributeCriteria();
        rAssignedCriteria.setClassCode(SearchOperator.EQUALS, RoleClass.ASSIGNED);
        rAssignedCriteria.setId(SetSearchOperator.ANY, practitionerIIs);
//Particular practitioner
        RoleAttributeCriteria rEmpCriteria =
queryComponentFactory.newRoleAttributeCriteria();
        rEmpCriteria.setClassCode(SearchOperator.EQUALS, RoleClass.EMP);
        rEmpCriteria.setId(SetSearchOperator.ANY, practitionerIIs); //Particular
practitioner
        RoleAttributeCriteria rCriteria = (RoleAttributeCriteria)
queryComponentFactory.or(rAssignedCriteria, rEmpCriteria);

        // ADM participation
        ParticipationAttributeCriteria pCriteria =
queryComponentFactory.newParticipationAttributeCriteria();
        pCriteria.setRoleCriteria(rCriteria, QueryComponentFactory.VERSION_
INDEPENDENT);
        pCriteria.setTypeCode(SearchOperator.EQUALS, ParticipationType.ADM);

        // act criteria
        ActAttributeCriteria aCriteria =
queryComponentFactory.newActAttributeCriteria();
        aCriteria.setCurrentVersion(true);
        aCriteria.setClassCode(SearchOperator.EQUALS, ActClass.ENC);
        aCriteria.setMoodCode(SearchOperator.EQUALS, ActMood.EVN);

        //    aCriteria.setCode(SearchOperator.EQUALS, actCd);
        aCriteria.setParticipationCriteria(pCriteria,
QueryComponentFactory.VERSION_INDEPENDENT);

        // create an act criteria to retrieve new and active acts
        ActAttributeCriteria activeActStatus =
queryComponentFactory.newActAttributeCriteria();
        activeActStatus.setStatusCode(SearchOperator.EQUALS, ActStatus.ACTIVE);
        activeActStatus.setClassCode(SearchOperator.EQUALS, ActClass.ENC);
        activeActStatus.setMoodCode(SearchOperator.EQUALS, ActMood.EVN);
        CD actCd = dataTypeFactory.newCD(dataTypeFactory.newST("AMB"),
                dataTypeFactory.nullUID(NullFlavor.NI),
dataTypeFactory.newST("ActCode"), dataTypeFactory.nullST(NullFlavor.NI),
                dataTypeFactory.nullED(NullFlavor.NI));
```

```
        activeActStatus.setCode(SearchOperator.EQUALS, actCd);
        activeActStatus.setCurrentVersion(true);
        activeActStatus.setParticipationCriteria(pCriteria,
QueryComponentFactory.VERSION_INDEPENDENT);
        ActAttributeCriteria newActStatus =
queryComponentFactory.newActAttributeCriteria();
        newActStatus.setStatusCode(SearchOperator.EQUALS, ActStatus.NEW);
        newActStatus.setClassCode(SearchOperator.EQUALS, ActClass.ENC);
        newActStatus.setMoodCode(SearchOperator.EQUALS, ActMood.EVN);
        newActStatus.setCurrentVersion(true);

        newActStatus.setCode(SearchOperator.EQUALS, actCd);
        newActStatus.setParticipationCriteria(pCriteria,
QueryComponentFactory.VERSION_INDEPENDENT);
        ActCriteria activeOrNewStatus = (ActCriteria)
queryComponentFactory.or(activeActStatus, newActStatus);

        ActFetch actFetchMain =
queryComponentFactory.newActFetch(activeOrNewStatus,
                QueryComponentFactory.VERSION_INDEPENDENT);
        actFetchMain.retrieveId(true);

        // Add location to a fetch
        RoleAttributeCriteria lrCriteria =
queryComponentFactory.newRoleAttributeCriteria();
        lrCriteria.setClassCode(SearchOperator.EQUALS, RoleClass.SDLOC);
        RoleFetch lrFetch = queryComponentFactory.newRoleFetch(lrCriteria,
                QueryComponentFactory.VERSION_INDEPENDENT);
        lrFetch.retrieveId(true);
        lrFetch.addPlayerEntityFetch(queryComponentFactory.newEntityFetch());
        ParticipationAttributeCriteria lpCriteria =
queryComponentFactory.newParticipationAttributeCriteria();
        lpCriteria.setTypeCode(SearchOperator.EQUALS, ParticipationType.LOC);
        ParticipationFetch lpFetch =
queryComponentFactory.newParticipationFetch(lpCriteria,
                QueryComponentFactory.VERSION_INDEPENDENT);
        lpFetch.addRoleFetch(lrFetch);

        actFetchMain.addParticipationFetch(lpFetch);

        // Add patient to a fetch
        EntityAttributeCriteria eCriteria =
queryComponentFactory.newEntityAttributeCriteria();
        eCriteria.setClassCode(SearchOperator.EQUALS, EntityClass.PSN);
        EntityFetch eFetch = queryComponentFactory.newEntityFetch(eCriteria);
        eFetch.retrieveName(true);
        eFetch.retrieveAdministrativeGenderCode(true);
        eFetch.retrieveBirthTime(true);
        eFetch.retrieveCode(true);
        RoleAttributeCriteria patRoleCriteria =
queryComponentFactory.newRoleAttributeCriteria();
        patRoleCriteria.setClassCode(SearchOperator.EQUALS, RoleClass.PAT);
        RoleFetch patRoleFetch =
queryComponentFactory.newRoleFetch(patRoleCriteria,
                QueryComponentFactory.VERSION_INDEPENDENT);
        patRoleFetch.retrieveId(true);
        patRoleFetch.addPlayerEntityFetch(eFetch);
        ParticipationAttributeCriteria patPartCriteria =
queryComponentFactory.newParticipationAttributeCriteria();
```

```
        patPartCriteria.setTypeCode(SearchOperator.EQUALS, ParticipationType.SBJ);
        ParticipationFetch patPartFetch =
queryComponentFactory.newParticipationFetch(patPartCriteria,
                QueryComponentFactory.VERSION_INDEPENDENT);
        patPartFetch.addRoleFetch(patRoleFetch);

        actFetchMain.addParticipationFetch(patPartFetch);

        Iterator fetchedActs =
ServiceLocator.getInstance().queryActs(serviceLocator, actFetchMain);

        System.err.println("EncountersByStaffPractitioner Query Results:");
        System.err.println("*****************************************");
        for (; fetchedActs.hasNext();) {
            Act fetchedAct = (Act) fetchedActs.next();
            System.err.println("Act class code: " + fetchedAct.getClassCode());
            System.err.println("Act mood code: " + fetchedAct.getMoodCode());
            System.err.println("Act status code: " + fetchedAct.getStatusCode());

            Iterator fetchedPatientParticipations =
fetchedAct.getParticipations(patPartFetch);
            for (; fetchedPatientParticipations.hasNext();) {
                Participation fetchedPatientParticipation = (Participation)
fetchedPatientParticipations.next();
                System.err.println("Participation type code: " +
fetchedPatientParticipation.getTypeCode());

                Role fetchedPatientRole = (Role)
fetchedPatientParticipation.getRole(patRoleFetch);
                System.err.println("Role class code: " +
fetchedPatientRole.getClassCode());

                Entity fetchedPlayerEntity =
fetchedPatientRole.getPlayerEntity(eFetch);
                System.err.println("Player entity class code: " +
fetchedPlayerEntity.getClassCode());
            }
        }
    }

    private ServiceLocator getServiceLocator() throws IOException, CommonException
{
        Properties props = new Properties();

props.load(this.getClass().getClassLoader().getResourceAsStream("jndi.properties")
);
        return ServiceLocator.getInstance(props);
    }
}
```

Original Coded Attributes

Coded attributes are commonly coerced into equivalent concepts during persistence into HDR. By default these coerced concepts are retrieved by Rim Query. You can retrieve the original data, using the oracle.hsgbu.hdr.hl7.util.CodedTypeUtility.getOriginalAttributes(CD) method.

Since qualifiers are not copied from original to coerced code, qualifiers can only be retrieved from CD datatypes that have been obtained via the getOriginalAttributes method.

See Also:

- oracle.hsgbu.hdr.hl7 package for further information about HDR HL7 interfaces.
- oracle.hsgbu.hdr.hl7.query package for further information about HDR Query interfaces.

DCTB Subqueries

The SQL queries generated by the RIM query API can be optimized in three different ways. By default, in all nested SELECT sub queries, the WHERE conditions will be generated as a SQL IN condition. This can be modified by configuring the HDR managed server start-up JVM argument CTB_SUBQRY_OPT_METHOD with one of the values NONE, EXISTS, or JOIN. Based on the database version and configuration, you can choose an option that results in the best database SQL execution plans for the HDR generated SQL queries.

-DCTB_SUBQRY_OPT_METHOD=NONE

This is the default behavior where all nested select sub queries in the where condition will be generated as the SQL IN condition.

For example, SELECT CtbCoreRolesEO.ROLE_ID, CtbCoreRolesEO.ROLE_VERSION_NUM FROM CTB_CORE_ROLES CtbCoreRolesEO WHERE CtbCoreRolesEO.CLASS_CODE = 'PAT' AND(CtbCoreRolesEO.PLAYER_ID, CtbCoreRolesEO.PLAYER_VERSION_NUM) IN (SELECT CtbCoreEntitiesEO.ENTITY_ID, CtbCoreEntitiesEO.ENTITY_VERSION_NUM FROM CTB_CORE_ENTITIES CtbCoreEntitiesEO WHERE CtbCoreEntitiesEO.CLASS_CODE = 'PSN' AND CtbCoreEntitiesEO.DETERMINER_CODE = 'INSTANCE' AND CtbCoreEntitiesEO.ENTITY_ID IN (SELECT CtbCoreEntyIIEO.ENTITY_ID FROM CTB_CORE_ENTY_II CtbCoreEntyIIEO WHERE CtbCoreEntyIIEO.ROOT_ID = '1.2.3' AND CtbCoreEntyIIEO.EXTENSION_TXT = 'Person1')) ;

-DCTB_SUBQRY_OPT_METHOD=EXISTS

By setting sub-query optimization method to EXISTS, all nested select sub queries in the where condition will be generated as the SQL EXISTS condition.

For example, SELECT CtbCoreRolesEO.ROLE_ID, CtbCoreRolesEO.ROLE_VERSION_NUM FROM CTB_CORE_ROLES CtbCoreRolesEO WHERE CtbCoreRolesEO.CLASS_CODE = 'PAT' AND EXISTS (SELECT 1 FROM CTB_CORE_ENTITIES CtbCoreEntitiesEO WHERE CtbCoreEntitiesEO.CLASS_CODE = 'PSN' AND CtbCoreEntitiesEO.DETERMINER_CODE = 'INSTANCE' AND CtbCoreEntitiesEO.ENTITY_ID = CtbCoreRolesEO.PLAYER_ID AND CtbCoreEntitiesEO.ENTITY_VERSION_NUM = CtbCoreRolesEO.PLAYER_VERSION_NUM AND EXISTS (SELECT 1 FROM CTB_CORE_ENTY_II CtbCoreEntyIIEO WHERE CtbCoreEntyIIEO.ROOT_ID = '1.2.3' AND CtbCoreEntyIIEO.EXTENSION_TXT = 'Person1' AND CtbCoreEntyIIEO.ENTITY_ID = CtbCoreEntitiesEO.ENTITY_ID)) ;

-DCTB_SUBQRY_OPT_METHOD=JOIN

By setting sub-query optimization method to JOIN, all nested select sub queries in the where condition will be converted to the SQL JOIN condition.

For example, SELECT CtbCoreRolesEO1.ROLE_ID, CtbCoreRolesEO1.ROLE_VERSION_NUM FROM CTB_CORE_ROLES CtbCoreRolesEO1, CTB_CORE_ENTITIES CtbCoreEntitiesEO1, CTB_CORE_ENTY_II CtbCoreEntyIIEO1 WHERE CtbCoreRolesEO1.CLASS_CODE = 'PAT' AND CtbCoreEntitiesEO1.CLASS_CODE = 'PSN' AND CtbCoreEntitiesEO1.DETERMINER_CODE = 'INSTANCE' AND CtbCoreEntitiesEO1.ENTITY_ID = CtbCoreRolesEO1.PLAYER_ID AND CtbCoreEntitiesEO1.ENTITY_VERSION_NUM = CtbCoreRolesEO1.PLAYER_VERSION_NUM AND CtbCoreEntyIIEO1.ROOT_ID = '1.2.3' AND CtbCoreEntyIIEO1.EXTENSION_TXT = 'Person1' AND CtbCoreEntyIIEO1.ENTITY_ID = CtbCoreEntitiesEO1.ENTITY_ID;

HDR RIM Services

Core functions:

- Use RIM Services
- Use Master Catalog API

Use RIM Services

- Use The RIM Service
- Reference Modifiers
- Exception Handling

Use The RIM Service

The RimService provides the primary mechanism to persist or query Entities, Roles, Acts and related objects through HDR. Both persist and query operations are supported through the RimService.submit(ControlAct) method.

The control act passed to submit is expected to contain one outbound ActRelationship with a typeCode of SUBJ with an instance of a subclass of Act as its target, which can similarly be related to other objects to form a graph of objects to be persisted. If the target Act of the Act Relationship is an instance of a QueryAct, a data query results; see Section 7.5 for more information about querying.

These code samples help you to:

- Create an Organization Using a Registry Event
- Create an Encounter Using a CREATE_OR_UPDATE Reference to a Patient Object
- Pass an Invalid Code to Generate an Exception
- Rim Validation Exception Output

Example 7–16 Create an Organization Using a Registry Event

The following example illustrates the registration of an Entity—in this case an Organization is created. To create any Entity in HDR you must provide a Role and an Act; typically an AssignedEntity Role is used, and a RegistryEvent is used for the Act.

```
// Create the ORG Entity to be registered
        SET_II orgId = dataTypeFactory.newSET_II("9.989898.5.3.1", "ORG002",
true);
        Organization organization = entityFactory.newOrganization(
            dataTypeFactory.nullCE(NullFlavor.NI),
            EntityDeterminer.INSTANCE, orgId);
```

```
        // Build up the organization name using
        SET<CS> orgNameUseCode= dataTypeFactory.newSET_
CS(dataTypeFactory.newCS("L"));
        ENXP orgNameUsePart = dataTypeFactory.newENXP("Pro Health Systems", null,
         dataTypeFactory.nullSET<CS>(NullFlavor.NI));
        ENXP[] orgNameUsePartArray = new ENXP[] {orgNameUsePart};
        ON orgName = dataTypeFactory.newON(orgNameUsePartArray, orgNameUseCode,
         dataTypeFactory.nullIVL<TS>(NullFlavor.NI));

        // Set the organization name, desc and status
        organization.setName(dataTypeFactory.newBAG<EN>(orgName));
        organization.setStatusCode(EntityStatus.ACTIVE);

        // build a TEL type and addto the telecom attribute.
        SET<CS> orgTelUseCode= dataTypeFactory.newSET_
CS(dataTypeFactory.newCS("H"));
        TEL orgTel = dataTypeFactory.newTEL("tel", "1-510-555-1234",
         dataTypeFactory.nullGTS(NullFlavor.NI),orgTelUseCode);
        organization.addTelecom(orgTel);

        // build an ADXP type and add the address
        ADXP adxp1 = dataTypeFactory.newADXP("100 Oracle Parkway",
dataTypeFactory.newCS("SAL"));
        ADXP adxp2 = dataTypeFactory.newADXP("Redwood Shores",
dataTypeFactory.newCS("CTY"));
        ADXP adxp3 = dataTypeFactory.newADXP("CA", dataTypeFactory.newCS("STA"));
        ADXP adxp4 = dataTypeFactory.newADXP("94065",
dataTypeFactory.newCS("ZIP"));
        ADXP adxp5 = dataTypeFactory.newADXP("US", dataTypeFactory.newCS("CNT"));

        ADXP[] addrPartArray = new ADXP[] {
            adxp1,
            adxp2,
            adxp3,
            adxp4,
            adxp5
        };

        AD addr = dataTypeFactory.newAD(addrPartArray,
            dataTypeFactory.newSET<CS>(dataTypeFactory.newCS("WP")),
                dataTypeFactory.nullGTS(NullFlavor.NI));

        organization.addAddr(addr);

        // Create an ASSIGNED Role for the organization,
        // with the organization as the player
        // CD roleCode = dataTypeFactory.newCD("000172","HDR Supplemental");
        SET_II assignedRoleId = dataTypeFactory.newSET_II("9.989898.5.49.1",
"ASSIGNED0002", true);
        Role role =

roleFactory.newAssignedEntity(dataTypeFactory.newCE("000174","2.16.840.1.113894.10
04.100.100.2.5"),
        organization, null, assignedRoleId);


        // Create the registry act. Registry acts are used to denote a
        // registration of a Role or Entity, in this case an ORG Entity.
        // The registry act will later be associated with the control act
```

```
        // using an act relationship
        SET_II regActId = dataTypeFactory.newSET_II("9.989898.5.42.1", "REG0012",
true);
        Act regAct  = actFactory.newRegistryAct(ActMood.EVN,
dataTypeFactory.nullCD
         (NullFlavor.NI), regActId);

        // Create a "SBJ" Participation between the registry act
        // and the identified role
        regAct.addParticipation(ParticipationType.SBJ, role);

        // Create the control act, providing an Id and a null trigger event
        SET_II ctrlActId = dataTypeFactory.newSET_II("9.989898.5.28.1",
"CACT0012", true);
        ControlAct controlAct =
actFactory.newControlActEvent(dataTypeFactory.nullCD
         (NullFlavor.NI), ctrlActId);

        // Create an outbound Act Relationship between the control act
        // and registry act
        controlAct.addOBActRelationship(ActRelationshipType.SUBJ, regAct);

        // Submit the control act. The returned control act will be
        // null unless a query act was specified in the act relationship
        // on the control act
        ControlAct returnedControlAct = rimService.submit(controlAct);
```

In addition to illustrating the basic structure of a submission, Example 7-16 highlights the following:

- Construction of datatypes: AD, ADXP, CD, CS, ENXP, EN, II, ON, TEL.
- Construction of sets of datatypes: SET_CS, SET_II.
- Construction of NullFlavor objects as the appropriate datatype class.
- Elements of Organizations that are necessary to pass TCA validation. For example: Organization Name use code of H; Telecom Scheme of TEL.

Reference Modifiers

In the prior example, all objects are newly created. To update an object, fetch it using the query mechanism and use the createNewVersion method to create a new version.

HDR also provides a mechanism to create a reference to an object that already exists and to permit an object to be created or updated without first querying the object. This is achieved by making a reference to an object using the ReferenceModifier class. This approach may also improve an application's performance because it eliminates the need to fetch the affected objects to the client tier prior to updating.

The simplest reference that can be created is the MustExist reference. A MustExist reference mandates that an object with a specified Instance Identifier has already been created. It is useful when creating ActRelationships or Participations to objects that have already been persisted.

Two other reference types worth discussing in more detail are the CreateOrOverlay and the CreateOrUpdate types. These reference types either create the object with the supplied attributes if it does not already exist, or they cause the creation of a new version of the object with attributes set in a manner consistent with the reference modifier used. For more information about the difference between the CreateOrOverlay and CreateOrUpdate methods, and for information about other reference types see the ReferenceModifier class description.

Example 7–17 Create an Encounter Using a CREATE_OR_UPDATE Reference to a Patient Object

```
// Create the Patient II
        SET_II patientId = dataTypeFactory.newSET_II("9.989898.5.2","PAT1001",
true);

        // Use a Create Or Update reference for the Patient
        Patient patientRole = (Patient)roleFactory.makeReference(
            ReferenceModifier.CREATE_OR_UPDATE, RoleClass.PAT,
            dataTypeFactory.nullCE(NullFlavor.NP),
            null, null, patientId, 0);

        // Create the Patient Encounter Event (Class Code:ENC)
        // The Encounter will later be associated with the control act
        // using an act relationship
        SET_II encActId = dataTypeFactory.newSET_II("9.989898.5.6.100","ENC1001",
true);

        // Use "ActCode" for the codingscheme; UID is 2.16.840.1.113883.5.4
        Act encAct  = actFactory.newPatientEncounter(ActMood.EVN,
            dataTypeFactory.newCD("IMP","2.16.840.1.113883.5.4"), encActId);

        // Create a "SBJ" Participation between the registry act
        // and the identified role
        encAct.addParticipation(ParticipationType.SBJ, patientRole);

        // Create the control act, providing an Id and a null trigger event
        SET_II ctrlActId = dataTypeFactory.newSET_II
            ("9.989898.5.28", "CACT001032", true);
        ControlAct controlAct = actFactory.newControlActEvent(
            dataTypeFactory.nullCD(NullFlavor.NP), ctrlActId);

        // Create an outbound Act Relationship between the control act
        // and registry act
        controlAct.addOBActRelationship(ActRelationshipType.SUBJ, encAct);

        // Submit the control act. The returned control act will be
        // null unless a query act was specified in the act relationship
        // on the control act
        ControlAct returnedControlAct = rimService.submit(controlAct);
```

Exception Handling

Invalid objects submitted in a graph cause HDR to bundle validation exceptions. These bundles consist of one exception per validation error; there may be multiple exceptions relating to an object. The exceptions contain methods to return the Ids of the related invalid object.

Example 7–18 Pass an Invalid Code to Generate an Exception

```
// Create the Patient Encounter Event (Class Code:ENC)
        // The Encounter will later be associated with the control act
        // using an act relationship
        SET_II encActId = dataTypeFactory.newSET_II("9.989898.5.6.100", "ENC1001",
true);

        // UNKNOWN_CODE is expected to cause validation to fail
        Act encAct  = actFactory.newPatientEncounter(ActMood.EVN,
        dataTypeFactory.newCD("UNKNOWN_CODE","2.16.840.1.113883.5.4"), encActId);
```

```
        // Create the control act, providing an Id and a null trigger event
        SET_II ctrlActId = dataTypeFactory.newSET_II("9.989898.5.28",
"CACT001032", true);
        ControlAct controlAct =
actFactory.newControlActEvent(dataTypeFactory.nullCD
         (NullFlavor.NP), ctrlActId);

        // Create an outbound Act Relationship between the control act
        // and registry act
        controlAct.addOBActRelationship(ActRelationshipType.SUBJ, encAct);
        try
        {
            // Submit the control act. The returned control act will be
            // null unless a query act was specified in the act relationship
            // on the control act
            ControlAct returnedControlAct = rimService.submit(controlAct);
        }
        catch (HDRRimException e)
        {
            CommonException[] exceptions = e.getBundledExceptions();
            for (int i=0; i<exceptions.length; i++)
            {
                if(exceptions[i] instanceof RimObjectException)
                {
                    System.out.println(
                        ((RimObjectException)exceptions[i]).getIds());
                }
            }
            e.printStackTrace();
        }
```

This code generates the following output:

```
{9.989898.5-34789202; 9.989898.5.6.100-ENC1001}
```

...where 9.989898.5-34789202 is the internal Instance Indicator created on all objects by HDR and 9.989898.5.6.100-ENC1001 is the *II* submitted with the Encounter object. Additionally, the full exception stack trace is< printed, and the exception details attributes that fail validation.

Example 7–19 Rim Validation Exception Output

```
oracle.hsgbu.hdr.exception.HDRRimException
CODE = CMN_001: Multiple exception occured while processing. Please see the
individual exceptions for details.
MESSAGE = HDR_CMN_001: Multiple exception occured while processing. Please see the
individual exceptions for details.
      [1]
oracle.hsgbu.hdr.base.persist.exception.CorePersistenceValidationException
            CODE = HDR_CORE_UNKNOWN_CODE
            MESSAGE = HDR_CPS_PVE006: Unknown Code for attributeName:CODE,
code:UNKNOWN_CODE, codeOid:2.16.840.1.113883.5.4, codeSystemName:null,
codeVersionName:2.01.4 and token:EncounterEventResource

...
Child Exceptions:

oracle.hsgbu.hdr.exception.HDRRimException
CODE = HDR_CORE_UNKNOWN_CODE
MESSAGE = HDR_CPS_PVE006: Unknown Code for attributeName:CODE, code:UNKNOWN_CODE,
codeOid:2.16.840.1.113883.5.4, codeSystemName:null, codeVersionName:2.01.4 and
```

```
token:EncounterEventResource

Original Stacktrace:
HDR_CPS_PVE006: Unknown Code for attributeName:CODE, code:UNKNOWN_CODE,
codeOid:2.16.840.1.113883.5.4, codeSystemName:null, codeVersionName:2.01.4 and
token:EncounterEventResource
...
...
```

Use Master Catalog API

Master Catalog Entries

During persistence all Acts, Entities, and Role objects are validated against the Master Catalog. Master Catalog defines the ETS values for class code, mood code, determiner code, code, player entity, and/or scoper entity that are allowed for these objects. HDR supports three types of ETS values: (1) explicit values specifying the ID of the concept, (2) null values when no values are required, or (3) any value when all ETS concepts are valid.

To ensure that Master Catalog entries exist for all objects being persisted prior to submission, HDR provides APIs to query and create Master Catalog data.

HDR provides MasterCatalogService APIs to create, update, and query Master Catalog entries.

> **See Also:** ■*Oracle Healthcare Data Repository Implementation Guide* (Implementing Master Catalog) for more information about Master Catalog API.

Outbound Message Processor (OMP) uses the Masters Catalog identifiers to create the Act Concept Configuration; see section 10.2.1 for more details.

Concepts

Master Catalog entry defines constraints based on:

- Act: Class Code, Mood Code and Code.
- Entity: Class Code, Determiner Code and Code.
- Role: Class Code, Code and the Player and Scoper Master Catalog IDs.

All Master Catalog entries define a code type—ID, ANY, or NULL.

- ID: Code attribute of the entity should be a valid ETS concept code.
- ANY: Code attribute of the entity should be null.
- NULL: Code attribute of the entity should be null.

For all Master Catalog getXxx() search operations, if the codeType is ID, then the getXxx() operation goes by the following order of precedence:

- Looks for the exact concept code.
- If the exact code not found, it looks for equivalent concept code.

- If equivalent concept code not found, it looks for code type ANY.

The Master Catalog supports concept equivalence. It considers a concept valid if it is equivalent to a concept used in an entry. You need not update the Master Catalog when new a coding scheme or version is loaded in ETS, provided equivalence information is also loaded with the new coding scheme or version. Concept equivalence is also used to verify whether user is trying to set up a duplicate Master Catalog entry. If one Master Catalog entry exists for a concept, no other entry with its equivalent concepts is permitted.

See Also: ■*Oracle Healthcare Data Repository API Documenation* for more information about Master Catalog interfaces.

- *Oracle Healthcare Data Repository Implementation Guide* for more information about Master Catalog validations.

Example 7–20 Check for the Existence of a Master Catalog Entry Matching the Control Act

```
// Find the Master Catalog Entry with parameters:
        // class code = CACT (From the ActClass Domain)
        // mood code = EVN (From the ActMood Domain)
        // code = null
        // code type = "NULL"
        MasterCatalog mc =
masterCatalogService.getMasterCatalogActEntry(ActClass.CACT, ActMood.EVN,
         null, "NULL");
        if (mc.getMasterCatalogId() != null)
        {
            System.out.println("Master Catalog Record found. ID: " +
mc.getMasterCatalogId());
        }
        else
        {
              System.out.println("No Master Catalog Record found");
        }
```

Example 7–21 Check for the Existence of a Master Catalog Entry Matching the Person

```
// Find the Master Catalog Entry with parameters:
        // class code = PSN (From EntityClass Domain)
        // determiner code = INSTANCE (From EntityDeterminer Domain)
        // code = null
        // code type = "NULL"
        MasterCatalog mc = masterCatalogService.getMasterCatalogEntityEntry
         (EntityClass.PSN, EntityDeterminer.INSTANCE, null, "NULL");
        if (mc.getMasterCatalogId() != null)
              {
            System.out.println("Master Catalog Record found. ID: " +
mc.getMasterCatalogId());
              }
        else
             {
             System.out.println("No Master Catalog Record found");
             }
```

```
      If suitable entries do not exist, use the persistMasterCatalog API to
```

create one.

Example 7–22 Create a Master Catalog Entry for a Diagnostic Image

```
// Create a Master Catalog Entry with parameters:
      // class code = DGIMG (From the ActClass Domain)
      // mood code = EVN (From the ActMood Domain)
      // code = null
      // code type = "NULL"
      MasterCatalog mc = masterCatalogFactory.newMasterCatalog();
      mc.setClassCode(ActClass.DGIMG);
      mc.setCodeType(MasterCatalog.NULL_CODE_TYPE);
      mc.setMoodCode(ActMood.EVN);
      mc.setEntryTypeCode(MasterCatalog.ACT_ENTRY_TYPE_CODE);
      mc.setActiveFlag(true);

      masterCatalogService.persistMasterCatalogEntry(mc);
```

Example 7–23 Create an Array of Master Catalog Entries

You can create or update an array of MasterCatalog entries using MC bulk persists API. For each entry in array, the API checks, if it is a valid entry, as follows and persists it. In the case of one or more invalid entries, the API rejects all of them.

```
MasterCatalog mc1 = masterCatalogFactory.newMasterCatalog();
 MasterCatalog mc2 = masterCatalogFactory.newMasterCatalog();

// For each master catalog entry defined above, to set the values follow the steps
described in Example 7-32.

//Create an array of mastercatalog with the above mentioned entries
MasterCatalog [] mcArray = new MasterCatalog [] {mc1,mc2};

//Call bulk persist method
 masterCatalogService.persistMasterCatalogEntries(mcArray);
```

Example 7–24 Search Master Catalog Entries Using Query Criteria

You can retrieve MasterCatalog entries based on MasterCatalogQueryCriteria using this API, which takes the MasterCatalogQueryCriteria as the parameter. MasterCatalogQueryCriteria defines methods that can be used to build a query for retrieving MasterCatalog records based on the criteria.

```
// Defining the MasterCatalogQueryCriteria to find Master Catalog Entry, for code
type = "ID" and coding scheme
// name = "HDR Supplemental", coding scheme version name = "HDR Supplemental
(2005-04-08)" and concept code = "001827"

// get an instance of the Master Catalog Criteria

MasterCatalogQueryCriteria masterCatalogSearchCriteria =
masterCatalogFactory().newMasterCatalogQueryCriteria();
SearchTerm searchTerm = masterCatalogSearchCriteria.equalsCode("HDR Supplemental",
"HDR Supplemental (2005-04-08)", "001827");

// set the code type to ID
searchTerm = masterCatalogSearchCriteria.and(searchTerm,
masterCatalogSearchCriteria.equalsCodeType(MasterCatalog.ID_CODE_TYPE));

// set the root search term
masterCatalogSearchCriteria.setRootSearchTerm(searchTerm);
```

```
// retrieve master catalog entry
MasterCatalog [] fetchedMasterCatalog =
masterCatalogService.findMasterCatalogEntries(masterCatalogSearchCriteria);
```

Example 7–25 Remove Master Catalog Entries from Cache

The Master Catalog API uses the CTBCacheService for caching the Master Catalog and Master Catalog Focal Class State Transition entries. The following table lists the cache names used to store the object:

Cache Names

Cache Name	Description	Key
CtbCoreMcActEntriesCache	Cache name for Master Catalog Act entries	ClassCode
CtbCoreMcEntityEntriesCache	Cache name for Master Catalog Entity entries	ClassCode
CtbCoreMcRoleEntriesCache	Cache name for Master Catalog Role entries	ClassCode
CtbCoreMcStTrnstnsCache	Cache name for Master Catalog Focal Class State Transition entries	ControlActMasterCatalogId

Example 7–26 Remove Master Catalog Entries from Cache

There are 2 APIs, which you can use to remove Master Catalog entries from the cache.

```
//This will remove all the entries from the Act Master Catalog cache, here we only
supply the cache name as parameter.

cacheService.invalidate("CtbCoreMcActEntriesCache");

//There is also an option to selectively delete entries from the cache using the
key.
//Here the class code is being used as a key for the Act Master Catalog entries
cache.
//This will remove all the act master catalog entries having that class code from
the cache.

cacheService.invalidate("CtbCoreMcActEntriesCache", "OBS");
```

Focal Class State Transitions

Focal Class State Transitions define the valid state transitions for a given combination of the Control Act Master Catalog ID and the Focal Class Master Catalog ID. These transitions must be a subset of the valid transitions for the RIM object type (Act, Entity, and Role). These transitions should be valid according to the generic state transition rules. The following are the possible state transition scenarios during creation or update of Act, Entity, and Role objects.

- null -> null (Always allowed and is never validated)
- not null -> not null (Allowed only after validation)
- null -> not null (Allowed only after validation)
- not null -> null (Not Allowed)

The examples below illustrate how to query for the existence of a Focal Class State Transition, and how to create a required Focal Class State Transition.

See Also: ■*Oracle Healthcare Data Repository API Documenation* for focal class description.

■ *Oracle Healthcare Data Repository Implementation Guide* for more information about Focal Class State Transitions.

Example 7–27 Check for Existence of a Focal Class State Transition (for Control Act and Focal Class)

```
String controlActMasterCatalogId = "";
        String focalClassMasterCatalogId = "";

        // Find the Master Catalog Entry for the Control Act
        MasterCatalog mc = masterCatalogService.getMasterCatalogActEntry(
            ActClass.CACT, ActMood.EVN, null, "NULL");
        controlActMasterCatalogId = mc.getMasterCatalogId();

        // Find the Master Catalog Entry for the Focal Class
        mc = masterCatalogService.getMasterCatalogEntityEntry(
            EntityClass.PSN, EntityDeterminer.INSTANCE, null, "NULL");
        focalClassMasterCatalogId = mc.getMasterCatalogId();

        // Retrieve the Focal Class State Transition for the given
        // Control Act, Focal Class where
        // Focal Class start state = "null" and
        // Focal Class end state = "active"
              MasterCatalogFocalClassStateTransition mcfcst =
            masterCatalogService.getFocalClassStateTransitionEntry(
            controlActMasterCatalogId, focalClassMasterCatalogId,
            "null","active");
        if (mcfcst != null)
        {
            System.out.println("Valid Focal Class State Transition found.");
        }
              else
              {
            System.out.println("Focal Class State Transition not found.");
              }
```

Example 7–28 Create a Focal Class State Transition

```
// Create a Focal Class State Transition for the
        // Master Catalog entry for DGIMG, EVN, NULL

        // Find the Control Act Master Catalog Entry
              MasterCatalog mcCact =
masterCatalogService.getMasterCatalogActEntry(
            ActClass.CACT,
              ActMood.EVN, null,
            "NULL");
        // Find the DGIMG Act Master Catalog Entry
              MasterCatalog mcDgimg =
masterCatalogService.getMasterCatalogActEntry(
            ActClass.DGIMG,
              ActMood.EVN, null,
            "NULL");

        MasterCatalogFocalClassStateTransition mcfcst =
            masterCatalogFactory.newMasterCatalogFocalClassStateTransition();
```

```
mcfcst.setControlActMasterCatalogId(mcCact.getMasterCatalogId());
mcfcst.setFocalClassMasterCatalogId(mcDgimg.getMasterCatalogId());
mcfcst.setActiveFlag(true);
mcfcst.setStartState("any");
mcfcst.setEndState("any");

masterCatalogService.persistFocalClassStateTransitionEntry(mcfcst);

// Now check it was persisted
mcfcst =
    masterCatalogService.getFocalClassStateTransitionEntry(
        mcCact.getMasterCatalogId(),
        mcDgimg.getMasterCatalogId(),
        "any", "any");
if (mcfcst == null) throw new HDRConfigException(
      "Master Catalog Focal Class State Transition not found");
```

Example 7–29 Create an Array of Focal Class State Transition Entries

You can create or update an array of MasterCatalogFocalClassStateTransition entries using MC bulk persists API. For each entry in the array, the API checks if it is a valid entry as follows and persist it. If the API finds one or more invalid entries, it rejects all of them.

```
MasterCatalogFocalClassStateTransition mcfcst1 =
masterCatalogFactory.newMasterCatalogFocalClassStateTransition();
MasterCatalogFocalClassStateTransition mcfcst2 =
masterCatalogFactory.newMasterCatalogFocalClassStateTransition(); // For
each mastercatalogfocalstatetransition entry defined above, to set the
values follow the steps

// described in Example 7-37. //Create an array of
mastercatalogfocalstatetransition with the above mentioned entries
MasterCatalogFocalClassStateTransition [] mcfcstArray =
MasterCatalogFocalClassStateTransition[] { mcfcst1, mcfcst1 }; //Call bulk
persist method
masterCatalogService.persistFocalClassStateTransitionEntries(mcfcstArray);
```

HDR HL7 Data Types

This package contains an implementation of a subset of the HL7 Data Types specification, to meet HDR RIM interface requirements. The attributes (id, class, status,...) of the HL7 version 3 RIM objects are defined in terms of these Data Types, as are the parameters and return types of the RIM API. In accordance with the specification, these data type objects are immutable—their value cannot be changed once created.

You can do the following:

- Use the DataTypeFactory
- HL7 Null Flavors
- Unsupported Operations
- Coded Types
- Collections (SET, BAG, LIST, IVL)
- HL7 Timing Specification (GTS, PIVL, EIVL, IVL<TS>, TS)

See Also: The UML class diagram at the HL7 website for information about available Data Types; refer to the following tips to view the diagram:

- *Select View > Full Screen [F11]* to expand to a full screen.
- Choose the Expand button to view the diagram; use the Windows scroll bars to navigate.
- Use the Windows back-arrow to return.

Use the DataTypeFactory

New Data Type objects should always be instantiated using the DataTypeFactory methods exclusively. Most of the factory methods throw CommonException or ETSException if there is a problem creating the new instance.

Creating Constants

As the factory methods throw checked exceptions, we do not recommend defining static final constants for Data Type values. If you find it useful to define constants, it is preferable to declare them as final member variables that can be initialized in the constructor (where exceptions can be handled more easily).

The CNE Domain Constants (defined in the package oracle.hsgbu.hdr.hl7.domain) define values for the CS type RIM coded attributes.

Abstract Types (ANY, BIN, QTY)

There are no factory methods for the ANY Data Type, because it is an abstract type. No value can be just an ANY without belonging to one of the concrete Data Types. For RIM attributes of type ANY (such as Observation.value, call the factory method for the appropriate concrete type (ST, ED, CD, GTS, etc.) and pass that type as the parameter to the RIM API.

Similarly, there is no factory method available for the Data Type BIN. It is a protected type that is only declared within the HL7 Data Type Specification. For properties of type BIN, you can substitute either ED or ST as the parameter to the factory method.

QTY is another example of an abstract type with no factory method. The specific subtype of Quantity (PQ, INT, REAL, and so on.) should be substituted instead.

HL7 Null Flavors

The Data Types Specification defines a hierarchy of different kinds of exceptional values, or null values. You can use the isNull method to determine whether a Data Type is one of these exceptional values; the nullFlavor method returns a CS code that specifies what kind of exceptional value it is.

In general, a Java null should never be passed to any of the parameters of the Data Types methods. You should use the DataTypeFactory to create an object with the appropriate HL7 null flavor instead. NI is the default null flavor, and should be used if you are uncertain about whether any of the more specific flavors of null are applicable.

The null flavor factory methods are guaranteed to return a valid Data Type with a null flavor, and *never* throw an exception. The CS codes for the null flavor values themselves are provided as convenience constants in the HL7.domain package.

Unsupported Operations

The present implementation of the HL7 Data Types Specification is only partially complete. Several methods throw a runtime UnsupportedOperationException, such as the arithmetic and comparison operations on Quantities, and many of the properties of the Timing Specification (GTS, PIVL, ...).

Coded Types

The HL7 Coded Types (CD, CE, CV and so on.) are stored in both HDR and ETS. Accordingly, several types of factory methods are provided for each coded type:

- Methods that accept any of the HL7 properties (Examples: newCE(ST code, UID codeSystem, ST codeSystemName, ST codeSystemVersion, ED originalText)) (Example 7-41).
- Methods that simply accept an ETS ID (Examples: newCE(String etsID)). The DataTypeFactory uses the ServiceLocator to retrieve the corresponding properties from ETS (Example 7-42); *note that use of this method should be avoided because of the additional communication overhead with the server.*
- Convenience methods that accept only the mandatory attributes, and default all of the others to null flavors (Examples: newCE(String code, String codeSystem). Note that the second parameter of this method is the code system OID, and not the code system name (Example 7-43).

The complete mapping of code system names to OIDs can be retrieved in the following ways:

- By viewing corresponding coding schemes in ETS UI *(preferred)*.
- For code systems registered with HL7, an online OID repository is provided as a free service.

These code samples help you do the following:

- Create Coded Data Type through Full Set of Properties (Recommended)
- Create Coded Data Type through ETS ID (Not Recommended)
- Create Coded Data Type through Code and Code System OID
- Create Coded Data Type through Code and Code System Name

Example 7–30 Create Coded Data Type through Full Set of Properties (Recommended)

```
DataTypeFactory dtf = DataTypeFactory.getInstance();
 CD code = df.newCD(df.newST("233604007"), df.newUID("2.16.840.1.113883.6.96"),
    df.newST("SNOMED-CT"), df.newST("v1"), df.newST("Pneumonia"));
```

Example 7–31 Create Coded Data Type through ETS ID (Not Recommended)

```
DataTypeFactory dtf = DataTypeFactory.getInstance();
 CD code = df.newCD("CON-2217");
```

Example 7–32 Create Coded Data Type through Code and Code System OID

```
DataTypeFactory dtf = DataTypeFactory.getInstance();
 CD code = df.newCD("233604007", "2.16.840.1.113883.6.96");
```

Example 7–33 Create Coded Data Type through Code and Code System Name

```
DataTypeFactory dtf = DataTypeFactory.getInstance();
 CD code = df.newCD(df.newST("233604007"), df.nullUID(NullFlavor.NI),
```

```
df.newST("SNOMED-CT"), df.newST("v1"), df.newST("Pneumonia"));
```

Collections (SET, BAG, LIST, IVL)

The factory methods for aggregate Data Types accept an array of elements (such as newSET_II(II[] identifiers)). The promotion operation is also provided, from a single Data Type into a trivial collection containing only that one element: newSET_II(II element).

HL7 Timing Specification (GTS, PIVL, EIVL, IVL<TS>, TS)

The factory methods for the time-related Data Types accept a literal string formatted according to the syntax defined in the HL7 specification. This literal is parsed to extract the properties (boundaries of an Interval, elements of a GTS) of the Data Type. If there is a syntax error in the string, a CommonException is thrown by the factory method.

RIM Service Examples

- Use CD Qualifiers
- Query Based on Observation Value Attribute

The code samples below help you do the following:

- Use CD Qualifiers
- Query PQ Values
- Query CE/CD Values
- Query BL Values
- Query INT values
- Query TS valuesl

Use CD Qualifiers

Example 7–34 Use CD Qualifiers

```
// Creates an Observation Event showing the use of qualifiers
        // on coded data to provide additional information about the code
        // Uses the SNOMED-CT coding scheme, which must be loaded to pass
validation

        // Use Act Code of DISDX/ActCode.
        // DISDX represents a "Discharge Diagnosis"
        // ActCode OID is 2.16.840.1.113883.5.4
        CD actCode = dataTypeFactory.newCD("DISDX", "2.16.840.1.113883.5.4");
        Observation obs = actFactory.newObservation(ActMood.EVN, actCode,
            dataTypeFactory.nullSET_II(NullFlavor.NA));

        obs.setText(dataTypeFactory.newST("Acute sudden-onset severe appendicitis,
first episode."));


        // Construct a code for observation value with qualifiers
        // Code is Appendicitis/SNOMED-CT
        // Qualifiers represent the following name - value information -
```

```
        // Severity (246112005) - Severe (24484000)
        // Clinical Course (263502005) - Acute Onset (373933003)
        // Episodicity (246456000) - First Episode (255217005)
        // All qualifiers use the SNOMED-CT coding scheme

        // It is expected that qualifiers will normally be taken from the same
        // coding scheme as the code. However, HDR does not validate this.

        CR[] qualifiers =
        {
            dataTypeFactory.newCR(
                dataTypeFactory.newCV("246112005", "2.16.840.1.113883.6.96"),
                dataTypeFactory.newCD("24484000", "2.16.840.1.113883.6.96"),
                dataTypeFactory.newBN(false)),
            dataTypeFactory.newCR(
                dataTypeFactory.newCV("263502005", "2.16.840.1.113883.6.96"),
                dataTypeFactory.newCD("373933003", "2.16.840.1.113883.6.96"),
                dataTypeFactory.newBN(false)),
            dataTypeFactory.newCR(
                dataTypeFactory.newCV("246456000", "2.16.840.1.113883.6.96"),
                dataTypeFactory.newCD("255217005", "2.16.840.1.113883.6.96"),
                dataTypeFactory.newBN(false))
        };


        // use Appendicitis (74400008) from SNOMED-CT coding scheme
        CD code = dataTypeFactory.newCD(
            dataTypeFactory.newST("74400008"),
            dataTypeFactory.newUID("2.16.840.1.113883.6.96"),
            dataTypeFactory.nullST(NullFlavor.NA),
            dataTypeFactory.nullST(NullFlavor.NA),
            dataTypeFactory.nullED(NullFlavor.NA),
            dataTypeFactory.newLIST<CR>(qualifiers));
        obs.setValue(code);

        // Add participations to a Medical Practitioner and to a Patient
        // Participation to the Patient is typically SBJ
        // Participation to the Provider is typically AUT

        // Omitting the Person (player) and Organization (scoper)
        // for the purposes of the example
        Patient pat =
roleFactory.newPatient(dataTypeFactory.nullCE(NullFlavor.NA),
            null, null, dataTypeFactory.nullSET_II(NullFlavor.NA));
        LicensedEntity prov = roleFactory.newHealthcareProvider(
            dataTypeFactory.nullCE(NullFlavor.NA),
            null, null, dataTypeFactory.nullSET_II(NullFlavor.NA));

        obs.addParticipation(ParticipationType.SBJ,pat);
        obs.addParticipation(ParticipationType.AUT,prov);

        // Attach the Observation to a Control Act and submit
        ControlAct cact = actFactory.newControlActEvent(
            dataTypeFactory.nullCD(NullFlavor.NA),
            dataTypeFactory.nullSET_II(NullFlavor.NA));
        cact.addOBActRelationship(ActRelationshipType.SUBJ,obs);

        rimService.submit(cact);
```

Query Based on Observation Value Attribute

The data type specification defines a hierarchy of various exceptional values or null values. You can use the isNull method to determine whether a data type is an exceptional value. The nullFlavor method returns a CS code that specifies the type of exceptional value.

Example 7–35 Query PQ Values

Query for all patients with a "Post-Prandial Blood Glucose Measurement" greater than 180 mg/dL (OBS.EVN.302788006//SNOMED-CT with Observation.value (PQ) = greater than 180 mg/dL).

```
// Show how to perform a query on Observation Value.
        // It is normal to specifiy the Class Code, Mood code, code and value
        // attributes for such queries

        // Build a Role Fetch to bring back the Patient
        RoleFetch rf = queryComponentFactory.newRoleFetch();

        // Decide which attributes to retrieve on the Patient Role
        rf.retrieveId(true);

        // Build a Participation Fetch,with a ParticipationCriteria
        // specifying that the participation type is SBJ
        ParticipationAttributeCriteria pc =
queryComponentFactory.newParticipationAttributeCriteria();
        pc.setTypeCode(ParticipationCriteria.EQUALS,ParticipationType.SBJ);
        ParticipationFetch pf =
queryComponentFactory.newParticipationFetch(pc,queryComponentFactory.VERSION_
DEPENDENT);
        pf.addRoleFetch(rf);

        // Put together act critera, specifying class code, mood code, code
        // and value attributes
        ActAttributeCriteria ac = queryComponentFactory.newActAttributeCriteria();
        ac.setClassCode(ActCriteria.EQUALS,ActClass.OBS);
        ac.setMoodCode(ActCriteria.EQUALS,ActMood.EVN);

        // Build code and value criteria objects
        CodedTypeAttributeCriteria ctac =
queryComponentFactory.newCodedTypeAttributeCriteria();
        ctac.setCode(dataTypeFactory.newST("302788006"));
        ctac.setCodeSystem(dataTypeFactory.newUID("2.16.840.1.113883.6.96"));
        ac.setCode(ctac);

        PQAttributeCriteria pqac = queryComponentFactory.newPQAttributeCriteria();
        pqac.setValue(PQAttributeCriteria.GREATER_
THAN,dataTypeFactory.newREAL("180"));
        pqac.setUnit(PQAttributeCriteria.EQUALS,dataTypeFactory.newCS("mg/dL"));
        ac.setValue(pqac);

        ActFetch af = queryComponentFactory.newActFetch(ac,
queryComponentFactory.VERSION_DEPENDENT);
        af.addParticipationFetch(pf);
        af.retrieveAll();

        Iterator iter = rimService.queryActs(serviceLocator,af);
        while (iter.hasNext())
        {
            Observation obs = (Observation)iter.next();
```

```
            System.out.println("== Observation ==");
            System.out.println("== id ==>" + obs.getId());
            System.out.println("== code ==>" + obs.getCode());
            System.out.println("== value ==>" + ((PQ)obs.getValue()).literal());
            Iterator partIter = obs.getParticipations();
            while (partIter.hasNext())
            {
                Role role = ((Participation)partIter.next()).getRole();
                System.out.println("== Role id ===> " + role.getId());
            }
        }
```

Example 7–36 Query CE/CD Values

Query for all patients with a diagnosis of "Asthma" or "Pneumonia" (OBS.EVN.DISDX//ActCode with Observation.value (CE) = 195967001// SNOMED-CT or 233604007// SNOMED-CT).

```
// use the same fetch and criteria objects as before.
        // change the code and value attributes for the different criteria
        ctac = queryComponentFactory.newCodedTypeAttributeCriteria();
        ctac.setCode(dataTypeFactory.newST("DISDX"));
        ctac.setCodeSystem(dataTypeFactory.newUID("2.16.840.1.113883.5.4"));
        ac.setCode(ctac);

        CodedTypeAttributeCriteria asthmaCriteria =
queryComponentFactory.newCodedTypeAttributeCriteria();
        asthmaCriteria.setCode(dataTypeFactory.newST("195967001"));

asthmaCriteria.setCodeSystem(dataTypeFactory.newUID("2.16.840.1.113883.6.96"));

        CodedTypeAttributeCriteria pneumoniaCriteria =
queryComponentFactory.newCodedTypeAttributeCriteria();
        pneumoniaCriteria.setCode(dataTypeFactory.newST("233604007"));

pneumoniaCriteria.setCodeSystem(dataTypeFactory.newUID("2.16.840.1.113883.6.96"));

ac.setValue((CodedTypeAttributeCriteria)queryComponentFactory.or(asthmaCriteria,
pneumoniaCriteria));

        af = queryComponentFactory.newActFetch(ac, queryComponentFactory.VERSION_
DEPENDENT);
        af.addParticipationFetch(pf);
        af.retrieveAll();

        iter = rimService.queryActs(serviceLocator,af);
        while (iter.hasNext())
        {
            Observation obs = (Observation)iter.next();
            System.out.println("== Observation ==");
            System.out.println("== id ==>" + obs.getId());
            System.out.println("== code ==>" + obs.getCode());
            System.out.println("== value ==>" + obs.getValue());
            Iterator partIter = obs.getParticipations();
            while (partIter.hasNext())
            {
                Role role = ((Participation)partIter.next()).getRole();
                System.out.println("== Role id ===> " + role.getId());
            }
        }
```

Example 7–37 Query BL Values

Query for all patients with a positive "Mantoux Test" (OBS.EVN.268376005//SNOMED-CT with Observation.value (BL) = true).

```
ctac = queryComponentFactory.newCodedTypeAttributeCriteria();
        ctac.setCode(dataTypeFactory.newST("268376005"));
        ctac.setCodeSystem(dataTypeFactory.newUID("2.16.840.1.113883.6.96"));
        ac.setCode(ctac);


        ac.setValue(ActAttributeCriteria.EQUALS,dataTypeFactory.newBL(true));

        af = queryComponentFactory.newActFetch(ac, queryComponentFactory.VERSION_
DEPENDENT);
        af.addParticipationFetch(pf);
        af.retrieveAll();

        iter = rimService.queryActs(serviceLocator,af);
        while (iter.hasNext())
        {
            Observation obs = (Observation)iter.next();
            System.out.println("== Observation ==");
            System.out.println("== id ==>" + obs.getId());
            System.out.println("== code ==>" + obs.getCode());
            System.out.println("== value ==>" + ((BL)obs.getValue()).literal());
            Iterator partIter = obs.getParticipations();
            while (partIter.hasNext())
            {
                Role role = ((Participation)partIter.next()).getRole();
                System.out.println("== Role id ===> " + role.getId());
            }
        }
```

Example 7–38 Query INT values

Query for all patients with a zero "Missing Tooth Count" (OBS.EVN.251317003//SNOMED-CT with Observation.value (INT) = 0).

```
ctac = queryComponentFactory.newCodedTypeAttributeCriteria();
        ctac.setCode(dataTypeFactory.newST("251317003"));
        ctac.setCodeSystem(dataTypeFactory.newUID("2.16.840.1.113883.6.96"));
        ac.setCode(ctac);


        ac.setValue(ActAttributeCriteria.EQUALS,dataTypeFactory.newINT(0));

        af = queryComponentFactory.newActFetch(ac, queryComponentFactory.VERSION_
DEPENDENT);
        af.addParticipationFetch(pf);
        af.retrieveAll();

        iter = rimService.queryActs(serviceLocator,af);
        while (iter.hasNext())
        {
            Observation obs = (Observation)iter.next();
            System.out.println("== Observation ==");
            System.out.println("== id ==>" + obs.getId());
            System.out.println("== code ==>" + obs.getCode());
            System.out.println("== value ==>" + ((INT)obs.getValue()).literal());
            Iterator partIter = obs.getParticipations();
            while (partIter.hasNext())
```

```
            {
                Role role = ((Participation)partIter.next()).getRole();
                System.out.println("== Role id ===> " + role.getId());
            }
        }
```

Example 7–39 Query TS values

Query for all patients with a "Time of Onset" greater than or equal to 00:02:01 (OBS.EVN.263501003//SNOMED-CT with Observation.value (TS) = 00:02:01).

```
ctac = queryComponentFactory.newCodedTypeAttributeCriteria();
        ctac.setCode(dataTypeFactory.newST("263501003"));
        ctac.setCodeSystem(dataTypeFactory.newUID("2.16.840.1.113883.6.96"));
        ac.setCode(ctac);


        try
        {
            ac.setValue(ActAttributeCriteria.EQUALS,dataTypeFactory.newTS(new
SimpleDateFormat("yyMMddHHmmss").parse("080101000201")));
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }

        af = queryComponentFactory.newActFetch(ac, queryComponentFactory.VERSION_
DEPENDENT);
        af.addParticipationFetch(pf);
        af.retrieveAll();

        iter = rimService.queryActs(serviceLocator,af);
        while (iter.hasNext())
        {
            Observation obs = (Observation)iter.next();
            System.out.println("== Observation ==");
            System.out.println("== id ==>" + obs.getId());
            System.out.println("== code ==>" + obs.getCode());
            System.out.println("== value ==>" + ((TS)obs.getValue()).literal());
            Iterator partIter = obs.getParticipations();
            while (partIter.hasNext())
            {
                Role role = ((Participation)partIter.next()).getRole();
                System.out.println("== Role id ===> " + role.getId());
            }
        }
```

Constraints on the HL7 V3 RIM Model

The constraints to some of the HL7 V3 data types allow HDR to have a physical data model that gives a significant boost to the API performance.

HL7 V3 Datatype Constraints

ED.Reference.Use

HDR allows only one use code for `ED.Reference` instead of many. `ED.Reference` is generally used to store the external URL of an image or document represented by this ED. Since URL is a type of addressing mechanism, HL7 allows `Use` based on the base definition of an address. HDR allows just a single `Use` to be specified on the `ED.Reference` to cater to any use cases where an `Use` may be needed on `ED.Reference`.

AD.Use

HDR allows only 3 use codes for `AD.Use`. A single address may be used as home, office, and the third one can be used for another purpose.

AD.ADXP

The following additional constraints are placed on the number of `ADXP` values used in `AD` datatype:

1. Only 5 street address lines (ADXP with part type `SAL`) are supported. Each street address line can have a maximum of 240 characters.
2. There can be only one ADXP each with part type other than `SAL`. For example, there can be only one building number or one country in the address.

EN.Use

HDR allows only 3 use codes for `EN.Use`. A single name may be used as legal, call-by, and the third one can be used for another purpose.

EN.ENXP

There can be only one `ENXP` each with any part type. For example, there can be only one given name in the name. Each name part can have a maximum length of 1000 characters.

RIM Query API Constraints

- The top-level RIM fetch object must have criteria with RIM structural codes specified.
- Criteria is optional in child fetches but is strongly recommended in all cases where the queried RIM object model is well defined.
- Criteria on any fetch object cannot be connective criteria specifying different structural attributes, for example: a different class code. The recommended approach to restructure such queries is to create different fetch objects for each class code. Connective criteria can still be used at any level provided the RIM structural codes are same on each of the attribute criteria. Each structural code represents a particular type of clinical information such as encounter or an observation and HDR mandates that one fetch is used for each type of clinical data retrieved.

8

Merge, Unmerge, Link and Unlink Person Data in HDR

The code samples below help you to:

- Classes Used by EMPI Examples
- Create a Person Entity
- Revise a Person Entity
- Persist Domains
- Persist a Deduplication Context
- Persist a Linking Context
- Use Reference Modifiers

Most of the basic operations described in this guide are based on the RIM Services APIs. Usage of RIM Service APIs is described in "RIM Service" on page 1-7.

Example 8–1 Classes Used by EMPI Examples

The following code sample implements classes used by the following examples:

```
import oracle.hsgbu.hdr.configuration.InternalOID;
import oracle.hsgbu.hdr.configuration.OIDService;
import oracle.hsgbu.hdr.empi.Domain;
import oracle.hsgbu.hdr.fwk.base.common.ETSException;
import oracle.hsgbu.hdr.fwk.serviceLocator.common.ServiceLocator;
import oracle.hsgbu.hdr.hl7.RimService;
import oracle.hsgbu.hdr.hl7.domain.ActMood;
import oracle.hsgbu.hdr.hl7.domain.ActRelationshipType;
import oracle.hsgbu.hdr.hl7.domain.EntityClass;
import oracle.hsgbu.hdr.hl7.domain.NullFlavor;
import oracle.hsgbu.hdr.hl7.domain.ParticipationType;
import oracle.hsgbu.hdr.hl7.domain.RoleClass;
import oracle.hsgbu.hdr.hl7.factories.ActFactory;
import oracle.hsgbu.hdr.hl7.factories.DataTypeFactory;
import oracle.hsgbu.hdr.hl7.factories.EntityFactory;
import oracle.hsgbu.hdr.hl7.factories.QueryComponentFactory;
import oracle.hsgbu.hdr.hl7.query.ActFetch;
import oracle.hsgbu.hdr.hl7.query.ActRelationshipFetch;
import oracle.hsgbu.hdr.hl7.query.CoreFetch;
import oracle.hsgbu.hdr.hl7.query.ENAttributeCriteria;
import oracle.hsgbu.hdr.hl7.query.ENXPAttributeCriteria;
import oracle.hsgbu.hdr.hl7.query.EntityAttributeCriteria;
import oracle.hsgbu.hdr.hl7.query.EntityFetch;
import oracle.hsgbu.hdr.hl7.query.ParticipationFetch;
```

```
import oracle.hsgbu.hdr.hl7.query.RoleAttributeCriteria;
import oracle.hsgbu.hdr.hl7.query.RoleFetch;
import oracle.hsgbu.hdr.hl7.query.SearchOperator;
import oracle.hsgbu.hdr.hl7.query.SetSearchOperator;
import oracle.hsgbu.hdr.hl7.rim.Act;
import oracle.hsgbu.hdr.hl7.rim.ActRelationship;
import oracle.hsgbu.hdr.hl7.rim.ControlAct;
import oracle.hsgbu.hdr.hl7.rim.Entity;
import oracle.hsgbu.hdr.hl7.rim.InfrastructureRoot;
import oracle.hsgbu.hdr.hl7.rim.Organization;
import oracle.hsgbu.hdr.hl7.rim.Participation;
import oracle.hsgbu.hdr.hl7.rim.Person;
import oracle.hsgbu.hdr.hl7.rim.QueryAct;
import oracle.hsgbu.hdr.hl7.rim.Role;
import oracle.hsgbu.hdr.hl7.types.AD;
import oracle.hsgbu.hdr.hl7.types.ADXP;
import oracle.hsgbu.hdr.hl7.types.BAG<AD>;
import oracle.hsgbu.hdr.hl7.types.BAG<EN>;
import oracle.hsgbu.hdr.hl7.types.CE;
import oracle.hsgbu.hdr.hl7.types.CS;
import oracle.hsgbu.hdr.hl7.types.EN;
import oracle.hsgbu.hdr.hl7.types.ENXP;
import oracle.hsgbu.hdr.hl7.types.II;
import oracle.hsgbu.hdr.hl7.types.SET_II;
import oracle.hsgbu.hdr.hl7.types.ST;
import oracle.hsgbu.hdr.hl7.types.common.CSImpl;
import oracle.hsgbu.hdr.hl7.types.common.NullFlavorImpl;
import oracle.hsgbu.hdr.empi.Context;
import oracle.hsgbu.hdr.empi.DedupContext;
import oracle.hsgbu.hdr.empi.ContextDomain;
import oracle.hsgbu.hdr.empi.LinkingContextDomain;
import oracle.hsgbu.hdr.empi.LinkingContext;
import oracle.hsgbu.hdr.empi.LinkingContextAttributeCriteria;
import oracle.hsgbu.hdr.empi.LinkingContextFetch;
import oracle.hsgbu.hdr.empi.common.LinkingContextFetchImpl;
import oracle.hsgbu.hdr.empi.common.LinkingContextDomainImpl;
import oracle.hsgbu.hdr.empi.common.LinkingContextDomainFetchImpl;
import oracle.hsgbu.hdr.empi.EmpiService;
import oracle.hsgbu.hdr.empi.EmpiConfigurationService;
import oracle.hsgbu.hdr.hl7.rim.Person;
import oracle.hsgbu.hdr.empi.EmpiFactory;
import oracle.hsgbu.hdr.empi.PersonFetch;
import oracle.hsgbu.hdr.empi.DomainAttributeCriteria;
import oracle.hsgbu.hdr.empi.common.DomainAttributeCriteriaImpl;
import oracle.hsgbu.hdr.empi.common.DomainFetchImpl;
import oracle.hsgbu.hdr.empi.common.DomainImpl;
import oracle.hsgbu.hdr.empi.PersonAttributeCriteria;
```

The following variables are used throughout the examples that follow (Examples 8-2 through 8-13):

Variables

Variable Name	Class
mSL	ServiceLocator
mOIDService	OIDService
mRimService	RimService
mAF	ActFactory

mDTF	DataTypeFactory
mEF	EntityFactory
mQCF	QueryComponentFactory
mEMF	EmpiFactory
mConfigService	EmpiConfigurationService

Example 8–2 Create a Person Entity

This code sample illustrates creation of a Person entity using RIM Services. Person attributes such as name, address, gender, and marital status have been set on the Person. The process involves creation of the Person entity using the RIM Service EntityFactory class and populating the attributes required on this object. All data types must be created using the DataTypeFactory class.

The process for creation and persistence of a person includes the following steps:

- An Organization is created; the organization is the scoper of the Identification Role.
- The EntityFactory returns a new Person and assigns a unique identifier to it. The status of the Person is set to Active.
- Various attributes (such as name, address, gender,...) are set on the Person.
- A Domain is attached to the person; the Domain is associated with the person in the form of a Type 2 role played by the Person. In the case of a Domain Enabled Environment, every person created must belong to a Domain.
- A unique identification for the Person (such as Social Security Number) is also attached in the form of a Type 2 role played by the Person.
- The Person is persisted using the standard RIM Services persistence mechanism.

Figure 8–1 Person Registry RMIM

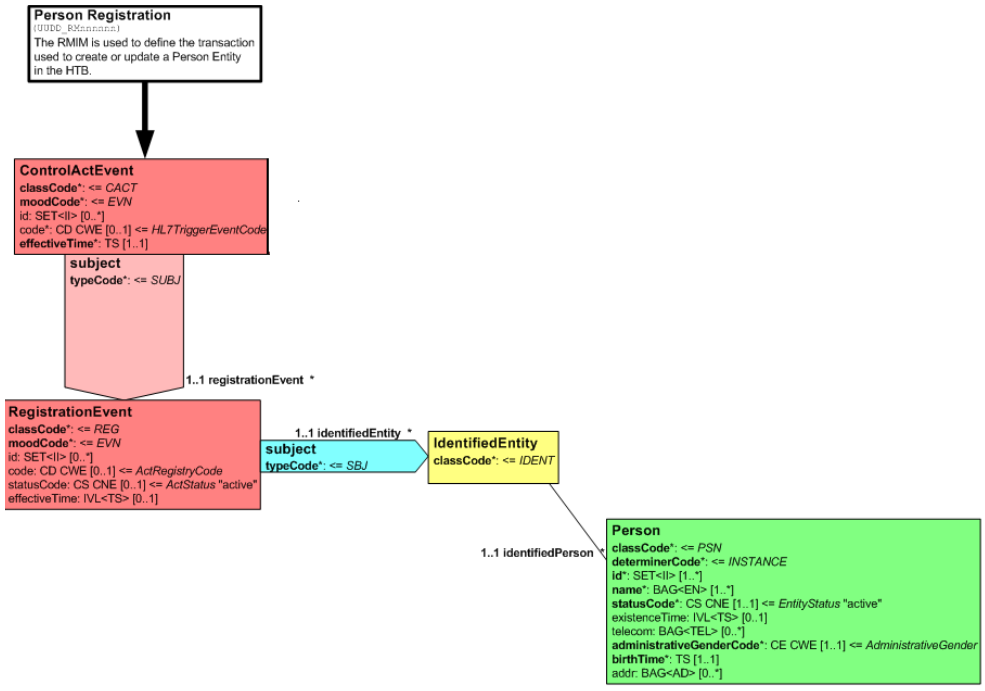


Figure 8–1 illustrates the Person Registry RMIM, including related classes and the interactions between them.

To create a person in a domain-enabled environment, a type II role (the *Domain Role*) must be attached to the person; in a domain-disabled environment a type II role is not required. The domain role is a *played role*, which is identified by concept code 001896, coding scheme HDR Supplemental, and root OIDService.EMPI_DOMAIN. The domain id (extension associated with the domain role) is used to identify the person's domain. Note that the person must be associated with a single valid domain role.

When a person is created, a played type II role (the *Master Person Role*) is attached to the person by the EMPI service. This type II role is identified by concept code 001897, coding scheme HDR Supplemental, and root OIDService.EMPI_MASTER_ID. The master person id (extension associated with the master person role) is used as the logical identifier to represent persons belonging to the same linked set.

Create a person in a domain-enabled mode by attaching a valid domain role, using the following code sample:

Person Creation in Domain Enabled Scenario

```
public void testCreatePersonsDomainEnabled() throws ETSException
{
   //Assign the II for Sally Andrews
   String rootForPerson = "9.989898.5.1";
   String extension = "PSN0001";
   String domainId = "2147483640";

   person = persistPerson(rootForPerson , extension , domainId);

   personII = getPersonII(person);
   System.out.println("Person created with II " +personII);
   System.out.println("Person 1 created and persisted successfully");

   rootForPerson = "9.989898.5.1";
   extension = "PSN0002";
   person = persistPerson(rootForPerson , extension , domainId);

   System.out.println("Person created with II " +personII);
   System.out.println("Person 2 created and persisted successfully");
}


public Person persistPerson(String rootForPerson , String extension ,String
domainId) throws ETSException
{
   //Create a Person object with the provided root and extension
   person = getPerson(rootForPerson , extension );

   if(domainId != null)

{
   //Create a domain role and attach it to the person
   addDomainRole(person , domainId );
}

   //Add a type II ident role
   addIdentToPerson(person, "PASSPORT", "1234567890");

   String rootForIdent = "9.515151.5";
   String extIdent = "535-77-3737";
```

```
   //Create the ident role with the person as the player
   Role identRole = getIdentRole(person, rootForIdent,extIdent );

   CS statusCode = mDataTypeFactory.newCS("active");

   //Create the Registry Act and set the status code to active
   Act registryAct = mActFactory.newRegistryAct(ActMood.EVN, null , null);
   registryAct.setStatusCode(statusCode);
   registryAct.setTitle(mDataTypeFactory.newST("Person Persistence Registry
Act"));

   //Add the Participation to the ident role
   registryAct.addParticipation(ParticipationType.SBJ, identRole);

   //Get the Control Act
   ControlAct controlAct = mActFactory.newControlActEvent(null,null);
   controlAct.setStatusCode(statusCode);

   //Create an Act Relationship between Control Act and Registry Act
   ActRelationship ar = controlAct.addOBActRelationship(ActRelationshipType.SUBJ,
registryAct);

   // Submit the Control Act and commit the data
   ControlAct returnedControlAct = mRimService.submit(controlAct);

   return person;
}


public Person getPerson(String rootForPerson , String extension ) throws
ETSException
{
   //Create a SET_II object using the Persons roor and extension
   SET_II setII = mDataTypeFactory.newSET_II(mDataTypeFactory.newII(rootForPerson
, extension, true));

   // Create a CS object for the determiner code 'INSTANCE'
   CS entityDeterminer = mDataTypeFactory.newCS("INSTANCE");

   //Create the Person object
   person = mEntityFactory.newPerson(null,entityDeterminer,setII);

   // Create a CS object for the status code 'active'
   CS statusCode = mDataTypeFactory.newCS("active");

   //Set the person's status to active
   person.setStatusCode(statusCode);

   //Create a new person with name as "Sally Andrews"
   String givenName = "Sally";
   String famName = "Andrews";

   //Populate the First Name
   CS namePart1 = mDataTypeFactory.newCS("GIV");
   ENXP namePartObj1 =
mDataTypeFactory.newENXP(givenName,namePart1,mDataTypeFactory.nullSET<CS>
    (CSImpl.nullCS(NullFlavorImpl.NI)));

   //Populate the Last Name
```

```
   CS namePart2 = mDataTypeFactory.newCS("FAM");
   ENXP namePartObj2 =
mDataTypeFactory.newENXP(famName,namePart2,mDataTypeFactory.nullSET<CS>
    (CSImpl.nullCS(NullFlavorImpl.NI)));

   ENXP[] namePartList = new ENXP[] {namePartObj1, namePartObj2 };

   //Set the Legal Name
   EN name = mDataTypeFactory.newEN(namePartList ,
mDataTypeFactory.newSET<CS>(mDataTypeFactory.newCS("L")),
    mDataTypeFactory.nullIVL<TS>(CSImpl.nullCS(NullFlavorImpl.NI)));
   BAG<EN> nameBag = mDataTypeFactory.newBAG<EN>(new EN[] {name});

   person.setName(nameBag);

   //Create the address parts
   ADXP addrPart1 = mDataTypeFactory.newADXP("WA", mDataTypeFactory.newCS("STA"))
;
    //Name of the State
   ADXP addrPart2 = mDataTypeFactory.newADXP("Seattle",
mDataTypeFactory.newCS("CTY"));
    //Name of the City
   ADXP addrPart3 = mDataTypeFactory.newADXP("98105",
mDataTypeFactory.newCS("ZIP"));
    //Zip Code
   ADXP addrPart4 = mDataTypeFactory.newADXP("3545 18th Avenue",
mDataTypeFactory.newCS("SAL"));
    //Street Address Line
  ADXP addrPart5 = mDataTypeFactory.newADXP("US", mDataTypeFactory.newCS("CNT"));
    //Name of the Country

   ADXP[] addrParts = new ADXP[]{addrPart1, addrPart2, addrPart3, addrPart4,
addrPart5};

   //Set the address as Home Primary Address
   AD addr = mDataTypeFactory.newAD(addrParts, mDataTypeFactory.newSET_
CS(mDataTypeFactory.newCS("HP")),
    mDataTypeFactory.nullGTS (CSImpl.nullCS(NullFlavorImpl.NI)));

   //Create an address bag
   BAG<AD> addrBag = mDataTypeFactory.newBAG<AD>(addr);

   //Set the address bag on the person
   person.setAddr(addrBag);

   //Add the Gender
   CE genderCode =  mDataTypeFactory.newCE(mDataTypeFactory.newST("000202"),
                                     mDataTypeFactory.newUID(mCodingSchemeOid),
                                     mDataTypeFactory.newST("HDR
Supplemental"),

mDataTypeFactory.nullST(oracle.hsgbu.hdr.hl7.domain.NullFlavor.NI),

mDataTypeFactory.nullED(oracle.hsgbu.hdr.hl7.domain.NullFlavor.NI));

    person.setAdministrativeGenderCode (genderCode);

    //Add the Marital Status
    CE maritalCode =    mDataTypeFactory.newCE(mDataTypeFactory.newST("000282"),
```

```
mDataTypeFactory.newUID(mCodingSchemeOid),
                                                    mDataTypeFactory.newST("HDR
Supplemental"),

mDataTypeFactory.nullST(oracle.hsgbu.hdr.hl7.domain.NullFlavor.NI),

mDataTypeFactory.nullED(oracle.hsgbu.hdr.hl7.domain.NullFlavor.NI));

    person.setMaritalStatusCode(maritalCode);

    //Add the Birth Time
    person.setBirthTime(mDataTypeFactory.newTS(new java.util.Date()));

    //Setting various other attributes for the person

person.setLivingArrangementCode(mDataTypeFactory.newCE(mDataTypeFactory.newST("H")
,

mDataTypeFactory.nullUID(oracle.hsgbu.hdr.hl7.domain.NullFlavor.NI),

mDataTypeFactory.newST("LivingArrangement"),

mDataTypeFactory.nullST(oracle.hsgbu.hdr.hl7.domain.NullFlavor.NI),

mDataTypeFactory.nullED(oracle.hsgbu.hdr.hl7.domain.NullFlavor.NI)));


person.setReligiousAffiliationCode(mDataTypeFactory.newCE(mDataTypeFactory.newST("
1041"),

mDataTypeFactory.nullUID(oracle.hsgbu.hdr.hl7.domain.NullFlavor.NI),

mDataTypeFactory.newST("ReligiousAffiliation"),

mDataTypeFactory.nullST(oracle.hsgbu.hdr.hl7.domain.NullFlavor.NI),

mDataTypeFactory.nullED(oracle.hsgbu.hdr.hl7.domain.NullFlavor.NI)));
    person.setDeceasedInd(mDataTypeFactory.newBL(false));
    person.setMultipleBirthInd(mDataTypeFactory.newBL(false));
    person.setOrganDonorInd(mDataTypeFactory.newBL(false));

person.setTelecom(mDataTypeFactory.newBAG<TEL>(mDataTypeFactory.newTEL(mDataTypeFa
ctory.newCS("H"),
    //Set as Home Number

mDataTypeFactory.newST("+91-80-5108 0001"),

mDataTypeFactory.nullGTS(oracle.hsgbu.hdr.hl7.domain.NullFlavor.NI),

mDataTypeFactory.nullSET<CS>(oracle.hsgbu.hdr.hl7.domain.
                                                    NullFlavor.NI))));
    return person;
}


public Role getIdentRole(Person player, String rootForIdent , String extIdent )
throws ETSException
{
    //Adding the identification role to the Person, assuming the identification
exists
```

```
    CE identRoleCode =
mDataTypeFactory.newCE(mDataTypeFactory.newST("398093005"),null,

mDataTypeFactory.newST("SNOMED-CT"),

mDataTypeFactory.nullST(oracle.hsgbu.hdr.hl7.domain.NullFlavor.NI),

mDataTypeFactory.nullED(oracle.hsgbu.hdr.hl7.domain.NullFlavor.NI));

    CS statusCode = mDataTypeFactory.newCS("active");

    II[] identRoleII = new II[] {mDataTypeFactory.newII(rootForIdent,extIdent,
true)};
    SET_II identRoleSetII = mDataTypeFactory.newSET_II(identRoleII);

    //Set the identification role on the person
    Role identRole = mRoleFactory.newIdentifiedEntity(null, player , null , null);

    //Role identRole = person.addPlayedRole(RoleClass.IDENT, identRoleCode,
org,identRoleSetII);
    identRole.setStatusCode(statusCode);

    return identRole;
}


public Role addDomainRole(Person person , String domainId) throws ETSException
{
    //For the domain enabled scenario, adding the domain information to the person
assuming
    //the domain already exists
    String domCode = "001896"; //Domain Code
    String domCodeSystemName = "HDR Supplemental";
    CE domainRoleCode = mDataTypeFactory.newCE(mDataTypeFactory.newST(domCode),
                                        mDataTypeFactory.newUID(mCodingSchemeOid),
                                        mDataTypeFactory.newST(domCodeSystemName),

mDataTypeFactory.nullST(oracle.hsgbu.hdr.hl7.domain.NullFlavor.NI),

mDataTypeFactory.nullED(oracle.hsgbu.hdr.hl7.domain.NullFlavor.NI));

    OIDService oidService = mServiceLocator.getOIDService();

    //Getting the domain root from the OIDService
    String rootForDomain = oidService.getOID(OIDService.EMPI_
DOMAIN).getRootId().stringValue();

    //Create the Domain II from the root and extension
    II[] domainRoleII = new II[]
                {mDataTypeFactory.newII(mDataTypeFactory.newUID(rootForDomain),
                 mDataTypeFactory.newST(domainId), mDataTypeFactory.newBL(true))};
    SET_II domainRoleSetII = mDataTypeFactory.newSET_II(domainRoleII);

    //Add the person to a Domain and set status to active
    Role domainRole = person.addPlayedRole(RoleClass.IDENT, domainRoleCode, null,
domainRoleSetII);

    CS statusCode = mDataTypeFactory.newCS("active");
    domainRole.setStatusCode(statusCode);
```

```
    return domainRole;
}


        //Method returns the II of the person
        public static II getPersonII(Person person1)
{
    II ii1 = null;
    try
    {
                SET_II setII = person1.getId();
                Set set = setII.toSet();
                Iterator iter = set.iterator();

                while(iter.hasNext())

                {
                        ii1 = (II) iter.next();
                        System.out.println("The root is "
+ii1.root().stringValue());
                        System.out.println("The corresponding extension is "
+ii1.extension().stringValue());

if(ii1.root().stringValue().equals(mRimService.getInternalOID().stringValue()))
                        {
                 break;
                        }
                }
    }
        catch(Exception e)
        {
        e.printStackTrace();
        }
        return ii1
}

public static Role addIdentToPerson(Person person, String conceptCode, String
personIdent) throws ETSException
{
        String codingSchemeName =
getCodingSchemeNameFromConceptListName("LOOKUPS","CTB_ID_TYPES",conceptCode);
        CE identRoleCode =
mDataTypeFactory.newCE(mDataTypeFactory.newST(conceptCode),

mDataTypeFactory.nullUID(NullFlavor.NI),

mDataTypeFactory.newST(codingSchemeName),
                                        mDataTypeFactory.nullST(NullFlavor.NI),

mDataTypeFactory.nullED(NullFlavor.NI));

        II identRoleII = getII(Constants.IDENT_ROLE_ROOT,personIdent,null);
        SET_II identRoleSetII = mDataTypeFactory.newSET_II(identRoleII);

        //Add Identification to Person
        Role identRole =
person.addPlayedRole(RoleClass.IDENT,identRoleCode,null,identRoleSetII);
        identRole.setStatusCode( mDataTypeFactory.newCS("active"));

        return identRole;
```

```
}

public static String getCodingSchemeNameFromConceptListName(String
groupName,String conceptListName,String conceptCode) throws ETSException
{
        String codingSchemeName = null;
        try
        {
        ConceptList conceptList = mEtsService
.getConceptList(groupName,conceptListName);
        Concept[] concept = conceptList.getConceptsByConceptCode(conceptCode);
        if(concept != null && concept.length > 0)
        {
        for(int count=0;count<concept.length;count++)
                        {
                        String returnedConceptCode =
concept[count].getConceptCode();
                        CodingSchemeVersion codingScheVersion
=concept[0].getCodingSchemeVersion();
                        CodingScheme codingScheme =
codingScheVersion.getCodingScheme();
                        codingSchemeName = codingScheme.getName();
                        }
                    }
                                }
                                catch(ETSException e)
                                {
                    throw new ETSException("ETS_EXCEPTION");
                }
                                return codingSchemeName;
}

public static II getII(String root,String extension,BL displayable)throws
ETSException
{
                II ii=null;
                UID uid=mDataTypeFactory.newUID(root);
                ST st=mDataTypeFactory.newST(extension);
                ii=mDataTypeFactory.newII(uid,st,displayable);
                return ii;
}
```

The following code sample creates and persists the person in domain-disabled mode; the person persisted is associated with one linking and one dedup context:

Person Creation in Domain Disabled Scenario

```
public void testCreatePersonsDomainDisabled() throws ETSException
{
    //Assign the II for Sally Andrews
    String rootForPerson = "9.989898.5.1";
    String extension = "PSN0003";

    person = persistPerson(rootForPerson , extension , null);

    personII = getPersonII(person);
    System.out.println("Person created with II " +personII);
    System.out.println("Person 1 created and persisted successfully");

    rootForPerson = "9.989898.5.1";
    extension = "PSN0004";
    person = persistPerson(rootForPerson , extension , null);
```

```
    System.out.println("Person created with II " +personII);
    System.out.println("Person 2 created and persisted successfully");
}
```

Example 8–3 Revise a Person Entity

This code sample illustrates the revision of an existing Person entity:

- The Person entity is fetched by using the Internal Identifier for the person; the fetch mechanism used is a standard RIM Services fetch (Section 7.5, RIM Query, Fetches).
- The address and the domain are updated.
- The Person is persisted back.

In this scenario the address of the person is updated with a new value, and a new version of the fetched person is created by using the createNewVersion method. The new address is set on this new version of the person. The persistence mechanism for the updated person uses the standard RIM persistence mechanism illustrated in this code sample.

In the domain-enabled environment a revised person must have an associated domain role to get persisted. Accordingly, if the existing domain role is removed from a person being revised, a new domain role must be attached to the person. If no changes are made to the domain role, the existing domain role is copied and attached to the person; the revised person retains the old domain id. Note that the revised person must be associated with a single valid domain role. In the domain-disabled environment no domain role is required on the revised person.

While revising a person, a new master person role can be attached to the person by removing the existing master person role. If no master person role is attached, the existing master person role is copied and attached to the revised person. The person thus retains the old master person id. Note that the revised person must be associated with a single master person role.

The following code sample illustrates a domain-enabled scenario, where the person is revised with a new domain ID:

```
//Test method to revise an existing Person. Retrieving all type 2 roles during
update
//Person Sally Andrews with Internal Identifier as 9.989898.5.1-PSN0001 is revised
to reflect change in address
public void testRevisePerson() throws ETSException
{
 //Providing the II to retrieve the person

 String rootForPerson = "9.989898.5.1";
 String extension = "PSN0001";

 II ii = mDataTypeFactory.newII(rootForPerson,extension, true);

 //Creating an entity criteria object
 EntityAttributeCriteria entityCriteria =
mQueryComponentFactory.newEntityAttributeCriteria();

 //Retrieve the person assuming the person already exists in the database

 entityCriteria.setClassCode(SearchOperator.EQUALS , EntityClass.PSN); //Set the
```

```
criteria as entity class is
        Person (PSN)
 entityCriteria.setId(SetSearchOperator.ANY,new II[] {ii});
 entityCriteria.setCurrentVersion(true);

 //Creating the entity fetch based on the criteria
 EntityFetch entityFetch = mQueryComponentFactory.newEntityFetch(entityCriteria);

 //Specifiying the attributes of the person to be retrieved
 entityFetch.retrieveId(true);
 entityFetch.retrieveAddr(true);
 entityFetch.retrieveName(true);

 //Creating the role fetch to retrieve roles owned by the person
 RoleFetch roleFetch = mQueryComponentFactory.newRoleFetch();
 roleFetch.retrieveId(true);
 roleFetch.retrieveClassCode(true);

 //Adding the roleFetch to the entityFetch to retrieve roles owned by the entity
 entityFetch.addOwnedPlayedRoleFetch(roleFetch);

 //Execute the query
 InfrastructureRoot fetchedEntities[] = executeQuery(entityFetch);

 //Throw ETSException if person cannot be fetched
 if(fetchedEntities == null || fetchedEntities.length == 0)
 {
        throw new ETSException("NULL_PERSON_RETURNED");
 }
        else
        {
                //Fetching the person to be updated
                Person personA = null;
                {
                        for (int k = 0; k < fetchedEntities.length; k++)
                        {
                                personA = (Person) fetchedEntities[k];
                                if (personA != null)
                                {
                                        //Revising Person
                                        //Updating person Address attribute
                                        personA = (Person)
personA.createNewVersion();

                                        //Create the address parts

                                        ADXP addrPart1 =
mDataTypeFactory.newADXP("WA",
                                         mDataTypeFactory.newCS("STA")); //Name of
the State
                                        ADXP addrPart2 =
mDataTypeFactory.newADXP("Seattle",
                                         mDataTypeFactory.newCS("CTY")); //Name of
the City
                                        ADXP addrPart3 =
mDataTypeFactory.newADXP("98105",
                                         mDataTypeFactory.newCS("ZIP"));  //Zip
Code
                                        ADXP addrPart4 =
mDataTypeFactory.newADXP("35 Elm Avenue",
```

```
                                        mDataTypeFactory.newCS("SAL")); //Street
Address Line
                                       ADXP addrPart5 =
mDataTypeFactory.newADXP("US",
                                        mDataTypeFactory.newCS("CNT")); //Name of
the Country

                                       ADXP[] addrParts = new ADXP[]{addrPart1,
addrPart2, addrPart3,
                                        addrPart4, addrPart5};

                                       //Set the address as Home Primary Address
                                       AD addr =
mDataTypeFactory.newAD(addrParts,

mDataTypeFactory.newSET<CS>(mDataTypeFactory.newCS("HP")),
                                        mDataTypeFactory.nullGTS
(CSImpl.nullCS(NullFlavorImpl.NI)));

                                       //Create an address bag
                                       BAG<AD> addrBag =
mDataTypeFactory.newBAG<AD> (addr);

                                       //Set the address bag on the person
                                       personA.setAddr(addrBag);

                                       //Retrieving Type 2 roles and printing the
role code
                                       Iterator iter =
personA.getOwnedPlayedRoles();
                                       if (iter != null)
                                       {
                                               while (iter.hasNext())
                                               {
                                                       Role role = (Role)
iter.next();
                                                       if (role != null)
                                                       {
                                                               CE roleCode =
role.getCode()
                                                               if (roleCode !=
null)
                                                               {
                                                                       ST code =
roleCode.code();

if(code!=null)
                                                                       {

String codeValue =

code.stringValue();

                                                                                //
Remove the current domain and

associate the person to a new domain

if("001896".equals(codeValue))
                                                                                {
```

```
iter.remove();

addDomainRole

(personA,"2147483646");

System.out.println("

The domain id is updated");
                                                                        }

System.out.println("

Role Code is " +codeValue);
                                                                        }
                                                                    }
                                                                }
                                                            }
                                                        }
                                                        //Persisting the updated person
                                                        //Set the status code to active
                                                        CS statusCode =
mDataTypeFactory.newCS("active");

                                                        //Create an ident role
                                                        Role identRole =
personA.addPlayedRole(RoleClass.IDENT,
                                                         null ,null,null);

identRole.setStatusCode(statusCode);

                                                        //Create the control act
                                                        ControlAct controlAct =

mActFactory.newControlActEvent(null,null);

controlAct.setStatusCode(statusCode);

                                                        //Creating the registry act
                                                        Act act =
mActFactory.newRegistryAct(ActMood.EVN,
                                                         null , null);
                                                        act.setStatusCode(statusCode);

act.setTitle(mDataTypeFactory.newST
                                                         ("Person Persistence Registry
Act"));

                                                        //Add the participation to the act

act.addParticipation(ParticipationType.SBJ, identRole);

                                                        //create an act relationship
between the control act and
                                                         registry act
                                                        ActRelationship ar =
controlAct.addOBActRelationship
                                                         (ActRelationshipType.SUBJ,act);
```

```
                                                    //Submit the control act, which
persists the person
                                                    ControlAct returnedControlAct =
                                                     mRimService.submit(controlAct);
                                            }
                        }
        }
}

//Method to execute the query

public static InfrastructureRoot[] executeQuery(CoreFetch coreFetch) throws
ETSException
{
//Wrapping the fetch in a control act
ControlAct queryAct = wrapFetchInControlAct(coreFetch);

//Submitting the control act
ControlAct queryResponseCACT = mRimService.submit(queryAct);

return getFetchedResultsFromQueryResult(queryResponseCACT,coreFetch);

}

//Method to wrap the coreFetch in a control act

public static ControlAct wrapFetchInControlAct(CoreFetch coreFetch) throws
ETSException
{
                QueryAct queryAct = mActFactory.newQueryAct(coreFetch);
                ControlAct queryCACT = mActFactory.newControlActEvent(null,null);

                //Create an out-bound act relationship between the query act and
the control act
                queryCACT.addOBActRelationship(ActRelationshipType.SUBJ,queryAct);
                return queryCACT;
}

//Method to fetch the results from QueryResult

public static InfrastructureRoot[] getFetchedResultsFromQueryResult(ControlAct
controlAct, CoreFetch fetch) throws ETSException
{
        List queryResults = new ArrayList();
        try
        {
                // Navigate from the returned control act to the QueryAct
                // ControlAct -> OB act relationship -> QueryAct -- all are pseudo
objects so are linked via empty
                        fetches
                Iterator obControlActRelationships =

controlAct.getOBActRelationships(mQueryComponentFactory.newActRelationshipFetch())
;
                QueryAct queryResultAct = null;
                while(obControlActRelationships.hasNext() && queryResultAct ==
null)
                {
                        ActRelationship currentActRelationship = (ActRelationship)
                         obControlActRelationships.next();
```

```
                        Act targetAct =
currentActRelationship.getTarget(mQueryComponentFactory.newActFetch());
                        if(targetAct instanceof QueryAct)
                        {
                                queryResultAct = (QueryAct) targetAct;
                        }
        }

// Navigate down each OB act relationship from the returned QueryAct to fetched
objects. There is one OB relationship per fetched result

   Iterator obQueryActRelationships =

queryResultAct.getOBActRelationships(mQueryComponentFactory.newActRelationshipFetc
h());
   While(obQueryActRelationships != null && obQueryActRelationships.hasNext())
        {
                ActRelationship currentActRelationship = (ActRelationship)
obQueryActRelationships.next();
                if(currentActRelationship != null)
                {
                        if(fetch instanceof ActFetch)
                        {
                                queryResults.add(getAct(currentActRelationship,
(ActFetch)fetch));\
                        }
                                else if(fetch instanceof ActRelationshipFetch)
                        {

queryResults.add(getActRelationship(currentActRelationship,
                                 (ActRelationshipFetch)fetch));
                        }
                        else if(fetch instanceof ParticipationFetch)
                        {

queryResults.add(getParticipation(currentActRelationship,
                                 (ParticipationFetch)fetch));
                        }
                        else if(fetch instanceof RoleFetch)
                        {
                                queryResults.add(getRole(currentActRelationship,
(RoleFetch)fetch));
                        }
                        else if(fetch instanceof EntityFetch)
                        {
                                queryResults.add(getEntity(currentActRelationship,
(EntityFetch)fetch));
                        }
                }
        }
}
catch(Exception e)
{
        System.out.println("Error");
        e.printStackTrace();
}
return (InfrastructureRoot[]) queryResults.toArray(new
InfrastructureRoot[queryResults.size()]);
}
```

```
private static Act getAct(ActRelationship actRelationship, ActFetch fetch) throws
ETSException
{
        return actRelationship.getTarget(fetch);
}

private static ActRelationship getActRelationship(ActRelationship actRelationship,
ActRelationshipFetch fetch) throws ETSException

        Act pseudoAct = getAct(actRelationship,
mQueryComponentFactory.newActFetch());
        Iterator obActRelationships = pseudoAct.getOBActRelationships(fetch);
        return (ActRelationship) obActRelationships.next()
}

private static Participation getParticipation(ActRelationship actRelationship,
ParticipationFetch fetch) throws ETSException
{
        Act pseudoAct = getAct(actRelationship,
mQueryComponentFactory.newActFetch());
        Iterator participations = pseudoAct.getParticipations(fetch);
        return (Participation)participations.next();
}

private static Role getRole(ActRelationship actRelationship, RoleFetch fetch)
throws ETSException
{
        Participation pseudoParticipation = getParticipation(actRelationship,
         mQueryComponentFactory.newParticipationFetch());
        return pseudoParticipation.getRole(fetch);
}

private static Entity getEntity(ActRelationship actRelationship, EntityFetch
fetch) throws ETSException
{
        Role pseudoRole = getRole(actRelationship,
mQueryComponentFactory.newRoleFetch());
        return pseudoRole.getPlayerEntity(fetch);
```

Example 8–4 Persist Domains

The EmpiConfigurationService provides functionality to persist and query the Domain object and assigns a system-generated domain identifier.

Domains enable a logical partitioning of Person objects for easier data management. In a domain-enabled implementation, every Person must belong to a single domain, and every domain must belong to a single Linking Context and a single Deduplication Context. The Domain Name must be unique across all Domains.

```
private static void persistDomain() throws ETSException {
            String name1 = "EMPI_PUG_DOMAIN_1";
            String name2 = "EMPI_PUG_DOMAIN_2";

            domain1 = mEMF.newDomain(name1);
            domain2 = mEMF.newDomain(name2);
            //Persisting the 2 domains by calling the Empi configuration service
            Domain[] createDomainArr = new Domain[] { domain1, domain2 };
            mConfigService.persistDomains(createDomainArr);
            //Querying the persisted domains to assign the domain ids
            DomainAttributeCriteriaImpl domainAttribCriteria = new
DomainAttributeCriteriaImpl();
```

```
domainAttribCriteria.setName(SearchOperator.LIKE, name1);
DomainFetchImpl domainFetch = new DomainFetchImpl();
domainFetch.retrieveDescription(true);
domainFetch.retrieveName(true);
domainFetch.setTopLevelDomainCriteria(domainAttribCriteria);
Domain[] createdDomains = mConfigService.queryDomains(domainFetch);
System.out.println("the no of domains being created are "
      + createdDomains.length);
domainId1 = createdDomains[0].getId();
domainAttribCriteria = new DomainAttributeCriteriaImpl();
domainAttribCriteria.setName(SearchOperator.LIKE, name2);
domainFetch = new DomainFetchImpl();
domainFetch.retrieveDescription(true);
domainFetch.retrieveName(true);
domainFetch.setTopLevelDomainCriteria(domainAttribCriteria);
createdDomains = mConfigService.queryDomains(domainFetch);
System.out.println("the no of domains being created are "
      + createdDomains.length);
domainId2 = createdDomains[0].getId();
}
```

Example 8–5 Persist a Deduplication Context

The EmpiConfigurationService includes context setup for the deduplication context type.

A Deduplication Context must contain a single domain, and the context name must be unique across all contexts.

```
private static void persistDedupContext(String domainId, String contextName)
            throws ETSException {

            //create Dedup Context
            DedupContext dedupContext = mEMF.newDedupContext();
            if (domainId != null) {
                  //create Context Domain
                  ContextDomain conDomain = mEMF.newContextDomain(domainId);

                  //set Context Domain in DedupContext
                  dedupContext.setContextDomain(conDomain);
            }

            mConfigService.persistContexts(new Context[] { dedupContext });
```

Example 8–6 Persist a Linking Context

The EmpiConfigurationService includes context setup for linking context type.

Multiple domains are permitted for each context, but the same domain cannot be included in more than one context. Domain precedence is required to be set and must be unique if more than one domain is defined in a context. The context name must be unique across all contexts.

```
private static void persistLinkingContext(String[] domainIds)
            throws ETSException {

            // Linking context with match conflict code,upper threshold,lower
            // threshold,match conflict code,dominant person conflict code and
            // default operation flag
            LinkingContext linkContext = mEMF.newLinkingContext;

            //adding the linking context doamins to linking context domain array
```

```
            LinkingContextDomain[] linkingContextDomainArr = new
LinkingContextDomain[domainIds.length];
            //Linking context domain with domain id and domain precedence
            LinkingContextDomain linkingContextDomain = null;
            String domainPrecedence = null;
            for (int i = 0; i < domainIds.length; i++) {
                  domainPrecedence = new Integer(i).toString();
                  linkingContextDomain =
mEMF.newLinkingContextDomain(domainIds[i],
                        domainPrecedence);
                  linkingContextDomainArr[i] = linkingContextDomain;
            }

            //set LinkingContext Domain in LinkingContext
            linkContext.setLinkingContextDomain(linkingContextDomainArr);


                  mConfigService.persistContexts(new Context[] { linkContext });
                  //Querying the context to retrieve the context id
                  Context[] createdContext = null;

                  //create Linking Context Attribute Criteria
                  LinkingContextAttributeCriteria linkSearchCriteria = mEMF
                        .newLinkingContextAttributeCriteria();
                  linkSearchCriteria.setName(SearchOperator.EQUALS, contextName);


                  LinkingContextFetch contextFetchImpl = mEMF
                        .newLinkingContextFetch(linkSearchCriteria);
                  contextFetchImpl.retrieveName(true);

                  LinkingContextDomainFetchImpl linkingContextDomainFetch =
(LinkingContextDomainFetchImpl)
                                    mEMF.newLinkingContextDomainFetch();
                  linkingContextDomainFetch.retrieveDomainId(true);
                  linkingContextDomainFetch.retrieveDomainPrecedence(true);
                  ((LinkingContextFetchImpl) contextFetchImpl)

.addLinkingContextDomainFetch(linkingContextDomainFetch);

                  createdContext = mConfigService.queryContexts(contextFetchImpl);

                  //checking the values of created contexts

                  LinkingContext createdLinkContext = (LinkingContext)
createdContext[0];
                  linkingContextId = createdLinkContext.getId();
```

Example 8–7 Use Reference Modifiers

This code sample illustrates how to use reference modifiers to update the domain id:

CREATE_OR_OVERLAY **and** OVERLAY

Using reference modifiers CREATE_OR_OVERLAY and OVERLAY in the domain-enabled environment, a person can be attached to a new domain id. If no domain role is attached the existing domain role is copied and attached to the person; the overlaid person retains the old domain id.

The master person id cannot be updated with these reference modifiers. Accordingly, the existing master person role is copied and attached to the overlaid person, with the

overlaid person retaining the old master person id. The EMPI service thus copies the two type II roles (domain and master person roles) as appropriate in the domain-enabled environment, and only one type II role (master person role) in the domain-disabled environment.

```
public void testOverlayPerson() throws ETSException
{
String rootForPerson = "9.989898.5.1";
String extension = "PSN0001";
SET_II setII = mDataTypeFactory.newSET_II(mDataTypeFactory.newII(rootForPerson ,
 extension, true));

// Make a reference using OVERLAY reference modifier and resubmit the same
Person person1 = (Person) mEntityFactory.makeReference
( oracle.hsgbu.hdr.hl7.factories.ReferenceModifier.OVERLAY ,
 EntityClass.PSN, null, mDataTypeFactory.newCS("INSTANCE"), setII , 0);
person1.setStatusCode(mDataTypeFactory.newCS("active"));

String domainId = "2147483644";
//Create a new domain role and attach it to the person
addDomainRole(person1 , domainId );

String rootForIdent = "9.515151.5";
String extIdent = "111-222-333";

// Create the ident role with the person as the player
Role identRole = getIdentRole(person1, rootForIdent,extIdent );

CS statusCode = mDataTypeFactory.newCS("active");

//Create the Registry Act and set the status code to active
Act registryAct = mActFactory.newRegistryAct(ActMood.EVN, null , null);
registryAct.setStatusCode(statusCode);
registryAct.setTitle(mDataTypeFactory.newST("Person Persistence Registry Act"));

//Add the Participation to the ident role
registryAct.addParticipation(ParticipationType.SBJ, identRole);

//Get the Control Act
ControlAct controlAct = mActFactory.newControlActEvent(null,null);
controlAct.setStatusCode(statusCode);

//Create an Act Relationship between Control Act and Registry Act
ActRelationship ar = controlAct.addOBActRelationship(ActRelationshipType.SUBJ,
 registryAct);

// Submit the Control Act and commit the data
ControlAct returnedControlAct = mRimService.submit(controlAct);

System.out.println("Person overlaid successfully");
}
```

MUST_EXIST

With MUST_EXIST as the reference modifier, neither the domain role nor the master person role can be altered. The person in the repository is undisturbed and the object graph remains unchanged.

CREATE_IF

With the reference modifier CREATE_IF the person object is created if it not persisted earlier. The master person id is automatically generated internally by the EMPI service—the master person role must not be supplied externally.

In the domain enabled environment the domain id must be provided externally, but in the domain-disabled environment it is not necessary.

The following code sample of illustrates using the CREATE_IF reference modifier in the domain enabled mode:

```
public void testCreateIfPerson() throws ETSException
{
String rootForPerson = "9.989898.5.1";
String extension = "PSN0005";

SET_II setII = mDataTypeFactory.newSET_II(mDataTypeFactory.newII(rootForPerson ,
 extension, true));

// Make a reference using OVERLAY reference modifier and resubmit the same
Person person1 = (Person) mEntityFactory.makeReference
 (oracle.hsgbu.hdr.hl7.factories.ReferenceModifier.CREATE_IF, EntityClass.PSN,
null,
 mDataTypeFactory.newCS("INSTANCE"), setII , 0);
person1.setStatusCode(mDataTypeFactory.newCS("active"));

String domainId = "2147483642";

//Create a new domain role and attach it to the person
addDomainRole(person1 , domainId );

String rootForIdent = "9.515151.5";
String extIdent = "111-222-333";

//Create the ident role with the person as the player
Role identRole = getIdentRole(person1, rootForIdent,extIdent );

CS statusCode = mDataTypeFactory.newCS("active");

//Create the Registry Act and set the status code to active
Act registryAct = mActFactory.newRegistryAct(ActMood.EVN, null , null);
registryAct.setStatusCode(statusCode);
registryAct.setTitle(mDataTypeFactory.newST("Person Persistence Registry Act"));

//Add the Participation to the ident role
registryAct.addParticipation(ParticipationType.SBJ, identRole);

//Get the Control Act
ControlAct controlAct = mActFactory.newControlActEvent(null,null);
controlAct.setStatusCode(statusCode);

//Create an Act Relationship between Control Act and Registry Act
ActRelationship ar = controlAct.addOBActRelationship(ActRelationshipType.SUBJ,
registryAct);

// Submit the Control Act and commit the data
ControlAct returnedControlAct = mRimService.submit(controlAct);
}
```

9

Enterprise Terminology Server (ETS)

Enterprise Terminology Services (ETS) provides a repository for coded vocabulary content used across the HDR platform, and an API suite used by applications to access its content. At its core, ETS manages collections of *concepts* that function as a single representation of a distinct idea. ETS provides services that include versioning of terminologies, maintenance of changes between versions, mapping between terminologies, and management of concept domains into *concept lists*.

This section provides an overview of core terminology entities in ETS and describes how programmers can use them.

Generic Terminology Model

ETS uses a generic terminology model that captures the essential features of disparate terminology systems. The generic terminology model provides:

- Real-time access to terminology content.
- A generic API that provides basic terminology services for all terminologies.
- A data model for custom terminologies.

Terminologies are represented in ETS as Coding Schemes. A Coding Scheme is a generic structure that contains terminology meta-data, such as name, description, and active versions. Actual terminology content is loaded and stored in a Coding Scheme Version. Names of Coding Scheme Versions are decided by the user and are specified when the version is loaded. You can load new Coding Scheme Versions as required. For Coding Schemes that have multiple versions, exactly one version can be designated as default. Note that editable terminologies have only one version; Concepts in an editable terminology can be modified without loading a new version.

A Coding Scheme Version contains a definite set of Concepts, Descriptions, Attributes, and Relationships. A concept is the basic unit of information in a coding scheme version: It corresponds to a specific unit of meaning in the native terminology. Every concept has a Concept Code, which is the code given to it by the terminology. ETS identifies concepts and other ETS components (Descriptions and Relationships) using a system-generated identifier called ETS ID. Concepts (and other components) from different versions of a terminology have different ETS IDs, as the concept code may not correctly identify a concept in a different version.

A concept may have one or more textual descriptions. ETS supports multiple descriptions for a concept in the languages supported by that coding scheme version – whether defined by the terminology, or added later by the user. For concepts that have multiple descriptions, exactly one description must be designated as the preferred description for every language supported by the Coding Scheme Version. All other descriptions of a given language, associated with that concept are designated as synonyms. An application may use specific descriptions for designated contexts. This is done by defining Usage Contexts and associating local descriptions to those contexts.

Managing Usage Contexts

Usage Contexts let an HDR solution specify and determine which concept list or local description (of a specific ETS concept) should be used in a given application context. Usage contexts are an attribute of concept lists and local descriptions. A user may specify a usage context when a concept list or a local description is created or later.

When accessing a concept list or local description, HDR solutions may specify a usage context. Based on the specified usage context, HDR retrieves a matching concept list or local description. For example, a Utilization Review department may require diagnoses to be displayed as short names or abbreviations. For this to be implemented in HDR, first a usage context with a name such as Utilization Review must be created. Then, the required local descriptions (short names or abbreviations) with the Utilization Review usage context can be created for the appropriate concepts. Subsequently, applications developed for the department can use ETS APIs with the Utilization Review usage context to display the required local descriptions.

Each local description of a concept must have a single usage context that is unique for that concept. If a local description is assigned a usage context and a local description for that concept already exists with the same usage context, the operation succeeds—but the usage context is removed from the first local description.

A usage context can similarly be used by a concept list. associated with an organization can by an application. For example, a healthcare enterprise might have a concept list of medical services called ENT_MED_SERVICES. A hospital owned by the enterprise may require a specialization of that concept list that contains a subset of the original values. For this to be implemented in HDR, first create a usage context with a name such as Fair Oaks, and associate it with a hospital unit. Then, a specialization of the ENT_MED_SERVICES concept list can be created, with a name such as FAIR_ OAKS_MED_SERVICES, and associated with the Fair Oaks usage context.

Use ETS API to manage usage context.

Associating a Usage Context with an Organization

Use the following method to associate a usage context with an organization:

`associateUsageContextWithExternalOwner`.

A Relationship represents a directed relation between two concepts: from a source concept to a target concept. Relationships can be defined between concepts in the same Coding Scheme Version. These are usually provided as part of the terminology itself.

The generic terminology model serves as the base for a number of standard-based terminologies for which ETS provides special support. These terminologies are referred to as core terminologies in HDR. The following terminologies are referred to as core terminologies in ETS:

- FDB4
- HL7 v3 Code Systems (Seeded)
- LOINC
- SNOMED CT

Special support for core terminologies is in the form of:

- Terminology-specific APIs (in addition to the generic APIs).
- Specific loaders (and associated integrity checking) for loading the terminologies into ETS.

As core terminologies are based on the generic terminology model, they support all features of generic terminologies, such as local descriptions, usage contexts, equivalence, attributes, and cross maps. However, core terminologies are not editable: new Coding Scheme Versions have to be loaded and activated to update the terminology content.

Note: The following terminologies supported by ETS cannot be designated as editable. They are created as non-editable by default when ETS is installed:

- CPT4
- FDB4
- HCPCS-2
- HL7 v3 Code Systems (Seeded)
- ICD-10
- ICD-9-CM-DRG
- ICD-9-CM-MDC
- ICD-9-CM-V1
- ICD-9-CM-V3
- LOINC
- SNOMED CT

Although editable coding schemes can be updated using ETS APIs, editable coding schemes cannot be updated by loading a new version, because they can have only one version.

Caution: Do not attempt to load new versions for an editable coding scheme, other than the original version.

Verify Different Terminology Versions Using Change Files

Change files are used to document the differences between successive versions of a terminology that are loaded into ETS, in a format that is acceptable for loading purposes. Change files are loaded at the same time as the terminology version data

using the same loader and importer. Change files contain the following types of information:

- **Reassignment:** The meaning of one concept is occasionally *reassigned* to another concept represented by a different concept code. Following are some of the situations in which reassignment occurs:
  - Duplicate concepts are detected. One of them is elected to continue representing the meaning while the other is retired or deleted and reassigned to the retained concept.
  - A concept is detected to be erroneous. The erroneous concept is retired or deleted and reassigned to a correct concept.
  - The classification of a concept is changed. If the concept codes are hierarchical (as in ICD-9-CM), the change in classification translates to a change in concept code, necessitating reassignment.
  - A reassignment is indicated in a change file by a row containing an entry of type *S* (semantic reassignment), followed by a *source* concept code (the concept whose meaning is being reassigned to another code), and a *target* concept code (the concept whose code now captures the meaning).

  See Also

  /ets/hdr-ets-1.0.0-SNAPSHOT/db/execute/readme/Change_File_Formats_General.txt from hdr-1.0.0-SNAPSHOT.zip for more information about the format of the change file.

  If both the source and target concepts of a reassignment are from the new version, the reassignment is said to be *intra-version*. For example, in SNOMED-CT, if a duplicate or erroneous concept is detected, the new version carries forward the duplicate or erroneous concept in an inactive status. The reassignment in this case is from the inactive concept in the new version to an active concept in the new version.

  If the source of a reassignment is from the previous version and the target is from the new version, the reassignment is said to be *inter-version*. For example, in ICD-9-CM, if an erroneous concept is detected, it is deleted and excluded from the new version. A correct concept is provided in the new version and a reassignment is created between the concepts in the previous version and the new version. To process an inter-version reassignment contained in a change file, the ETS importer looks for the source code in the non-quarantined version (of the terminology in question) that has the latest load date. The non-quarantined version can be either *active* or *retired*.

- Reuse: Occasionally, a concept code used in the previous version is *reused* to represent a concept with a different meaning in the new version. Note that this is considered bad terminology practice and should only be used to account for inadvertent errors. *Unless a reuse is explicitly called out in the change file, a concept in the previous version is always considered equivalent to a concept with the same concept code in the new version.*

Change files are preseeded for the following terminologies:

- HL7
- HDR Supplemental

For each of the following terminologies, new versions and their change files are available to customers, possessing licenses, from the vendor of the terminology:

- IETF RFC 1766

- ISO 3166-1 alpha2
- NUBC-UB92
- CPT-4
- FDB
- HCPCS-2
- ICD-10
- ICD-9-CM-DRG
- ICD-9-CM-MDC
- ICD-9-CM-V1
- ICD-9-CM-V3
- LOINC
- SNOMED-CT

For all other terminologies that are loaded into ETS, change files must be created separately and loaded for each new version.

Note that change files must be loaded concurrently with the new version of a terminology. It is not possible to load a change file for a version of a terminology *after* both versions have been loaded.

Note also that a change file can only equivalence concepts between two consecutively imported versions of a terminology. Hence the order in which versions are imported is significant if change files are being used. The following scenarios illustrate this constraint:

- A concept (concept code X) exists in version 1 of a coding scheme. The concept neither appears in version 2 nor is reassigned to an equivalent version 2 concept. A concept with code X reappears in version 3 with the same meaning as in version 1. It is not possible to indicate to ETS that concept X from version 1 is equivalent to concept X from version 3-because it spans a version.
- A concept (concept code X) exists in version 1 of a coding scheme. The concept neither appears in version 2 nor is reassigned to an equivalent version 2 concept. In version 3, another concept (concept code Y) is created that is equivalent to concept X from version 1. It is not possible to indicate to ETS that concept X from version 1 is equivalent to concept Y from version 3.

  Note: To support concept equivalence, the ETS importer does not import a second quarantined version of a terminology if one already exists. This facilitates verification of equivalence between the quarantined version and the previous version of the terminology before the quarantined version is published.

See also: Loading and Activating Coding Scheme Versions

The *Implementation Guide* for a detailed procedure on how to define interterminology mapping using cross maps.

Loading and Activating Coding Scheme Versions

Terminologies have to be loaded into ETS (as coding scheme versions), imported, and activated before they are used. This includes initial versions of core terminologies, which have to be loaded, imported and activated at implementation.

ETS provides terminology loader and importer jobs that load and import a terminology after performing required validations. ETS provides custom loader and importer jobs for core terminologies, and generic loader and importer jobs for generic and custom terminologies.

Note:

New versions of the following terminologies can be loaded if required; HDR does not seed versions of these terminologies (they are available from Apelon, Inc.):

- IETF RFC 1766
- ISO 3166-1 alpha-2
- NUBC-UB92

New versions of the following terminologies can be loaded if required; HDR does not seed versions of these terminologies (they are available from Apelon, Inc.):

Caution:*Do not load versions of terminologies that are seeded in HDR. These include:*

- HL7
- HDR Supplemental

If you have already loaded such versions, mark them as retired and non-default.

The procedure for implementing a new coding scheme version is the same for both generic and core terminologies. To load and activate coding scheme versions, perform the following steps:

Steps:

1. Prepare the terminology files.
2. Load the terminology into ETS as a coding scheme version.
3. Publish the coding scheme version.
4. Activate the coding scheme version.

You can use HDR Terminology Jobs to load and publish coding scheme versions.

See Also

- /ets/hdr-ets-1.0.0-SNAPSHOT/db/execute/readme from hdr-1.0.0-SNAPSHOT.zip for additional information about loading.

  The notes can be categorized into DBA, General, and Terminology related notes.

  - *DBA Notes:* Gives a brief description of some common database management issues related to ETS, such as sizing issues, rollback adjustments for loads and imports, intermedia text indexes, and load/import performance, as well as a general description of database access patterns of ETS.
  - *General Notes:* Gives the basic principals common to all terminology file formats. This section must be read before moving on to the details of specific loader file formats.
  - *Terminology Notes:* Gives additional information relating to the core terminologies supported in ETS. Each of the core terminologies have a separate notes file. These files are to be referenced for information relating to the respective loader file formats.

Preparing Terminology Content and Control Files

To create the terminology files and move them into the correct folder, perform the following steps:

Steps

1. If the terminology is a generic terminology, create the terminology files in the format expected by the ETS generic loader. Otherwise, ensure that the files are in the format expected by the appropriate terminology loader. (If the terminology is supported by Apelon, this step is not required.)

   See Also

   - /ets/hdr-ets-1.0.0-SNAPSHOT/db/execute/readme from hdr-1.0.0-SNAPSHOT.zip, for details regarding file formats required by ETS terminology loaders.
   - /ets/hdr-ets-1.0.0-SNAPSHOT/db/execute/readme/Change_File_Formats_General.txt from hdr-1.0.0-SNAPSHOT.zip, for details about change files.

   Note:

   - To load equivalence information for the terminology version being loaded, the change file must be specified while loading the version-ETS does not support retrospective loading of change information.
   - For equivalence processing to be performed correctly, versions must be loaded in order. Equivalence processing assumes that the codes referenced in the change file are from the version currently being loaded and its immediate predecessor.
2. Move the terminology files to a directory in the same file system as the Applications instance-a directory that is accessible by the Oracle Database Scheduler (DBMS_SCHEDULER).
3. Create a control file that reflects the locations of the terminology files and move it to a directory in the same file system as the Applications instance-a directory that is accessible by the Oracle Database Scheduler (DBMS_SCHEDULER).

Creating New Generic Coding Schemes

ETS lets users define and implement custom terminologies for specific needs. Custom terminologies must be based on the generic ETS terminology model. Coding schemes that implement the generic terminology model are known as generic coding schemes.

See also:Coding Scheme

Loading a Coding Scheme Version

To load a new coding scheme version, use the Oracle Database Scheduler (DBMS_SCHEDULER).

See Also:

/ets/hdr-ets-1.0.0-SNAPSHOT/db/execute/readme from hdr-1.0.0-SNAPSHOT.zip, for information about control files.

Using Oracle Database Scheduler (DBMS_SCHEDULER)

Select the HDR Loader Job, and enter values for the Control File (absolute path), Coding Scheme Name, and Coding Scheme Version Name parameters.

Publishing a Coding Scheme Version

A loaded terminology is staged for importation into ETS. Use HDR Importer Job directly for publishing a coding scheme version.

The published coding scheme version is in the quarantined state by default. The coding scheme version must be activated before it can be used.

Note: In order to support concept equivalence, the HDR Importer job process does not publish a second quarantined version of a coding scheme if one already exists. This facilitates verification of equivalence between the quarantined version and the previous version of the terminology before the quarantined version is published.

Using Oracle Database Scheduler (DBMS_SCHEDULER)

For publishing a staged coding scheme version, select the HDR Importer job, and enter values for the Load Sequence Number and Dry Run Mode parameters. You can get the Load Sequence Number from the log file of the HDR Loader job that has successfully loaded the data (coding scheme versions, classifications, and cross maps).

See Also:

- *Oracle Applications System Administrator's Guide*

Note: In order to support concept equivalence, the HDR Importer job does not publish a second quarantined version of a terminology if one already exists. This facilitates verification of equivalence between the quarantined version and the previous version of the terminology before the quarantined version is published.

See Also:

/ets/hdr-ets-1.0.0-SNAPSHOT/db/execute/readme, for details regarding file formats for ETS terminologies and loaders.

Activating a New Terminology Version

A quarantined coding scheme version must be activated before it can be used. A quarantined version can be activated only ETS API.

Interterminology Mapping

Inter-terminology Mapping provides support for any type of relationship, including equivalence, between two different terminologies. Relationships that can be defined include (but not restricted to) broader-than, narrower-than, and clinical-to-administrative relationships.

Inter-terminology Mapping is implemented using cross maps. A cross map defines the relationship between a source concept and a target concept. A number of cross maps are aggregated into a Map Set. ETS specifies file formats for map sets and cross maps. The HDR Loader job loads these files into ETS tables.

Interterminology Mapping Using Cross Maps

Interterminology mapping provide a mechanism by which concepts from a source version in one terminology can be mapped to concepts from a target version in another terminology. Mappings are typically tailored for a specific application. For example, a data-aggregating or reporting application may require a mapping between specialized SNOMED-CT codes and coarse ICD-9-CM codes. A data retrieval application may use mappings with the opposite semantics (from less granular

classifier codes to more detailed codes). These examples serve to illustrate that mappings serve multiple purposes, and not all cross maps indicate equivalence. Those cross maps that truly do indicate equivalence must be explicitly flagged by the author of the cross maps. This section describes the steps you should follow to indicate equivalence between concepts from two different terminologies using interterminology mapping files.

See Also:

The *Implementation Guide* for a detailed procedure on how to define interterminology mapping using cross maps and how to load the map set files.

Guidelines: Cross Maps

The ETS Cross mapping model is based on the SNOMED CT cross mapping model. Cross-mapping mechanisms provide support for the following:

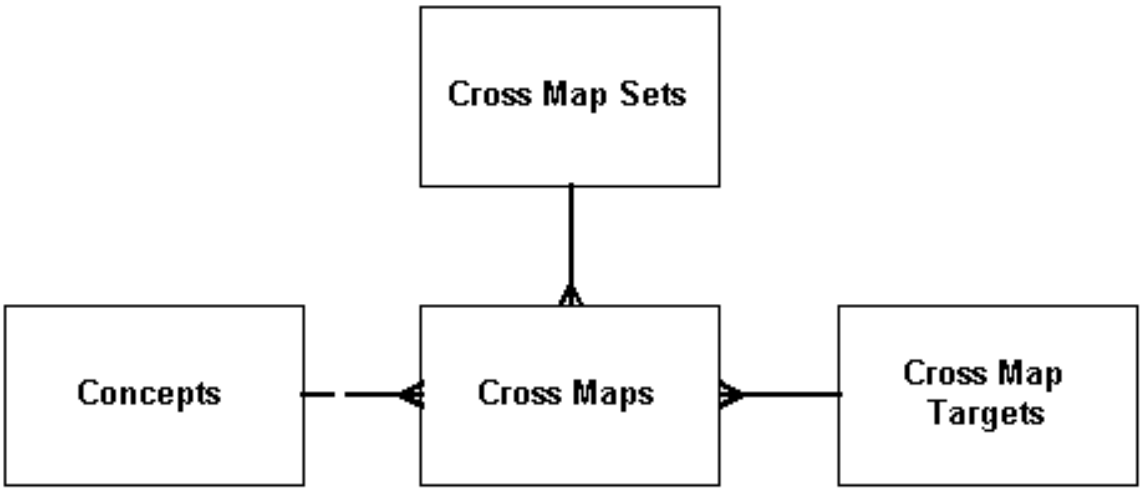
- Mapping a single concept to a target code (a one-to-one mapping).
- Mapping to a set of Target codes (a one-to-many mapping).

The current structure does not support:

- Mapping a set of Concepts to a Target.

The relationship between these tables is shown by the following chart:

Figure 9–1 ETS Cross Mapping Relationship



A map set defines a mapping between two coding scheme versions, such as Terminology A version 2003 and Terminology B version 2.01. Each map set is composed of multiple cross maps. Each cross map consists of a source concept and one or more target concepts, such as a source concept from Terminology A and one or more target concepts from Terminology B—to which it maps.

Loading Cross Maps Provided by the College of American Pathologists

The principal difference between cross map files distributed by the College of American Pathologists (with SNOMED CT) and those expected by ETS loaders is that the SNOMED CT files could contain data regarding multiple map sets in a single file.

The map set file may contain multiple rows, each pertaining to a different map set. The cross map file may contain cross maps relating to multiple map sets, and the map targets file may contain targets used by multiple map sets (targets related to multiple coding schemes).

To make the SNOMED CT files suitable for ETS loading, split the files into map sets. The map set file should contain only one row, representing one map set. The cross maps file should contain only rows containing the map set ID of the chosen map set. The map targets file should contain only targets related to the target coding scheme specific to the map set.

See Also:

/ets/hdr-ets-1.0.0-SNAPSHOT/db/execute/readme from hdr-1.0.0-SNAPSHOT.zip for the cross-map table structures.

ETS Object Model

The ETS base code consists of a core set of objects and services required for ETS to operate. The following sections provide a brief review of some of these core terminology objects:

- Coding Scheme
- Coding Scheme Version
- Concept
- Concept Description
- Relationship
- Usage Context

Coding Scheme A coding scheme is a representation of a single terminology across all of its versions. CPT-4, ICD-9, LOINC and SNOMED are examples of common coding schemes. Coding schemes can be created as needed by calling the createCodingScheme method in the ETSAdministrationService. Coding Schemes can be added into the repository but can never be removed.

Coding Scheme Version A coding scheme version represents a single release of a coding scheme. A new coding scheme version is added every time a new release of a coding scheme is loaded into HDR. The 2004 release of CPT-4 and the v2.10 release of LOINC are examples of coding scheme versions. Coding scheme versions are created by running the HDR Terminology Jobs using a set of terminology files. Although coding scheme versions can never be removed from the system, they can be retired. The number of coding scheme versions in a system typically grows over time.

Concept A concept is a single representation of an idea in a terminology and is associated with the coding scheme version that defines it. For example, a concept with concept code ACTN is included in version 2.01.4 of coding scheme ActClass. The number of concepts in a coding scheme version can range from less than a hundred to thousands. A concept is required to have a content-publisher defined concept code, which must be unique within a coding scheme version. A concept cannot be removed from a system, but its parent coding scheme version can be retired.

Concept Description A concept description, sometimes also called a description, is a human-readable text description for a single concept. There are two types of concept descriptions: terminology-specified and locally-specified. A terminology-specified

concept description is supplied by a terminology provider and populated by the ETS Loader/Importer. Terminology-specified concept descriptions are immutable once they are in the system and cannot be deleted. Every concept is required to have exactly one preferred terminology-specified description that can be used as the default when displaying a concept without any other contextual information. For example, concept ACTN in version 2.01.4 of coding scheme ActClass has the terminology-specified description action. In contrast, locally-specified descriptions are supplied by an organization using ETS. A locally-specified concept description can be associated with a UsageContext to aid the application by providing a contextually-appropriate description of a concept.

Relationship A relationship is a connection that ties concepts together into graphs representing ontological information. Relationships can be used to represent hierarchies, classifications, or arbitrarily connected graphs within terminologies. Relationships are built as a combination of three concepts: a source concept, a relationship type concept, and a target concept. BACTERIAL PNEUMONIA (source), IS TREATED_BY (relationship type), ANTIBIOTIC (target) is an example of a typical relationship.

Some coding schemes, such as SNOMED-CT, contain explicit relationship information. Others have implicit relationships, which can be derived from other fields. In ICD-9, for example, a hierarchical relationship is implicitly defined by the ICD-9 concept codes. In another example, Concept 710.x, is automatically known as a child of concept 710.

Usage Context A usage context is a mechanism that lets customers define an application specific context for descriptions and customized child concept lists. Concepts provide a method for getting a description based on a usage context; concept lists have a method, getChildConceptList(UsageContext), to get the child concept list based on the usage context. The number of usage contexts in a typical system is expected to be small (less than 50).

ETS Concept Lists

An ETS concept list is a container for ETS concepts; each concept is associated with a membership code that is distinct within that list. This container abstraction has many practical applications within HDR and for customers interacting with ETS directly. A concept list is useful for presenting a commonly used list of values to a user, such as a list of valid lab tests, a list of commonly prescribed pain medications, or a list of surgical procedures. The concepts inside a concept list can be derived from any ETS coding scheme version. This permits the construction of a concept list to represent Concepts that represent Influenza, for example, containing one concept from each relevant terminology loaded into ETS.

ETS supports a number of advanced concept list features such as concept list groups, access restrictions (extensibility), and parent-child relationships between lists including inheritance options, specialization of lists, and integration with concept equivalence. Concept list objects can be specialized by implementing usage contexts linking a specialized child concept list to its parent concept list . Concept list objects have activation and retirement dates for the membership of their concept members, stored as a time stamp rounded to the nearest second.

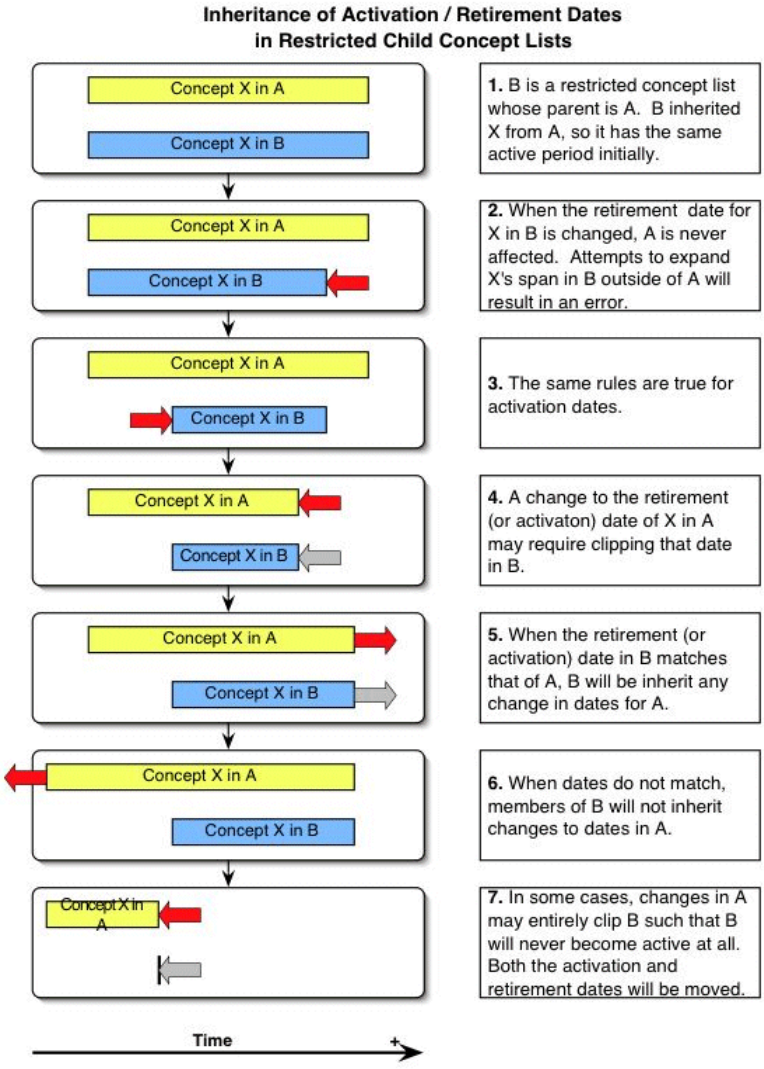
Concept list objects have access restrictions, which are stored as extensibility values. Third-party developers do not have access to set the extensibility of a concept list. Concept list objects can be defined as system, extensible, and user. System-defined concept list objects are created and seeded with each HDR release. Customers cannot add, remove, or update concept list objects within this list. User-defined concept

objects are created by customers and permit the addition and modification of concept objects. Extensible concept list objects are delivered with HDR, but let customers add concept objects into the list.

Parent-child concept list relationships are implemented in ETS. The child concept list can be restricted. Restricted concept list objects can only add concept objects or have concept objects that are already included in their parent concept list objects. Child concept list objects can also inherit additions or deletions of concept objects from their parent concept lists.

Figure 9-2 illustrates the inheritance behavior of restricted members.

Figure 9–2 Inheritance of Activation/Retirement Dates



Creating and Updating a Concept List

Use ETS API to create a concept list. For more information on Concept Lists API, refer to the *Oracle Healthcare Data Repository Javadoc*.

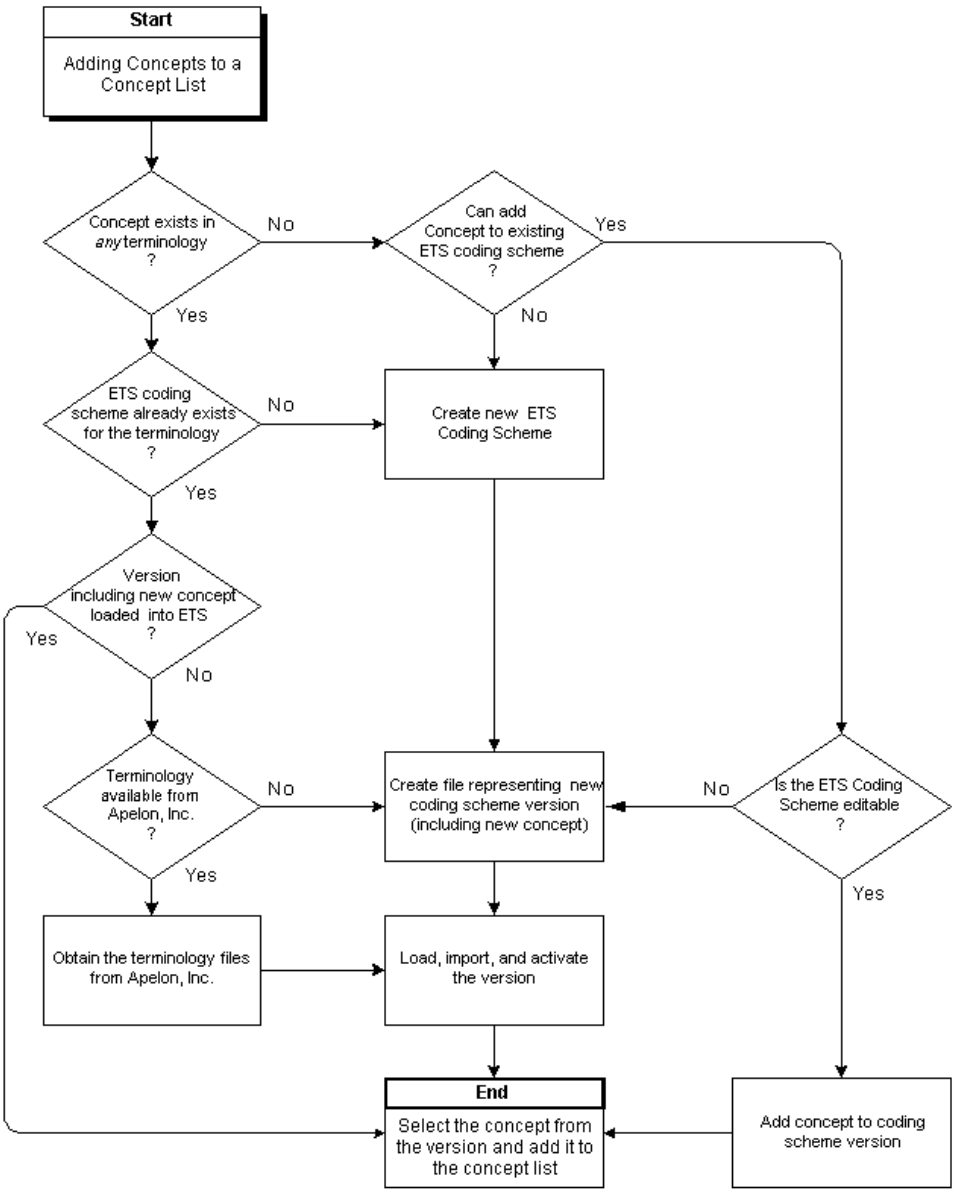
Adding Concepts to a Concept List

Concepts can be added to extensible concept lists. Concepts to be added to a concept list must be contained in a coding scheme version that has already been loaded into ETS. Determine if the concept exists in ETS and the coding scheme version in which it is contained.

Perform the steps described in this section to add concepts to a concept list.

Caution: Certain extensible concept lists are empty and must be populated with concepts before using their respective functionality.

Adding Concepts to a Concept List



Note:

A concept cannot be added under the following conditions: the concept is already active or pending in the selected list; the selected list is the parent of an additive child

specialization and the concept is already active or pending on the child; the selected list is a SYSTEM list; or the selected list is a restricted child specialization and the concept is not active or pending on the parent lists.

If any (but not all) concepts chosen for addition are not addable, a warning diagnostic lists the concepts that will not be added. You can elect to continue or return to list selection.

If all concepts selected for addition are not addable, the List Selection window is reloaded with an information box listing these concepts. To continue, select another list.

If any of these conditions exist, select a new membership code or activation date as appropriate, and click Next. You can alternatively click Back to return to the List Selection page, or click Cancel to exit the addition process.

Note: If the concept is being added to a restricted specialization, and the selected activation date would cause the concept's active period to exceed that of the corresponding concept in the parent list, an exception occurs; a dialog warns you that the concept has not been added.

See Also:

Oracle Healthcare Data Repository Javadoc

Caution: We strongly recommend that wherever possible, you only add concepts from the same terminology to a single concept list. Concept meanings can be sensitive to the context in which they are included in a terminology; mixing them with concepts from other terminologies may distort those meanings.

Specializing a Concept List

Concept lists can be specialized. A specialization of a concept list is a child concept list that initially inherits the active members of the parent list. It is a separate concept list, distinct from the parent list. Subsequent behavior of the specialization (a child concept list) with respect to the parent concept list depends upon the setting of its inheritance type:

- *Addition Inheritance:* Any concept added to the parent list is added to the child list.
- *Deletion Inheritance:* Any concept retired from the parent list is retired from the child list.
- *Restricted Inheritance:* A child list cannot contain any concept not contained in its parent list; before a concept is added to the child list it must first either exist in the parent list or be added to the parent list. A concept in a restricted child list also inherits certain changes to the activation and retirement dates of the corresponding concept in the parent list.

The inheritance types are not mutually exclusive. A restrictive list must also exhibit deletion inheritance. You can update a list's addition inheritance and deletion inheritance by turning them on or off, but you cannot update its restricted inheritance.

A specialization can be associated with a usage context, as can a concept local description. A usage context can in turn be associated with an organization. Accordingly, a concept list can have multiple specializations, each associated with a particular organization.

A concept list specialization is created in the same manner as any other concept list (Steps).

Values in a concept list specialization can be added or retired as for any concept list.

Subsetting a Concept List

It may be desirable to subset a concept list—using only a subset of a concept list's members, for UI display purposes, or for validating data to be stored by an application. The following sections describe how to subset a concept list:

Subsetting a User Concept List

A concept list of extensibility type User can be subsetted by retiring unwanted members from the list. A member is retired by updating its retirement date and time.

Note: This subsetting procedure does not apply to System or System extensible concept lists.

Subsetting a Concept List of any Extensibility Type

A concept list of any extensibility type (including user) can be subsetted using either of two additional procedures. These procedures are especially useful if the list to be subsetted is a System or System Extensible list, from which members cannot be retired (two methods):

Method 1: Using the Core Member Setting of List Members

The core set of members in the list can then be retrieved using the method:

getCoreSet

Checks of individual members of the list can be performed using the method:

isCoreMember

Method 2: Using a Specialization of the Concept List and Retiring Members

Reference:

Oracle Healthcare Data Repository Javadoc

Table 9–1 Service and Methods: Specializing Concept Lists

Level	Detail
Package	oracle.hsgbu.ets.base
Class	ConceptList
Methods	getChildConceptList

To subset a concept list of any extensibility type using Procedure-2, do the following:

Create a specialization of the concept list, and specify a usage context for the child list. You can then subset the child concept list, and you can use the subsetted list as required by your application.

Access the specialization using the method:

getChildconceptList

See Also: ■HDR Concept Lists Index for further information about HDR supported concept lists.

- HDR ConceptList Interface for further information about the ConceptList_Class

ETS Editable Terminologies

Editable terminologies let developers add contents to a coding scheme identified as *editable*. After creating an editable coding scheme, you can add concepts, descriptions and relationships to a version of this coding scheme. An editable coding scheme can have only one version.

Editable Coding Scheme

Using the HDR API, you can create an editable coding scheme by creating a new generic coding scheme with the editable attribute set to true. After the coding scheme is created, you can edit the original version, but a new version is not permitted.

Editable Terminology

An initial version of the editable coding scheme must be created and loaded before it is used. Use the generic terminology loader to load an editable coding scheme version.

Editable Coding Scheme

Editing concepts, descriptions, and relationships is limited to addition and removal of attributes, changing status, and changing a description's preferred status. Other changes are made by retiring a component and adding a new component in its place. For example, a description's text cannot be changed, but the description can be retired and a new description added to replace it. The new description can optionally be designated preferred.

Equivalence

As an editable terminology has only a single version, equivalence in an editable terminology must be intraversion – in other words, equivalence may only be declared between concepts in the same version. In the initial version load, equivalence may be declared using a change file, just as for any terminology being loaded with intraversion equivalence information. After the initial load, reassignments (introduction of a new concept that has the same meaning as an existing concept but has a different concept code than the existing concept) may be declared when a new concept is added. The new concept's code may be declared, using the relevant API, to be a reassignment from an existing concept's code.

No reuse of codes (introduction of a concept whose concept code is the same as an existing code, but where the concept has a different meaning than the existing code represents) is permitted in an editable terminology.

See Also:

- Implementing Interterminology and Intraterminology Equivalence
- Implementing Interterminology Mapping Using Cross Maps

Reference

Oracle Healthcare Data Repository API Documentation

The following table provided information about the ETSAuthoringService interface used to implement editable terminologies:

Oracle Healthcare Data Repository Javadoc

Table 9–2 Service and Methods: Editable Terminologies

Level	Detail
Package	*oracle.hsgbu.ets.authoring*

Table 9–2 (Cont.) Service and Methods: Editable Terminologies

Interface	*ETSAuthoringService*
Methods	■ *addAttributes* ■ *addConcepts* ■ *addConceptDescriptions* ■ *addRelationships* ■ *changeStatus* ■ *removeAttributes*

Refer to the following sections to edit the components of an Editable Coding Scheme:

- Adding Components
- Changing Component Status
- Adding and Removing Component Attributes

Note: These changes are exclusive; any other changes to concepts, descriptions, and relationships can only be made by retiring the component and adding a new one.

Adding Components

Use the *addConcepts*, *addConceptDescriptions*, and *addRelationships* methods to add concepts, descriptions, relationships and attributes to an editable coding scheme. *Candidate* components are created first and passed to the *add* methods.

Changing Component Status

Components such as concepts, relationships and descriptions of an editable coding scheme cannot be edited or removed directly. To modify a component, it must be retired and replaced. A component can be retired or made active by changing the status flag associated with the component. Use the changeStatus method to change component status.

Adding and Removing Attributes

Use the *addAttributes* and *removeAttributes* methods to add and remove attributes, respectively.

The code samples below help you to:

- Create an Editable Coding Scheme
- Create a Coding Scheme Version and Making it the Default
- Create a Concept
- Create a Relationship between Two Concepts

> **Note:** The RelationshipTypecode of the IS_A concept used in Example 9-4 should be set to CandidateConcept.REL._TYPE_YES.

Example 9–1 Create an Editable Coding Scheme

```
etsAdministrationService.createCodingScheme
(codingSchemeName, codingSchemeDescription,CodingScheme.CODINGSCHEME_MODEL_
GENERIC, oid , true);
```

Example 9–2 Create a Coding Scheme Version and Making it the Default

```
CodingSchemeVersion csv =
etsAuthoringService.createCodingSchemeVersion(codingSchemeVersionName,
codingSchemeVersionDescription , codingScheme);etsAdmin.makeDefaultVersion(csv);
```

Example 9–3 Create a Concept

```
CandidateConcept[] candConcepts = new CandidateConcept[1];
candConcepts[0] = etsAuthoringService.newCandidateConcept();
CandidateConceptDescription[] candDescs = new CandidateConceptDescription[1];

candDescs[0] = etsAuthoringService.newCandidateConceptDescription();
candDescs[0].setStatusCode('A');
candDescs[0].setTermText(conceptDescription  );
candDescs[0].setPreferredDescription(true);
candConcepts[0].setCandidateDescriptions(candDescs);


candConcepts[0].setConceptCode(conceptCode);
candConcepts[0].setStatusCode('A');
candConcepts[0].setRelationshipType(relationshipType);

etsAuthoringService.addConcepts(csv, candConcepts, false);
```

Example 9–4 Create a Relationship between Two Concepts

```
String sourceConceptId = csv.getConceptByConceptCode(conceptCode).getETSID();

Concept targetConcept = csv.getConceptByConceptCode("TargetConcept");
String targetConceptId = targetConcept.getETSID();

Concept relationshipConcept = csv.getConceptByConceptCode( "IS_A");
String relationshipConceptId = relationshipConcept.getETSID();

CandidateRelationship[] candRelation = new CandidateRelationship[1];
candRelation[0] = etsAuthoringService.newCandidateRelationship();

candRelation[0].setSourceConceptID(sourceConceptId);
candRelation[0].setTargetConceptID(targetConceptId);
candRelation[0].setRelationshipTypeConceptID(relationshipConceptId);
candRelation[0].setStatusCode('A');

etsAuthoringService.addRelationships(csv, candRelation, false);
```

See Also: *Oracle Healthcare Data Repository API Documenation* for more information about using ETS APIs .

ETS Classifications

ETS Classifications are containers for grouping existing ETS concepts from different coding scheme and versions. Classifications are intended for large categorizations of concepts, while concept lists are intended for smaller sets for the purposes of validation or display. ETS Classifications provide the following features:

- Classifications describe a set of concepts that are all part of the same category. For example, the classification Antibiotics might include concepts for penicillin and telithromycin, among others.
- Classifications can be arranged in a hierarchy.

- Tests for containment in a classification search down the levels of the hierarchy.
- Classifications can be created and populated through an API or through creating and loading text files.
- Classification contents incorporate equivalence.
- Concepts added to an ETS Classification retain their equivalence information and characteristics.
- ETS Classifications are themselves ETS Concepts--components of a special, predefined editable terminology called ETSClassifications. Accordingly the following applies:
  - A classification has a concept identifier.
  - A classification can have multiple descriptions, including local descriptions.
  - A classification's local descriptions can be associated with usage contexts.
- ETS Classifications can be defined using the API or using the HDR Loader job and HDR Importer job.
- Classification contents are created with declarations that can leverage ontological relationships already present in many terminologies.
- All members of a classification implicitly include their semantic equivalents.
- Classifications are computed asynchronously by a new feature of the HDR Maintenance Job, supporting fast runtime response to classification queries.

Every classification created is backed by an ETS concept, and inherits all functionality that ETS concepts provide. Classification concepts are created on demand in a special coding scheme.

The following table summarizes the principal interfaces referenced by this section:

Table 9–3 Service and Methods: ETS Classifications

Level	Detail
Package	*oracle.hsgbu.ets.base*
Class	*ETSAdministrationService*
Methods	■ *addConceptToConceptList* ■ *addDeclarationToClassification* ■ *createClassification* ■ *createCodingScheme* ■ *createConceptList* ■ *removeDeclarationFromClassification*
Class	*ETSService*
Methods	■ *findClassificationByCode* ■ *findcodingScheme* ■ *findCodingSchemeVersion* ■ *findConcept* ■ *findConcepts* ■ *isEquivalent*
Class	*Classification*

Table 9–3 (Cont.) Service and Methods: ETS Classifications

Methods	■ *contains* ■ *findChildClassifications* ■ *findConceptsInClassification*

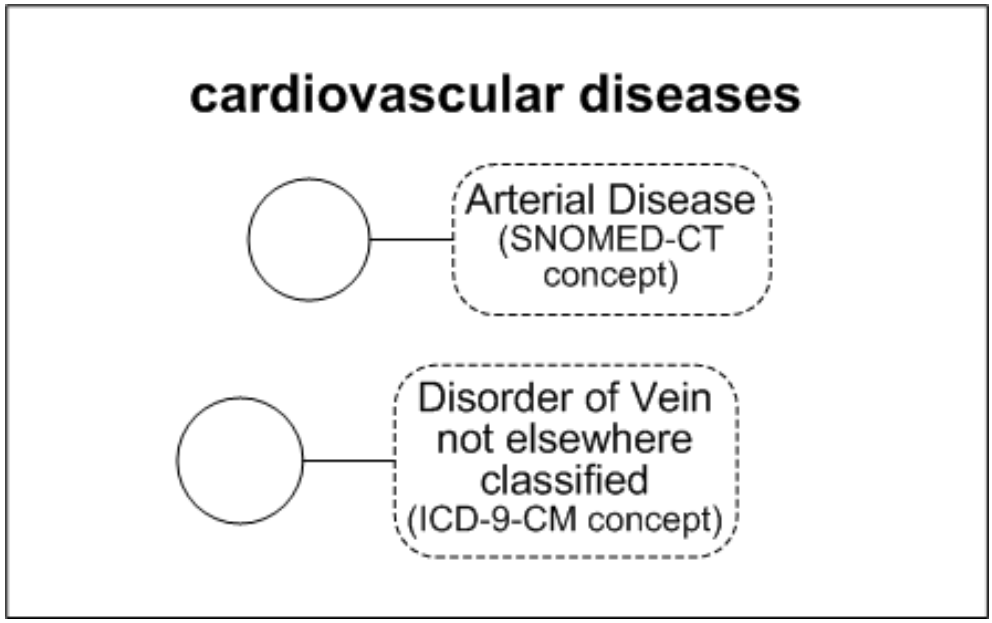
Large amounts of healthcare data are difficult to use unless they are well organized. Creating classifications is the most common means of organizing healthcare data. As institutions generally use a combination of (standard and local) terminologies, classifications need to incorporate concepts from different coding schemes and versions.

ETS Classifications provide a mechanism for grouping concepts from different coding schemes and versions, and arranging the groups in hierarchies navigable by the ETS API. ETS Classifications facilitate:

- Viewing a large number of concepts
- Selection of concepts
- Class-based query of information

For example, a classification called *cardiovascular diseases* could be created and populated with concepts that represent different cardiovascular diseases from different terminologies. The following chart shows this classification:

Figure 9–3 Simple ETS Classification

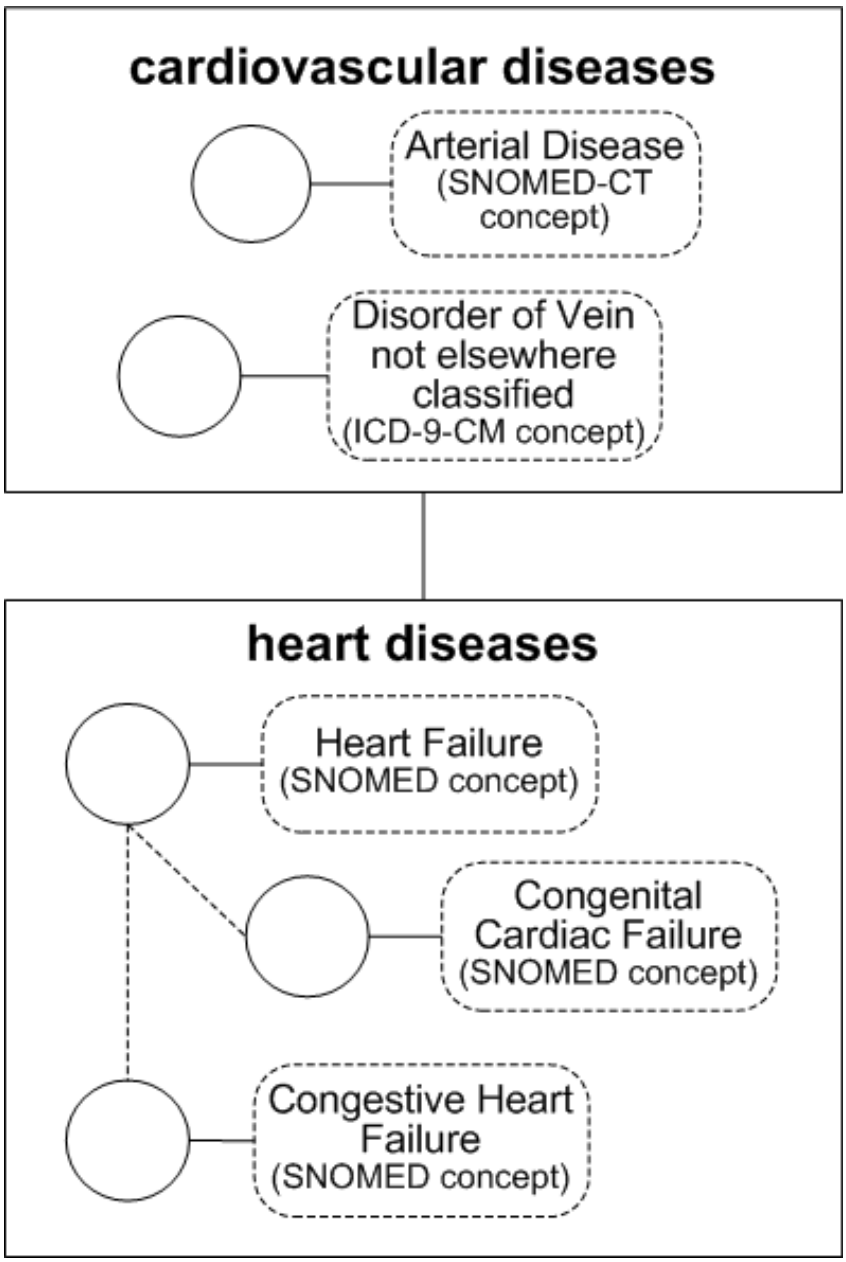


ETS Classifications are internally represented as concepts in an editable generic coding scheme called ETSClassification. In this example, the classification, cardiovascular diseases is stored as a concept in the coding scheme ETSClassification.

But you could also create another classification such as heart diseases and make it a child classification of the cardiovascular diseases classification. Thereafter, any concept in the heart diseases classification, such as congestive heart failure, would be

considered by the Classifications interface to be implicitly contained in the cardiovascular diseases classification.

Figure 9–4 Parent and Child ETS Classification



ETS Classifications are basically containers of ETS concepts. In this chart, the concepts in each classification node could derive from different coding schemes.

ETS classification interfaces are oblivious to relationships between the classification concepts in their native terminologies. For example, in this chart, congestive heart failure and congenital cardiac failure are children of heart failure in their native terminology, but the heart disease ETS Classification views its contents as a flat list of concepts. There is no classification interface that is aware of relationships between the classification concepts in their native terminologies. However, those relationships can

be navigated by querying the individual concepts, using the ETS generic terminology interface or a terminology-specific interface.

Classifications are themselves ETS Concepts – components of a special, pre-defined editable terminology called ETSClassifications. Creating a new classification actually is creating a new concept in the ETSClassifications terminology. Since classifications are concepts:

- A classification may have multiple descriptions, one of which is designated preferred
- Local descriptions may be associated with usage contexts
- A classification has an ETS ID

Classifications can be Linked Hierarchically

Classifications may be linked in strict or multiple hierarchies (a classification may have multiple parent classifications) to form a network of linked classifications forming an acyclic graph.

Testing Containment

ETS classifications support testing of containment across levels of a hierarchy. For example, given the hierarchy in the chart Sample ETS Classification, if the concept congestive heart failure is in the heart disease classification, and the heart disease classification is a child of the cardiovascular diseases classification, you can test if congestive heart failure is a cardiovascular disease, and the answer will be Yes.

Equivalence is incorporated into Classifications.

The concepts contained in an ETS Classification retain their equivalence information. For example, what is actually considered as included in cardiovascular diseases are groups of concepts that are equivalent to heart failure, congenital cardiac failure, congestive heart failure, and arterial anuerysm. Consequently, if concept X is equivalent to congestive heart failure, and the question “Does ‘cardiovascular diseases’ contain X?” is asked, the answer will be “yes”.

Creating and Populating Classifications

Classifications are created by generating a hierarchical network of classification nodes and populating those nodes with individual ETS concepts. An individual classification node may contain concepts from multiple terminologies.

A single concept can be used in multiple classifications, including multiple sub-classifications of the same parent classification, but it cannot appear more than once in the same sub-classification. However, equivalent concepts can be inserted into the same sub-classification. As for all ETS concepts, concepts in a classification can have multiple text representations.

Classifications may be created and populated by two methods:

- Specifying classification data in files, and loading the files using HDR Terminology Jobs
- Using ETS APIs directly

A new classification will have a pending state when it is created and loaded. It will become effective only when the HDR Maintenance job is run. Until then, the new classification will be unusable. This enables new classifications to be created and readied for use without requiring a downtime.

Whenever changes are made that could affect classification contents (declarations are added or removed, a version of a terminology that contains classification concepts is loaded, or a mapping involving a version that contains classification concepts is loaded), the classification moves from the active state to a dirty state. In this state the classification in its former active state is still usable. The effects of the changes that placed the classification into the dirty state will not be usable until the classification is moved from the dirty state to the active state. Classifications are moved from the dirty state to the active state by running the HDR Maintenance Job.

See Also:

Classification contents are defined declaratively. This enables a *short-cut* for adding concepts declared a terminology to be children of another concept. For example, the concepts *heart failure*, *congenital cardiac failure*, and *congestive heart failure* could have been added to *heart diseases* by a single statement*(add heart failure and its descendents to heart diseases).* This adds *heart failure* and its children as defined within its native terminology. A declaration can also add a concept and only those concepts deemed to be direct children in its native terminology, add a concept's descendents but not the concept itself, or add a concept's direct children but not the concept itself. The various insertion choices are called *insert options*.

See Also

Oracle Healthcare Data Repository Javadoc

A declaration with an insert option of *concept only* adds a single concept ((the concept with none of its children). The contents of a classification amount to a series of declarations. Classification contents are augmented or reduced by adding or removing declarations.

To retrieve a concept's children when implementing a declaration, the ETS classification build process must know which relationships in the concept's native terminology represent parent-child relationships. Accordingly, for each core terminology, HDR has identified certain relationships as defining parent-child associations. The ETS classification build process queries for such relationships when called upon to find a concept's children. For generic terminologies, the build process queries for relationships in which the relationship type concept is identified as type *IS_PARENT_OF* or *IS_CHILD_OF.*

See Also

- Documentation at the Apelon, Inc. web site for descriptions of the treatment of specific terminologies.
- /ets/hdr-ets-1.0.0-SNAPSHOT/db/execute/readme /Terminology_File_Formats_Generic.txt from hdr-1.0.0-SNAPSHOT.zip

Note:

Each ETS concept defined in a generic terminology must be identified as being valid or invalid for use as a relationship type, and is indicated by the *RELATIONSHIPTYPEFLAG* column in the Concepts fie. This flag can contain one of the following values (Table):

Table 9–4 Concept File: RELATIONSHIPTYPEFLAG Legal Values

Value	Description
N	The concept cannot be used as a relationship type.
Y	The concept can be used as a relationship type.

Table 9–4 (Cont.) Concept File: RELATIONSHIPTYPEFLAG Legal Values

IS_PARENT_OF	The concept can be used as a relationship type, and the type indicates that the source concept is the *parent* of the target.
IS_CHILD_OF	The concept can be used as a relationship type, and the type indicates that the source concept is the *child* of the target.

Those concepts designated as valid relationship types in the concepts file can subsequently be used in the relationships file in the *RELATIONSHIPTYPECONCEPTCODE* column.

Use the following HDR interfaces to define and use ETS classifications:

- *ETSAdministrationService*
- *ETSService*
- *Classification*

Creating a Classification

To create and populate a classification via the loader, perform the following steps:

Steps:

1. Create a Classifications file. This file lists the classifications to be created and their properties.
2. Create a Classifications descriptions file: This file lists the descriptions to be associated with the classifications and their properties. Just as multiple descriptions for a concept can be listed in a terminology descriptions file, multiple descriptions can be created for a classification.
3. Create a Classifications declarations file. This file lists the declarations (each declaration consisting of a concept and an insert option) that will be added to classifications. The classifications referenced in this file can be new classifications listed in the classifications file, or pre-existing classifications.

   Note: Declarations must be removed through an API call; they cannot be removed through the loader.
4. Create a control file specifying the Classifications, Classification Descriptions, and Classification Declarations files.

   See Also:

   /ets/hdr-ets-1.0.0-SNAPSHOT/db/execute/readme/Terminology_File_Formats_Classifications.txt from hdr-1.0.0-SNAPSHOT.zip for file formats.
5. Load the classification. The procedure for loading a classification is the same as that for loading a Coding Scheme Version. Enter *ETSClassifications* in the Coding Scheme Name field on the Parameters page. Enter any text for the Coding Scheme Version Name (this field must contain text, but the actual text is ignored by the loader).
6. Import the classification. The procedure for importing a classification is the same that for importing a Coding Scheme Version.
7. Run the Healthcare HDR Maintenance Job to build the classification.

Building a Classification with the HDR Maintenance Job

To build a classification run the HDR Maintenance Job with CLASSIFICATIONS in run mode.

Note: Run the HDR Maintenance job in the *full* mode whenever a significant amount of new classification data is created (including the first time classifications are created and populated in ETS.

Updating Published Coding Scheme Versions

After a coding scheme version is imported (published), you can update its properties (description, status, and default status) through the ETS API.

Running the HDR Maintenance Job

The HDR maintenance Job performs several database tasks. These tasks include:

- Maintaining the ETS stage tables. For example, cleanup of incomplete or obsolete data in the staged tables.
- Maintaining the ETS data in the active tables. For example, cleanup of failed imports in the active tables.
- Building classifications by processing data for classifications in the pending or dirty state.
- Building and synchronizing intermedia indexes.
- Gathering statistics for the Cost-based Optimizer.
- Ensuring that there is an entry in the language mapping table for each combination of coding scheme version and installed languages.

The maintenance job must be run in either *FULL* mode or *CLASSIFICATIONS* mode to move a classification from the 'dirty' or pending state to the active state. In general, it is a good idea to run the maintenance job periodically to keep ETS running optimally.

Note: You should run the HDR Maintenance Job in the *FULL* mode:

- Each time you apply a patch to ETS.
- Whenever a significant amount of new classification data is created (including the first time classifications are created and populated in ETS).

Scheduling the Maintenance Job

Note: In job arguments, select the desired Run Mode. The available choices are:

- *CLASSIFICATIONS:* Builds classifications by matching definitions of classifications in the *pending* or *dirty* state. Classifications that have been created since the last time the maintenance job was run in CLASSIFICATIONS or FULL mode will be in the pending state. Classifications that have been modified since the last time the maintenance job was run in CLASSIFICATIONS or FULL mode will be in the dirty state. Successfully processed classifications obtain the active state.
- *CLEAN_ACTIVE:* Performs maintenance on the ETS data in the active tables. Removes data from failed imports in the active tables, and rebuilds the intermedia indexes, if necessary. Ensures that there is an entry in the language mapping table for each combination of coding scheme version and installed languages.
- *CLEAN_STAGE:* Performs maintenance on ETS stage tables. Removes incomplete or obsolete data from the staged tables, and rebuilds the intermedia indexes, if necessary.
- *DEFAULT:* Performs maintenance on ETS stage, active, and language mapping tables, but does not build classifications. This mode is composed of CLEAN_ ACTIVE and CLEAN_STAGE modes of the maintenance job.

- *FULL:* Performs all operations, including maintenance of stage, active and language mapping tables, and building of classifications. This mode is composed of CLEAN_ACTIVE, CLEAN_STAGE, and CLASSIFICATION modes of the maintenance job.
- *TRUNCATE_STAGE:* Removes all staged contents, regardless of their status. This mode is faster than CLEAN_STAGE for large datasets.

  Caution: The TRUNCATE_STAGE option can cause data that could have been used by the importer to be lost.

See Also:

- *Oracle Applications System Administrator's Guide*

See Also:
- HDR ConceptList Interface for more information about the ETS ConceptList Class.
- HDR Concept List Index for more information about ETS supported concept lists.

ETS Equivalence

ETS manages changes to terminology data over time by maintaining complete copies of each version of a terminology. ETS equivalence provides a mechanism to determine all of the instances of concepts that have the same semantic meaning, both within the history of a coding scheme and across coding schemes.

Equivalence may be Intra-terminology or Inter-terminology. Intra-terminology equivalence is defined between concepts from the same terminology. For example, the concept Cholera is represented in ICD-9-CM 2002 and in ICD-9-CM 2003 by the same concept code [001_CHOLERA]. Because both concepts have the same meaning, they can be treated as equivalents---ETS treats them as equivalent by default, as they belong to the same coding scheme and have the same concept code.

ETS allows concepts from two different terminologies to be defined as semantic equivalents. Equivalence between concepts from different terminologies, or Inter-terminology equivalence, is defined using cross maps. For example, the concept for the disease Cholera in the ICD-9-CM terminology (2002 version), and the concept representing the same disease in the SNOMED-CT terminology (2002 version), can be defined as equivalents using a cross map.

The information used to determine equivalence is provided by terminology vendors, either as an integral part of the terminology data, or as a separate change file. There are two types of equivalences in ETS:

- *Intra-terminology equivalence* captures exact, unambiguous equivalence between concepts in the same terminology. Declaring two concepts semantically equivalent indicates that their meaning is identical.
- *Inter-terminology equivalence* uses mappings to capture explicit equivalence between concepts from different terminologies.

Intraterminology Equivalence

Intraterminology equivalence deals with identical concepts (those with the same meaning) within a single terminology. When a new version of a terminology is released, there may be several changes to the representation and meaning of concepts when compared to the previous version. Because there is no way for ETS to automatically determine these changes, by default it treats concepts from the previous

and new versions as distinct and unrelated. However, it is possible to explicitly indicate the changes that have occurred between a prior version and a new version in a change file that is loaded with the new version. Using this information, Intraterminology equivalence services can determine whether two concepts from the previous version and the new version have the same meaning.

Given a concept from a version of a terminology, ETS can retrieve equivalent concepts from all contiguous versions that have change files loaded. Given two concepts from different versions of a terminology, ETS can verify if they are equivalent, provided that the more recent version and all the intermediate versions have change files loaded.

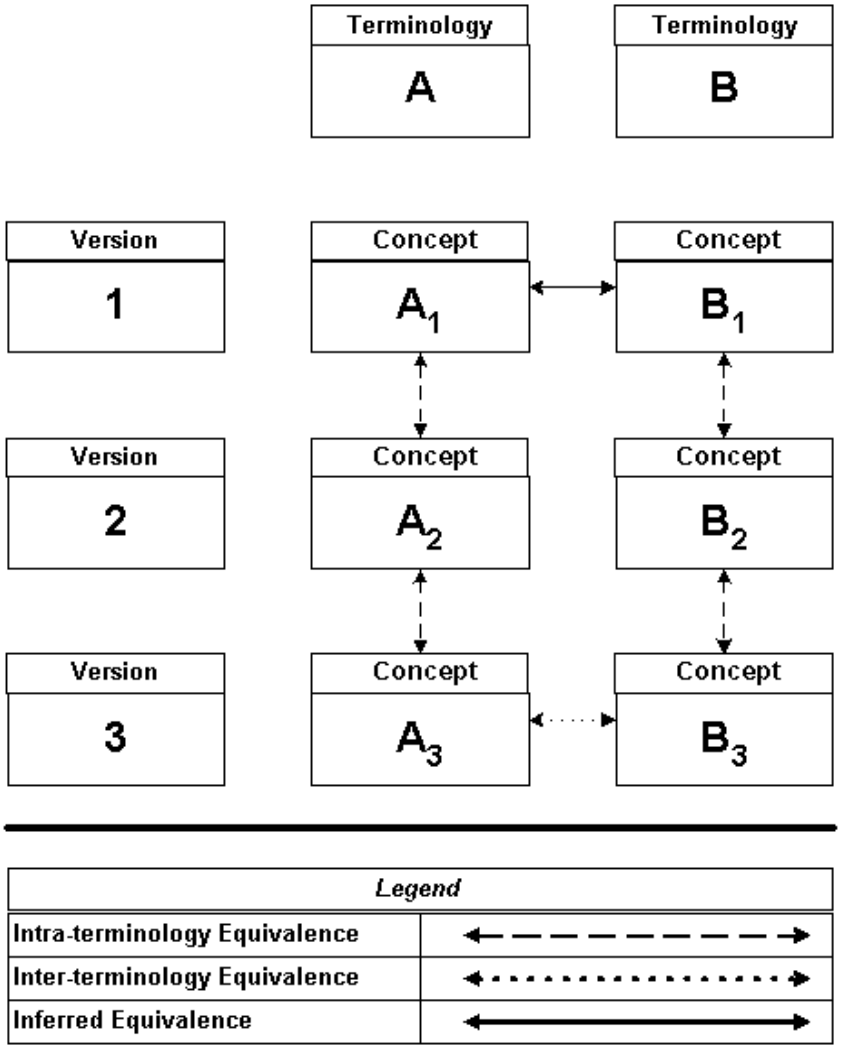
Interterminology Equivalence

Interterminology equivalence deals with identical concepts (those with the same meaning) from different terminologies. Concepts from two different terminologies can vary widely in their granularity and coverage of a domain. Because there is no way for ETS to automatically determine these differences, by default it treats concepts from the two terminologies as distinct and unrelated. However, it is possible to explicitly indicate equivalence between concepts from two versions of different terminologies in the form of an Interterminology Mapping. Using this information, Interterminology equivalence services can determine whether two concepts from different terminologies have the same meaning.

Combining Intraterminology and Interterminology Equivalence

Equivalence between concepts is a transitive relationship. In the following chart, if Concept A1 is equivalent to Concept A2, and Concept A2 is equivalent to Concept A3, it can be inferred that Concept A1 is equivalent to Concept A3. Consistent with this logic, Concept Equivalence services in ETS can determine if concepts from two terminologies are equivalent—provided that an inter terminology mapping exists between versions of the two terminologies, and, change files have been loaded for all versions.

Concept Equivalence Model



In this chart, ETS transitively combines intraterminology equivalence and interterminology equivalence information to infer that Concept A1 is equivalent to Concept B1.

Example 9–5 Using ETS Equivalence

```
import java.util.Properties;
import oracle.hsgbu.ets.base.*;
import oracle.hsgbu.ets.authoring.*;
import oracle.hsgbu.ets.fwk.servicelocator.common;

public class ProgrammerGuideTest2 {

public static void main(String argv[]) throws Exception {
 /* Add a concept with concept code "REPC_TE000005" into a concept list,
 which has a name of "CL_TEL_USE" and belongs to a group "HTB_RIM".
 The steps will be:

 1. find the concept and the concept list in the system;
 2. check to see if the concept (and its equivalent concept) is already in the
concept list;
```

```
 3. if not, add the concept into the concept list.*/

addConceptToConceptList("REPC_TE000005","CL_TEL_USE","HTB_RIM");
 }

public static void addConceptToConceptList(String conceptCode,String
conceptListName, String groupName) throws Exception{

  Properties props = new Properties();
  props.setProperty(ServiceLocator.CLIENT_MODE,
   ServiceLocator.REMOTE); //Specify Client mode: Local or Remote
  ServiceLocator slocator = ServiceLocator.getInstance(props);
  slocator.login("sysadmin","sysadmin");
  ETSService ets = slocator.getEtsService();
  //get the ETSService through ServiceLocator
  ETSAdministrationService etsAdmin =
  ets.getAdministrationServiceInstance();
  //get the getAdministrationService
  //1. find the CodingScheme and its default version by calling
   ets.getCodingScheme(String) and cs.getDefaultVersion();

  // if using this example code in your own code, replace "My Terminology"
   with the terminology name in which you are interested
  CodingSchemeVersion csv = ets.getCodingScheme("My
Terminology").getDefaultVersion();

  //2. find the concept by calling csv.getConceptByConceptCode(String)
   Concept con = csv.getConceptByConceptCode(conceptCode);

  //3.find the concept list by calling ets.getConceptList(String, String)
   ConceptList cl = ets.getConceptList(conceptListName, groupName);

  //4. validation: check to see if the concept list contains this concept
   by calling cl.contains(Concept,int)

  //5. if not, add the concept to the concept list by calling
   etsAdmin.addConceptToConceptList(Concept,String,ConceptList)
   if (!cl.contains(con,Concept.EQUIVALENCE_TYPE_MAPPING))
    {
    etsAdmin.addConceptToConceptList(con,conceptCode,cl);
    // here we use the concept code as the membership code, but you may use
     something else
    }
  }
}
```

Note: ETS does not support authoring of equivalence information. Equivalence must be provided to ETS in one of the specified ETS file formats.

Intra-terminology equivalence information is provided to ETS as a “change file”, when loading a new version of a terminology. Change files identify reuse of codes (codes that represent different meanings than the previous version), and reassignment of codes (meanings that are now represented with different codes).

Note: ETS does not support the authoring of change files.

HDR Terminology Jobs

ETS includes a set of DB SCHEDULER jobs that let you load and import terminology data files, perform routine database maintenance, and delete unwanted map sets. You can run these jobs through the scripts provided.

- **Loader:** Running the HDR Loader Job is the first step in the process of populating terminology data into the ETS active repository. The loader job is used to load terminology data from flat, textual files into terminology staging tables. The loader interprets or converts data from its native format into a format that aligns with the internal ETS representations, and it also validates the format of the files being loaded.
- **Importer:** The HDR Importer Job takes data from a set of stage tables (populated by the loader), runs a series of validation checks on it, and publishes it to the ETS active tables.
- **Maintenance:** The HDR Maintenance Job cleanses data from the stage and active tables, gathers statistics for the stage and active tables, rebuilds intermedia indexes, and builds classifications.

ETS Multiple Language Support (MLS)

ETS lets concepts to be described in any language. ETS languages are derived by appending the ISO Language code (in UPPER case) to the ISO Territory code (in UPPER case). The territory represents either a geographic region or a language dialect. For example, ENGB represents the English language used in Great Britain.

You can use one of the following methods to create a description in a different language than the base language:

- Use the HDR Loader Job
- Create local descriptions

This is achieved by locale enabling a subset of APIs. These locale enabled APIs permit retrieval of concepts and creation of descriptions in specific locales. The non-locale enabled APIs return descriptions in the base language. However, you can only create editable terminologies in the base language.

Mappings exist for every coding scheme version, ISO Language code and ISO Territory code. This resolves situations where a description does not exist for the installed language. For example, if Simplified Chinese is mapped to American English, American English descriptions are returned when requesting descriptions in Simplified Chinese. Local descriptions are an exception to this rule. Simplified Chinese local descriptions are returned when requesting descriptions in Simplified Chinese.

- Terminology-specified concept descriptions of non-editable terminologies.
- Locally-specified concept descriptions of terminologies (editable and non-editable) and Classifications.

MLS in ETS lets you load Coding Scheme versions with Concept descriptions in multiple languages. Each Coding Scheme version can support Concept descriptions in multiple languages, but every Concept in a Coding Scheme version must be supplied with a terminology-preferred description in the languages supported by that version.

You can create local descriptions in multiple languages. Concept descriptions (both terminology-specified and local) based on a language can be obtained by calling methods that accept a language.

Creating Local Descriptions

You can specify local descriptions for any ETS concept. These descriptions can be used in place of terminology-specified descriptions for display purposes. You can assign a single usage context to each local description.

The usage context for each local description must be unique; no two local concept descriptions share the same usage context.

Note: Local descriptions of a concept can be created for any language, not just for languages loaded with the coding scheme version.

Use ETS API to create local description. You can create local descriptions for retired or active concepts, but the typical procedure is to create a description for concepts in the active default version of the terminology.

Note:

- The terminology-specified Concept descriptions of editable terminologies and Classifications are created in the base language of the HDR installation.
- ETS will not translate terminology content - whether seeded or loaded.
- ETS does not perform any validation to ascertain whether a description is actually in the language that it claims to be in. You can load pseudo translated text as concept descriptions for a supported language along with the real description text. For example, you can load the data given in the following table as Concept descriptions into ETS without getting any errors:

Table 9–5 Sample Concept Descriptions

Version ID	Concept ID	Description Text	Language
v20070101	A25.66	Fracture of Femur	American English
v20070101	A25.66	Fracture of Femur	Korean
v20070101	A25.66	Fracture of Femur	Spanish
v20070101	A25.66	Fracture of Femur	Japanese
v20070101	A25.66	Fracture of Femur	German

Understanding Language (Locale) Mappings

Mappings let ETS return descriptions even though the terminology may not contain descriptions in the requested language. For example, the mappings can be set up so that if British English descriptions are requested, American English descriptions are returned. Every coding scheme version has a set of mappings that let the system determine which language to return.

Mappings can be created through one of the following mechanisms:

- By migration (upgrading from a prior version).
- By the HDR Loader Job.
- By using the HDR Maintenance Job.

See Also: ■The HDR Loader Job Readme files for information about loading terminologies, including language mapping setup.

You can use the HDR Maintenance Job to create missing mappings. For example, you can run the maintenance program after installing a language to create mappings from the new language to the base language.

Scenario

Assume an installation in London, England with base (or default) language in American English.

Assume a terminology that includes French and American English descriptions but not British English. Users requesting British English are given an American English description because no British English description exists. This is represented in the following table:

Mappings

Requested Locale	Description Locale
en_GB	en_US
fr_FR	fr_FR
en_US	en_US

When a description is requested in a particular locale, ETS locates this mapping and determines which language to return. In this scenario:

- If a British English description is requested, ETS returns an American English description.
- If a French description is requested, ETS returns a French description.
- If an American English description is requested, ETS returns an American English description.

If a description is requested and the locale is not specified (non locale enabled APIs), ETS assumes that the requested locale is in the base language and uses the mapping to determine which language the description should return. In this scenario, the American English description is returned because the base language (American English) maps to American English.

To provide British English descriptions, create British English local descriptions. When a description is requested in British English, ETS returns the British English local description and ignores the mapping.

Locale Enabled APIs

- getLocale
- findConcepts
- createLocalConceptDescription
- getDescription
- getLocallyPreferredDescription
- getTerminologyPreferredDescription

- getAllDescriptions

A code sample below can help you to:

- Use a Locale-Enabled API

Mixing calls to Locale enabled and non-Locale enabled APIs is not recommended. For example, calling getLocallyPreferredDescription() without a Locale and then calling it with a Locale.

There are two constructors for a java.util.Locale: one that accepts a language code and a country code, and one that also accepts a variant code. Locale variants are not supported by ETS; parameter constructor 3 should never be used to construct a parameter for the ETS API. Both the ISO 639 language code and the ISO 3166 country code are mandatory for a java.util.Locale that is passed to ETS. For example, it is valid to pass new Locale("en", "US") for American English or Locale("en", "GB") for British English, but not Locale("en", null) or Locale("en", "")

Errors

Locale enabled APIs may throw the following exceptions for MLS specific reasons.

- An ETSBadParameterException is thrown when the Locale is not valid. The following Locales are considered invalid:
  - Locales where the language code or territory code is missing.
  - Locales that contain a variant.
- An ETSDataException is thrown when the language mapping is missing.

Example 9–6 Use a Locale-Enabled API

```
ServiceLocator slocator = ...;
  ETSService etsService = slocator.getEtsService();
  Locale locale = new Locale("en","US");

  String codingSchemeName = "A Terminology";
  CodingScheme codingScheme = etsService.getCodingScheme(codingSchemeName);
  CodingSchemeVersion csv = codingScheme.getDefaultVersion();
  String conceptCode = "A Concept Code";
  Concept concept = csv.getConceptByConceptCode(conceptCode);

  // get descriptions in the locale of the current session
  ConceptDescription[] descriptions = concept.getAllDescriptions(locale);

  System.out.println("Description of conceptCode: " + conceptCode + " is " +
descriptions[0].getTerm());
```

See Also:

- Implementing Master Catalog
- *HDR Concept Lists Index, Oracle Healthcare Data Repository Javadoc (click HDR Concept Lists link at bottom of Javadoc pages), for seeded concept lists and values.*

See Also:
- Character Sets & Conversion - Frequently Asked Questions.
- Unicode character sets in the Oracle database.
- IETF RFC 3066: The terminology data files that are processed by the ETS Loader validate against RFC 3066 (2-letter language—hyphen—2-letter territory). This representation is the combination of ISO 639-1 alpha-2 language codes and ISO 3166-1 alpha-2 territory codes.
- ISO 639-1 (alpha-2 codes)
- ISO 3166-1 (territory codes)
- java.util.Locale

10

HDR Messaging Services

HDR Inbound Message Processor

The Inbound Message Processor (IMP) lets HDR receive inbound HL7 version 3.0 messages, compliant with HDR messaging schema, from sending systems and persist information to the HDR repository.

IMP provides a processMessage API to persist a HL7 V3 message, which returns a Result object containing the acknowledgement. IMP provides an invalidateCache API that will invalidate the configuration cache.

To persist a message, user must configure sender, sender interaction, and side effect for the message and create the receiver of the message as an organization in HDR. You can use Messaging Configuration Service or IMP Configuration Administration Service to do IMP configuration.

IMP extracts Sender, Receiver, and Interaction Id available in the message, and picks up the associated side effect configuration. Based on the side effect configuration, IMP sets the Reference Modifier on RIM Objects of the message. Based on which, RIM objects available in the message is created, updated, overlaid or ignored. There are certain rules that influence the value of Reference Modifier to be set on RIM Objects, which is described in the section Side Effect Configuration Rules.

After persisting the message, IMP returns a successful acknowledgment (AA), if the message is persisted successfully. IMP rejects a message with an Application Error (AE) typecode, if the content or format of the message is incorrect (such as identified object validation failed, mandatory code attribute missing). IMP rejects a message with an Application Reject (AR) typecode, if message processing fails for any reason unrelated to the content or format of the message (such as system down, internal error, and so on).

Messaging Schema

HDR includes messaging schemas for all supported message types. Messaging schema includes the following for each message type:

- Schema (XSD Files) for the Payload of the Message Type

- Composite Message Schema (XSD File) for each Interaction ID of the Message Type
- Model Interchange Files (MIF files) for the Payload of the Message Type

In addition, messaging schema contains a common Vocabulary schema and data type schema for all message types.

The Composite Message Schema (CMS) has three parts: Message Wrapper, Control Act Wrapper, and Payload Reference. If there are three Interaction IDs seeded for the same Payload, there will be three composite message schemas; one for each Interaction ID and all of them will refer to the same Payload.

For samples, refer to the schemas for Lab Result available at the following locations:

- Payload Schema for a Message Type (Lab Result)

  <hdr_domain_home>/config/hdr/message/defs/rim214101/schemas/POLB_MT004000HT01.xsd

- Composite Message Schema for Interaction Ids POLB_IN004003, POLB_IN004004 (Lab Result)

  <hdr_domain_home>/config/hdr/message/defs/rim214101/schemas/POLB_IN004003.xsd

  <hdr_domain_home>/config/hdr/message/defs/rim214101/schemas/POLB_IN004004.xsd

- Common Data Type Schemas

  <hdr_domain_home>/config/hdr/message/defs/rim214101/coreschemas/datatypes.xsd

  <hdr_domain_home>/config/hdr/message/defs/rim214101/coreschemas/datatypes-base.xsd

- Common Vocabulary Schemas

  <hdr_domain_home>/config/hdr/message/defs/rim214101/coreschemas/datatypes.xsd

  <hdr_domain_home>/config/hdr/message/defs/rim214101/coreschemas/datatypes-base.xsd

For more information on message types supported, refer to the *Oracle Healthcare Data Repository HL7 Version 3 Conformance Specification*.

Acknowledgement Processing

Upon receipt of a message from the sending application (the source of the message), IMP synchronously processes the message into HDR, and returns an Application Acknowledgment (AA), an Application Error (AE), or an Application Reject (AR).

- **Application Acknowledgement (AA):** An AA response indicates that the message was successfully processed and persisted in HDR.
- **Application Error (AE):** An AE response indicates an error reported by HDR, including error information in message content or format (error type code, error detail code, free text). It is the responsibility of the interface engine to determine if the acknowledgement message is returned to the sending system or if the message should be resent to HDR or skipped (abandoning the message). IMP does not support sequence number protocol--the interface engine is responsible for assuring that messages are delivered in order.

- **Application Reject (AR):** An AR response indicates that the message is rejected, for reasons unrelated to its content or format (system or network down, network transmission errors). For most such problems, the receiving system may be able to accept the message at a later time. The sending system or interface engine must decide on an application-specific basis whether the message should be sent again. Ultimately, the AR is resolved to either an AA (upon successful retransmission) or an AE--which thence generates a call to error processing.

The acknowledgement message contains the following XML segments:

Table 10–1 XML Segments in an Acknowledgement Message

Components	XPATH	Sample values
Acknowledgement Type	MCCI_MT002300HT01.Message/ acknowledgement/typeCode/@code	`<typeCode code="AA"/> ,` `<typeCode code="AE"/>,` `<typeCode code="AR"/>`
Acknowledgement Detail Code	MCCI_ MT002300HT01.Message/acknowledgement/ acknowledgementDetail/ code/@code	`<code code="NS250" codeSystemName="AcknowledgementDetailCode"/>`
Acknowledgement Error Text	MCCI_ MT002300HT01.Message/acknowledgement/ acknowledgementDetail/ text	`<text mediaType="text/plain" encoding="TXT">` `Application: CTB, Message Name: CTB_MS_INVALID_PROCESS_MD_CD. Tokens: PROCESSING_MODE_CODE = T1;` `</text>`
Acknowledgement Error Location	MCCI_ MT002300HT01.Message/acknowledgement/ acknowledgementDetail/location	`<location>` `CTB_MS_IMP_EXCEPTION_LOCATION2 :Error occurred while processing XML data located at line 6, column 30. XPATH: /PRPA_IN400000[1] COMPLEX_TYPE: MCCI_MT000100HT04.Message</location>`
HDR Error Code	MCCI_ MT002300HT01.Message/acknowledgement/ acknowledgementDetail/text	`CTB_MS_INVALID_PROCESS_MD_CD`
Responder Information	MCCI_ MT002300HT01.Message/sender/device/id	`<sender type="CommunicationFunction">` `<typeCode code="SND"/>` `<device type="Device" classCode="DEV" determinerCode="INSTANCE">` `<id root="9.989898.5.100" extension="ORG1000"/>` `<asAgent type="RoleHeir" classCode="AGNT">` `<representedOrganization type="Organization" classCode="ORG" determinerCode="INSTANCE">` `<id root="9.989898.5.100" extension="ORG1000"/>` `</representedOrganization>` `</asAgent>` `</device>` `</sender>`

Sender Configuration

Before processing a message, the message type must be configured for the sender. IMP extracts Sender, Receiver, and Interaction Id available in the message, and picks up the associated side-effect configuration. If Interaction ID is not configured for the Sender and Receiver, IMP rejects the message.

Based on the side-effect configuration, IMP sets the reference modifier on RIM objects available in the message. If a particular RIM object is not configured for side-effect, IMP defaults the value of reference modifier for the RIM object to MUST_EXIST. There are certain side-effect rules in IMP that influences the value of reference modifier of a RIM Object.

Message Validation

In addition to being compliant with messaging schema, IMP imposes certain validations on messages before processing the message. Major validations that affect messages are described in this section.

Identified Object Processing

All RIM objects containing ids are identified objects. If a message instance contains repeating objects with same ids, IMP merges the information of repeating objects and persists union of data from different instances into HDR Repository. This is called *Identified Object Processing*. If the repeating objects in the message contain inconsistent information, IMP rejects the message. For example, if the age of a particular person (having same II) has different values at different segments of the message, IMP rejects the message. For information on complete set of rules to merge information of repeating objects, refer to the *Oracle Healthcare Data Repository Programmer's Guide* and *Oracle Healthcare Data Repository Conformance Specification Guide*.

Media Type and MIME Type Validation for CDA Messages

For CDA Message Types, IMP supports only certain Media Type and MIME Type. Refer to the CDA Message Type section of the *Oracle Healthcare Data Repository Message Conformance Specification V6.1*.

Master Catalog Validation

Master Catalog entries must exist in HDR Repository for all Acts, Entities, and Roles submitted to HDR for persistence.

Vocabulary Validation

Code System Names used in the message must be loaded into ETS and the Codes used should be part of Coding Scheme.

State Transition Validation

All Acts, Entities, and Roles submitted to HDR for persistence is subjected to Generic State Transition validation. The Focal object in the message is subjected to focal class state transition.

Immutable Attributes Validation

An update message cannot modify values of structural attributes and code (example, act.ClassCode) of an already persisted object.

RIM Service Validation

Every message is persisted as a control act graph in HDR Repository and subjected to the validations done by RIM Persistence Service.

Messaging Metadata

To process a message, IMP needs the following RMIM schematic information about the message elements:

- Name of RIM Foundation Class of the RIM Object available in the message element.
- Type of RIM Association.
- Constrained RIM Data Type of the attribute.
- If the association is a choice.

The RMIM schematic information is not available in the schemas for message types, but present in the MIF files for the same message type. The information is extracted from the MIF file and loaded into the database after installing HDR. This information is known as *Messaging Metadata*.

To load Messaging Metadata, use ConcurrentProgService.loadMessagingMetadata() API.

Profile Options and System Properties

Use the *CTB: Store Incoming Message* profile option to indicate whether the incoming message has to be stored or not. The valid values are *Y* and *N*. If the value is *Y*, the incoming message is stored in the submission unit table. If the value is *N*, the incoming message is not stored.

The following system properties impacts behavior of the IMP engine:

Table 10–2 IMP System Properties

Property Name	Valid Values	Description
IgnoreUnrecognizedElements	*Y* or *N*	With value 'N' throws an exception when an unrecognized RIM attribute is encountered. With 'Y', just skips it. If *N*, IMP throws an exception when an unrecognized RIM attribute is encounters. If *Y*, IMP skips the validation. The default value is *N*.
IMP_NONDESTRUCTIVE_MODE	*Y* or *N*	If *Y*, IMP rollbacks all transactions and does not update the audit log. The default value is *N*.
IMP_BUNDLED_MODE	*Y* or *N*	If *Y*, IMP collects all non-runtime exceptions, and continues processing. If *N*, each exception aborts processing immediately. The default value is *N*.

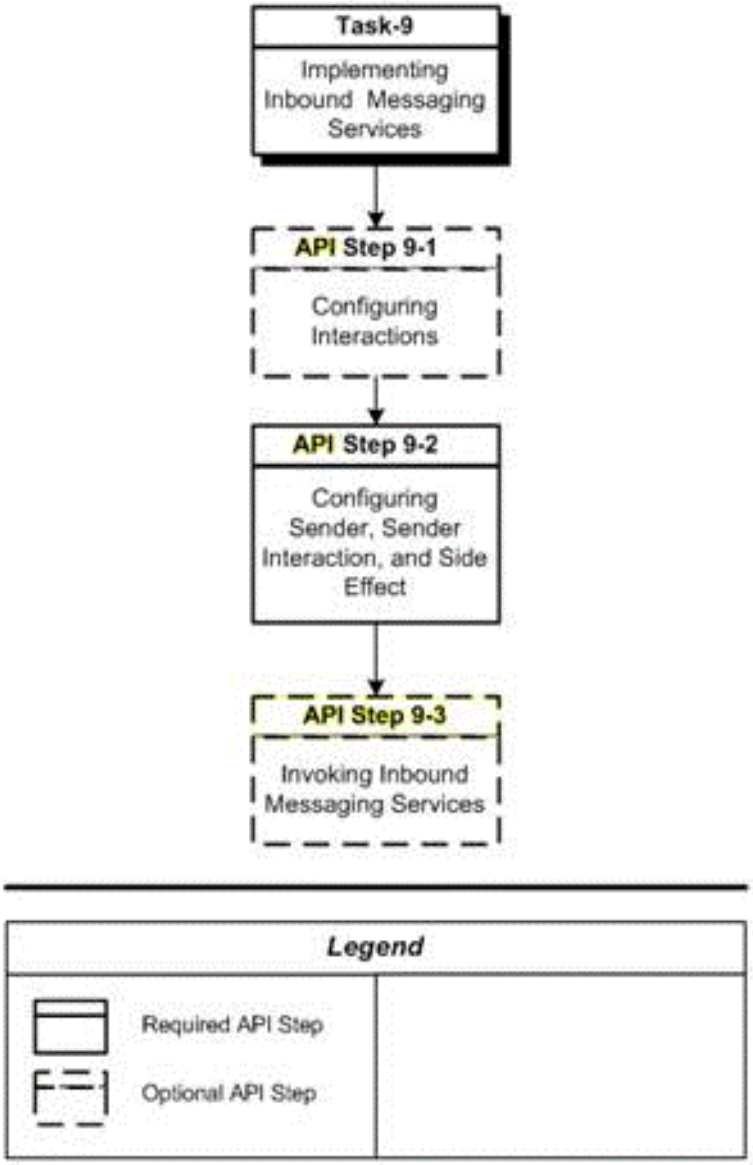
See Also

- *Oracle Healthcare Data Repository Javadoc*
- *Oracle Healthcare Data Repository Conformance Specification*

Procedures:

The following chart provides an overview of the implementation process for Inbound Messaging Services:

Figure 10–1 Implementation Process: Inbound Messaging Services



To implement Inbound Messaging Services, refer to the following procedure table:

Table 10–3 HDR Implementation Procedures: Inbound Messaging Services

Task-Step	Description	Optional?	Interface
9-1	Configuring Interactions	Yes	API
9-2	Configuring Sender, Sender Interaction, and Side Effect	No	API
9-3	Invoke Inbound Messaging Services	Yes	API

Configuring Interactions

IMP extracts Interaction Id and Trigger Event Code from the incoming message and checks whether it is configured or not. The following table lists the location of the parameters in the message

Table 10–4 Location of the Parameters in the Message

Parameter	XPath
Interaction Id	Top Level Element in the message. Example, PRPA_IN400000
Trigger Event Code	PRPA_IN400000/controlActProcess/code/@code

Interaction Ids for all supported message types are seeded. Refer to the *Oracle Healthcare Data Repository HL7 Version 3 Conformance Specification* for the list of seeded interaction ids. You can also configure new Interactions Id for supported messages. Use the Interactions window to configure new Interaction Id. For more information on the Interactions window, refer to *Oracle Healthcare Data Repository User Interface Guide*.

When you configure a new Interaction Id, an Interaction schema is generated by the Healthcare Data Repository User Interface and stored at the following location with the name of {InteractionId}.xsd::

<hdr_domain_
home>/config/hdr/message/defs/customSchema/newMessageType/interaction.

Configuring Sender, Sender Interaction, and Side Effect

IMP extracts the following information (in the table) from the message and validates them for the configuration:

Table 10–5 Information Extracted and Validated by IMP

Parameter	XPath
Interaction Id	Top Level Element in the message. Example, PRPA_IN400000
Sender Root	PRPA_IN400000/sender/device/ id/@root
Sender Extension	PRPA_IN400000/sender/device/id/@extension
Receiver Root	PRPA_IN400000/receiver/device/asAgent/representedOrganization /id/@root
Receiver Extension	PRPA_IN400000/receiver/device/as Agent/representedOrganization/id/@extension
Trigger Event Code	PRPA_IN400000/controlActProcess/code/@code

If the Sender Root and Extension and Receiver Root and Extension is not configured, IMP rejects the message. This configuration thus controls a valid sender and HDR enterprises authorized to send messages. This is called the *Sender Configuration*.

Important: You must only use Organization's external II while creating sender configuration. You must not use any of the Internal IIs that are automatically generated in HDR.

Upon validation of the Sender Configuration, IMP uses its configuration to determine if the Interaction Id is valid for the Sender Configuration. If it is not configured for that Sender Configuration, IMP rejects the message. This configuration thus controls which types of Interaction Id a sender is permitted to send to a receiver. This is called the *Sender Interaction Configuration*.

Upon validation of the Sender Configuration and Sender Interaction Configuration combination, IMP processes the message payload. The focal object is created or updated in the HDR Repository. However, for non-focal objects, IMP inspects its side effect configuration to determine its behavior. You can configure IMP to let each non-focal object type create or not create the object if *it is not* present in the repository, and to update or overlay or not update or overlay the object *if it is* present in the repository. This configuration of side effects is called the *Side Effect Configuration*.

Use the IMPConfigAdminIntrService to configure sender and side effects.

See Also

- *Oracle Healthcare Data Repository HL7 Version 3 Conformance Specification* for a list of side effect configuration records required for each message type.

Invoke Inbound Messaging Services

Reference

- *Oracle Healthcare Data Repository Javadoc*
- *Oracle Healthcare Data Repository HL7 Version 3 Conformance Specification*

The following table lists the principal IMP service and methods:

Table 10–6 Service and Methods: IMP

Level	Detail
Package	oracle.hsgbu.hdr.message.improcessor
Class	IMPService
Methods	▪ processMessage
Class	RawIMPService
Methods	▪ processMessage
Class	Result
Methods	▪ getResponseXML ▪ getStatus ▪ getControlActId ▪ getTriggerEvent

Login

This is an API-based implementation procedure.

Navigation

This is an API-based implementation procedure.

Steps

1. Use the Service Locator to access the IMP Service.

   Note:

   `RawIMPService` is implemented as a container-managed transactions (CMT) bean, and does not create `SubmissionUnit`. Use `RawIMPService` if you want to use the Java Transaction API (JTA) support of IMP.

2. Use the processMessage method with an HDR-compliant message (see following Note) as a parameter to invoke message processing services; a Result object is returned.
3. Use the following methods to inspect the result of processing the message:
   - getResponseXML
   - getStatus

Note:

IMP supports XML formatted inbound messages that conform to the HL7 version 3 messaging standard. The messages must conform to the messaging schema for the message types supported in HDR. The schemas for all supported message type is available at the following location:

```
<hdr_domain_home>/config/hdr/messge/defs/rim214101/schemas
```

The list of supported message types is provided in *Oracle Healthcare Data Repository HL7 Version 3 Conformance Specification*. Using Messaging Tool Kit, additional message types can be supported. For more information, refer to *Oracle Healthcare Data Repository Messaging Tool Kit User Guide*.

See Also

- *Oracle Healthcare Data Repository HL7 Version 3 Conformance Specification*, for information about message types supported by IMP.

See Also: ■Oracle Healthcare Data Repository Implementation Guide describes how to implement and configure messaging services, including HDR Gateway

- Oracle Healthcare Data Repository HL7 Version 3 Messaging Conformance Specification describes the HL7 message types supported by the current HDR release

---

Note: IMP provides Java Transaction API (JTA) support through RawIMPService. RawIMPService provides the same functionality as IMPService except that RawIMPService is implemented as a container-managed transactions (CMT) bean, and does not create SubmissionUnit. Use RawIMPService if you want to use the Java Transaction API (JTA) support of IMP.

IMP Configuration API Usage

The Interaction ID based IMP Configuration Administration Service provides functions to create, remove, and find Sender Configurations. HDR Configuration Service also provides same functionality. To update a Sender Configuration, the client application removes it and creates the new Sender Configuration, including all of its child objects (sender interaction configuration and sender side effect configuration). HDR APIs support the following:

- Moving configuration data from one environment to another
- Writing loader applications

This section contains the following topics:

- Sender Interaction Configuration Attributes
- Sender Side Effect Configuration Attributes
- Sender Configuration Search Parameters
- IMP Sender Interaction Configuration Administration Service

Note: ■The client application must call the Master Catalog Service to resolve Master Catalog IDs.

■ You can use Messaging Configuration Service for updates.

Sender Configuration Attributes

Attribute Name	Field Type	Length	Mandatory	Description
Sender Id Root	`String`	240	Yes	Root part of the OID of the sending device
Sender Id Extension	`String`	240	Yes	Extension part of the OID of the sending device
Receiver Id Root	`String`	240	Yes	Root part of the OID of a valid organization registered in the system
Receiver Id Extension	`String`	240	No	Extension part of the OID of a valid organization registered in the system
Sender Interaction Configuration	`SenderInteractionConfiguration[]`	-	No	An array of Sender Interaction Configurations for this Sender Configuration

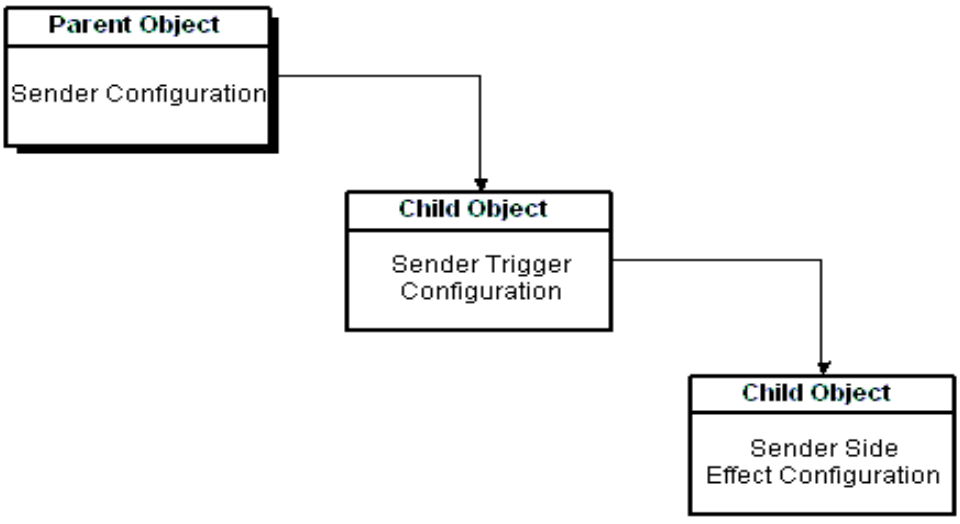
Sender Interaction Configuration Attributes

Attribute Name	Field Type	Length	Mandatory	Description
Interaction Id	`String`	80	Yes	Valid Interaction Id for this Sender Interaction Configuration
Sender Side Effect Configuration	`SenderSideEffectConfiguration[]`	-	No	An array, of Sender Side Effect Configurations, that defines permissions (create if/create or overlay/update/must exist/create or update) on referenced objects

Sender Side Effect Configuration Attributes

Attribute Name	Field Type	Length	Mandatory	Description
Master Catalog Id	`String`	-	Yes	The Master Catalog ID of the RIM object for which the side effect is configured. You can retrieve the Master Catalog details from the `MasterCatalogService`
Object Reference Modifier Code	`String`	-	No	Object reference modifier code value: `CREATE_IF, OVERLAY, CREATE_OR_OVERLAY, MUST_EXIST, UPDATE, CREATE_OR_UPDATE`
Player Reference Modifier Code	`String`	-	No	Player reference modifier code value: `CREATE_IF, OVERLAY, CREATE_OR_OVERLAY, MUST_EXIST, UPDATE, CREATE_OR_UPDATE`
Scoper Reference Modifier Code	`String`	-	No	Scoper reference modifier code value: `CREATE_IF, OVERLAY, CREATE_OR_OVERLAY, MUST_EXIST, UPDATE, CREATE_OR_UPDATE`

Figure 10–2 Sender Configuration Objects



Sender Configuration is the top-level object, with Sender Interaction Configuration object as its direct child. Sender Interaction Configuration object in turn has Sender Side Effect Configuration object as its child. The client application must create the

complete hierarchy before invoking the create functionality. Before removing a Sender Configuration, the client application must invoke the finder API to get the handle associated with the target configuration item to delete.

Sender Configuration Search Parameters

`SenderConfigSearchConfig` is a criteria object passed to the finder API as a parameter. It provides methods to set search attributes used for retrieving the Sender Configuration and associated Sender Interaction Configurations and Sender Side Effect Configurations, based on the following parameters:

- `Sender Id Root`
- `Sender Id Extension`
- `Receiver Id Root`
- `Receiver Id Extension`

Because `Receiver Id` Extension can have a null value, in order to support a query where in the client application searches specifically for items with Receiver Id Extension as null value, a boolean flag `ReceiverIdExtensionIsNull` can be set to *True*. If `ReceiverIdExtensionIsNull` is not set or set to *False*, the criteria matches the given root and *any* extension.

IMP Sender Interaction Configuration Administration Service

The IMP Sender Interaction Configuration Administration Service lets the client application create, remove, and find Sender Configurations. The code samples that follow illustrate each of these use cases:

- Create Function
- Find Function
- Remove Function

> **Note:** SenderConfigFactory class is not available from 8.0.

Create Function

This code sample creates a new sender:

Example 10–1 Sample Code to Create Sender Configuration Using the newXXX Functions Returning Empty Objects

```
public SenderConfiguration createSenderConfiguration()
{
SenderConfiguration senderConfig = new SenderConfigurationImpl();
        senderConfig.setSenderIdRoot(SND_ROOT);
        senderConfig.setSenderIdExtension(SND_EXTN);
        senderConfig.setReceiverIdRoot(RCV_ROOT);
        senderConfig.setReceiverIdExtension(RCV_EXTN);
        SenderInteractionConfiguration senderIntrConfig = new
SenderInteractionConfigurationImpl();
        senderIntrConfig.setInteractionId(INTERACTION_ID[0]);
        SenderSideEffectConfiguration sseConfig = new
SenderSideEffectConfigurationImpl();
        sseConfig.setObjectRefModifierCode(SenderSideEffectConfiguration.CREATE_
OR_OVERLAY);

sseConfig.setPlayerRefModifierCode(SenderSideEffectConfiguration.CREATE_OR_
```

```
OVERLAY);
        sseConfig.setMasterCatalogId(MC_IDS[0]);
        SenderSideEffectConfiguration[] sseConfigs = { sseConfig };
        senderIntrConfig.setSenderSideEffectConfiguration(sseConfigs);
        SenderInteractionConfiguration[] senderIntrConfigs = { senderIntrConfig };
        senderConfig.setSenderInteractionConfigurations(senderIntrConfigs);
        SenderConfiguration createReturn =
configService.createSenderConfiguration(senderConfig);
              return createReturn;
```

It is not mandatory to set the Receiver Extension on the Sender Configuration because this attribute is nullable. If a reference modifier code is not set, the value, the values defaults to null.

Find Function

This code sample lets the client application retrieve a Sender Configuration using the Sender Configuration Search Criteria:

Example 10–2 Sample Code to Find Sender Configuration Using the newXXX Functions Returning Empty Search Criteria

```
public SenderConfiguration[] findSenderConfiguration()
{
        SenderConfigSearchCriteria criteria = new
SenderConfigSearchCriteriaImpl();
        criteria.setReceiverIdRoot(RCV_ROOT);
        criteria.setReceiverIdExtension(RCV_EXTN);
        criteria.setSenderIdRoot(SND_ROOT);
        criteria.setSenderIdExtension(SND_EXTN);
        SenderConfiguration[] queriedSenderConfigs =
configService.findSenderConfiguration(criteria);
        return queriedSenderConfigs;
```

Because all Search Criteria setters are optional, all of the records are returned if you do not set any criteria. The Receiver Extension is the only attribute that is *nullable*, and is thus treated differentially here. You must either set the `Receiver Id` Extension to a particular non-null value or set the flag using the function `setReceiverIdExtensionIsNull(boolean)`. Setting both results in a query that ignores the value set by `setReceiverIdExtension(String)`. The boolean flag thus takes precedence over setting of the `ReceiverId` Extension to a particular value. Failing to set of them either lets the user search for all records-records with both null value `and non-null value Receiver Id` Extension attributes.

Remove Function

This code sample permits the deletion of a particular Sender Configuration and all its child elements. The client application is expected to invoke the find function to get a handle on the Sender Configuration it wants to delete.

Example 10–3 Sample Code to delete Sender Configuration

```
public void removeSenderConfiguration()
{
    SenderConfigSearchCriteria criteria = new SenderConfigSearchCriteriaImpl();

        criteria.setReceiverIdExtension(RCV_EXTN);
        criteria.setSenderIdRoot(SND_ROOT);
        criteria.setSenderIdExtension(SND_EXTN);
        criteria.setReceiverIdExtensionIsNull(false);
     (RCV_ROOT, RCV_EXTN, SND_ROOT,SND_EXTN, false);
criteria.setReceiverIdRoot(RCV_ROOT);
```

```
    SenderConfiguration[] senderConfigs =
configService.findSenderConfiguration(criteria);
    if (senderConfigs != null && senderConfigs.length > 0)
    {
            for (int i = 0; i < senderConfigs.length; i++)
            {
              configService.removeSenderConfiguration(senderConfigs[i]);
            }
          }
}
```

Examples: This section contains the following examples:

- Persist a Sample Message
- A Sample Acknowledgement Message

Example 10–4 Persist a Sample Message

Perform the following steps to persist a sample encounter message:

1. Create the Receiver Organization (root="9.989898.5.100" extension = "ORG1000"), using RIM Service.
2. Load sender and side effect configuration for the message using Bulk Load Service.
3. Execute the program to persist the message. On successful persistence, IMP returns an acknowledgment.

Side Effect Configuration Rules

Following are the side effect processing rules enforced by IMP:

Also, there are additional rules applied by RIM Services.

1. RIM objects without IDs are always created.
2. If side effect is not configured for a RIM Object, IMP defaults the reference modifier to `MUST_EXIST`.
3. For Focal Objects, IMP overrides any side effect configuration with reference modifier to `CREATE_OR_OVERLAY`.
4. If Focal Object is a role, player and scoper of the role are processed as per the side effect configuration for these entities, except Person and Unmerge messages. IMP overrides the configuration for player and scoper attached to the Focal Role of Person and Unmerge messages, with `CREATE_OR_UPDATE`.
5. All side effect processing follows the traversal path through the message model. This means that side effect processing is disabled for objects downstream from an object that has been configured as `MUST EXIST`.
6. If a backward/inbound participation to a role is referenced, the act that directly participates with that participation is processed as if `CREATE_OR_UPDATE` modifier is set on the act.
7. Owned roles are processed following the side effect processing of the Owning Entity. If the Owning Entity is configured for the side effect reference modifier as:
   - `CREATE_IF`, and if it exists, the Owned Role and the associated entity are ignored.
   - `MUST_EXIST`, the Owned Role and the associated entity are ignored.

- For other configurations, the Owned Role is always created or overlaid.

8. While processing Identified Objects, the merged object is assigned with the side effect of the object with the most restrictive side effect configuration.

Example 10–5 A Sample Acknowledgement Message

Acknowledgement message returned by IMP is compliant with HL7 V3 standard. The message contains the following elements:

- Receiver (same as respond to in the processed message).
- Sender (same as receiver in the processed message).
- Acknowledgement typecode.
  - AA - Message processed successfully.
  - AE - Message processing failed.
  - AR - Failed to process (reject) the message for reasons unrelated to its content or format (system down, internal error, and so on).
- Error message, if message processing fails.
- acknowledgementDetail. The element may be repeated couple of times, if there are multiple errors in the message. This element contains the following attributes:
  - Code: corresponding HL acknowledgement detail code.
  - Location: contains XPATH, XSD Complex Type Name, and line number of the message element responsible for the error.
  - Text: contains error message. The error text message is represented in the following format: {ERROR CODE NAME}: {ERROR MESSAGE TEXT}.

The following is a complete xml message, as it appears in an acknowledgment message:

```
<MCCI_MT002300HT01.Message xmlns="urn:hl7-org:v3" type="Message"
xmlns:htb="http://xmlns.oracle.com/apps/ctb/messaging">
     <id root="Messsage Id root value" extension="Message Id extension value"/>
    <creationTime value="Acknowledgement creation time"/>
     <responseModeCode code="D"/>
    <interactionId root="Interaction Id root value" extension="Interaction Id
extension value"/>
    <processingMode code="P"/>
     <processingModeCode code="T"/>
     <acceptAckCode code="NE"/>
     <acknowledgement type="Acknowledgement">
        <typeCode code="Type Code (AA or AE)"/>

   <!--Use <acknowledgementDetail> only for typecode = AE, skipped for typecode =
AA -->
    <acknowledgementDetail type="AcknowledgementDetail">
      <typeCode code="E"/>
      <code code="Error Code" codeSystemName="AcknowledgementDetailCode"/>
      <text mediaType="text/plain" encoding="TXT">Error Text</text>
  <location>Error location</location>
    </acknowledgementDetail>
      <targetMessage type="Message">
         <id root="Inbound Wrapper Message Id root value"
    extension="Inbound Wrapper Message Id extension value"/>
        </targetMessage>
      </acknowledgement>
```

```
      <receiver type="CommunicationFunction">
        <typeCode code="RCV"/>
        <device type="Device" classCode="DEV" determinerCode="INSTANCE">
          <id root="Inbound Wrapper Respond To Application Id root value"
    extension="Inbound Wrapper Respond To Application Id extension value"/>
          <asAgent type="RoleHeir" classCode="AGNT">
            <representedOrganization type="Organization" classCode="ORG"
determinerCode="INSTANCE">
              <id root="Inbound Wrapper Respond To Enterprise Id root value"
      extension="Inbound Wrapper Respond To Enterprise Id extension value"/>
            </representedOrganization>
          </asAgent>
        </device>
      </receiver>
      <sender type="CommunicationFunction">
        <typeCode code="SND"/>
        <device type="Device" classCode="DEV" determinerCode="INSTANCE">
          <id root="Inbound Wrapper Target Enterprise Id root value"
extension="Inbound Wrapper Target Enterprise Id extension value"/>
          <asAgent type="RoleHeir" classCode="AGNT">
            <representedOrganization type="Organization" classCode="ORG"
determinerCode="INSTANCE">
              <id root="Inbound Wrapper Receiver Organization Id root value"
         extension="Inbound Wrapper Receiver Organazation Id extension value"/>
            </representedOrganization>
          </asAgent>
        </device>
      </sender>
    </MCCI_MT002300HT01.Message>
```

HDR Message Submission Unit

The HDR Message Submission Unit defines a structure that contains messages processed by OMP or messages received and processed by IMP. The Submission Unit is used to audit information that is related to the processing of an xml message by IMP or OMP. Every SubmissionUnit is uniquely identified by the Instance Identifier message attribute.

This package consists of two interfaces: SubmissionUnit and SubmissionUnitService. The SubmissionUnitService interface defines the mechanism for finding and updating a persisted submission unit. The SubmissionUnit interface provides methods for accessing and updating the attributes of the SubmissionUnit.

See the following section for more information about Message Submission Unit interfaces:

Message Submission Unit

This package includes two interfaces [SubmissionUnit, SubmissionUnitService] that can be used to audit the information related to the processing of an XML message by IMP or OMP. These interfaces are described in the following sections:

- Submission Unit Interface
- Submission Unit Service Interface Methods

Task:

Find Submission Unit: Check and Resend

Submission Unit Interface

The SubmissionUnit interface defines a structure t'hat contains the messages processed by IMP or OMP. It contains get methods that can be used to track both inbound and outbound message processing.

The IMPService.processMessage method creates and updates the SubmissionUnit with the message id, acknowledgement typecode, acknowledgement message text, acknowledgement date, send date, sender id, receiver id, responder id, control act id, control act author id, original message, trigger event code, name of the application that submitted the SubmissionUnit and other related attributes. The processMessage method updates SubmittedByAppName to *HTBIMP*.

You can use SubmissionUnitService find methods to find a SubmissionUnit for a particular submission unit identifier or control act identifier.

Submission Unit Service Interface Methods

The SubmissionUnitService interface defines the methods for finding and updating a persisted submission unit. It contains the following finder methods to find persisted SubmissionUnits:

Submission Unit Service Interface Method

Method	Description
`findSubmissionUnitByControlAct(II controlActId)`	Returns a `SubmissionUnit` object with the specified control act identifier; throws `ETSException`
`findSubmissionUnitById(java.lang.String submissionUnitId)`	returns a SubmissionUnit object with the specified submission unit id; throws `ETSException`
`findSubmissionUnitByMessage(II messageId)`	Returns a SubmissionUnit object with the specified submission unit id; throws `ETSException`
`updateSubmissionUnit(SubmissionUnit submissionUnit)`	Updates `SubmissionUnit; throws ETSException`
`createSubmissionUnit(SubmissionUnit submissionUnit)`	Persists `SubmissionUnit; throws ETSException`

Example 10–6 Find Submission Unit: Check and Resend

Find the submission unit and check if the message generated successfully; if failed, resend the modified payload back to the receiver:

```
public void resendMessage(String submissionUnitId, II receiverId, II messageId)
    throws CustomerApplicationException
  {
    try {
      // Find SubmissionUnit for particular submission unit id
      SubmissionUnit subUnit =
        mSubmissionUnitService.findSubmissionUnitById(submissionUnitId);
      String ackTypeCode = subUnit.getAckTypeCode();      // If submission unit
acknowledgement type code is AE, modify original message payload and
         send it back.
      if ("AE".equals(ackTypeCode)) {
        // Get original message
        String originalMessageText = subUnit.getOriginalMessageText();
        // Modify original message
        String payload = modifyPayload(originalMessageText);
        // Resend the message
        String subId = mOMPService.resendMessage(payload, receiverId, messageId);
```

```
    }
  } catch (ETSException ETSException) {
    throw new CustomerApplicationException(ETSException.getMessage());
  }
}  public String modifyPayload(String originalMessageText) {
  ...
  ...
  return payload;
}
```

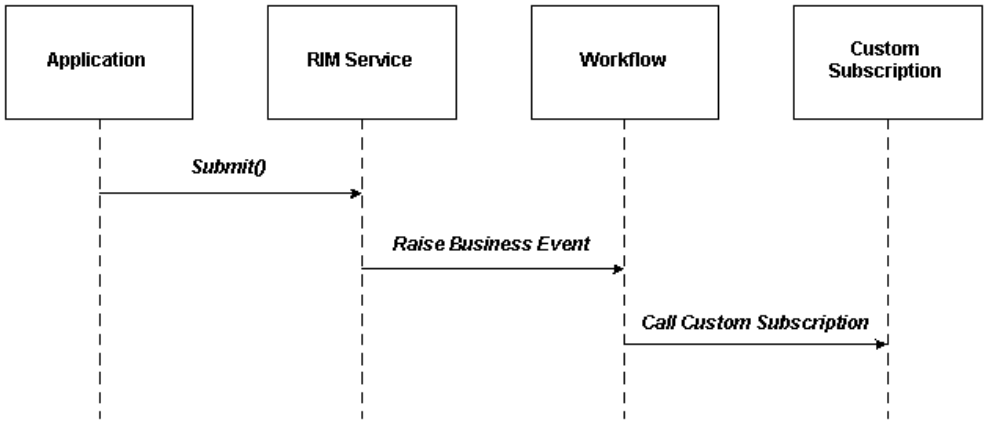
HDR RIM Service Hook

RIM Services raise workflow business events to which applications can subscribe. You can write subscription classes to listen to such events. This provides a loose coupling with the RIM Service that lets you integrate or disintegrate unique functionality by subscribing or unsubscribing to business events.

With this hook, you can code a Subscription Class subscribed to a business event; you are expected to include custom functionality in this class. The following sequence figure 10.4.1 illustrates the interaction between an application persisting data and a custom Subscription class:

Figure 10-2

Figure 10–3 Interaction between App Persisting Data and Custom Subscription Class



The following sections describe the RIM Service Event and Subscription, and include a Subscription Code Example:

Event and Subscription

The RIM Service raises event oracle.hsgbu.hdr.hl7.persist.event.ControlActSubmissionPost on submit. It passes trigger event and control act identifiers of the persisted data to the event in key-value pairs:

Keys

Key	Description
TRG_EVNT	Trigger event of persisted message
CONTROL_ACT_ID	Control act identifier root of persisted message

CONTROL_ACT_EXT	Control act identifier extension of persisted message.

Example 10–7 Event and Subscription

When RIM Service raises the event oracle.hsgbu.hdr.hl7.persist.event.ControlActSubmissionPost, workflow calls the onBusinessEvent method of the customer subscription class. All of the forgoing parameters are bundled in the BusinessEvent object and passed to the subscription class. The Subscription class can read those parameters in its onBusinessEvent method as illustrated in the following example:

```
public void onBusinessEvent(Subscription eo, BusinessEvent event, WorkflowContext
wfCxt)
    throws BusinessEventException
 {
   try {
     String TriggerEvent = event.getStringProperty("TRG_EVNT");
     String ControlActId_Root = event.getStringProperty("CONTROL_ACT_ID");
     String ControlActId_Extension = event.getStringProperty("CONTROL_ACT_EXT");
     ...
   } catch (Exception e) {
     e.printStackTrace();
   }
 }
```

Subscription Code Sample

The RIM Service receives an HL7 message, persists and resends the message to all organizations registered for that trigger event.

The functionality to resend the message to all organizations registered for the trigger event can be implemented in a subscription class to be executed when the RIM Service persists that message.

Example 10–8 Subscription Code Example

```
package oracle.hsgbu.hdr.sample.scenario;
import java.io.IOException;
import java.io.InputStream;
import java.io.PrintWriter;
import java.io.StringWriter;
import java.util.Properties;
import oracle.hsgbu.hdr.fwk.base.common.ETSException;
import oracle.hsgbu.hdr.fwk.serviceLocator.common.ServiceLocator;
import oracle.hsgbu.hdr.hl7.domain.NullFlavor;
import oracle.hsgbu.hdr.hl7.factories.DataTypeFactory;
import oracle.hsgbu.hdr.hl7.types.II;
import oracle.hsgbu.hdr.hl7.types.ST;
import oracle.hsgbu.hdr.hl7.types.UID;
import oracle.hsgbu.hdr.message.omprocessor.ControlActRequest;
import oracle.hsgbu.hdr.message.omprocessor.OMPHelper;
import oracle.hsgbu.hdr.message.omprocessor.OMPService;
import oracle.hsgbu.hdr.security.Responsibility;
import oracle.hsgbu.hdr.security.SessionContext;
import oracle.hsgbu.hdr.security.SessionService;
import oracle.apps.fnd.wf.bes.BusinessEvent;
import oracle.apps.fnd.wf.bes.BusinessEventException;
import oracle.apps.fnd.wf.bes.SubscriptionInterface;
import oracle.apps.fnd.wf.bes.server.Subscription;
import oracle.apps.fnd.wf.common.WorkflowContext;
```

```
 public class WorkflowListenerSubscription implements SubscriptionInterface {
 protected ServiceLocator mServiceLocator = null;
 public void onBusinessEvent(Subscription eo,
 BusinessEvent event,
 WorkflowContext wfCxt)
 throws BusinessEventException
 {
   try {
   // Get OMPService
   OMPService mOMPService = getServiceLocator().getOMPService();
   // Read parameres passed by RIM Service persistence.
   String triggerEvent = event.getStringProperty("TRG_EVNT");
   String controlActId_Root = event.getStringProperty("CONTROL_ACT_ID");
   String controlActId_Ext = event.getStringProperty("CONTROL_ACT_EXT");
   DataTypeFactory mDataTypeFactory =
DataTypeFactory.getInstance(getServiceLocator());
   UID uidVal = mDataTypeFactory.newUID(controlActId_Root);
   ST extVal = mDataTypeFactory.newST(controlActId_Ext);
   II controlActId = mDataTypeFactory.newII(uidVal, extVal,
mDataTypeFactory.nullBL(NullFlavor.NI));
   // Create ControlActRequest
   OMPHelper ompHelper = new OMPHelper();
   ControlActRequest ompCACTRequest = ompHelper.newControlActRequest();
   ompCACTRequest.setControlActId(controlActId);
   ompCACTRequest.setTriggerEvent(triggerEvent);
   // Generate message
   mOMPService.generateMessage(ompCACTRequest);r
   } catch (Exception e) {
     throw new BusinessEventException(e.getMessage());
   }
 }
 protected ServiceLocator getServiceLocator() throws ETSException,IOException {
   if (mServiceLocator != null) {
     return mServiceLocator;
   }
   ClassLoader loader = Thread.currentThread().getContextClassLoader();
   Properties props = new Properties();
   props.load(loader.getResourceAsStream("jndi.properties"));
   //Specify Client mode: Local or Remote
   props.setProperty(ServiceLocator.CLIENT_MODE, ServiceLocator.REMOTE);
   ServiceLocator mServiceLocator = ServiceLocator.getInstance(props);
   mServiceLocator.login("sysadmin","sysadmin");
   return mServiceLocator;
   }
  }
```

See Also:

- *Oracle Workflow Administrator's Guide*
- *Oracle Workflow API Reference*
- *Oracle Workflow Developer's Guide*
- *Oracle Workflow User's Guide*

HDR Messaging Toolkit

Oracle Healthcare Data Repository (HDR) is a healthcare application development platform that exchanges healthcare information with external applications using the HL7 Version 3 Messaging Standard.

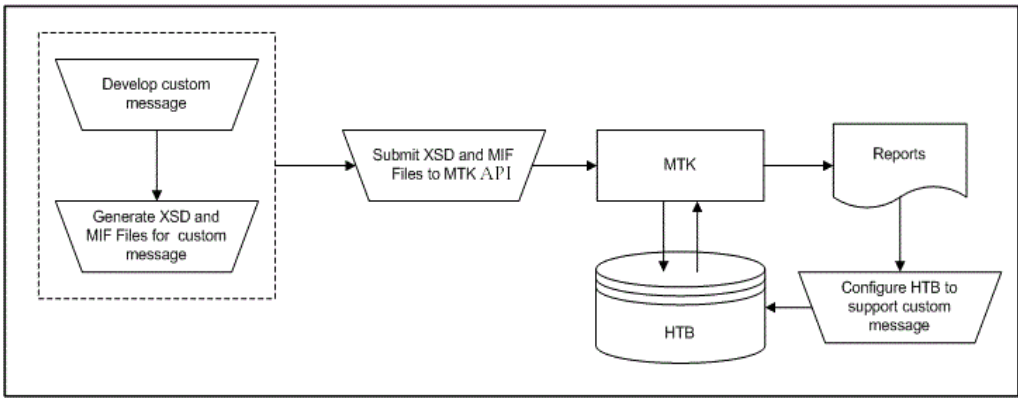
HDR ships with predefined message specifications for a comprehensive set of domains (subject areas), such as Encounter Management, Patient Care, Lab, and Public Health. Each such specification is called a Message Type. For each predefined Message Type, the inbound and outbound message processors have access to Message Type related files and data (XSD, MIF, Composite Message Schema, cmetinfo.coremif, and seeded metadata) that help them use and generate conformant XML instances.

The HDR Messaging Toolkit (MTK) provides you with a mechanism to specify custom message types, to facilitate HDR interoperation with other healthcare applications, and to suit your specific business requirements. MTK lets you perform validation and setups necessary to support custom messages types.

MTK Workflow

The following figure displays the process for creating and persisting custom message types using MTK:

Figure 10–4 Workflow for Creating and Persisting Custom Message Types Using MTK



To create and persist a new custom Message Type using MTK:

1. Create an RMIM for a custom message type in any one of the following ways:
   - Designing the RMIM from scratch.
   - Modifying an Oracle-published artifact to suit your business requirements. For more information on Oracle-published artifacts, refer to the *Oracle Healthcare Data Repository Conformance Specification*.
   - Using an HL7 normative artifact. For more information, refer to the HL7 website (www.hl7.org).
2. Generate the associated schema and MIF files. If the Message Type has any Common Message Element Type (CMET) references (either Oracle-published or custom) you must create a cmetinfo.coremif file.
3. Load the XSD and MIF files into MTK. MTK performs some tasks as described in the following sections and generates reports.
4. Use these reports to configure HDR to support these custom Message Types.

For more information on how to implement a new Message Type, refer to the Implement a New Message Type section.

> **Note:** The process of testing a custom message using MTK does not affect the data in HDR or existing configurations of HDR.

MTK accepts the XSD, MIF, and *cmetinfo.coremif* files of a custom message type as the input and performs the following:

1. Validates inputs.
2. Generates the configuration reports.
3. Generates instances with minimal manual intervention.
4. Tests the instances for compatibility with HDR.
5. Sets up Message Type.

Validating Inputs

MTK validates the inputs for the following:

- The Message Type must have both the XSD and MIF files.
- The custom CMETS referenced by Message Types must be present.
- The *cmetinfo.coremif* file must be provided if the Message Type has any reference to the CMETs.
- The contents of the corresponding XSD and MIF files must match.

After successful input validation, MTK loads the submitted artifacts into a test folder and generates related metadata.

Generating Configuration Reports

MTK generates configuration reports to help you port (or create) configurations related to the custom message into the production environment. These reports are generated in plain text or CSV format. The downloaded configuration can be loaded into a different HDR instance by using the respective messaging configuration service or by converting the data into SQL loader format.

> **Note:** Use these reports as an aid for creating the configuration reports to set up the message type in the production environment. These reports are not to be considered as the actual report.

Configuration Reports

Report Name	Report Format	Description
Master Catalog	CSV	Lists the Acts, Roles, and Entities missing from Master Catalog for a given Message Type. You can use this report to create the appropriate Act Concept configurations in the production environment.

Side Effect Configuration	CSV	Lists all objects that require side effect configurations for a given Message Type. This report lists the side effect configurations for a given Message Type with the master catalog entries relating to the code type *ANY*. The reference modifier used in the report for the objects will be *Create or Overlay*. You can use this report to create the appropriate Act Concept configurations in the production environment.
Act Concept Configuration	Plain Text	Lists the ambiguous complexes found in the Custom Message Type Schema. You can use this report to create appropriate Act Concept configurations in the production environment.

Generating Instances

Based on the constructs available, MTK generates one or more message instances. MTK uses dummy data to generate message instance. The data for the message instance will be drawn from the data seeded in HDR. MTK follows certain rules and logic while generating the instance.

Testing Instance

MTK tests the instance for the following:

- Generation of configuration data: Inbound Message Processor (IMP) and Outbound Message Processor (OMP) configurations required by HDR to process a Message Type are created by MTK. These configurations will be rolled back after completing the test. For information on Sender, Receiver, Interaction, and Trigger event ID, refer to the *Oracle Healthcare Data Repository HL7 Version 3 Conformance Specification*.
- Persistence of the generated MTK message instance into HDR using IMP: Once the instance(s) are generated, MTK invokes the IMP services to persist the message.
- Generation of OMP instance: After persisting the message instance(s) successfully using IMP, MTK invokes OMP services to generate outbound instances.
- Logging of output status along with Test ID: A unique Test Id will be generated for each test requested. The format of the Test Id is *{ArtifactID}_{User ID}_{Test Date}*.

Setting Up Message Type

MTK also helps users to upload the successfully tested Message Type's files onto the production environment. The Message Types loaded on the production environment can be browsed and managed using MTK Services.

Messaging Toolkit (MTK) Services

Use the `oracle.apps.ctb.message.mtk` package for this purpose.

Following are the MTK services:

- `oracle.apps.ctb.message.mtk.MtkTestService`
- `oracle.apps.ctb.message.mtk.MtkProductionLoadService`

For detailed description of the APIs provided by these MTK services, refer to the HDR Javadoc.

HL7 Message Development Process

To utilize MTK to set up custom message types, you must understand the HL7 V3 process by which message types are specified.

The following steps summarize the MTK process for specifying message types:

1. Develop storyboards and use cases that provide requirements for the content and transactions of messages.
2. Use the HL7 RMIM Designer to produce an RMIM for a message type based on the content requirements from the previous step. Refer to the HL7 website (www.hl7.org) for more information on RMIM Designer.
3. Use the HL7 Schema Generator to convert RMIM Designer output into XML schema definition files (XSDs) and Message Interchange Format files (MIF). Refer to the HL7 website (www.hl7.org) for more information on Schema Generator.

Note: For each type of message specified, one pair of XSD and MIF files are produced that define the structure and constraints of the message. In addition, if CMETs were used in defining the message a *cmetinfo.corem*if file is generated.

Implement a New Message Type

A new Message Type can be implemented in HDR using the MTK Services.

The following are the prerequisites for using MTK:

- **Tools**

  Either download the tools provided by HL7 or use your own tools to generate the Schema and MIF files of a new Message Type. The tools used to generate a schema must also generate the required MIF files.

  Note: MTK is tested with artifacts created on:

  - HL7 RIM repository v2.14.1
  - RMIM Designer - 4.3.2
  - RoseTree - 4.0.12
  - Schema Generator - 3.0.4

- **Concept Lists**

  Identify from the *Javadoc* the Concept List used by the attributes of the new Message Type's objects. Ensure those Concept List have concepts in them.

  Note: If any of the Concept List used by a new Message Types Object attribute is empty, an error is thrown by MTK and the Message Type is not processed.

- **Datatypes and VOC files**

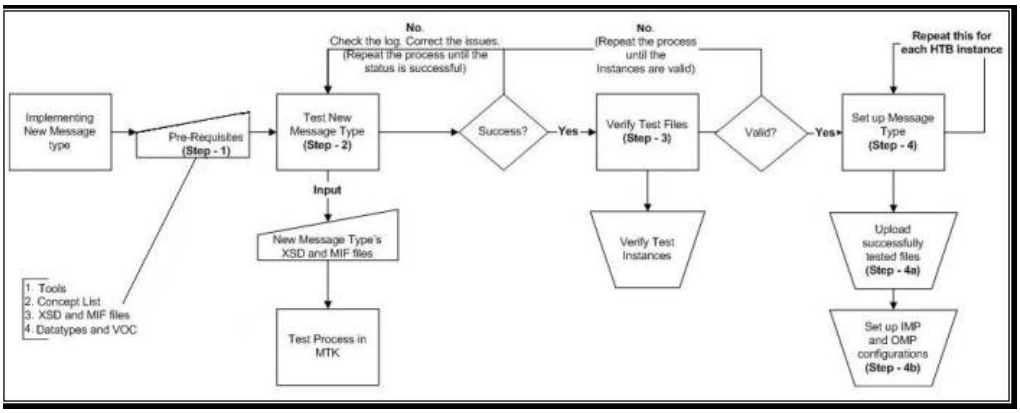
  datatypes.xsd, datatypes-base.xsd, and voc.xsd files are available for download from the Test Message Types window of the MTK Service. You can use these files

in the schema generation tool to create a Schema and MIF files for the new Message Type.

Procedure

The following figure illustrates the process of implementing a new message type.

Figure 10–5 Implement New Message Type Using MTK



Test a New Message Type

Use the MTK Service Functions or MTK Services to test a new Message Type. Ensure that you have the following inputs for the testing process:

- Message Type's Schema and MIF files.
- Schema and MIF files of the custom CMETs referenced by the Message Type.
- *cmetinfo.coremif* file listing all the CMETS (Oracle published and custom) referenced by the Message Type. Note that the entries of the non-referenced CMET in the *cmetinfo.coremif* file are ignored.
- Comments relating to the test.

MTK uses the following process to test a new Message Type:

1. **Initial Validation:**

   Once the input files are submitted, MTK validates if:

   - Message Type is present.
   - Message Type has the XSD and MIF files.
   - Custom CMETs referenced by Message Type are present.
   - *cmetinfo.coremif* file with proper entries is provided.

2. **Loading onto Server:**

   MTK loads the test files provided on the server in a test folder. This folder will be named in the format {Artifact ID}_{User ID}_{Test Date}.

3. **Generating Metadata:**

   Metadata of those files is generated and persisted in the server.

4. **Interaction:**

A test interaction ID based on the artifact ID is created and inserted in the interaction table.

5. **Composite Message Schema**

   Using the test interaction created Composite Message Schema is generated.

6. **Metadata for Composite Message Schema:**

   From the Composite Message Schema, metadata for the Composite Message Schema is generated and stored in the server.

7. **Reports:**

   Reports for ambiguous complex types, missing Master Catalog entries, sender side effect configurations is generated.

8. **Generating Test Instance:**

   Data for instance will be fetched from:

   - Schema for the structural attribute;
   - Database for non-structural, non-identity, and non-coded attributes (Use the data snippets seeded in HDR);
   - Concept list for coded attributes tied to a concept list;
   - Coding scheme based on their type for coded attributes not tied to a concept list.

9. **Generating and Loading Configurations:**

   - *Missing Master Catalog Data*: Based on the schema file the Master catalog entries are scanned to check if any of the entries are missing. The master catalog entries scanned is of the ID type *ANY*. If they are found missing, those entries are created.
   - *Act Concept*: If the Message Type has any ambiguous complex types, Act Concept configurations for those complexes will be created in the server.
   - *Sender and Receiver Configurations*: Using the test Sender and Receiver IDs, records relating to sender and receiver configurations will be created.

10. **Persisting and Generating Instance Into and From HDR:**

    Invoke the HDR IMP Service to persist the MTK generated instance. HDR supports object attributes, datatype and its attributes, and vocabulary content.

    Invoke the HDR OMP service to generate the persisted MTK instance.

11. **Display the Status:**

    If the test is successful, a message indicating that the message type has been tested successfully is displayed. If some error occurs during the processes described above, error message is displayed.

    Hyperlinks to download the files generated by the process (including test status file, IMP, and OMP instances of the test message type) are displayed.

Using the MTK Services for Testing

For more information on the services you can use for testing, refer to the Using the MTK Services section.

Verify the Test Files

This step is the final step in testing the Message Type and is required to verify if the instance generated from MTK and the one generated from HDR using OMP are the same with the exception of known differences. Once the test process is successful, download the MTK and OMP generated instance using the hyperlinks displayed in the MTK Service. Use a file compare utility, to verify the instances.

Note: MTK does not provide an option or tools to compare MTK and OMP generated instances.

Set Up Message Types

Setting up Message Types is required if messages based on the Message Type have to be processed by HDR. To set up successfully tested Message Types in HDR:

- Loading Message Types
- Configuring IMP and OMP

Load Message Types

For messages to be processed, the files relating to the Message Type must be loaded onto the server. To load Message Type provide the following inputs:

- XSD and MIF files of the Message Types.
- XSD and MIF files of the CMETS referenced by the Message Types. Note that only custom CMETS must be provided. Do not provide any Oracle published CMETS if the Message Type references these.
- *cmetinfo.coremif* file must have a list of all the CMETS (Oracle published and custom) referenced by the Message Type.
- MTK Services: Refer to the Setting Up a New Message Type section for details on loading successfully tested custom Message Types onto a production server using Java API.

Use the MTK Services for Testing

For more information on the services you can use for loading Message Types, refer to the Using the MTK Services section.

Configure IMP and OMP

In IMP, configure the following:

- Trigger Event
- Interactions
- Sender
- Sender Interaction
- Sender Side Effect

In OMP, configure the following:

- Receiver
- Receiver vocabulary

- Act Concept

> **Note:** *Oracle Healthcare Data Repository Implementation and System Administrator Guide* for more information on configuring IMP and OMP.

Use the MTK Services

This section lists details of services that help to test custom Message Types and setup custom Message Types, along with code samples. For more information on MTK Services, refer to *Oracle Healthcare Data Repository Javadoc*.

Test Custom Message Types

To test a custom Message Type, perform the following steps:

1. Initialize the `ServiceLocator`.
2. Create `MtkTestService`.
3. Invoke the `testCustomMessageType()` method of `MtkTestService`.

If the Message Type testing is successful, the method returns an unique Test Id. If the testing is not successful, the method throws `CTBException` and the Test Id can be obtained from TEST_ID parameter of the `CTBException` instance. The files submitted for testing as well the ones generated by MTK will be loaded in a folder with the test ID name.

The following table summarizes the service and methods referenced by this section:

Service and Methods: Testing Message Type

Level	Detail
Package	`oracle.apps.ctb.message.mtk`
Class	`MtkTestService`
Methods	■ `downloadTestFilesForATest` ■ `getMatchingTestIds` ■ `getTestStatus` ■ `removeTestFilesForATest` ■ `testCustomMessageType`

> **Note:** *Oracle Healthcare Data Repository Javadoc* (`oracle.apps.ctb.message.mtk.MtkTestService`)

Example 10–9 Use the MTKTestService API

```
//Initializing Service Locator:
  ServiceLocator serviceLocator = ServiceLocator.getInstance();
  serviceLocator.login("username", "password");

   //Payload Xsd and Mif
   String  payloadXsdName = {"prpa_mt203000ht04.xsd"};
   String  payloadMifName = {"prpa_mt203000ht04.mif"};

   //Cmet Xsds and Mifs, referred by the Payload.
```

```
 String [] cmetXsdNames  =  { "coct_mt030202ht04.xsd",
"coct_mt030200ht04.xsd", "coct_mt150002ht02.xsd",
"coct_mt150000ht04.xsd", "coct_mt030202ht04.xsd"};

String [] cmetMifNames  =  { "coct_mt030202ht04.mif",
"coct_mt030200ht04.mif", "coct_mt150002ht02.mif",
"coct_mt150000ht04.mif", "coct_mt030202ht04.mif"};

  //CoreMif name.
 String coreMifName ="cmetinfo.coremif";

//Initializing SchemaInformation
SchemaInformation [] schemaInfo = new SchemaInformation [7];

//Populating SchemaInformation List with values.
for(int i=0; i< cmetXsdNames.length; i++)
{
    String xsdContent = readFileContent(cmetXsdNames[i]);
    String mifContent = readFileContent(cmetMifNames[i]);
    String artifactID = cmetXsdNames[i].replaceAll(".xsd", "");
    schemaInfo[i] = new SchemaInformation();
    schemaInfo[i].setXsd(xsdContent);
    schemaInfo[i].setMif(mifContent);
    schemaInfo[i].setSchemaType(SchemaInformation.CMET);
    schemaInfo.setArtifactID(artifactID);
}

//Populating payload.
String xsdContent = readFileContent(payloadXsdName);
String mifContent = readFileContent(payloadMifName);
String artifactID = payloadXsdName.replaceAll(".xsd","");
schemaInfo[i] = new SchemaInformation();
schemaInfo[i].setXsd(xsdContent);
schemaInfo[i].setMif(mifContent);
schemaInfo[i].setSchemaType(SchemaInformation.PAYLOAD);
schemaInfo.setArtifactID(artifactID);

//Populating CoreMif.
i++;
String coreMifContent = readFileContent(coreMifName);
schemaInfo[i] = new SchemaInformation();
schemaInfo[i].setMif(coreMifContent);
schemaInfo[i].setSchemaType(SchemaInformation.CORE_MIF);
schemaInfo[i]. setArtifactID(coreMifName);

String testDesc = "PRPA_MT203000HT04  - Message Type testing on 16-Jan-2008.";
boolean generateReport = true;
String testID = null;
MtkTestService mtkTestService = null;

//calling MtkTestService#testCustomMessageType()
try
{
   mtkTestService = serviceLocator.getMtkTestService();
   testID =
   mtkTestService.testCustomMessageType(schemaInfo,testDesc,
   generateReport);
  }
 catch(CTBException ctbException)
 {
```

```
      //Incase of CTBException, the TEST_ID can be retrieved from
      // CTBException instance.
      testID = (String)ctbException.getParameter("TEST_ID");
   }
```

Example 10–10 Searching for Test ID

```
//Initializing Service Locator:
  ServiceLocator serviceLocator = ServiceLocator.getInstance();
  serviceLocator.login("username", "password");
  //search key
  String searchKey = "%PRTS%MT%";
  String [] matchingTestIDs = null;
  MtkTestService mtkTestService = null;

  //invoke getMatchingTestIDs()  API.
  try
  {
    mtkTestService = serviceLocator.getMtkTestService();
    matchingTestIDs  = mtkTestService.getMatchingTestIDs(searchKey);
  }
  catch(CTBException ctbException)
  {
      //handle or re-throw the exception
  }
```

Example 10–11 Downloading Test ID Files

```
//Initializing Service Locator:
 ServiceLocator serviceLocator = ServiceLocator.getInstance();
 serviceLocator.login("username", "password");

// Input Test ID
String testID = "PRTS_MT000008TK01_611521202";
TestIDData testIDData = null;
 MtkTestService mtkTestService = null;

//Invoke downloadTestFilesForATest() API
 try
 {
   mtkTestService = serviceLocator.getMtkTestService();
   testIDData = mtkTestService.downloadTestFilesForATest(testID);
}
catch(CTBException ctbException)
{
  //handle or re-throw the exception
}

//retrieving data from TestID object
String messageTypeID = testIDData.getPayloadID();
String masterCatalogReportContent = testIDData.getMasterCatalogReport();
String sIDeEffectReportContent = testIDData. getSIDeEffectReport();
String actConceptReportContent = testIDData.getAmbiguousComplexTypeReport();
String userComments = testtIDData.getUserComment();
TestMessage[] testMessage = testIDData.getTestMessage();

 for (int i = 0; i < testMessage.length; i++)
 {
   String testMessage = testMessage[i].getTestMessage();
   String generatedMessage = testMessage[i].getGeneratedMessage();
 }
```

```
 SchemaInformation[] schemaInfo = testIDData.getSchemaInformation();
```

Example 10–12 Deleting Test ID Files

```
//Initializing Service Locator:
 ServiceLocator serviceLocator = ServiceLocator.getInstance();
serviceLocator.login("username", "password");

//Input Test ID
String testID = "PRTS_MT000008TK01_611521202";
MtkTestService mtkTestService = null;

//Invoke removeTestFilesForATest() API
try
{
mtkTestService = serviceLocator.getMtkTestService();
mtkTestService. removeTestFilesForATest(testID);
}
catch(CTBException ctbException)
{
//handle or re-throw the exception
}
```

Setting Up a New Message Type

To load and manage the successfully tested new Message Types onto a new environment,

1. Initialize ServiceLocator.
2. Create `MtkProductionLoadService`.

The following table summarizes the service and methods referenced by this section:

Service and Methods: Setting Up a New Message Type

Level	Detail
Package	`oracle.apps.ctb.message.mtk`
Class	`MtkProductionLoadService`
Methods	■ `createCompositeMessageSchema` ■ `deleteInteractionSchema` ■ `deleteSchemas` ■ `fetchCompositeMessageShemasByPayload` ■ `fetchCustomCMETsForPayload` ■ `fetchSchemas` ■ `loadCustomMessageTypeToProductionServer` ■ `loadSchemas`

Note: *Oracle Healthcare Data Repository Javadoc* (`oracle.apps.ctb.message.mtk.MtkProductionLoadService`)

Example 10–13 Search for Message Types

```
//Initializing Service Locator:
 ServiceLocator serviceLocator = ServiceLocator.getInstance();
```

```
serviceLocator.login("username", "password");

//Message Type ID
String [] messageTypeID= { "PRPA_MT203000HT04"};
SchemaInformation [] schemaInfo = null;

try
 {
  //Getting MtkProductionLoadService
  mtkProductionLoadService = serviceLocator.getMtkProductionLoadService();

 //Calling fetch API
 schemaInfo = mtkProductionLoadService.fetchSchemas (messageTypeID) ;
}
catch(CTBException ctbException)
{
  //…handle or re-throw the exception
}

if (schemaInfo.length == 0 ) // If  Message Type not already uploaded.
{
//…Invoke loadCustomMessageTypeToProductionServer() API
}
```

Example 10–14 Loading Custom Message Types

```
//Initializing Service Locator:
 ServiceLocator serviceLocator = ServiceLocator.getInstance();
 serviceLocator.login("username", "password");

   //Payload Xsd and Mif
 String  payloadXsdName = {"prpa_mt203000ht04.xsd"};
 String  payloadMifName = {"prpa_mt203000ht04.mif"};

 //Cmet Xsd and Mif, referred by the Payload.
 String [] cmetXsdNames  =  { "coct_mt030202ht04.xsd", "coct_mt030200ht04.xsd",
"coct_mt150002ht02.xsd", "coct_mt150000ht04.xsd", "coct_mt030202ht04.xsd"};
String [] cmetMifNames  =  { "coct_mt030202ht04.mif", "coct_mt030200ht04.mif",
"coct_mt150002ht02.mif", "coct_mt150000ht04.mif", "coct_mt030202ht04.mif"};

 //CoreMif name.
  String coreMifName ="cmetinfo.coremif";

 //Initializing SchemaInformation
  SchemaInformation [] schemaInfo = new SchemaInformation [7];

  //Populating SchemaInformation List with values.
  for(int i=0; i< cmetXsdNames.length; i++)
{
      String xsdContent = readFileContent(cmetXsdNames[i]);
      String mifContent = readFileContent(cmetMifNames[i]);
      String artifactID = cmetXsdNames[i].replaceAll(".xsd", "");
      schemaInfo[i] = new SchemaInformation();
      schemaInfo[i].setXsd(xsdContent);
      schemaInfo[i].setMif(mifContent);
      schemaInfo[i].setSchemaType(SchemaInformation.CMET);
      schemaInfo.setArtifactID(artifactID);
  }

 //Populating payload.
```

```
  String xsdContent = readFileContent(payloadXsdName);
  String mifContent = readFileContent(payloadMifName);
  String artifactID = payloadXsdName.replaceAll(".xsd","");
  schemaInfo[i] = new SchemaInformation();
  schemaInfo[i].setXsd(xsdContent);
  schemaInfo[i].setMif(mifContent);
  schemaInfo[i].setSchemaType(SchemaInformation.PAYLOAD);
  schemaInfo.setArtifactID(artifactID);

//  Populating CoreMif.
  i++;
 String coreMifContent = readFileContent(coreMifName);
 schemaInfo[i] = new SchemaInformation();
schemaInfo[i].setMif(coreMifContent);
 schemaInfo[i].setSchemaType(SchemaInformation.CORE_MIF);
schemaInfo[i]. setArtifactID(coreMifName);

 MtkProductionLoadService mtkProductionLoadService = null;
  try
  {
     //Getting MtkProductionLoadService
     mtkProductionLoadService = serviceLocator. getMtkProductionLoadService();
     //Calling upload API
     mtkProductionLoadService.loadCustomMessageTypeToProductionServer(schemaInfo);
  }
 catch(CTBException ctbException)
 {
  //handle or re-throw the exception
 }
```

Example 10–15 Fetch Custom CMETs

```
String messageTypeID= "POXX_MT111000HT02";
SchemaInformation [] schemaInfo = null;
MtkProductionLoadService mtkProductionLoadService = null;

try
{
//Getting MtkProductionLoadService
mtkProductionLoadService = serviceLocator.getMtkProductionLoadService();
//Calling fetch API
schemaInfo = mtkProductionLoadService. FetchCustomCMETsForPayload
(messageTypeID) ;
}
catch(CTBException ctbException)
{
//…handle and re-throw the exception
}
```

Example 10–16 Fetch Composite Message Schema

```
String messageTypeID= "POXX_MT111000HT02";
SchemaInformation [] schemaInfo = null;
try
{
//Getting MtkProductionLoadService
mtkProductionLoadService = serviceLocator.getMtkProductionLoadService();
//Calling fetch API
schemaInfo = mtkProductionLoadService. FetchCompositeMessageShemasByPayload
(messageTypeID) ;
}
```

```
catch(CTBException ctbException)
{
//…handle and re-throw the exception
}
```

> **Note:** SchemaInformation object will not contain MIF content for an Interaction Schema.

Example 10–17 Deleting a Schema

```
String [] schemas = {  "coct_mt930002ht03.xsd", "poxx_mt111000ht02.xsd"};

MtkProductionLoadService mtkProductionLoadService = null;

try
{
   //Getting MtkProductionLoadService
    mtkProductionLoadService = serviceLocator.getMtkProductionLoadService();
    //Calling delete API
    mtkProductionLoadService. deleteSchemas (schemas) ;
}
catch(CTBException ctbException)
{
  //…handle or re-throw exception
 }
```

Sample Exercise Using MTK Services

- Prerequisites and Tools
- Creating a New Message Type or Modifying an Existing Message Type
- Generating Schema (XSD) and MIF Files for the Modified Messages

Prerequisites and Tools

The following tools are used in the process of testing a custom message and generating message instances for the custom messages.

- Microsoft® Visio 2002 SP - 2 version 10.0.6871
- HL7 RIM repository version 2.14.1
- HL7 RMIM Designer version 4.3.2
- HL7 Schema Generator version 3.0.4

> **Note:** The steps described in the following sections were performed using the tools listed above, and may not produce expected results if executed using versions other than those specified above.

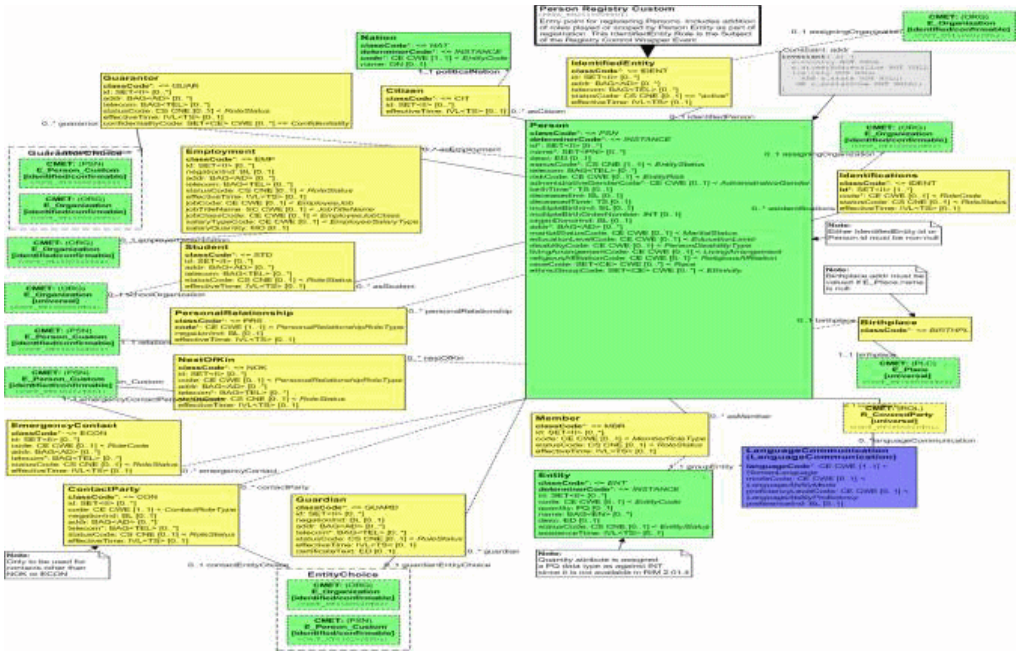
Creating a New Message Type or Modifying an Existing Message Type

For the purpose of this sample exercise, a modified version of the HDR-published domain Message Type Person Registry (PRPA_RM201000HT03) and a modified version of the HDR-published CMET E_Person (COCT_RM030202HT04) are used.

1. Use the RMIM Designer to modify the CMET E_Person (COCT_RM030202HT04) as follows:

1. In the focal class Person, add an attribute educationLevelCode: CE CWE [0..1] < EducationLevel
2. Change the name of the CMET to E_Person_Custom and the CMET code ID to COCT_RM030202TK01

Figure 10–6



3. Validate and save the Visio file as COCT_RM030202TK01.vsd. When you save the Visio file, the RMIM designer will automatically create the COCT_RM030202TK01.xml file.

2. In the CMETInfo.txt file that is referred by RMIM designer, include an entry for the custom CMET as follows:

   `E_Person_Custom,COCT_MT030202TK01,PSN,Entity,Custom Person CMET for testing MTK,identified/confirmable.`

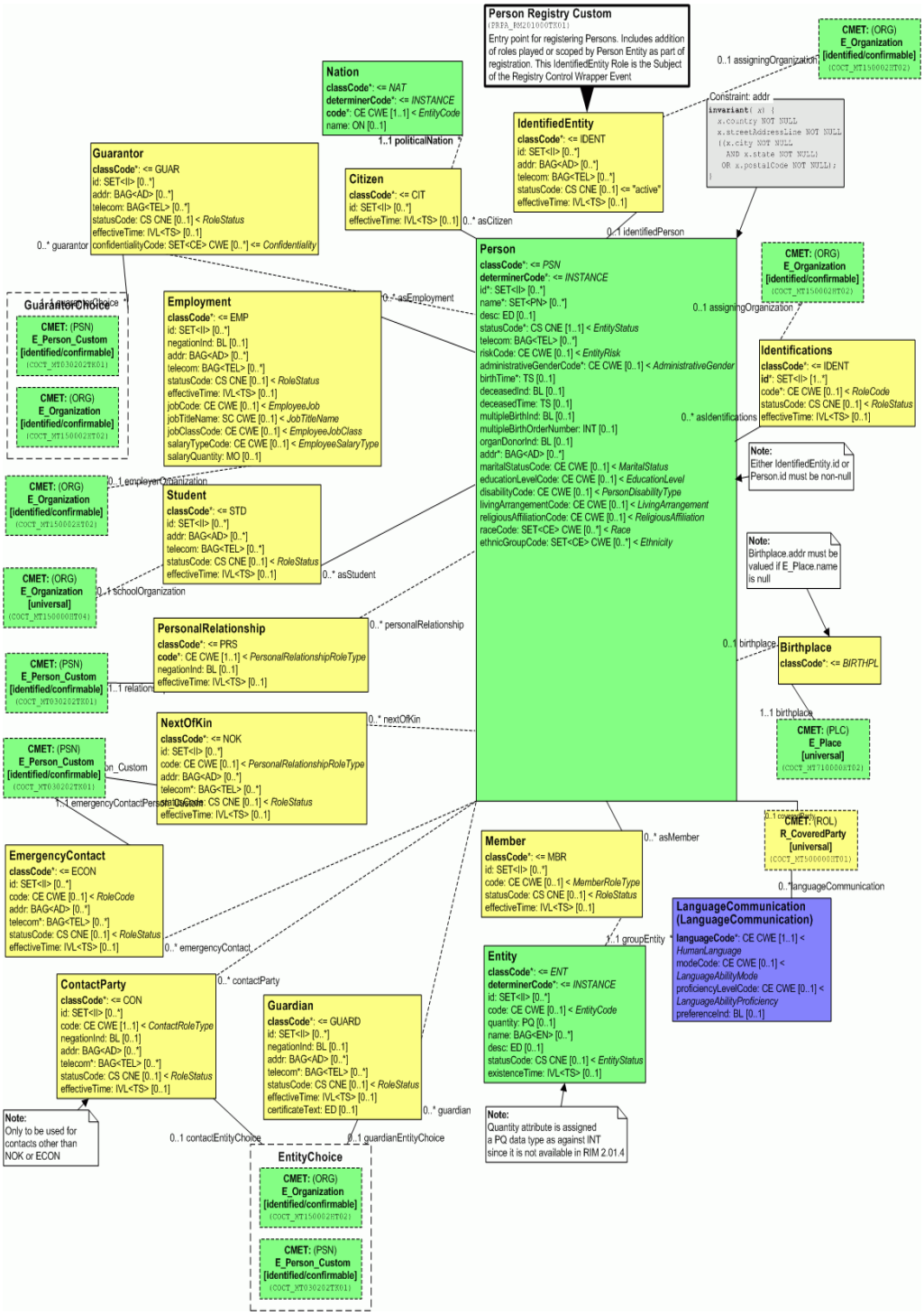
3. Modify the message domain Person Registry (PRPA_RM201000HT03) as follows:

   1. In the class Employment (Person => Employment), add the following attributes:

```
salaryTypeCode: CE CWE [0..1] < EmployeeSalaryType
salaryQuantity: MO [0..1]
```

   2. Replace all references to the `CMET E_Person (PSN) [identified/confirmable] (COCT_MT030202HT04)` with `E_Person_Custom (PSN) [identified/confirmable] (COCT_MT030202TK01).`
   3. Change the name of the RMIM to *Person Registry Custom* and the RMIM IDcode to *PRPA_RM201000TK01.*

Figure 10–7 Custom Message Person_Registry_Custom (PRPA_RM201000TK01)



4. Validate and save the Visio file as 'PRPA_RM201000TK01.vsd'.
5. Verify that Visio automatically generates a file 'PRPA_RM201000TK01.xml'.

Generate Schema (XSD) and MIF Files for the Modified Messages

Use the Schema Generator generate schema and MIF files for the modified messages as follows:

1. Copy the files PRPA_RM201000TK01.xml and COCT_RM030202TK01.xml in the following folder:

   `<Schema Generator Root Folder>\InputFiles\VisioModelXmlFiles\`

   Note that this folder must contain the xml files corresponding to all the CMETs that are used directly or indirectly by the message PRPA_RM201000TK01.

2. Modify the file <Schema Generator Root Folder>\InputFiles\CommonSourceFiles\cmetInfoExport.txt to include following entry:

   ```
   E_Person_Custom,COCT_MT030202TK01,PSN,Entity,Custom Person CMET for testing MTK,identified/confirmable
   ```

3. Ensure that the file `<Schema Generator Root Folder>\InputFiles\configuration.txt` includes the following setting:

   ```
   generateSchemas = true
   ```

4. Execute the file `<Schema Generator Root Folder>\runlogged.bat`

5. Upon completion of execution, view the file `<Schema Generator Root Folder>\OutputFiles\generator.log` for any errors and rectify them.

6. After successful execution:

   - Schema Generator should generate the schema and MIF files for custom messages in following folders:

     `<Schema Generator Root Folder>\OutputFiles\Schemas\`

     `<Schema Generator Root Folder>\OutputFiles\MIF\`

     Verify that Schema Generator has generated following files:

     PRPA_MT201000TK01.xsd, PRPA_MT201000TK01.mif

     COCT_MT030202TK01.xsd, COCT_MT030202TK01.mif

   - Schema Generator should automatically update the following file:

     `<Schema Generator Root Folder>\OutputFiles\MIF\cmetinfo.coremif`

     and include a new entry for the custom CMET 'COCT_RM030202TK01'

     Verify that appropriate details of the custom CMET are included in the file cmetinfo.coremif

Generate Test Messages Using MTK Service

Refer to the Use the MTK Services section to test the newly created message type artifacts and retrieve the test messages generated by MTK test API.

Generating Custom Message Types

You can create custom artifacts either by using RMIM or by using the artifacts published by HL7. Ensure that you give valid names for the custom messages. You must ensure that:

- Realm code in the artifact ID must be other than *HT*.

- CMET names must be different from the Oracle-published CMET names.

Note:

- Oracle will not provide tools to design artifacts and its associated files. You have to use the tools provided by HL7 or create your own tools.
- The tools used to generate schema must also generate MIF files.
- MTK is tested with artifacts created on:
    - HL7 RIM repository v2.14.1
    - HL7 RMIM Designer v4.3.2
    - RoseTree v4.0.12
    - HL7 Schema Generator v3.0.4

Generating Custom Artifacts Using RMIM

RMIMs can be created in one of the following ways:

- Modifying the RMIM of the Oracle-published Message Types.
- Downloading and modifying the RMIM of the HL7-published normative Message Types.
- Creating a new RMIM for a Message Type RMIM.

Once the RMIMs are created, save the RMIM in a repository. Provide the XML of the RMIM as input to the tools generating the Schema.

Generating Custom Artifacts Without Using RMIM

To generate a custom Artifact without using an RMIM, you can do one of the following:

- Modify Oracle-published XSD and MIF files.
- Download and modify HL7-published XSD and MIF files.

Logic for Instance Generation

MTK uses the following logic to generate a test instance:

- Custom Message Types and CMETs are filled with default data even if they are optional items.
- CMETs that are mandatory are filled with data irrespective of whether they are Oracle published or custom CMETs.
- Items (Classes, CMETs, and Attributes) in custom artifacts (Message Type, CMETs) are handled using the following logic:

  Mandatory Items

    - Classes and CMETs with {1..1, 1..N, and 1..*} cardinality are filled only once.
    - Recursions with {1..1, 1..N, and 1..*} cardinality are filled once and only one level is filled.
    - Coded Attributes with {1..1, 1..N, and 1..*} cardinality are filled only once.
    - Non-coded attributes with {1..1} cardinality are filled only once.

- Non-coded attributes with {1..N and 1..*} cardinality are filled twice.
- Choices with 0..* or 1..* are repeated as many number of times as the number of objects in the choice to fill all the objects within the choice.

Optional Items

- Classes and CMETs with {0..1, 0..N, and 0..*}, cardinality are filled once.
- Recursions with {0..1, 0..N, and 0..*} cardinality are filled once and only one level is filled.
- Attributes (Coded and Non-Coded) with {0..1, 0..N, and 0..*} cardinality are not filled.
- Choices with {0..1, 0..N, and 0..*}cardinality are filled only once.

- Items (Classes, CMETs, and Attributes) relating to Oracle-published CMETs are handled using the following logic:

  Mandatory Items

  - Classes and CMETs with {1..1, 1..N, and 1..*}, cardinality are filled only once.
  - Within the mandatory CMETs, only mandatory classes with {1..1, 1..N, and 1..*} cardinality are filled with data, and only the attributes with {1..1, 1..N, and 1..*}cardinality are populated with data.
  - Recursions with {1..1, 1..N, and 1..*} cardinality are filled once and only one level is filled.
  - Coded Attributes with {1..1, 1..N, and 1..*} cardinality are filled only once.
  - Non-coded attributes with {1..1} cardinality are filled only once.
  - Non-coded attributes with {1..N and 1..*} cardinality are filled twice.
  - Choices with {1..1, 1..N, and 1..*}cardinality are filled only once.

  Optional items

  - Classes and CMETs with {0..1, 0..N, and 0..*} cardinality are not filled.
  - Recursions with {0..1, 0..N, and 0..*} cardinality are not filled.
  - Attributes (Coded and Non-Coded) with {0..1, 0..N, and 0..*} cardinality are not filled.
  - Choices with {0..1, 0..N, and 0..*} cardinality are not filled.

- Data for Clone class attributes are populated using the following logic:
  - Data for the structural attributes is populated from the respective schema.
  - A unique test ID is assigned for each clone class object.
  - For coded attributes bound to concept lists, data is taken from the respective concept list within ETS and populated. Note that if the concept list is empty, an error is thrown
  - For coded attributes not bound to concept lists, a value from the appropriate HL7 coding scheme is populated based on their types (For example, code attribute of Act class has code from `ActCode` Code System)
  - Non-Coded attributes like address, name, and others are populated with data from the sample snippets seeded in HDR.
  - Attributes with *Any* datatype are defaulted to **ST** and a default value is populated.

- Message Wrapper and Control Act wrapper:
  - Data for the Message wrapper, Act Payload Control wrapper, and Role Payload Control wrapper is seeded in HDR.
  - The Interaction ID, Trigger event code, and Sender and Receiver II attribute values are generated during each instance generation.

Note:

- The objective of instance generation is to cover all the objects of custom artifacts (Message Type and CMETs) irrespective of whether they are optional or not.
- Within an instance, if the object of external artifact is covered once, the same object is not repeated unless it is mandatory.
- External artifacts with Roles having entity choices either as Player or Scoper will always result in more than one instance. Each of the Player/Scoper choice is included in one of the instances.
- The order of the objects in the instance generated depend on their order in the schema file.

Master Catalog and Side Effect Configuration Reports

Master Catalog Reports

This section details the validation logic followed by MTK for Master Catalog validations, and also the logic used to generate the Master Catalog report.

Master Catalog Validation Rules

MTK follows the rules listed below to validate the schema against the Master Catalog entries to verify if the entries exist:

General Rules

- If the object in the schema has a code attribute,
  - If the code attribute is *mandatory* in the schema, MTK checks if an entry for that object exists in the Master Catalog (MC) configuration with Code Type as ANY. If it is not present, MTK includes that object in the MC Report.
  - If the code attribute is *optional* in the schema, MTK checks if an entry for that object exists in the MC Configuration with Code Type as ANY. If it does not exist, MTK includes that object in the MC Report. In addition, MTK also checks if an entry exists for that object in the MC Configuration with Code Type as NULL. It it does not exist, MTK includes that object in the MC Report.
- If the object in the schema does **not** have a code attribute, MTK checks if an entry for that object exists in the MC Configuration with Code Type as NULL. If it is not present, MTK includes that object in the MC Report.

Act

The general rules apply to all objects in the schema that are Acts.

Role: Non-owned Role without Entities

The general rules apply to all objects in the schema that are Non-owned Roles without Entities.

Role: Non-owned Role with Entities

In addition to the general rules, MTK also checks if the corresponding Entity entry exists for the Non-owned Role in the MC configuration. The general rules apply for checking an individual Entity entry in the MC Configuration. If the entry does not exist, MTK includes that object in the MC Report.

ROLE: Owned Role without Non-Owning Entity

For Owned Role, in addition to the general rules, MTK also checks if the corresponding Owning Entity entry exists for the Owned Role in the MC Configuration. The general rules apply for checking an individual Entity entry in the MC configuration. If the entry does not exist, MTK includes all the corresponding objects (both missing role and the missing entities) in the MC Report.

ROLE: Owned Role with Non-Owning Entity

For Owned Role, in addition to the general rules, MTK also checks if the corresponding Owning and Non-Owning Entity for the Owned Role has appropriate entries in the MC Configuration. The general rules apply for checking an individual Entity entry in the MC Configuration. If the entry does not exist, MTK includes all corresponding objects (both missing role and the missing entities) in the MC Report.

> **Note:** In case of Owned Roles, MTK also checks if the entry in the ROLE_OWNER_CODE (in the MC Configuration) has *P* or *S*, depending on whether the Owning Entity is a Player or a Scoper. If the entry does not exist, MTK includes that object in the MC Report.

Master Catalog Reporting Logic

MTK populates a report based on the rules described in the Master Catalog Validation Rules section, and then based on the value in the Report column (Yes or No). If the Report column contains the value Yes, the following logic is applied while creating an entry in the Master Catalog report:

Act

- **With mandatory code attribute**: Create an entry with code type *ANY*.
- **With optional code attribute**: Create entries with code type *ANY* and *Null*.
- **Without code attribute**: Create an entry with code type *Null*.

Non-Owned Role with Entities

- **With mandatory code attribute:** Create an entry with code type ANY, and with appropriate Entity entries.
- **With optional code attribute**: Create entries with code type *ANY* and *Null*, and with appropriate Entity entries.
- **Without code attribute**: Create an entry with code type *Null*, and with appropriate Entity entries.

Non-Owned Role without Entities

- **With mandatory code attribute**: Create an entry with code type ANY, and without any entities.
- **With optional code attribute**: Create entries with code type *ANY* and *Null*, and without any entities.
- **Without code attribute**: Create an entry with code type *Null*, and without any entities.

Owned Role with non-owning Entity

- **With mandatory code attribute**: Create an entry with code type *ANY*, and with appropriate non-owning Entity entry.
- **With optional code attribute**: Create entries with code type *ANY* and *Null*, and with appropriate non-owning Entity entry.
- **Without code attribute**: Create an entry with code type *Null*, and with appropriate non-owning Entity entry.

Entity

- **With mandatory code attribute**: Create an entry with code type *ANY*.
- **With optional code attribute**: Create entries with code type *ANY* and *Null*.
- **Without code attribute**: Create an entry with code type *Null*.

Side Effect Configuration Report

MTK applies the following rules while creating the Side Effect Configuration report:

- **Act**: Create or Overlay
- **Non-Owned Role without Entities**: Create or Overlay.
- **Non-Owned Role with Entities**: Create or Overlay for both Role and its entities.
- **Owned Role without non-owning Entity**: No entry.
- **Owned (Type II) Role with non-owning Entity**: Null for Role and owning Entity. Create or Overlay for non-owning Entity.

MTK Message Types Construct Processing

- Constructs in External Artifacts (Message Type and CMETS)
- Constructs Within Oracle Published CMETS

> **Note:** The order of the objects in the instance generated depend on their order in the schema file. The objects listed in the examples might change when compared to actual instances.

Constructs in External Artifacts (Message Type and CMETs)

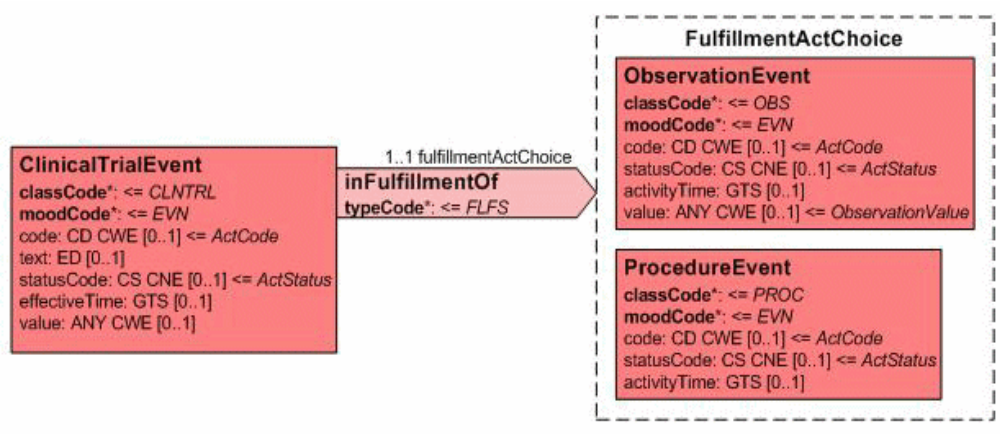
Act> ActChoice with {0..1 or 1..1 or 0..n or 1..n}

MTK generates *n* instances where n is the number of objects in the choice (and is not same as *). MTK generates two instances if a message type that has the construct described in Figure E.1. For each object within the choice, a message instance is generated.

- First message instance has ClinicalTrialEvent > inFulfillmentOf > ObservationEvent

- Second message instance has ClinicalTrialEvent > inFulfillmentOf > ProcedureEvent

Figure 10–8 Act > ActChoice with {1..1}

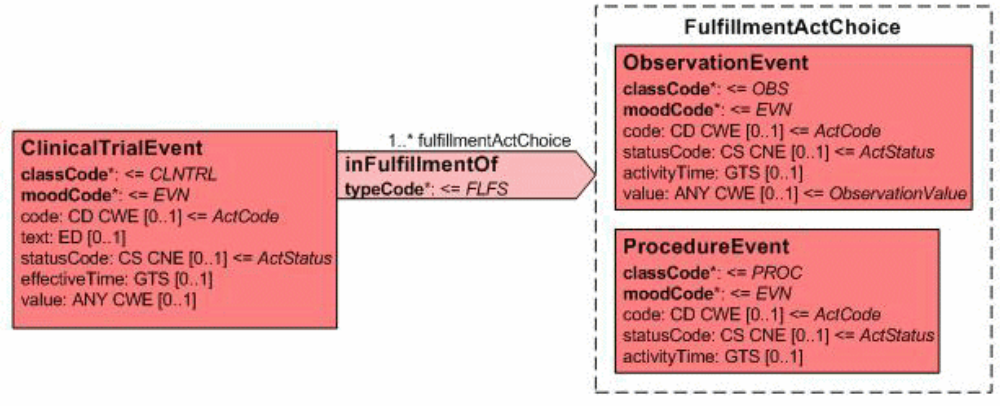


Act> ActChoice with {0..* or 1..*}

MTK generates one instance with two *inFulfillmentOf* ActRelationship, if a message type has the construct shown in Figure E.2. For each object within the choice, an inFulfillmentOf ActRelationship is generated.

- First instance of inFulfillmentOf ActRelationship would have ClinicalTrialEvent > inFulfillmentOf > ObservationEvent
- Second instance of inFulfillmentOf ActRelationship would have ClinicalTrialEvent > inFulfillmentOf > ProcedureEvent

Figure 10–9 Act > ActChoice with {1..*}



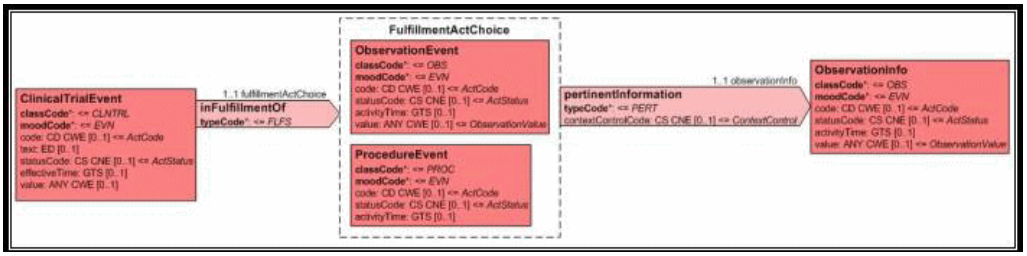
Act> ActChoice with {0..1 or 1..1 or 0..n or 1..n}> Act with {0..1 or 1..1 or 0..n or 1..n or 0..* or 1..*}

MTK generate n instances where n is the number of objects in the first choice. MTK generate two instances if a message type has the construct described in Figure E.3. For each object within the choice, a message instance is generated along with the target *pertinentInformation* ActRelationship.

- First message instance has ClinicalTrialEvent > inFulfillmentOf > ObservationEvent > pertinentInformation > ObservationInfo

- Second message instance has ClinicalTrialEvent > inFulfillmentOf > ProcedureEvent > pertinentInformation > ObservationInfo

Figure 10–10 Act > Choice with {1..1} > Act with {1..1}

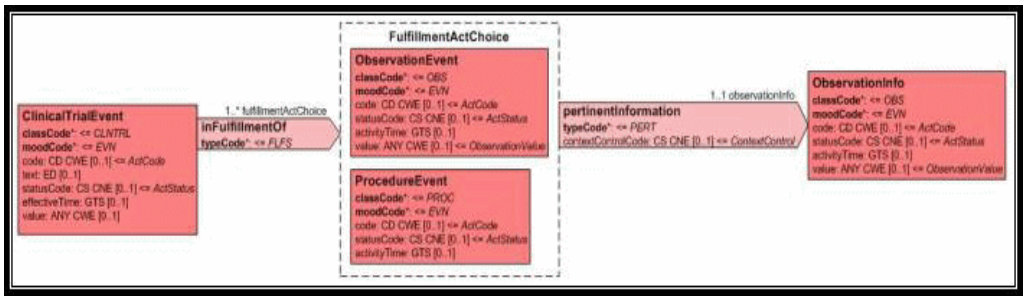


Act > ActChoice with {0..* or 1..*} > Act with {0..1 or 0..* or 0..n or 1..n or 1..1 or 1..*}

MTK generates one instance with two *inFulfillmentOf* ActRelationship, if a message type has the construct described in Figure E.4. For each object within the choice, an inFulfillmentOf ActRelationship is generated along with one target *pertinentInformation* ActRelationtionship.

- First instance of inFulfillmentOf ActRelationship has ClinicalTrialEvent > inFulfillmentOf > ObservationEvent > pertinentInformation > ObservationInfo
- Second instance of inFulfillmentOf ActRelationship has ClinicalTrialEvent > inFulfillmentOf > ProcedureEvent > pertinentInformation > ObservationInfo

Figure 10–11 Act > ActChoice with {1..*} > Act with {1..1}

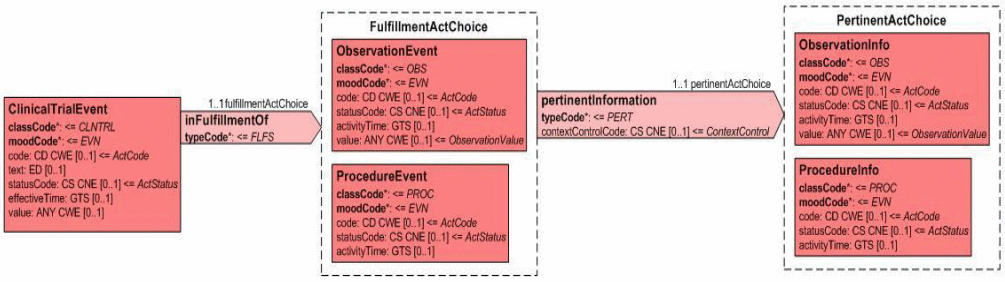


Act > ActChoice with {0..1 or 1..1 or 0..n or 1..n} > ActChoice with {0..1 or 1..1 or 0..n or 1..n}

MTK generates n instances where n is the number of objects in the first choice. MTK generates two instance if a message type has the construct described in Figure E.5. For each object within the fulfillmentActChoice, a message instance is generated along with an object from target pertinentActChoice.

- First message instance has ClinicalTrialEvent > inFulfillmentOf > ObservationEvent > pertinentInformation > ObservationInfo
- Second message instance has ClinicalTrialEvent > inFulfillmentOf > ProcedureEvent > pertinentInformation > ProcedureInfo

Figure 10–12 Act > ActChoice with {1..*} > ActChoice with {1..1}

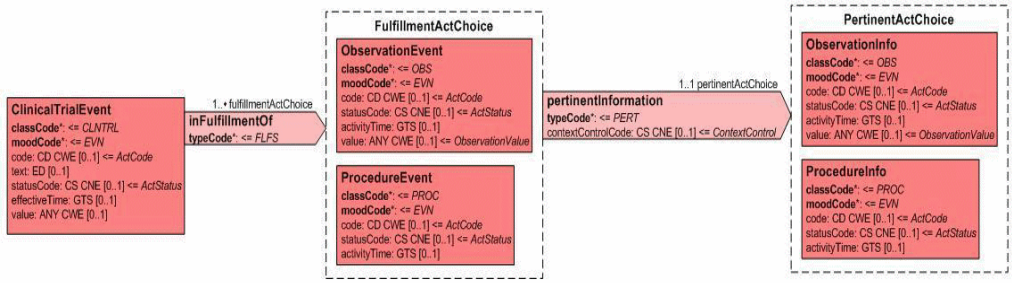


Act > ActChoice with {0..* or 1..*} > ActChoice with {0..1 or 1..1 or 0..n or 1..n}

MTK generate one instance with two *inFulfillmentOf* ActRelationship, if a message type has the construct described in Figure E.6. For each object within the choice, an inFulfillmentOf ActRelationship is generated along with an object from the target choice.

- First instance of inFulfillmentOf ActRelationship has ClinicalTrialEvent > inFulfillmentOf > ObservationEvent > pertinentInformation > ObservationInfo
- Second instance of inFulfillmentOf ActRelationship has ClinicalTrialEvent > inFulfillmentOf > ProcedureEvent > pertinentInformation > ProcedureInfo

Figure 10–13 Act > ActChoice with {1..*} > ActChoice with {1..1}

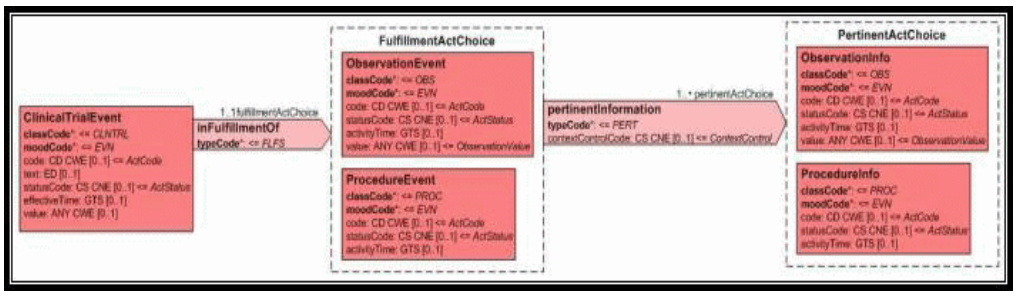


Act > ActChoice with {0..1 or 1..1 or 0..n or 1..n} > ActChoice with {0..* or 1..*}:

MTK generates n instances where n= number of objects is in the first choice. MTK generates two instance if a message type has the construct described in Figure E.7. For each object within the inFulfillmentOf, a message instance is generated along with two pertinentInformation ActRelationships.

- First message instance has ClinicalTrialEvent > inFulfillmentOf > ObservationEvent:
-
  1. pertinentInformation > ObservationInfo
  2. pertinentInformation > ProcedureInfo
- Second message instance has ClinicalTrialEvent > inFulfillmentOf > ProcedureEvent:
  1. pertinentInformation > ObservationInfo
  2. pertinentInformation > ProcedureInfo

Figure 10–14 Act > Choice with {1..1} > ActChoice with {1..*}

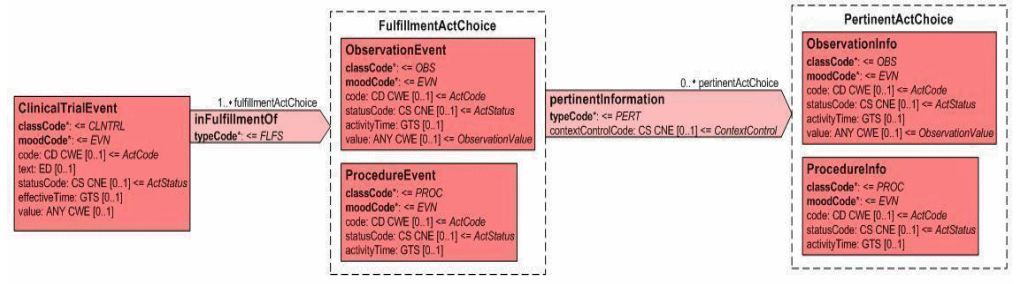


Act > ActChoice with {0..* or 1..*} > ActChoice with {0..*}

MTK generates one instance with two inFulfillmentOf ActRelationship, if a message type has the construct described in Figure E.8. The first inFulfillmentOf ActRelationship instance has the objects from the target pertinentActChoice. The other inFulfillmentOf ActRelationship instance does not have it.

- First instance of inFulfillmentOf ActRelationship has ClinicalTrialEvent > inFulfillmentOf > ObservationEvent:
  1. pertinentInformation > ObservationInfo
  2. pertinentInformation > ProcedureInfo
- Second instance of fulfillmentActChoice ActRelationship has ClinicalTrialEvent > inFulfillmentOf > ProcedureEvent

Figure 10–15 Act > ActChoice with {1..*} > ActChoice with {0..*}



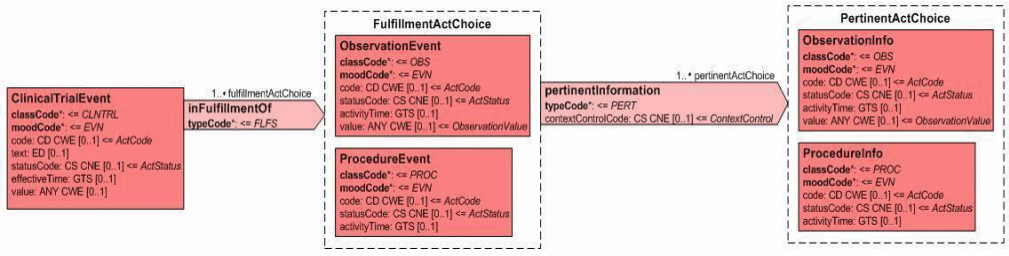
Act > ActChoice with {0..* or 1..*} > ActChoice with {1..n or 1..*}

MTK generates one instance with two inFulfillmentOf ActRelationship, if a message type has the construct described in Figure E.9. For each object within FullfillmentActChoice, a inFulfillmentOf ActRelationship is generated along with the object from the target pertinentActChoice.

- First instance of inFulfillmentOf ActRelationship has ClinicalTrialEvent > inFulfillmentOf > ObservationEvent:
-
  1. pertinentInformation > ObservationInfo
  2. pertinentInformation > ProcedureInfo
- Second instance of inFulfillmentOf ActRelationship has ClinicalTrialEvent > inFulfillmentOf > ProcedureEvent (one of the below):
  1. pertinentInformation > ObservationInfo

2. pertinentInformation > ProcedureInfo

Figure 10–16 Act > ActChoice with {1..*} > ActChoice with {1..*}

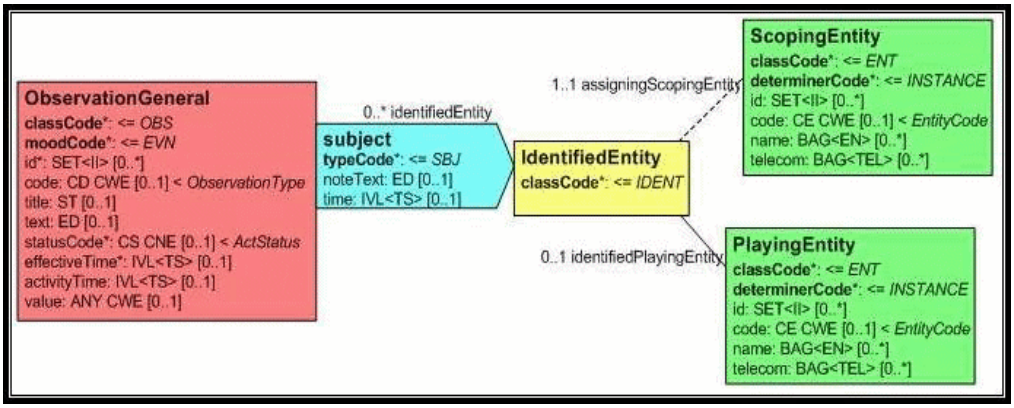


Act > Role with {0..1 or 0..n or 0..* or 1..1 or 1..n or 1..*} > Player with {0..1 or 1..1} and Scoper with {0..1 or 1..1}

MTK generates only one instance if a message type has the construct described in Figure E.10. The message instance has:

- Message Instance: ObservationGeneral > subject > IdentifiedEntity > ScopingEntity and PlayingEntity

Figure 10–17 Act > Role with {0..*} > Player with {0..1} and Scoper with {1..1}



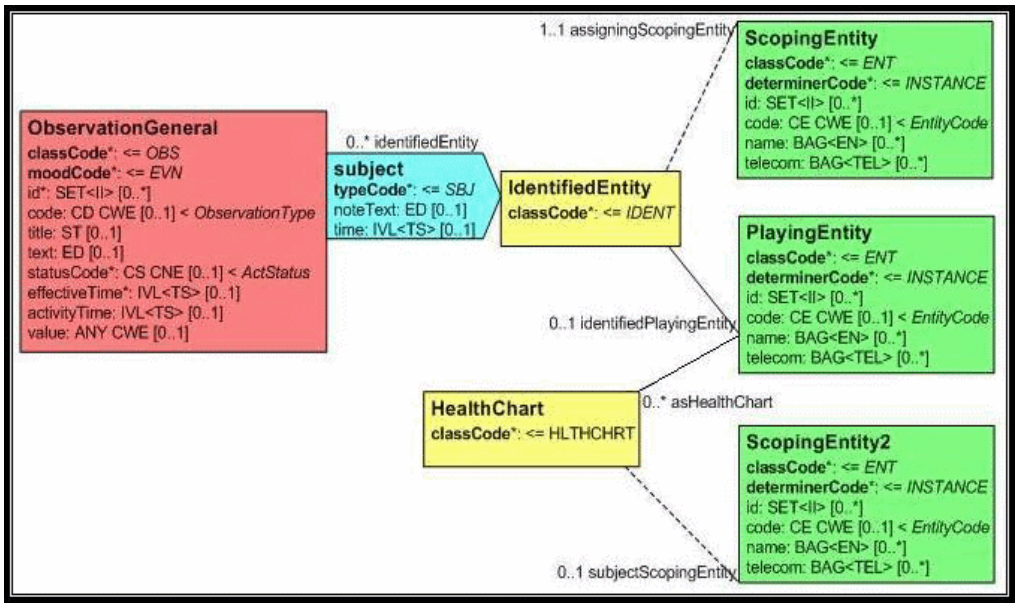
Act > Role with {0..1 or 0..n or 0..* or 1..1 or 1..n or 1..*} > Player with {0..1 or 1..1} and Scoper with {0..1 or 1..1} > Playedrole with {0..1 or 0..n or 0..* or 1..1 or 1..n or 1..*} > Scoper with {0..1 or 1..1}

MTK generates only one instance if a message type has the construct described in Figure E.11. The instance generated has the following:

- Message instance: ObservationGeneral > subject > IdentifiedEntity > ScopingEntity and Playingentity<

  1. PlayingEntity > HealthChart > ScopingEntity2

Figure 10–18 Act > Role with {0..*} > Player with {0..1} and Scoper with {1..1} > Playedrole with {0..*} > Scoper with {0..1}

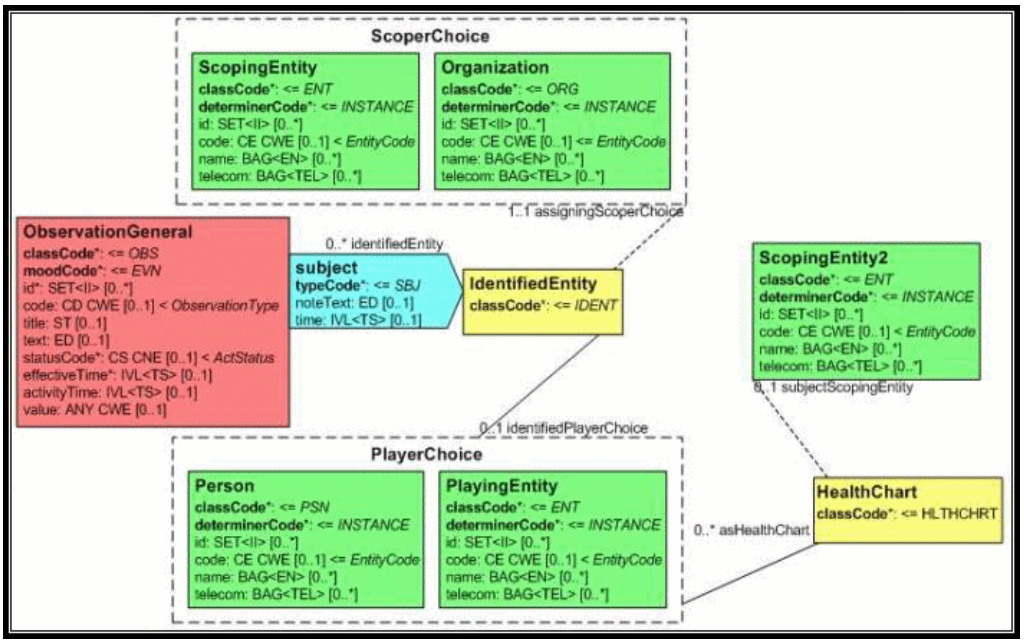


Act > Role with {0..1 or 0..n or 0..* or 1..1 or 1..n or 1..*} > PlayerChoice with {0..1 or 1..1} and ScoperChoice with {0..1 or 1..1} > PlayedRole with {0..1 or 0..n or 0..* or 1..1 or 1..n or 1..*} > Scoper with {0..1 or 1..1}

MTK generates n instances where n is the maximum number of entities present in a single choice. In the construct described in Figure E.12, MTK generates two instances. The instances generated might be:

- First Instance: ObservationGeneral > subject > IdentifiedEntity > ScopingEntity and PlayingEntity
  1. Playingentity > HealthChart > ScopingEntity2
- Second Instance: ObservationGeneral > subject > IdentifiedEntity > Organization and Person
  1. Person > HealthChart > ScopingEntity2

Figure 10–19 Act > Role with {0..*} > PlayerChoice with {0..1} and ScoperChoice with {1..1} > Playedrole with {0..*} > Scoper with {0..1}



Note: Even if the cardinality of participation is 0..*, MTK generates multiple instances of the above structure to incorporate all the entities. This behavior is different from ActRelationships.

Act> Role with {0..1 or 0..n or 0..* or 1..1 or 1..n or 1..*}> PlayerChoice with {0..1} and ScoperChoice with {0..1 or 1..1}> PlayedRole with {0..1 or 0..n or 0..* or 1..1 or 1..n or 1..*}> Scoper with {0..1 or 1..1}

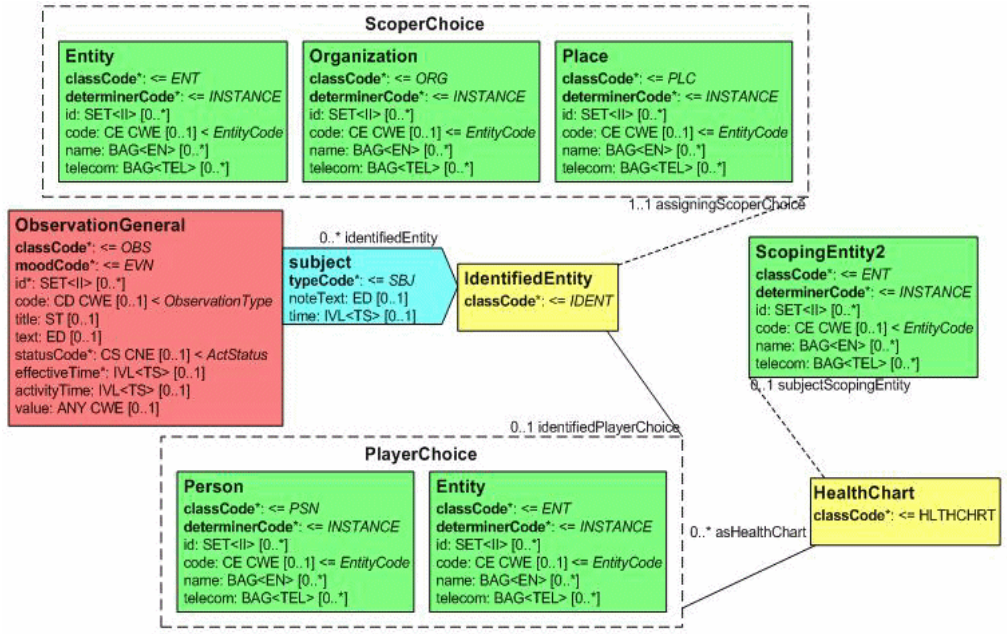
MTK generates n instances where n is the number of entities in the choice with Max entities. In the construct described in Figure E.13, MTK generates three instances:

- Instances with player entities has the Owned role
- One of the instances has only the scoper for IdentifiedEntity role. This instance does not have any Player as (i) the cardinality is 0..1 (ii) both the player entities would have been covered in the previous two messages.

The instance generated might be:

- First Instance: ObservationGeneral > subject > IdentifiedEntity > Entity and Person
  1. Person > HealthChart > ScopingEntity2
- Second Instance : ObservationGeneral > subject > IdentifiedEntity > Organization and Entity
  1. Entity > HealthChart > ScopingEntity2
- Third Instance : ObservationGeneral > subject > IdentifiedEntity > Place

Figure 10–20 Act > Role with {0..*} > PlayerChoice with {0..1} and ScoperChoice with {1..1} > Playedrole with {0..*} > Scoper with {0..1}



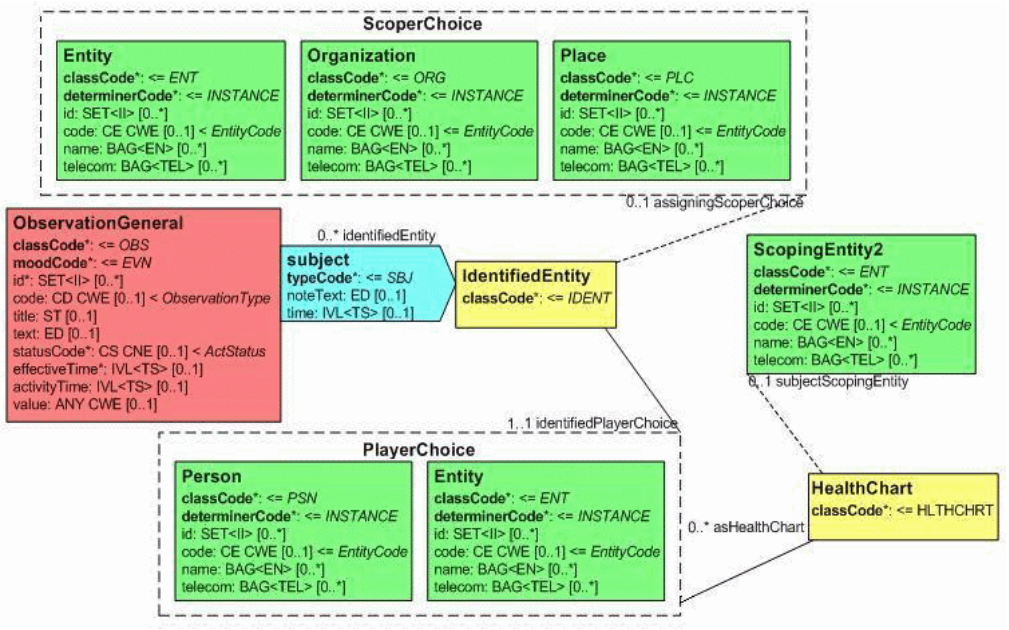
Act> Role with {0..1 or 0..n or 0..* or 1..1 or 1..n or 1..*}> PlayerChoice with {1..1} and ScoperChoice with {0..1 or 1..1}> PlayedRole with {0..1 or 0..n or 0..* or 1..1 or 1..n or 1..*}> Scoper with {0..1 or 1..1}

MTK generates n instances where n is the number of entities in the choice with Max entities. In the construct described in Figure E.14, MTK generates three instances. All the three instances have the Owned role due to the cardinality on the PlayerChoice.

The instance generated might be:

- First Instance: ObservationGeneral > subject > IdentifiedEntity > Entity and Person
  1. Person > HealthChart > ScopingEntity2
- Second Instance: ObservationGeneral > subject > IdentifiedEntity > Organization and Entity
  1. Entity > HealthChart > ScopingEntity2
- Third Instance: ObservationGeneral > Subject > IdentifiedEntity > Place and Person
  1. Person > HealthChart > ScopingEntity2

Figure 10–21 Act > Role with {0..*} > PlayerChoice with {1..1} and ScoperChoice with {0..1} > PlayedRole with {0..*} > Scoper with {0..1}



Constructs Within Oracle Published CMETS

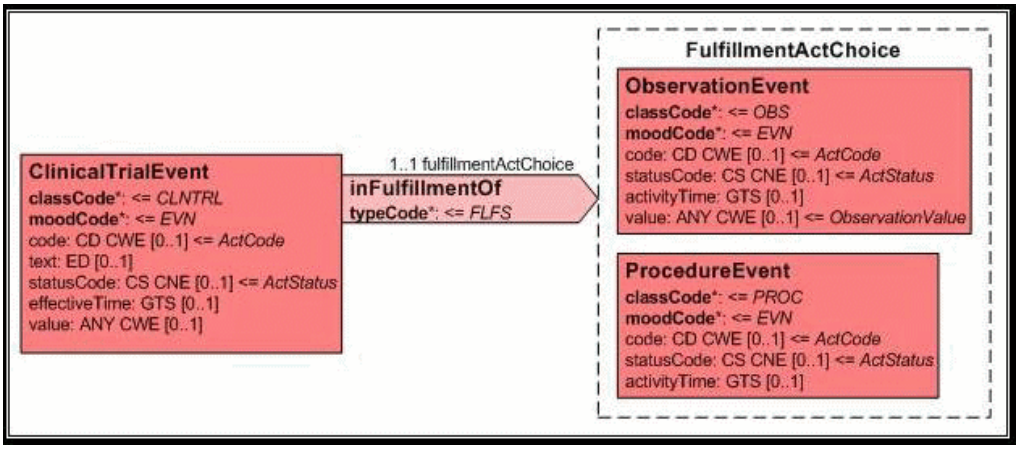
Act> ActChoice with {1..1 or 1..n or 1..*}

MTK will generate one instance. The generated instance will be one of the following:

- ClinicalTrialEvent > inFulfillmentOf > ObservationEvent
- ClinicalTrialEvent > inFulfillmentOf > ProcedureEvent

> **Note:** If the choice is {0..1 or 0..*}, the generated instance will not have the fulfillmentActChoice ActRelationship.

Figure 10–22 Act > ActChoice with {1..1}



Act> ActChoice with {1..1 or 1..n or 1..*}> Act with { 1..1 or 1..n or 1..*}

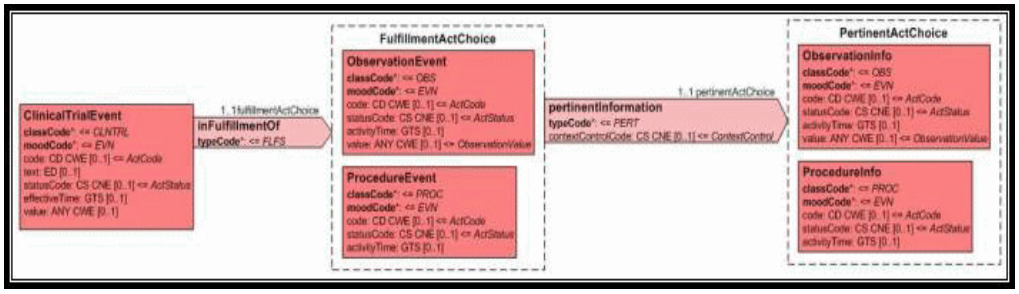
MTK generates one instance. The instance will have one of the following:

- ClinicalTrialEvent > inFulfillmentOf > ObservationEvent > pertinentInformation > ObservationInfo
- ClinicalTrialEvent > inFulfillmentOf > ProcedureEvent > pertinentInformation > ObservationInfo

Note:

- If the inFulfillmentOf has {0..1 or 0..*}, then the generated instance will not have the inFulfillmentOf ActRelationship and the other associated relationships
- If the inFulfillmentOf has {1..1 or 1..*} and observationInfo has {0..1 or 0..*} then the generated instance will have inFulfillmentOf ActRelationship but not the pertinentInformation relationships

Figure 10–23 Act > ActChoice with {1..1} > ActChoice with {1..1}



Act> ActChoice with {1..1 or 1..n or 1..*}> ActChoice with {1..1 or 1..n or 1..*}

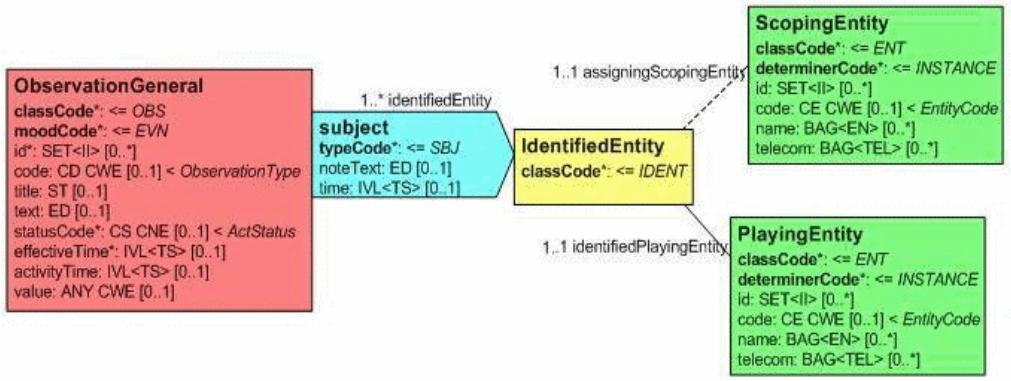
MTK generates one instance, if a message type has the construct described in Figure E.17. The generated message instance has one of the following:

- ClinicalTrialEvent > inFulfillmentOf > ObservationEvent > pertinentInformation > ObservationInfo
- ClinicalTrialEvent > inFulfillmentOf > ProcedureEvent > pertinentInformation > ProcedureInfo

Note:

- If the inFulfillmentOf has {0..1 or 0..*}, then the generated instance will not have the inFulfillmentOf ActRelationship and the other associated relationships.
- If the inFulfillmentOf has {1..1 or 1..*} and if pertinentInformation has {0..1 or 0..*} then the generated instance will have inFulfillmentOf ActRelationship but not the pertinentInformation relationships.

Figure 10–24 Act > Role with {1..} > Player with {1..1} and Scoper with {1..1}*



Act > Role with {1..1 or 1..n or 1..*} > PlayerChoice with {1..1} and ScoperChoice with {1..1} > PlayedRole with {1..1 or 1..n or 1..* } > Scoper with {1..1}

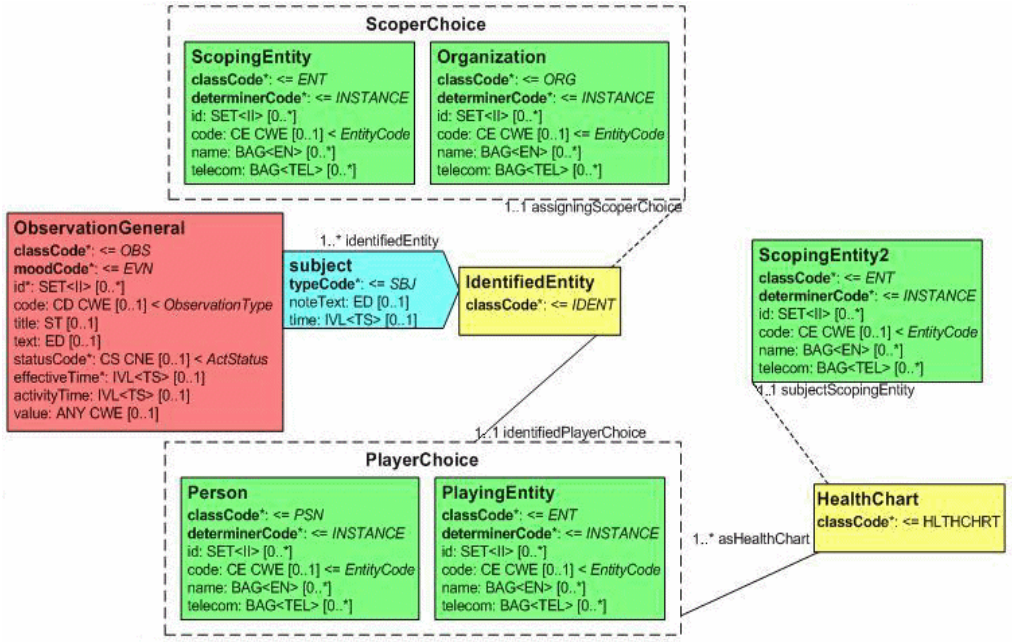
In the construct described in Figure E.19, MTK generates one instance. The instance generated might be:

- ObservationGeneral > subject > IdentifiedEntity > ScopingEntity and PlayingEntity
  1. Playingentity > HealthChart > ScopingEntity2
- ObservationGeneral > subject > IdentifiedEntity > Organization and Person
  1. Person > HealthChart > ScopingEntity2

Note:

- If the subject has {0..1 or 0..*} then the generated instance will not have the subject and its associated associations.
- If the subject has {1..1 or 1..*} and Scoper and Player choice with {0..1} then the generated instance will have: ObservationGeneral > subject > IdentifiedEntity
- If the subject has {1..1 or 1..*}, Player choice with {1..1} along with HealthChart with {1..1}, and ScoperChoice with {0..1} then the generated instance will have one of the below:
  1. ObservationGeneral > subject > IdentifiedEntity > PlayinngEntity > HealthChart > ScopingEntity2
  2. ObservationGeneral > subject > IdentifiedEntity > Person > HealthChart > ScopingEntity2

Figure 10–25 Act > Role with {1..*} > PlayerChoice with {1..1} and ScoperChoice with {1..1} > Playedrole with {1..*} > Scoper with {1..1}



XML Snippets of Seeded Data

Data seeded in HDR for MTK to generate instance for the provided Message Type schema file:

Datatypes	Value 1	Value 2

AD	<[attribute] use="H" > <delimiter partType="DEL">;</delimiter > <country partType="CNT">US</countr y> <state partType="STA">CA</state> <city partType="CTY">Fountain Hills</city> <postalCode partType="ZIP">85268</post alCode> <streetAddressLine partType="SAL">15542 OliveWest</streetAddressLin e> <houseNumberNumeric partType="BNN" >999777</houseNumberNum eric> <houseNumber partType="BNR">999777</ho useNumber> <direction partType="DIR">N</directio n> <streetName partType="STR">Church Street</streetName> <streetNameBase partType="STB">Church Street</streetNameBase> <streetNameType partType="STTYP">Avenue</ streetNameType> <unitID partType="UNID">222222</u nitID> <censusTract partType="CEN">testCensus Value</censusTract> <unitType partType="UNIT">Apartment </unitType> <useablePeriod xsi:type="IVL_TS" > <low value="20070127" inclusive="true"/> <high value="20070526" inclusive="false"/> </useablePeriod> </[attribute]>	<[attribute] use="H"> <delimiter partType="DEL">;</delimiter > <country partType="CNT">US</countr y> <state partType="STA">CA</state> <city partType="CTY">Fountain Hills</city> <postalCode partType="ZIP">852</postal Code> <streetAddressLine partType="SAL">142 OliveWest</streetAddressLin e> <houseNumberNumeric partType="BNN" >9977</houseNumberNumer ic> <houseNumber partType="BNR">9997</hous eNumber> <direction partType="DIR">N</directio n> <streetName partType="STR">Church Street</streetName> <streetNameBase partType="STB">Church Street</streetNameBase> <streetNameType partType="STTYP">Avenue</ streetNameType> <unitID partType="UNID">222234</u nitID> <censusTract partType="CEN">testCensus Value</censusTract> <unitType partType="UNIT">Apartment </unitType> <useablePeriod xsi:type="IVL_TS" > <low value="20070128" inclusive="true"/> <high value="20070527" inclusive="false"/> </useablePeriod> </[attribute]>
ANY	<[attribute] xsi:type="ST">MTK Test1</[attribute]>	<[attribute] xsi:type="ST">MTK Test2</[attribute]>
BL	<[attribute] value="true"/>	<[attribute] value="false"/>
BN	<[attribute] value="true"/>	<[attribute] value="true"/>
ControlActWrapper	<controlActProcess classCode="CACT" moodCode="EVN"> <subject typeCode="SUBJ"></subject> </controlActProcess>	<controlActProcess classCode="CACT" moodCode="EVN"> <subject typeCode="SUBJ"> <registrationEvent classCode="REG" moodCode="EVN"> <code code="T" codeSystemName="ActCode" /> <subject typeCode="SBJ"> </subject> </registrationEvent> </subject> </controlActProcess>

ED	<[attribute] mediaType="text/plain" language="en-US"> <reference value="http://example.org/xrays/128s8d9ej229se32s.png"> <useablePeriod xsi:type="IVL_TS"> <low value="200007200845"/> <high value="200008200845"/> </useablePeriod> </reference>ED.Text1 </[attribute]>	<[attribute] mediaType="text/plain" language="en-US">ED.Text2 <reference value="http://example.org/xrays/128s8d9ej229se32s.png"> <useablePeriod xsi:type="IVL_TS"> <low value="200007200846"/> <high value="200008200846"/> </useablePeriod> </reference>ED.Text2 </[attribute]>
EIVL	<[attribute] xsi:type="EIVL_TS"> <event code="AC" codeSystem="2.16.840.1.113883.5.139" codeSystemName="TimingEvent"/> <offset> <low value="10" unit="h"/> <high value="20" unit="h"/> </offset> </[attribute]>	<[attribute] xsi:type="EIVL_TS"> <event code="ACD" codeSystem="2.16.840.1.113883.5.139" codeSystemName="TimingEvent"/> <offset> <low value="11" unit="h"/> <high value="21" unit="h"/> </offset> </[attribute]>
EN	<[attribute] use="L"> <family partType="FAM">John</family> <given partType="GIV">Doe</given> <validTime xsi:type="IVL_TS"> <low value="20070127" inclusive="true"/> <high value="20070526" inclusive="false"/> </validTime> </[attribute]>	<[attribute] use="L"> <family partType="FAM">John</family> <given partType="GIV">Doe</given> <validTime xsi:type="IVL_TS"> <low value="20070129" inclusive="true"/> <high value="20070529" inclusive="false"/> </validTime> </[attribute]>
II	<[attribute] root="9.989898.5.100.10" extension="MTK1" assigningAuthorityName="ORACLE"/>	<[attribute] root="9.989898.5.100.10" extension="MTK1" assigningAuthorityName="ORACLE"/>
INT	<[attribute] xsi:type="INT" value="204" />	<[attribute] xsi:type="INT" value="205" />
IVL_INT	<[attribute] xsi:type="IVL_INT"> <low value="10"/> <high value="20"/> </[attribute]>	<[attribute] xsi:type="IVL_INT"> <low value="10"/> <high value="20"/> </[attribute]>
IVL_MO	<[attribute] xsi:type="IVL_MO"> <low value="200" currency="USD"/> <width value="205" currency="USD"/> </[attribute]>	<[attribute] xsi:type="IVL_MO"> <low value="1000" currency="USD"/> <width value="2000" currency="USD" /> </[attribute]>
IVL_PQ	<[attribute] xsi:type="IVL_PQ"> <low value="10" unit="g"/> <high value="20" unit="g"/> </[attribute]>	<[attribute] xsi:type="IVL_PQ"> <low value="1100" unit="g"/> <high value="2200" unit="g"/> </[attribute]>
IVL_REAL	<[attribute] xsi:type="IVL_REAL"> <low value="10.05"/> <high value="20.05"/> </[attribute]>	<[attribute] xsi:type="IVL_REAL" value="10.05">

IVL_TS	<[attribute] xsi:type="IVL_ TS"> <low value="20070127" inclusive="true"/> <high value="20070526" inclusive="false"/> </[attribute]>	<[attribute] xsi:type="IVL_ TS"> <low value="20070127" inclusive="true"/> <width value="2" unit="h"/> </[attribute]>
MO	<[attribute] value="204" currency="USD"/>	<[attribute] value="204" currency="INR"/>
ON	<[attribute] use="L"> <prefix partType="PFX">Prime Health</prefix> <suffix partType="SFX">Clinic</suffi x> </[attribute]>	<[attribute] use="L"> <prefix partType="PFX">Prime Health</prefix> <suffix partType="SFX">Speciality Center</suffix> </[attribute]>
PIVL	<[attribute] xsi:type="PIVL_ TS" alignment="DW" institutionSpecified="false"> <phase> <low value="20070319" inclusive="true"/> <high value="20070324" inclusive="false"/> </phase> <period value="1" unit="wk"/> </[attribute]>	<[attribute] xsi:type="PIVL_ TS" alignment="HD" operator="A"> <phase> <low value="200703190900" inclusive="true"/> <high value="200703191700" inclusive="false"/> </phase> <period value="1" unit="d"/> </[attribute]>
PN	<[attribute] use="L"> <family partType="FAM">Levin</fa mily> <given partType="GIV">Henry Jr</given> </[attribute]>	<[attribute] use="L"> <family partType="FAM" qualifier="VV">Levin</famil y> <given partType="GIV">Henry Sr</given> </[attribute]>
PQ	<[attribute] xsi:type="PQ" value="204" unit="g"/>	<[attribute] xsi:type="PQ" value="205" unit="g"/>
REAL	<[attribute] xsi:type="REAL" value="204" />	<[attribute] xsi:type="REAL" value="205" />
RTO_INT	<[attribute] xsi:type="RTO_ INT"> <numerator value="204"/> <denominator value="204"/> </[attribute]>	<[attribute] xsi:type="RTO_ INT"> <numerator value="205"/> <denominator value="205"/> </[attribute]>
RTO_MO_PQ	<[attribute] xsi:type="RTO_ MO_PQ"> <numerator value="204" currency="USD"/> <denominator value="204" unit="g"/> </[attribute]>	<[attribute] xsi:type="RTO_ MO_PQ"> <numerator value="205" currency="USD"/> <denominator value="205" unit="g"/> </[attribute]>
RTO_PQ	<[attribute] xsi:type="RTO_ PQ_PQ"> <numerator xsi:type="PQ" value="2" unit="g"/> <denominator xsi:type="PQ" value="1" unit="1"/> </[attribute]>	<[attribute] xsi:type="RTO_ PQ_PQ"> <numerator xsi:type="PQ" value="4" unit="g"/> <denominator xsi:type="PQ" value="2" unit="1"/> </[attribute]>
RTO_PQ_PQ	<[attribute] xsi:type="RTO_ PQ_PQ"> <numerator xsi:type="PQ" value="2" unit="g"/> <denominator xsi:type="PQ" value="1" unit="1"/> </[attribute]>	<[attribute] xsi:type="RTO_ PQ_PQ"> <numerator xsi:type="PQ" value="4" unit="g"/> <denominator xsi:type="PQ" value="2" unit="1"/> </[attribute]>

RTO_QTY_QTY	<[attribute] xsi:type="RTO_QTY_QTY"> <numerator xsi:type="PQ" value="2" unit="g"/> <denominator xsi:type="PQ" value="1" unit="1"/> </[attribute]>	<[attribute] xsi:type="RTO_QTY_QTY"> <numerator xsi:type="PQ" value="4" unit="g"/> <denominator xsi:type="PQ" value="2" unit="1"/> </[attribute]>
ST	<[attribute]>MTK Test1</[attribute]>	<[attribute]>MTK Test2</[attribute]>
SXCM_TS	<[attribute] xsi:type="SXPR_TS"> <comp xsi:type="IVL_TS"> <low value="20050603" /> <high value="20050603" /> </comp> <comp xsi:type="PIVL_TS" operator="A" institutionSpecified="true"> <period value="6" unit="h" /> </comp> </[attribute]>	<[attribute] xsi:type="SXPR_TS"> <comp xsi:type="IVL_TS"> <low value="20050604" /> <high value="20050604" /> </comp> <comp xsi:type="PIVL_TS" operator="A" institutionSpecified="true"> <period value="7" unit="h" /> </comp> </[attribute]
TEL	<[attribute] value="tel:1-690-555-1111" use="H" />	<[attribute] value="tel:1-690-555-2222" use="WP" />
TN	<[attribute]>Babel Fish</[attribute]>	<[attribute]>Red Herring</[attribute]>

TransmissionWrapper	<MCCI_ MT000100HT04.Message xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="urn:hl7-org:v3" xmlns:xsip="urn:hl7-org:v3" xmlns:htb="http://xmlns.oracle.com/apps/ctb/messaging" ITSVersion= "XML_1.0"> <creationTime value="20040719080000-0800" /> <responseModeCode code="C"/> <versionCode code="V3PR1"/> <interactionId root="9.989898.5.100" extension="PRPA_ HN400000"/> <profileId root="9.989898.5.1" extension="v2.14.1.01"/> <processingCode code="P"/> <processingModeCode code="T"/> <acceptAckCode code="AL"/> <receiver typeCode= "RCV"> <device classCode="DEV" determinerCode="INSTANCE" > <id root="9.989898.5.100" extension="DEV1000"/> <asAgent classCode="AGNT" > <representedOrganization classCode="ORG" determinerCode="INSTANCE" > <id root="9.989898.5.100" extension="ORG1000"/> </representedOrganization> </asAgent> </device> </receiver> <respondTo typeCode ="RSP"> <device classCode="DEV" determinerCode="INSTANCE" > <id root="9.481456.5.1" extension="DEV1001"/> <asAgent classCode="AGNT" > <representedOrganization classCode="ORG" determinerCode="INSTANCE"> <id root="9.481456.5.1" extension="ORG1001"/> </representedOrganization> </asAgent> </device> </respondTo> <sender typeCode ="SND"> <device classCode="DEV" determinerCode="INSTANCE" > <id root="9.481456.5.6" extension="MTKTest"/> <asAgent classCode="AGNT" > <representedOrganization classCode="ORG" determinerCode="INSTANCE"> <id root="9.481456.5.1" extension="ORG1001"/> </representedOrganization> </asAgent> </device> </sender> </MCCI_ MT000100HT04.Message>

TS	<[attribute] value="20070719080000-0800" />	<[attribute] value="20070719080000-0801" />
URL	<[attribute] value="www.Oracle.com"/>	<[attribute] value="http://www.hl7.org/" />

Expected Differences in Instances

Extra Internal ID Elements

The objects in the generated message can contain one extra ID attribute when compared to the corresponding object in the input message. This is not an issue, provided this extra id attribute represents the internal id of the object.

For person and organization entity objects, three IDs are generated and for other elements two IDs are generated.

Tasks:

Example 10–18 Input XML Element

```
<Person type="Person" classCode="PSN" determinerCode="INSTANCE"
htb:association="player">
<id root="9.989898.5.1" extension="PSN9002"/>
<name use="L">
<family partType="FAM" encoding="TXT">Mart</family>
<given partType="GIV" encoding="TXT">Bob</given>
</name>
</Person>
```

Example 10–19 Corresponding Output

```
<Person type="Person" classCode="PSN" determinerCode="INSTANCE"
htb:association="player">
<id root="9.101010.5" extension="23617" displayable="false"/>
<id root="9.989898.5.1" extension="PSN9002"/>displayable="false"/>
<id root="9.989898.5" extension="5321906" displayable="false"/>
<name use="L"><family partType="FAM" encoding="TXT">Mart</family>
<given partType="GIV" encoding="TXT">Bob</given>
</name>
</Person>
```

Extra XML Attributes in Coded Values

The XML representations of coded values will contain their full complement of attributes in the generated message, even though some of these attributes were absent in the persisted message. This is not an issue.

Tasks:

Example 10–20 Input XML Element

```
<code code="PORR_TE100001" codeSystemName="HDR Supplemental"/>
```

Example 10–21 Corresponding Output

```
<code code="PORR_TE100001"
codeSystemName="HDR Supplemental"
codeSystem="2.16.840.1.113894.1004.100.100.2.5"
codeSystemVersion="HDR Supplemental (2005-02-28)"
displayName="Christian: Assembly of God"/>
```

Order Differences in Collection Attributes

The order of the constituent elements in collection attributes (those of type SET or BAG) may not be preserved in the generated message.

Tasks:

Example 10–22 Input XML Element

```
<statusCode code = "active"/>
<telecom value="tel:1-690-555-1111" use="H" />
<telecom value="tel:1-690-555-1111" use="WP"/>
```

Example 10–23 Corresponding Output

```
<statusCode code = "active"/>
<telecom value="tel:1-690-555-1111" use="WP" />
<telecom value="tel:1-690-555-1111" use="H"/>
```

Order Differences Due to Choice Elements

When a ActRelationship has cardinality greater than one, and the target's type is a choice, then the order of the output elements may not match the order of the input elements.

Tasks:

Example 10–24 Input XML Element

```
<PORR_MT100001HT01.InvestigationEvent …>
…
<component typeCode="COMP" >
<SubstanceAdministrationEvent classCode="SBADM" moodCode="EVN"
negationInd="false"">
…
</SubstanceAdministrationEvent>
</component>
<component typeCode="COMP" >
<SpecimenObservationEvent classCode="SPCOBS" moodCode="EVN" negationInd="false" >
…
</SpecimenObservationEvent>
</component>
<component typeCode="COMP" >
```

```
<ObservationEventGeneral classCode="OBS" moodCode="EVN" negationInd="false" >
…
</ObservationEventGeneral>
</component>
<component typeCode="COMP" >
<EncounterEvent classCode="ENC" moodCode="EVN" >
…
</EncounterEvent>
</component>
…
</PORR_MT100001HT01.InvestigationEvent >
```

Example 10–25 Corresponding Output

```
<PORR_MT100001HT01.InvestigationEvent …>
…
<component typeCode="COMP" >
<SpecimenObservationEvent type="Observation" classCode="SPCOBS" moodCode="EVN"
negationInd="false" >
…
</SpecimenObservationEvent>
</component>
<component typeCode="COMP" >
<SubstanceAdministrationEvent classCode="SBADM" moodCode="EVN" negationInd="false"
>
…
</SubstanceAdministrationEvent>
</component>
<component typeCode="COMP" >
<EncounterEvent classCode="ENC" moodCode="EVN" >
…
</EncounterEvent>
</component>
<component typeCode="COMP" >
<ObservationEventGeneral type="Observation" classCode="OBS" moodCode="EVN"
negationInd="false" >
…
</ObservationEventGeneral>
</component>
…
</PORR_MT100001HT01.InvestigationEvent >
```

Differences in the Order of XML Attributes

The order of XML attributes of an element may be different in the input and output messages. The order of the attributes of an XML element is not significant.

Tasks:

Example 10–26 Input XML Element

```
<telecom value="tel:1-490-555-2222" use="H"/>
<POXX_MT121000HT02.SpecimenObservationOrder moodCode="RQO" classCode="SPCOBS"
negationInd="false" >
```

Example 10–27 Corresponding Output

```
<telecom use="H" value="tel:1-490-555-2222"/>
<POXX_MT121000HT02.SpecimenObservationOrder classCode="SPCOBS" moodCode="RQO"
negationInd="false">
```

Differences Due to Time Zones

A generated message may contain different literal values for time values than is present in the input message, as the generated time values are represented in the time zone of the database.

Example 10–28 Input XML Element

```
<id root="9.481456.5.30" extension="NC001"/>
<creationTime value="20040719080000-0800"/>
<versionCode code="V3PR1"/>
<profileId root="9.989898.5.1" extension=" v2.14.1.01"/>
<processingCode code="P"/>
```

Example 10–29 Corresponding Output

```
<id root="9.481456.5.30" extension="NC001"/>
   <creationTime value="20070719080000-0800"/>
   <versionCode code="V3PR1"/>
   <profileId root="9.989898.5.1" extension=" v2.14.1.01"/>
   <processingCode code="P"/>
```

Differences Due to Vocabulary Configuration

OMP can be configured to translate coded attributes before sending them out. In such cases, the input and output values for the coded attribute will be different.

Tasks:

Example 10–30 Input XML Element

```
<raceCode code="15754" codeSystemName="HL7"/>
```

Example 10–31 Corresponding Output

```
<raceCode code="15643001 " codeSystemName="SNOMED-CT"
codeSystem="2.16.840.1.113883.6.96 " codeSystemVersion="2.01.4"
displayName="White"/>
```

Differences in the Message Wrapper

Message wrapper of the input message is not persisted by inbound message processor. So, Outbound Message Processor will not generate the same message wrapper again.

MTK generates the test instance by using a seeded message wrapper. This message wrapper is the same for all the test messages. The only difference is the message ids. As outbound message generator does not know what is the content of the inbound message, there will be difference between the following attribute values in both the messages:

- Message id
- creationTime
- responseModeCode
- interactionId
- acceptAckCode
- receiver.device.id
- respondTo.device.id
- sender.deviceid

- receiver. representedOrganization .Id
- profileId

Tasks:

Example 10–32 Input XML Element

```
<MFFI_IN000101 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:hl7-org:v3"  xmlns:xsip="urn:hl7-org:v3"
xmlns:htb="http://xmlns.oracle.com/apps/ctb/messaging" ITSVersion= "XML_1.0">
<id root = "1.2.3" extension = "158933102"/>
<creationTime value="20040719080000-0800"/>
<responseModeCode code="C"/>
<versionCode code="V3PR1"/>
<interactionId root="9.989898.5.100" extension="PRPA_HN400000"/>
<processingCode code="P"/>
<processingModeCode code="T"/>
<acceptAckCode code="AL"/>
<receiver typeCode= "RCV">
<device classCode="DEV" determinerCode="INSTANCE" >
<id root="9.989898.5.100" extension="DEV1000"/>
<asAgent  classCode="AGNT" >
<representedOrganization  classCode="ORG" determinerCode="INSTANCE" >
<id root="9.989898.5.100" extension="ORG1000"/>
</representedOrganization>
</asAgent>
</device>
</receiver>
<respondTo typeCode ="RSP">
<device classCode="DEV" determinerCode="INSTANCE" >
<id root="9.481456.5.1" extension="DEV1001"/>
<asAgent  classCode="AGNT" >
<representedOrganization  classCode="ORG" determinerCode="INSTANCE">
<id root="9.481456.5.1" extension="ORG1001"/>
</representedOrganization>
</asAgent>
</device>
</respondTo>
<sender typeCode ="SND">
<device classCode="DEV" determinerCode="INSTANCE" >
<id root="9.481456.5.6" extension="MTKTest"/>
<asAgent  classCode="AGNT" >
<representedOrganization  classCode="ORG" determinerCode="INSTANCE">
<id root="9.481456.5.1" extension="ORG1001"/>
</representedOrganization>
</asAgent>
</device>
</sender>
---------------
```

Example 10–33 Output XML Element

```
< MFFI_IN000101 xmlns="urn:hl7-org:v3"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsip="urn:hl7-org:v3"
xmlns:htb="http://xmlns.oracle.com/apps/ctb/messaging" ITSVersion="XML_1.0">
   <id root="9.989898.5" extension="MTKTest"/>
   <creationTime value="20071129155115-0800"/>
   <responseModeCode code="D"/>
   <versionCode code="V3PR1"/>
   <interactionId root="9.989898.5" extension="MFPM_IN010000"/>
```

```
   <profileId root="9.989898.5" extension="2.14.1.01"/>
   <processingCode code="P"/>
   <processingModeCode code="T"/>
   <acceptAckCode code="NE"/>
   <receiver typeCode="RCV">
      <device classCode="DEV" determinerCode="INSTANCE">
         <id root="9.989898.5.100" extension="ORG1000" displayable="false"/>
         <asAgent classCode="AGNT">
            <representedOrganization classCode="ORG" determinerCode="INSTANCE">
               <id root="9.989898.5.100" extension="ORG1000" displayable="false"/>
            </representedOrganization>
         </asAgent>
      </device>
   </receiver>
   <respondTo typeCode="RSP">
      <device classCode="DEV" determinerCode="INSTANCE">
         <id root="9.989898.5" extension="1"/>
         <asAgent classCode="AGNT">
            <representedOrganization classCode="ORG" determinerCode="INSTANCE">
               <id root="9.989898.5" extension="1"/>
            </representedOrganization>
         </asAgent>
      </device>
   </respondTo>
   <sender typeCode="SND">
      <device classCode="DEV" determinerCode="INSTANCE">
         <id root="9.989898.5" extension="1"/>
         <asAgent classCode="AGNT">
            <representedOrganization classCode="ORG" determinerCode="INSTANCE">
               <id root="9.989898.5" extension="1"/>
            </representedOrganization>
         </asAgent>
      </device>
   </sender>
--------------
```

Sample Test Message and Corresponding Generated Message

Task Examples:

Example 10–34 Input XML Element

```
<MFFI_IN000101 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:hl7-org:v3"  xmlns:xsip="urn:hl7-org:v3"
xmlns:htb="http://xmlns.oracle.com/apps/ctb/messaging" ITSVersion= "XML_1.0">
<id root = "1.2.3" extension = "158933102"/>
<creationTime value="20040719080000-0800"/>
<responseModeCode code="C"/>
<versionCode code="V3PR1"/>
<interactionId root="9.989898.5.100" extension="PRPA_HN400000"/>
<profileId root="9.989898.5.1" extension="v2.14.1.01"/>
<processingCode code="P"/>
<processingModeCode code="T"/>
<acceptAckCode code="AL"/>
<receiver typeCode= "RCV">
<device classCode="DEV" determinerCode="INSTANCE" >
<id root="9.989898.5.100" extension="DEV1000"/>
<asAgent  classCode="AGNT" >
<representedOrganization  classCode="ORG" determinerCode="INSTANCE" >
<id root="9.989898.5.100" extension="ORG1000"/>
</representedOrganization>
```

```
</asAgent>
</device>
</receiver>
<respondTo typeCode ="RSP">
<device classCode="DEV" determinerCode="INSTANCE" >
<id root="9.481456.5.1" extension="DEV1001"/>
<asAgent  classCode="AGNT" >
<representedOrganization  classCode="ORG" determinerCode="INSTANCE">
<id root="9.481456.5.1" extension="ORG1001"/>
</representedOrganization>
</asAgent>
</device>
</respondTo>
<sender typeCode ="SND">
<device classCode="DEV" determinerCode="INSTANCE" >
<id root="9.481456.5.6" extension="MTKTest"/>
<asAgent  classCode="AGNT" >
<representedOrganization  classCode="ORG" determinerCode="INSTANCE">
<id root="9.481456.5.1" extension="ORG1001"/>
</representedOrganization>
</asAgent>
</device>
</sender>
<controlActProcess  classCode="CACT" moodCode="EVN">
<code code = "MFFI_TE000101" codeSystemName = "HDR Supplemental"/>
<subject typeCode="SUBJ">
<registrationEvent classCode="REG" moodCode="EVN">
<code code="T" codeSystemName="ActCode"/>
<subject typeCode="SBJ">
<employmentStaff classCode="EMP">
  <id root = "1.2.3" extension = "158931102"/>
  <code code = "DEPEN" codeSystemName = "RoleCode"/>
  <addr use="H" ><delimiter partType="DEL">;</delimiter>
<country partType="CNT">US</country>
<state partType="STA">CA</state>
<city partType="CTY">Fountain Hills</city>
<postalCode partType="ZIP">85268</postalCode>
<streetAddressLine partType="SAL">15542 OliveWest</streetAddressLine>
<houseNumberNumeric partType="BNN" >999777</houseNumberNumeric>
<houseNumber partType="BNR">999777</houseNumber>
<direction partType="DIR">N</direction>
<streetName partType="STR">Church Street</streetName>
<streetNameBase partType="STB">Church Street</streetNameBase>
<streetNameType partType="STTYP">Avenue</streetNameType>
<unitID partType="UNID">222222</unitID>
<censusTract partType="CEN">testCensusValue</censusTract>
<unitType partType="UNIT">Apartment</unitType>
<useablePeriod xsi:type="IVL_TS" >
    <low value="20070127" inclusive="true"/>
    <high value="20070526" inclusive="false"/>
</useablePeriod>
</addr>
  <addr use="H">
<delimiter partType="DEL">;</delimiter>
<country partType="CNT">US</country>
<state partType="STA">CA</state>
<city partType="CTY">Fountain Hills</city>
<postalCode partType="ZIP">852</postalCode>
<streetAddressLine partType="SAL">142 OliveWest</streetAddressLine>
<houseNumberNumeric partType="BNN" >9977</houseNumberNumeric>
```

```
<houseNumber partType="BNR">9997</houseNumber>
<direction partType="DIR">N</direction>
<streetName partType="STR">Church Street</streetName>
<streetNameBase partType="STB">Church Street</streetNameBase>
<streetNameType partType="STTYP">Avenue</streetNameType>
<unitID partType="UNID">222234</unitID>
<censusTract partType="CEN">testCensusValue</censusTract>
<unitType partType="UNIT">Apartment</unitType>
<useablePeriod xsi:type="IVL_TS" >
    <low value="20070127" inclusive="true"/>
    <high value="20070526" inclusive="false"/>
</useablePeriod>
</addr>
   <telecom value="tel:1-690-555-1111" use="H" />
  <telecom value="tel:1-690-555-1111" use="WP"/>
  <statusCode code = "active"/>
  <effectiveTime xsi:type="IVL_TS">
    <low value="20070127"/>
    <high value="20070526"/>
</effectiveTime>
  <employeePerson determinerCode="INSTANCE" classCode="PSN">
    <id root = "1.2.3" extension = "158931402"/>
    <id root = "1.2.3" extension = "158931502"/>
    <name use="L">
  <family partType="FAM" >Levin</family>
  <given partType="GIV" >Henry Jr</given>
</name>
     <name use="L">
  <family partType="FAM" qualifier="VV">Levin</family>
  <given partType="GIV">Henry Sr</given>
</name>
    <desc mediaType="image/png" language="en-US" compression="GZ"
integrityCheck="3454bfb019d0e7d47a253b59cf234bc49a0e0cf8">
   <reference value="http://example.org/xrays/128s8d9ej229se32s.png">
      <useablePeriod xsi:type="IVL_TS">
         <low value="200007200845"/>
         <high value="200008200845"/>
      </useablePeriod>
   </reference>
   <thumbnail mediaType="image/jpeg" language="en-US" representation="B64"/>
      MNYD83jmMdomSJUEdmde9j44zmMir6edjzMMIjdMDSsWdIJdksIJR3373jeu83
      6edjzMMIjdMDSsWdIJdksIJR3373jeu83MNYD83jmMdomSJUEdmde9j44zmMir
      omSJUEdmde9j44zmMiromSJUEdmde9j44zmMirdMDSsWdIJdksIJR3373jeu83
      4zmMir6edjzMMIjdMDSsWdIJdksIJR3373jeu83==</desc>
    <statusCode code = "active"/>
     <telecom value="tel:1-690-555-1111" use="H" />
    <telecom value="tel:1-690-555-1111" use="WP"/>
    <administrativeGenderCode code = "F" codeSystemName = "AdministrativeGender"/>
    <birthTime value="20060719080000-0800" />
    <deceasedInd value="true"/>
    <deceasedTime value="20060719080000-0800" />
    <multipleBirthInd value="true"/>
    <multipleBirthOrderNumber xsi:type="INT" value="204"/>
    <organDonorInd value="true"/>
    <addr use="H" ><delimiter partType="DEL">;</delimiter>
<country partType="CNT">US</country>
<state partType="STA">CA</state>
<city partType="CTY">Fountain Hills</city>
<postalCode partType="ZIP">85268</postalCode>
<streetAddressLine partType="SAL">15542 OliveWest</streetAddressLine>
```

```
<houseNumberNumeric partType="BNN" >999777</houseNumberNumeric>
<houseNumber partType="BNR">999777</houseNumber>
<direction partType="DIR">N</direction>
<streetName partType="STR">Church Street</streetName>
<streetNameBase partType="STB">Church Street</streetNameBase>
<streetNameType partType="STTYP">Avenue</streetNameType>
<unitID partType="UNID">222222</unitID>
<censusTract partType="CEN">testCensusValue</censusTract>
<unitType partType="UNIT">Apartment</unitType>
<useablePeriod xsi:type="IVL_TS" >
    <low value="20070127" inclusive="true"/>
    <high value="20070526" inclusive="false"/>
</useablePeriod>
</addr>
    <addr use="H">
<delimiter partType="DEL">;</delimiter>
<country partType="CNT">US</country>
<state partType="STA">CA</state>
<city partType="CTY">Fountain Hills</city>
<postalCode partType="ZIP">852</postalCode>
<streetAddressLine partType="SAL">142 OliveWest</streetAddressLine>
<houseNumberNumeric partType="BNN" >9977</houseNumberNumeric>
<houseNumber partType="BNR">9997</houseNumber>
<direction partType="DIR">N</direction>
<streetName partType="STR">Church Street</streetName>
<streetNameBase partType="STB">Church Street</streetNameBase>
<streetNameType partType="STTYP">Avenue</streetNameType>
<unitID partType="UNID">222234</unitID>
<censusTract partType="CEN">testCensusValue</censusTract>
<unitType partType="UNIT">Apartment</unitType>
<useablePeriod xsi:type="IVL_TS" >
    <low value="20070127" inclusive="true"/>
    <high value="20070526" inclusive="false"/>
</useablePeriod>
</addr>
    <maritalStatusCode code = "A" codeSystemName = "MaritalStatus"/>
    <disabilityCode code = "1" codeSystemName = "PersonDisabilityType"/>
    <livingArrangementCode code = "G" codeSystemName = "LivingArrangement"/>
    <religiousAffiliationCode code = "1001" codeSystemName =
"ReligiousAffiliation"/>
    <raceCode code = "1008-2" codeSystemName = "Race"/>
    <ethnicGroupCode code = "2135-2" codeSystemName = "Ethnicity"/>
    <languageCommunication>
      <languageCode code = "EncounterStatus" codeSystemName = "HDR Supplemental"/>
      <modeCode code = "ESGN" codeSystemName = "LanguageAbilityMode"/>
      <proficiencyLevelCode code = "E" codeSystemName =
"LanguageAbilityProficiency"/>
      <preferenceInd value="true"/>
    </languageCommunication>
  </employeePerson>
  <employerOrganization determinerCode="INSTANCE" classCode="ORG">
    <id root = "1.2.3" extension = "158931702"/>
    <id root = "1.2.3" extension = "158931802"/>
    <code code = "NDA17" codeSystemName = "EntityCode"/>
    <name use="L">
   <prefix partType="PFX" >Prime Health</prefix>
   <suffix partType="SFX" >Clinic</suffix>
</name>
    <name use="L">
   <prefix partType="PFX" >Prime Health</prefix>
```

```
   <suffix partType="SFX" >Speciality Center</suffix>
</name>
    <desc mediaType="image/png" language="en-US" compression="GZ"
integrityCheck="3454bfb019d0e7d47a253b59cf234bc49a0e0cf8">
   <reference value="http://example.org/xrays/128s8d9ej229se32s.png">
      <useablePeriod xsi:type="IVL_TS">
         <low value="200007200845"/>
         <high value="200008200845"/>
      </useablePeriod>
   </reference>
   <thumbnail mediaType="image/jpeg" language="en-US" representation="B64"/>
      MNYD83jmMdomSJUEdmde9j44zmMir6edjzMMIjdMDSsWdIJdksIJR3373jeu83
      6edjzMMIjdMDSsWdIJdksIJR3373jeu83MNYD83jmMdomSJUEdmde9j44zmMir
      omSJUEdmde9j44zmMiromSJUEdmde9j44zmMirdMDSsWdIJdksIJR3373jeu83
      4zmMir6edjzMMIjdMDSsWdIJdksIJR3373jeu83==</desc>
    <statusCode code = "active"/>
     <telecom value="tel:1-690-555-1111" use="H" />
    <telecom value="tel:1-690-555-1111" use="WP"/>
    <addr use="H" ><delimiter partType="DEL">;</delimiter>
<country partType="CNT">US</country>
<state partType="STA">CA</state>
<city partType="CTY">Fountain Hills</city>
<postalCode partType="ZIP">85268</postalCode>
<streetAddressLine partType="SAL">15542 OliveWest</streetAddressLine>
<houseNumberNumeric partType="BNN" >999777</houseNumberNumeric>
<houseNumber partType="BNR">999777</houseNumber>
<direction partType="DIR">N</direction>
<streetName partType="STR">Church Street</streetName>
<streetNameBase partType="STB">Church Street</streetNameBase>
<streetNameType partType="STTYP">Avenue</streetNameType>
<unitID partType="UNID">222222</unitID>
<censusTract partType="CEN">testCensusValue</censusTract>
<unitType partType="UNIT">Apartment</unitType>
<useablePeriod xsi:type="IVL_TS" >
    <low value="20070127" inclusive="true"/>
    <high value="20070526" inclusive="false"/>
</useablePeriod>
</addr>
    <addr use="H">
<delimiter partType="DEL">;</delimiter>
<country partType="CNT">US</country>
<state partType="STA">CA</state>
<city partType="CTY">Fountain Hills</city>
<postalCode partType="ZIP">852</postalCode>
<streetAddressLine partType="SAL">142 OliveWest</streetAddressLine>
<houseNumberNumeric partType="BNN" >9977</houseNumberNumeric>
<houseNumber partType="BNR">9997</houseNumber>
<direction partType="DIR">N</direction>
<streetName partType="STR">Church Street</streetName>
<streetNameBase partType="STB">Church Street</streetNameBase>
<streetNameType partType="STTYP">Avenue</streetNameType>
<unitID partType="UNID">222234</unitID>
<censusTract partType="CEN">testCensusValue</censusTract>
<unitType partType="UNIT">Apartment</unitType>
<useablePeriod xsi:type="IVL_TS" >
    <low value="20070127" inclusive="true"/>
    <high value="20070526" inclusive="false"/>
</useablePeriod></addr>
    <standardIndustryClassCode code = "001662" codeSystemName = "HDR
Supplemental"/>
```

```
    <asPartOfWhole classCode="PART">
      <id root = "1.2.3" extension = "158932002"/>
      <id root = "1.2.3" extension = "158932102"/>
      <code code = "DEPEN" codeSystemName = "RoleCode"/>
      <statusCode code = "active"/>
      <effectiveTime xsi:type="IVL_TS">
    <low value="20070127"/>
    <high value="20070526"/>
</effectiveTime>
    </asPartOfWhole>
    <contactParty classCode="CON">
      <id root = "1.2.3" extension = "158932302"/>
      <id root = "1.2.3" extension = "158932402"/>
      <code code = "DEPEN" codeSystemName = "RoleCode"/>
       <telecom value="tel:1-690-555-1111" use="H" />
      <telecom value="tel:1-690-555-1111" use="WP"/>
      <contactPerson determinerCode="INSTANCE" classCode="PSN">
        <name use="L">
  <family partType="FAM" >Levin</family>
  <given partType="GIV" >Henry Jr</given>
</name>
         <name use="L">
  <family partType="FAM" qualifier="VV">Levin</family>
  <given partType="GIV">Henry Sr</given>
</name>
         <telecom value="tel:1-690-555-1111" use="H" />
        <telecom value="tel:1-690-555-1111" use="WP"/>
        <asPersonDomain classCode="IDENT">
          <id root = "1.2.3" extension = "158932602"/>
          <code code = "DEPEN" codeSystemName = "RoleCode"/>
        </asPersonDomain>
      </contactPerson>
    </contactParty>
  </employerOrganization>
</employmentStaff>
</subject>
</registrationEvent>
</subject>
</controlActProcess>
</MFFI_IN000101>
```

Example 10–35 Output XML Element

```
<MFPM_IN000101  xmlns="urn:hl7-org:v3"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsip="urn:hl7-org:v3"
xmlns:htb="http://xmlns.oracle.com/apps/ctb/messaging" ITSVersion="XML_1.0">
   <id root="9.989898.5" extension="MTKTest"/>
   <creationTime value="20071129155115-0800"/>
   <responseModeCode code="D"/>
   <versionCode code="V3PR1"/>
   <interactionId root="9.989898.5" extension="MFPM_IN010000"/>
   <profileId root="9.989898.5" extension="2.14.1.01"/>
   <processingCode code="P"/>
   <processingModeCode code="T"/>
   <acceptAckCode code="NE"/>
   <receiver typeCode="RCV">
      <device classCode="DEV" determinerCode="INSTANCE">
         <id root="9.989898.5.100" extension="ORG1000" displayable="false"/>
         <asAgent classCode="AGNT">
            <representedOrganization classCode="ORG" determinerCode="INSTANCE">
```

```
               <id root="9.989898.5.100" extension="ORG1000" displayable="false"/>
            </representedOrganization>
         </asAgent>
      </device>
   </receiver>
   <respondTo typeCode="RSP">
      <device classCode="DEV" determinerCode="INSTANCE">
         <id root="9.989898.5" extension="1"/>
         <asAgent classCode="AGNT">
            <representedOrganization classCode="ORG" determinerCode="INSTANCE">
               <id root="9.989898.5" extension="1"/>
            </representedOrganization>
         </asAgent>
      </device>
   </respondTo>
   <sender typeCode="SND">
      <device classCode="DEV" determinerCode="INSTANCE">
         <id root="9.989898.5" extension="1"/>
         <asAgent classCode="AGNT">
            <representedOrganization classCode="ORG" determinerCode="INSTANCE">
               <id root="9.989898.5" extension="1"/>
            </representedOrganization>
         </asAgent>
      </device>
   </sender>
   <controlActProcess classCode="CACT" moodCode="EVN">
      <id root="9.989898.5" extension="158933802" displayable="false"/>
      <code code="EncounterStatus" codeSystemName="HDR Supplemental"
codeSystem="2.16.840.1.113894.1004.100.100.2.5" codeSystemVersion="HDR
Supplemental (2005-07-28)" displayName="EncounterStatus"/>
      <subject typeCode="SUBJ">
         <registrationEvent classCode="REG" moodCode="EVN">
            <id root="9.989898.5" extension="158933702" displayable="false"/>
            <code code="T" codeSystemName="ActCode"
codeSystem="2.16.840.1.113883.5.4" codeSystemVersion="2.01.4" displayName="tea
only"/>
            <subject typeCode="SBJ">
               <employmentStaff classCode="EMP">
                  <id root="1.2.3" extension="158931102" displayable="false"/>
                  <code code="DEPEN" codeSystemName="RoleCode"
codeSystem="2.16.840.1.113883.5.111" codeSystemVersion="2.01.4"
displayName="DEPEN"/>
                  <addr use="H">
                     <delimiter partType="DEL" representation="TXT">;</delimiter>
                     <country partType="CNT" representation="TXT">US</country>
                     <state partType="STA" representation="TXT">CA</state>
                     <city partType="CTY" representation="TXT">Fountain
Hills</city>
                     <postalCode partType="ZIP"
representation="TXT">85268</postalCode>
                     <streetAddressLine partType="SAL" representation="TXT">15542
OliveWest</streetAddressLine>
                     <houseNumberNumeric partType="BNN"
representation="TXT">999777</houseNumberNumeric>
                     <houseNumber partType="BNR"
representation="TXT">999777</houseNumber>
                     <direction partType="DIR" representation="TXT">N</direction>
                     <streetName partType="STR" representation="TXT">Church
Street</streetName>
                     <streetNameBase partType="STB" representation="TXT">Church
```

```
Street</streetNameBase>
                     <streetNameType partType="STTYP"
representation="TXT">Avenue</streetNameType>
                     <unitID partType="UNID" representation="TXT">222222</unitID>
                     <censusTract partType="CEN"
representation="TXT">testCensusValue</censusTract>
                     <unitType partType="UNIT"
representation="TXT">Apartment</unitType>
                     <useablePeriod xsi:type="IVL_TS">
                        <low value="20070127" inclusive="true"/>
                        <high value="20070526" inclusive="false"/>
                     </useablePeriod>
                  </addr>
                  <addr use="H">
                     <delimiter partType="DEL" representation="TXT">;</delimiter>
                     <country partType="CNT" representation="TXT">US</country>
                     <state partType="STA" representation="TXT">CA</state>
                     <city partType="CTY" representation="TXT">Fountain
Hills</city>
                     <postalCode partType="ZIP"
representation="TXT">852</postalCode>
                     <streetAddressLine partType="SAL" representation="TXT">142
OliveWest</streetAddressLine>
                     <houseNumberNumeric partType="BNN"
representation="TXT">9977</houseNumberNumeric>
                     <houseNumber partType="BNR"
representation="TXT">9997</houseNumber>
                     <direction partType="DIR" representation="TXT">N</direction>
                     <streetName partType="STR" representation="TXT">Church
Street</streetName>
                     <streetNameBase partType="STB" representation="TXT">Church
Street</streetNameBase>
                     <streetNameType partType="STTYP"
representation="TXT">Avenue</streetNameType>
                     <unitID partType="UNID" representation="TXT">222234</unitID>
                     <censusTract partType="CEN"
representation="TXT">testCensusValue</censusTract>
                     <unitType partType="UNIT"
representation="TXT">Apartment</unitType>
                     <useablePeriod xsi:type="IVL_TS">
                        <low value="20070127" inclusive="true"/>
                        <high value="20070526" inclusive="false"/>
                     </useablePeriod>
                  </addr>
                  <telecom use="H" value="tel:1-690-555-1111"/>
                  <telecom use="WP" value="tel:1-690-555-1111"/>
                  <statusCode code="active"/>
                  <effectiveTime xsi:type="IVL_TS">
                     <low value="20070127" inclusive="true"/>
                     <high value="20070526" inclusive="true"/>
                  </effectiveTime>
                  <employeePerson classCode="PSN" determinerCode="INSTANCE">
                     <id root="9.101010.5" extension="1086238"
displayable="false"/>
                     <id root="1.2.3" extension="158931502" displayable="false"/>
                     <id root="1.2.3" extension="158931402" displayable="false"/>
                     <id root="9.989898.5" extension="158933405"
displayable="false"/>
                     <name use="L">
                        <family partType="FAM" representation="TXT"
```

```
qualifier="VV">Levin</family>
                        <given partType="GIV" representation="TXT">Henry
Sr</given>
                     </name>
                     <name use="L">
                        <family partType="FAM" representation="TXT">Levin</family>
                        <given partType="GIV" representation="TXT">Henry
Jr</given>
                     </name>
                     <desc mediaType="image/png" compression="GZ"
integrityCheck="3454bfb019d0e7d47a253b59cf234bc49a0e0cf8"
representation="TXT">MNYD83jmMdomSJUEdmde9j44zmMir6edjzMMIjdMDSsWdIJdksIJR3373jeu8
3
      6edjzMMIjdMDSsWdIJdksIJR3373jeu83MNYD83jmMdomSJUEdmde9j44zmMir
      omSJUEdmde9j44zmMiromSJUEdmde9j44zmMirdMDSsWdIJdksIJR3373jeu83
      4zmMir6edjzMMIjdMDSsWdIJdksIJR3373jeu83==<reference
value="http://example.org/xrays/128s8d9ej229se32s.png"><useablePeriod
xsi:type="IVL_TS"><low value="200007200845" inclusive="true"/><high
value="200008200845" inclusive="true"/></useablePeriod></reference><thumbnail
mediaType="image/jpeg" representation="B64"></thumbnail></desc>
                     <statusCode code="active"/>
                     <telecom use="H" value="tel:1-690-555-1111"/>
                     <telecom use="WP" value="tel:1-690-555-1111"/>
                     <administrativeGenderCode code="F"
codeSystemName="AdministrativeGender" codeSystem="2.16.840.1.113883.5.1"
codeSystemVersion="2.01.4" displayName="Female"/>
                     <birthTime value="20060719080000-0800"/>
                     <deceasedInd value="true"/>
                     <deceasedTime value="20060719080000-0800"/>
                     <multipleBirthInd value="true"/>
                     <multipleBirthOrderNumber value="204"/>
                     <organDonorInd value="true"/>
                     <addr use="H">
                        <delimiter partType="DEL"
representation="TXT">;</delimiter>
                        <country partType="CNT" representation="TXT">US</country>
                        <state partType="STA" representation="TXT">CA</state>
                        <city partType="CTY" representation="TXT">Fountain
Hills</city>
                        <postalCode partType="ZIP"
representation="TXT">85268</postalCode>
                        <streetAddressLine partType="SAL"
representation="TXT">15542 OliveWest</streetAddressLine>
                        <houseNumberNumeric partType="BNN"
representation="TXT">999777</houseNumberNumeric>
                        <houseNumber partType="BNR"
representation="TXT">999777</houseNumber>
                        <direction partType="DIR"
representation="TXT">N</direction>
                        <streetName partType="STR" representation="TXT">Church
Street</streetName>
                        <streetNameBase partType="STB" representation="TXT">Church
Street</streetNameBase>
                        <streetNameType partType="STTYP"
representation="TXT">Avenue</streetNameType>
                        <unitID partType="UNID"
representation="TXT">222222</unitID>
                        <censusTract partType="CEN"
representation="TXT">testCensusValue</censusTract>
                        <unitType partType="UNIT"
```

```
representation="TXT">Apartment</unitType>
                        <useablePeriod xsi:type="IVL_TS">
                           <low value="20070127" inclusive="true"/>
                           <high value="20070526" inclusive="false"/>
                        </useablePeriod>
                     </addr>
                     <addr use="H">
                        <delimiter partType="DEL"
representation="TXT">;</delimiter>
                        <country partType="CNT" representation="TXT">US</country>
                        <state partType="STA" representation="TXT">CA</state>
                        <city partType="CTY" representation="TXT">Fountain
Hills</city>
                        <postalCode partType="ZIP"
representation="TXT">852</postalCode>
                        <streetAddressLine partType="SAL" representation="TXT">142
OliveWest</streetAddressLine>
                        <houseNumberNumeric partType="BNN"
representation="TXT">9977</houseNumberNumeric>
                        <houseNumber partType="BNR"
representation="TXT">9997</houseNumber>
                        <direction partType="DIR"
representation="TXT">N</direction>
                        <streetName partType="STR" representation="TXT">Church
Street</streetName>
                        <streetNameBase partType="STB" representation="TXT">Church
Street</streetNameBase>
                        <streetNameType partType="STTYP"
representation="TXT">Avenue</streetNameType>
                        <unitID partType="UNID"
representation="TXT">222234</unitID>
                        <censusTract partType="CEN"
representation="TXT">testCensusValue</censusTract>
                        <unitType partType="UNIT"
representation="TXT">Apartment</unitType>
                        <useablePeriod xsi:type="IVL_TS">
                           <low value="20070127" inclusive="true"/>
                           <high value="20070526" inclusive="false"/>
                        </useablePeriod>
                     </addr>
                     <maritalStatusCode code="A" codeSystemName="MaritalStatus"
codeSystem="2.16.840.1.113883.5.2" codeSystemVersion="2.01.4"
displayName="Annulled"/>
                     <disabilityCode code="1"
codeSystemName="PersonDisabilityType" codeSystem="2.16.840.1.113883.5.93"
codeSystemVersion="2.01.4" displayName="Vision impaired"/>
                     <livingArrangementCode code="G"
codeSystemName="LivingArrangement" codeSystem="2.16.840.1.113883.5.63"
codeSystemVersion="2.01.4" displayName="Group Home"/>
                     <religiousAffiliationCode code="1001"
codeSystemName="ReligiousAffiliation" codeSystem="2.16.840.1.113883.5.1076"
codeSystemVersion="2.01.4" displayName="Adventist"/>
                     <raceCode code="1008-2" codeSystemName="Race"
codeSystem="2.16.840.1.113883.5.104" codeSystemVersion="2.01.4"
displayName="Algonquian"/>
                     <ethnicGroupCode code="2135-2" codeSystemName="Ethnicity"
codeSystem="2.16.840.1.113883.5.50" codeSystemVersion="2.01.4"
displayName="Hispanic or Latino"/>
                     <languageCommunication>
                        <languageCode code="EncounterStatus" codeSystemName="HDR
```

```
Supplemental" codeSystem="2.16.840.1.113894.1004.100.100.2.5"
codeSystemVersion="HDR Supplemental (2005-07-28)" displayName="EncounterStatus"/>
                        <modeCode code="ESGN" codeSystemName="LanguageAbilityMode"
codeSystem="2.16.840.1.113883.5.60" codeSystemVersion="2.01.4"
displayName="Expressed signed"/>
                        <proficiencyLevelCode code="E"
codeSystemName="LanguageAbilityProficiency" codeSystem="2.16.840.1.113883.5.61"
codeSystemVersion="2.01.4" displayName="Excellent"/>
                        <preferenceInd value="true"/>
                     </languageCommunication>
                  </employeePerson>
                  <employerOrganization classCode="ORG" determinerCode="INSTANCE">
                     <id root="1.2.3" extension="158931702" displayable="false"/>
                     <id root="9.989898.5" extension="158933406"
displayable="false"/>
                     <id root="1.2.3" extension="158931802" displayable="false"/>
                     <id root="9.101010.5" extension="1086239"
displayable="false"/>
                     <code code="NDA17" codeSystemName="EntityCode"
codeSystem="2.16.840.1.113883.5.1060" codeSystemVersion="2.01.4"
displayName="NDA17"/>
                     <name use="L">
                        <prefix partType="PFX" representation="TXT">Prime
Health</prefix>
                        <suffix partType="SFX"
representation="TXT">Clinic</suffix>
                     </name>
                     <name use="L">
                        <prefix partType="PFX" representation="TXT">Prime
Health</prefix>
                        <suffix partType="SFX" representation="TXT">Speciality
Center</suffix>
                     </name>
                     <desc mediaType="image/png" compression="GZ"
integrityCheck="3454bfb019d0e7d47a253b59cf234bc49a0e0cf8"
representation="TXT">MNYD83jmMdomSJUEdmde9j44zmMir6edjzMMIjdMDSsWdIJdksIJR3373jeu8
3
      6edjzMMIjdMDSsWdIJdksIJR3373jeu83MNYD83jmMdomSJUEdmde9j44zmMir
      omSJUEdmde9j44zmMiromSJUEdmde9j44zmMirdMDSsWdIJdksIJR3373jeu83
      4zmMir6edjzMMIjdMDSsWdIJdksIJR3373jeu83==<reference
value="http://example.org/xrays/128s8d9ej229se32s.png"><useablePeriod
xsi:type="IVL_TS"><low value="200007200845" inclusive="true"/><high
value="200008200845" inclusive="true"/></useablePeriod></reference><thumbnail
mediaType="image/jpeg" representation="B64"></thumbnail></desc>
                     <statusCode code="active"/>
                     <telecom use="H" value="tel:1-690-555-1111"/>
                     <addr use="H">
                        <delimiter partType="DEL"
representation="TXT">;</delimiter>
                        <country partType="CNT" representation="TXT">US</country>
                        <state partType="STA" representation="TXT">CA</state>
                        <city partType="CTY" representation="TXT">Fountain
Hills</city>
                        <postalCode partType="ZIP"
representation="TXT">852</postalCode>
                        <streetAddressLine partType="SAL" representation="TXT">142
OliveWest</streetAddressLine>
                        <houseNumberNumeric partType="BNN"
representation="TXT">9977</houseNumberNumeric>
                        <houseNumber partType="BNR"
```

```
representation="TXT">9997</houseNumber>
                        <direction partType="DIR"
representation="TXT">N</direction>
                        <streetName partType="STR" representation="TXT">Church
Street</streetName>
                        <streetNameBase partType="STB" representation="TXT">Church
Street</streetNameBase>
                        <streetNameType partType="STTYP"
representation="TXT">Avenue</streetNameType>
                        <unitID partType="UNID"
representation="TXT">222234</unitID>
                        <censusTract partType="CEN"
representation="TXT">testCensusValue</censusTract>
                        <unitType partType="UNIT"
representation="TXT">Apartment</unitType>
                        <useablePeriod xsi:type="IVL_TS">
                           <low value="20070127" inclusive="true"/>
                           <high value="20070526" inclusive="false"/>
                        </useablePeriod>
                     </addr>
                     <addr use="H">
                        <delimiter partType="DEL"
representation="TXT">;</delimiter>
                        <country partType="CNT" representation="TXT">US</country>
                        <state partType="STA" representation="TXT">CA</state>
                        <city partType="CTY" representation="TXT">Fountain
Hills</city>
                        <postalCode partType="ZIP"
representation="TXT">85268</postalCode>
                        <streetAddressLine partType="SAL"
representation="TXT">15542 OliveWest</streetAddressLine>
                        <houseNumberNumeric partType="BNN"
representation="TXT">999777</houseNumberNumeric>
                        <houseNumber partType="BNR"
representation="TXT">999777</houseNumber>
                        <direction partType="DIR"
representation="TXT">N</direction>
                        <streetName partType="STR" representation="TXT">Church
Street</streetName>
                        <streetNameBase partType="STB" representation="TXT">Church
Street</streetNameBase>
                        <streetNameType partType="STTYP"
representation="TXT">Avenue</streetNameType>
                        <unitID partType="UNID"
representation="TXT">222222</unitID>
                        <censusTract partType="CEN"
representation="TXT">testCensusValue</censusTract>
                        <unitType partType="UNIT"
representation="TXT">Apartment</unitType>
                        <useablePeriod xsi:type="IVL_TS">
                           <low value="20070127" inclusive="true"/>
                           <high value="20070526" inclusive="false"/>
                        </useablePeriod>
                     </addr>
                     <standardIndustryClassCode code="001662" codeSystemName="HDR
Supplemental" codeSystem="2.16.840.1.113894.1004.100.100.2.5"
codeSystemVersion="HDR Supplemental (2005-07-28)" displayName="Snack"/>
                     <asPartOfWhole classCode="PART">
                        <id root="1.2.3" extension="158932002"
displayable="false"/>
```

```
                        <id root="1.2.3" extension="158932102"
displayable="false"/>
                        <code code="DEPEN" codeSystemName="RoleCode"
codeSystem="2.16.840.1.113883.5.111" codeSystemVersion="2.01.4"
displayName="DEPEN"/>
                        <statusCode code="active"/>
                        <effectiveTime xsi:type="IVL_TS">
                           <low value="20070127" inclusive="true"/>
                           <high value="20070526" inclusive="true"/>
                        </effectiveTime>
                     </asPartOfWhole>
                     <contactParty classCode="CON">
                        <id root="1.2.3" extension="158932302"
displayable="false"/>
                        <id root="1.2.3" extension="158932402"
displayable="false"/>
                        <code code="DEPEN" codeSystemName="RoleCode"
codeSystem="2.16.840.1.113883.5.111" codeSystemVersion="2.01.4"
displayName="DEPEN"/>
                        <telecom use="H" value="tel:1-690-555-1111"/>
                        <telecom use="WP" value="tel:1-690-555-1111"/>
                        <contactPerson classCode="PSN" determinerCode="INSTANCE">
                           <name use="L">
                              <family partType="FAM"
representation="TXT">Levin</family>
                              <given partType="GIV" representation="TXT">Henry
Jr</given>
                           </name>
                           <name use="L">
                              <family partType="FAM" representation="TXT"
qualifier="VV">Levin</family>
                              <given partType="GIV" representation="TXT">Henry
Sr</given>
                           </name>
                           <telecom use="H" value="tel:1-690-555-1111"/>
                           <telecom use="WP" value="tel:1-690-555-1111"/>
                           <asPersonDomain classCode="IDENT">
                              <id root="1.2.3" extension="158932602"
displayable="false"/>
                              <code code="DEPEN" codeSystemName="RoleCode"
codeSystem="2.16.840.1.113883.5.111" codeSystemVersion="2.01.4"
displayName="DEPEN"/>
                           </asPersonDomain>
                        </contactPerson>
                     </contactParty>
                  </employerOrganization>
               </employmentStaff>
            </subject>
         </registrationEvent>
      </subject>
   </controlActProcess>
</ MFPM_IN000101 >
```

Error Messages

Note: The & sign in the Error Message column of the above table indicates variable name. MTK displays the value of these variable at runtime.

Error Code	Exception Type	Error Message	Error Description
CTB_MTK_NO_SCHEMA_INFO	CTBValidationRuntimeException	There is no SCHEMA in the list provided to test. Client application has passed a null SCHEMAINFORMATION object.	This error shows that schemainformation object is not passed to the API .Please create schemainformation object array of xsds and mifs which are to be tested and pass it to the API.
CTB_MTK_NO_MIF_CONTENT	CTBValidationRuntimeException	The content of the MIF file `&MIF_FILE_NAME` is empty.	In schemainformation object only MIF file name is set but the content of the file is not set . Please set the MIF file content as well on the schemainformation object.
CTB_MTK_NO_XSD_CONTENT	CTBValidationRuntimeException	The content of the XSD file `&XSD_FILE_NAME` is empty.	In schemainformation object only XSD file name is set but the content of the file is not set . Please set the XSD file content as well on the schemainformation object.
CTB_MTK_NO_MIF_AND_XSD_CONTENT	CTBValidationRuntimeException	Both XSD `&XSD_FILE_NAME` and MIF `&MIF_FILE_NAME` are not present.	In schemainformation object both XSD and MIF file name is set but the content of the files are not set . Please set the XSD and MIF file content as well on the schemainformation object.
CTB_MTK_INVALID_SCHEMA_TYPE	CTBValidationRuntimeException	Validation of input parameters failed. SCHEMA INFORMATION TYPE `&SCHEMA_INFO_TYPE` is not supported.	SchemaInformation object requires a SCHEMA INFORMATION TYPE parameter to be set which can have following values 1.PAYLOAD 2.CMET 3. COREMIF. If another value other than these 3 values set for SCHEMA INFORMATION TYPE is encountered, this error is thrown . Please use the correct SCHEMA INFORMATION TYPE values in uploaded schemainformation object.
CTB_MTK_DUPLICATE_XSD	CTBValidationRuntimeException	XSD `&XSD_FILE_NAME` already exists as oracle ARTIFACTs. Please rename the ARTIFACT.	This error shows that the xsd which user is trying to upload is already present as an Oracle supplied artifact . Please change the name of xsd (other than oracle supported ones) and set it on the schemainformation object.
CTB_MTK_NO_CORE_MIF	CTBValidationRuntimeException	The content of file CMETINFO.COREMIF file is empty.	MTK throws this error when in the schemainformation object schema_type is set as coremif but the content of the files on the object are null. Please pass the coremif content as well in the schemainformation object.
CTB_MTK_PAYLOAD_NT_FOUND	CTBValidationRuntimeException	PAYLOAD XSD is not present in the SCHEMA information passed to MESSAGING TOOL KIT.	MTK throws this error when in the schemainformation object does not contain payload xsd . Please pass the payload xsd information in schemainformation object.
CTB_MTK_DUPLICATE_PAYLOAD	CTBValidationRuntimeException	More than one PAYLOAD present in the SCHEMA information passed to MESSAGING TOOL KIT.	MTK throws this error when in the schemainformation object array have more than one payload schemainformation objects . Please upload only one payload in schemainformation object array.

CTB_MTK_UNSUPPORTED_RIM_CLASS	CTBRuntimeException	PAYLOAD CLASS type '&PAYLOAD_TYPE' is unknown.	If the payload entry point class is not of type ROLE or ACT, this error is thrown. Please use only ROLE or ACT payload types i.e the payload must have ROLE or ACT as entry point class.
CTB_MTK_INTR_XSD_BUILD_FAILED	CTBRuntimeException	INTERACTION SCHEMA validation failed. INTERACTION SCHEMA path: &INTR_SCHEMA. CAUSE: &CAUSE	This error occurs when the uploaded xsds are incorrect or which have the datatypes or vocabulary values which are not supported in HDR. Please check that xsds generated by schema generator tool are correct and recheck the datatypes and vocabulary values used in RMIM (those have to be supported in HDR). Schema validation failure for interaction schema.
CTB_MTK_MSG_PARSING_FAILED	CTBRuntimeException	XML message parsing failed. CAUSE: &CAUSE	This is a program error where MTK is not able to parse the test message to persist it.
CTB_MTK_TRG_NOT_FOUND	CTBRuntimeException	TRIGGER CODE &TRIGGER_EVENT_CODE is not available in CODE SYSTEM &CODE_SYSTEM_NAME. CAUSE: &CAUSE	This is a program error where the triggerCode generated in test message is an invalid concept.
CTB_MTK_RIM_ASSOCIATION_NT_FND	CTBRuntimeException	RIM ASSOCIATION not found in METADATA for ELEMENT name: &ELEMENT_NAME and parent COMPLEX TYPE&PARENT_COMPLEX_TYPE. MIF or XSD for this complex type may be invalid.	MTK throws this error when it is unable to find the RIM association metadata for a RIM class from the database. As the metadata is generated from MIF file. Please check the MIF file is correct. The MIF file name can be found out from the complex type name given in the error message.
CTB_MTK_PARNT_CMPLX_TYP_NT_FND	CTBRuntimeException	The parent COMPLEX TYPE for the ELEMENT '&ELEMENT_NAME' is not found in METADATA. MIF or XSD for the complex type may be invalid	MTK throws this error when it is not able to find parent complex type for an element from MIF metadata. Please check the MIF file is correct. The MIF file name can be found out from the complex type name given in the error message.
CTB_MTK_CODING_SCHEME_NT_FOUND	CTBRuntimeException	Unable to get coding scheme, for CODE: &CODE, and CODE System Name: &CODE_SYSTEM_NAME. CAUSE: &CAUSE	This is a Program error where MTK is not able to find the code and code system name present in the test message from ETS.
CTB_MTK_PAYLOAD_INFO_NT_FOUND	CTBRuntimeException	Unable to parse PAYLOAD MIF file. PAYLOAD MIF file does not contain PAYLOAD ELEMENT, COMPLEX TYPE information.	This is a program error which occurs due to failure in parsing of payload mif file.
CTB_MTK_CMPLX_TYPE_NT_FOUND	CTBRuntimeException	XSD COMPEX TYPE object for the COMPLEX TYPE name &COMPLEX_TYPE_NAME is not found in the xsd.	This is a program error. This error occurs when MTK does not find a complex type in the xsd.

CTB_MTK_INCORRECT_FOCAL_CLASS	CTBValidationRuntimeException	The PAYLOAD XSD has unsupported focal CLASS. It should be either role or act.	If the payload entry point class is not of type ROLE or ACT, this error is thrown. Please use only ROLE or ACT payload types i.e the payload must have ROLE or ACT as entry point class.
CTB_MTK_FILE_READ_FAILED	CTBRuntimeException	Unable to read file &FILE_NAME	This is a program error when MTK is unable to read a file from disk.
CTB_MTK_FILE_WRITE_FAILED	CTBRuntimeException	Unable to write into the FILE: &FILE_NAME. CAUSE: &CAUSE	This is a program error when MTK is unable to write a file to disk.
CTB_MTK_METDATA_NOT_FOUND	CTBRuntimeException	Unable to find METADATA for COMPLEX TYPE &COMPLEX_TYPE_NAME	MTK throws this error when its not able to find metadata for a complex type. Please check the MIF file is correct. The MIF file name can be found out from the complex type name given in the error message.
CTB_MTK_CLASS_CODE_NT_FOUND	CTBRuntimeException	Unable to find CLASS CODE value for COMPLEX TYPE &COMPLEX_TYPE_NAME	This is program error where MTK is not able to find a value for class Code attribute for particular xsd complex type. The error occurs when MTK is trying to generate the test message.
CTB_MTK_PAYLOAD_NOT_FOUND	CTBValidationRuntimeException	Unable to find PAYLOAD in INTERACTION SCHEMA.	This is a program error, When MTK tries to find the payload complex type from interaction schema to generate the test instance, it does not find that and throws this error.
CTB_MTK_INCORRECT_INSTANCE	CTBValidationRuntimeException	Test message generation failed for INTERACTION SCHEMA &INTERACTIONS_SCHEMA. CAUSE: &CAUSE	This is a program error, This error occurs when MTK generates and incorrect test message and the schema validation for the message fails.
CTB_MTK_CODED_ATTR_NOT_CREATED	CTBRuntimeException	Unable to create XML fragment for CODED attribute for ELEMENT &ELEMENT_NAME. CAUSE: &CAUSE	This is a program error. This error occurs when MTK is unable to fetch ETS data for coded attributes to generate the test instance.
CTB_MTK_NOT_ENOUGH_VAL	CTBValidationRuntimeException	Concept list for CODEd attribute has less than two values for CONCEPT LIST &COCEPTLIST_NAME and XML ELEMENT name &ELEMENT_NAME	If a concept list for a coded attribute has less than 2 concepts then this error is thrown . Please insert minimum of 2 concepts in all the extensible concept list.
CTB_MTK_II_FRAGMNT_NT_CREATED	CTBRuntimeException	Unable to create XML fragment for CODED attribute for ELEMENT &ELEMENT_NAME. CAUSE: &CAUSE	This is a program error. This error occurs when MTK is not able to create id attribute data to generate a test instance.
CTB_MTK_XML_FRAGMENT_NT_FOUND	CTBRuntimeException	XML fragment is not seeded for data type &DATA_TYPE	When MTK does not find a xml fragment for a datatype to generate a test message, this error is thrown. Please verify the only HDR supported datatypes are used in the RMIM.
CTB_MTK_ELMT_METADATA_NOTFOUND	CTBRuntimeException	Metadata not found for an element name &ELEMENT_NAME and parent complex type name &COMPLEX_TYPE_NAME	MTK throws this error when its not able to find metadata for a complex type. Please check the MIF file is correct or not. The MIF file name can be found out from the complex type name given in the error message.

CTB_MTK_MSG_NT_PERSISTED	CTBValidationRuntimeException	IMP persistence failed. Acknowledgement message is &ACKNOWLEDGEMENT	When MTK tries to persist the message through IMP and it fails in persistence, this error is thrown. Please fix the error shown in acknowledgement message and rerun the test case.
CTB_MTK_ATTR_VAL_NT_FOUND	CTBRuntimeException	Unable to generate test INSTANCE . No default value found in XSD for ATTRIBUTE &ATTR_NAME present in COMPLEX TYPE &COMPLEX_TYPE_NAME	This is a program error. When MTK is not able to find a value for a structural attribute value from xsd or mif this error is thrown .
CTB_MTK_INVALID_ACTRELATION	CTBValidationRuntimeException	Validation of input SCHEMA failed. ACTRELATIONSHIP COMPLEX TYPE &COMPLEX_TYPE do not have TARGET ACT.	This error occurs when MTK finds the an act relationship in xsd does not have a target act . Please fix the RMIM and XSD to have a target act for the corresponding act relationship.
CTB_MTK_XSD_LOAD_FAILED	CTBRuntimeException	Loading of Schema &XSD_PATH has failed. CAUSE: &CAUSE	This error is thrown by MTK when the validation of uploaded xsd is failed . Please check the input xsds are correct.
CTB_MTK_INVALID_OB_PRTCPN	CTBValidationRuntimeException	Validation of input SCHEMA failed. OUTBOUND PARTICIPATION COMPLEX TYPE &COMPLEX_TYPE do not have target ROLE.	This error occurs when MTK finds the a participation in xsd does not have a participant role . Please fix the RMIM and XSD to have a participant role for the corresponding participation.
CTB_MTK_INVALID_IB_PRTCPN	CTBValidationRuntimeException	Validation of input SCHEMA failed. INBOUND PARTICIPATION COMPLEX TYPE &COMPLEX_TYPE do not have TARGET ACT.	This error occurs when MTK finds the an inbound participation in xsd does not have target act . Please fix the RMIM and XSD to have a target act for the corresponding participation.
CTB_MTK_XSD_MIF_ELMT_MISMATCH	CTBValidationRuntimeException	Input SCHEMA and MIF files does not match. Element &ELEMENT_NAME found in SCHEMA COMPLEX TYPE &COMPLEX_TYPE is missing in corresponding MIF file.	This error occurs when there is a mismatch between MIF and XSD file of the same artifact i.e an element found XSD file is missing from the MIF file.
CTB_MTK_XSDMIF_CMPXTY_MISMATCH	CTBValidationRuntimeException	Input SCHEMA and MIF files does not match. SCHEMA COMPLEX TYPE &COMPLEX_TYPE is missing in corresponding MIF file.	This error occurs when there is a mismatch between MIF and XSD file of the same artifact, that is, a complex type found XSD file is missing from the MIF file.
CTB_MTK_MS_SCM_VALIDATN_FAIL	CTBValidationRuntimeException	Message generated by OMP failed to validate against the schema. CAUSE: &CAUSE	This is a program error. When MTK generates the persisted test message using OMP, the generated message fails schema validation and this error is thrown . Please contact your system administrator.

CTB_MTK_ COREMIF_PARSE_ FAILED	CTBValidationRuntimeException	Unable to merge the existing CMETINFO.COREMIF with the one provided as input, because parsing of &CORE_MIF failed. CAUSE: &CAUSE.	For merging the CMETINFO.COREMIF file new uploaded CMETINFO.COREMIF has to be parsed, if the CMETINFO.COREMIF is not a valid xml file then MTK throws this error. Please fix CMETINFO.COREMIF and upload again.
CTB_MTK_NO_ DEFAULT_ CONCEPT	CTBValidationRuntimeException	CONCEPT LIST &CONCEPT_LIST should contain at least one CONCEPT from default CODING SCHEME VERSION.	MTK throws this error when the default coding scheme has no valid concepts . Please ensure that all coding schemes have default version and should have at least one valid concept and concept list. These should have concepts from default coding schemes.
CTB_MTK_ACTIVE_ STATUSCD_NTFND	CTBValidationRuntimeException	Test message generation failed. STATUS CODE CONCEPT LIST &CONCEPT_LIST do not have CONCEPT with CONCEPT CODE ACTIVE	MTK throws an error while generating a test instance,when it does not find an "ACTIVE" statusCode in the system conceptlist . Please ensure that ACTIVE concept is available in the concept list.
CTB_MTK_MANDT_ INPUT_MISSING	CTBValidationRuntimeException	The input parameter &TOKEN_NAME is passed as null. The request cannot be processed,because mandatory parameter &TOKEN_NAME has null value.	MTK throws this error when any mandatory input parameter is not passed by the user.
CTB_MS_CMET_ INFO_NOT_FOUND	CTBValidationRuntimeException	Unable to extract METADATA. CMET info file &CMETINFO_FILENAME not found.	MTK throws this error from metadataservice when metadataservice is not able to find the CMETINFO.COREMIF file. Please check the CMETINFO.CORE MIF file is available on the server.
CTB_MS_CMET_ TYPE_NOT_FOUND	CTBValidationRuntimeException	Unable to extract METADATA. CMET reference &REFERENCE in MIF file &MIFFILE is missing in CMETINFO.COREMIF file.	MTK throws this error from metadataservice when metadataservice is not able to find the an entry of cmet in CMETINFO.COREMIF file. Please check the corresponding CMET entry is present in CMETINFO.COREMIF file.
CTB_MS_MIFFILE_ NOT_FOUND	CTBValidationRuntimeException	Unable to extract METADATA. MIF file &MIFFILE not found .	MTK throws this error from metadataservice when metadataservice is not able to find the MIF file to load the metadata. Please check the corresponding MIF file is available on the server.
CTB_MS_MIF_ PARSING_FAILED	CTBValidationRuntimeException	Unable to extract METADATA. Unable to parse MIF file &MIFFILE.	MTK throws this error from metadataservice when metadataservice is not able to parse a MIF file. Please check all MIF files are valid xmls.
CTB_MS_ CMETINFO_ PARSING_FAILED	CTBValidationRuntimeException	Unable to extract METADATA. Unable to parse CMET info file &CMETINFO_FILENAME.	MTK throws this error from metadataservice when metadataservice is not able to parse CMETINFO.COREMIF. Please check CMETINFO.COREMIF file is valid xmls.

CTB_MS_INTR_ SCHEMA_NOT_ FOUND	CTBValidationRunti meException	Unable to extract METADATA. INTERACTION SCHEMA file [&INTERACTION_SCHEMA] not found.	MTK throws this error from metadataservice when metadataservice is not able to find interaction schema on the server.
CTB_MS_SCHEMA_ PARSING_FAILED	CTBValidationRunti meException	Unable to extract METADATA. Unable to parse INTERACTION SCHEMA &SCHEMA_FILE.	MTK throws this error from metadataservice when metadataservice is not able to parse interaction schema on the server. Please contact your system administrator.
CTB_MS_MIF_DIR_ NOT_FOUND	CTBValidationRunti meException	Unable to extract METADATA. MIF directory &DIR_PATH not found.	MTK throws this error from metadataservice when metadataservice is not able to find the directory for MIF files on the server.
CTB_MS_ WRAPPER_ METADT_ NOTFOUND	CTBValidationRunti meException	Unable to load metadata for an INTERACTION SCHEMA &SCHEMAFILE. Unable to find metadata for parent COMPLEX TYPE &COMPLEXTYPE of PAYLOAD &PAYLOAD_ ELEMENT.	MTK throws this error from metadataservice when metadataservice is not able to find metadata for the parent complex type element of the payload element in the database.
CTB_MTK_STATUS_ FILE_NT_FOUND	CTBValidationRunti meException	The STATUS file '&STATUS_FILE' is not available on the server	MTK throws this error when it tries to download the status file from TEST_ID folder but does not find the folder with that TEST_ID name on the server . Please pass the correct TEST_ID name.
CTB_MTK_TEST_ ID_DIR_NOT_FND	CTBValidationRunti meException	The TEST ID directory &TEST_ID is not present on the server	MTK throws this error when the Test ID folder is not found in the server.
CTB_MTK_CMET_ ID_CONFLICT	CTBValidationRunti meException	Unable to merge existing CMETINFO.COREMIF with user provided CMETINFO.COREMIF. CMET name &CMET_NAME already exists with different CMET ID. Existing CMET id &EXISTING_CMET_ID and new CMET ID &NEW_ CMET_ID.	This error occurs when user try to upload a CMETINFO.COREMIF file which has an entry with same cmet name as that of the one which is already uploaded but the corresponding CMET IDs for the new one and existing one are different .So if the new file is merged with the existing file , then there will be 2 entries of the same CMET name with a different CMET ID. This is incorrect. Please upload unique CMET names at all times and not applied to other CMET IDs.
CTB_MTK_CMET_ NAME_CONFLICT	CTBValidationRunti meException	Unable to merge existing CMETINFO.COREMIF with user provided CMETINFO.COREMIF. CMET id &CMET_ID already exists with different CMET name. Existing CMET name &EXISTING_CMET_NAME and new CMET name &NEW_CMET_NAME.	This error occurs when user try to upload a CMETINFO.COREMIF file which has an entry with same CMET ID as that of the one which is already uploaded. These have different CMET names .Therefore if the new file is merged with the existing file, then there will be two entries of the same CMET ID with different CMET names. To avoid this error please upload unique CMET IDs which are not already used with other CMET names.

CTB_MTK_DUPLICATE_INTR_ID	CTBRuntimeException	The INTERACTION detail for the INTERACTION ID &INTERACTION_ID already exists. Unable to create INTERACTION detail	When MTK tries to create a record for interaction ID in the database but the same interaction ID is already present then this error is thrown . Please use a different interaction ID which is not already seeded.
CTB_MTK_UNSUPPORTED_IP_PARAMS	CTBRuntimeException	Unable to create Composite Message Schema. The input parameter &TOKEN_VALUE for &TOKEN_NAME is not supported	MTK throws this error when any input parameter is passed incorrectly; for example if a transmission wrapper ID passed by the user does not match with the transmission wrapper ID which HDR supports MTK throws this error. Please pass the correct input values.
CTB_MTK_PAYLOAD_ID_NOT_EXISTS	CTBValidationRuntimeException	Unable to create INTERACTION SCHEMA PAYLOAD ID does not exist for the given INTERACTION ID: &INTERACTION_ID	MTK throws this error when it does not find payload xsd on the server to create an interaction schema for it. Please upload the payload xsd first and then create the interaction schemas for it.
CTB_MTK_PAYLOAD_XSD_NT_FOUND	CTBValidationRuntimeException	Unable to generate INTERACTION SCHEMA because Message Type XSD &XSD_PATH is not available.	This error occurs when MTK does not find payload xsd on server with name given by the user . Please pass the correct payload ID as input.
CTB_MTK_SCHEMA_IN_USE	CTBValidationRuntimeException	Unable to delete the SCHEMA &SCHEMA from server.The SCHEMA is referred by the PAYLOAD &PAYLOAD.	The error occurs when a schema is tried to be deleted, is referred by a payload present on the server. To delete the schema please delete the referring payload first.
CTB_MTK_HTB_ARTIFACT_FOUND	CTBValidationRuntimeException	Unable to delete ARTIFACTS. Customer cannot delete HDR specific ARTIFACT &ARTIFACT.	MTK throws this error when user tries to delete Oracle artifacts . You are not supposed to delete Oracle artifacts.
CTB_MTK_INTR_SCHEMA_REF	CTBValidationRuntimeException	Can not delete PAYLOAD SCHEMA &PAYLOAD_SCHEMA as INTERACTION IDS &INTERACTION_IDS are configured for this PAYLOAD.	MTK throws this error when you tries to delete a payload and that payload has a interaction ID configured in the table. Please delete the interaction IDs for the payload and then delete the payload.
CTB_MTK_XSD_NOT_EXISTS	CTBValidationRuntimeException	Unable to remove XSD &XSD_FILE, as it does not exist.	This error occurs when an xsd is tried to be deleted which is not present on the server. In this case the xsd file name might be incorrectly passed. Please pass the correct xsd file name.
CTB_MTK_PAYLOAD_FOUND	CTBValidationRuntimeException	Invalid input data because input contains a SCHEMA of a PAYLOAD. LOADSCHEMA API accepts SCHEMAS of CMETS only. Instead use API LOADCUSTOMMESSAGETYPETOPRODUCTIONSERVER.	MTK throws this error when the uploaded schema is of type PAYLOAD . Only CMETS are allowed to upload using this API. Please upload only CMETS.

CTB_MTK_INTR_DETAIL_NT_FND	CTBValidationRuntimeException	INTERACTION detail for INTERACTION ID `&INTERACTION_ID` is not available. Can not delete the INTERACTION detail entry.	When MTK tries to delete an interaction schema on the server ,and if it does not find a record for the corresponding interaction ID in the interactions table, it throws this error . Please ensure that interaction ID record is available in the interactions table before deleting it.
CTB_MTK_INTR_RECV_CONFIG_EXIST	CTBValidationRuntimeException	Can not delete INTERACTION SCHEMA for INTERACTION ID `&INTERACTION_ID` as RECEIVERS are configured for this INTERACTION ID.	When MTK tries to delete the interaction schema and for that interaction ID receivers are already configured then MTK cannot delete the interaction schema and throws this error. Please ensure that no receivers are configured for the interaction schema which is to be deleted.
CTB_MTK_INTR_SENDER_CONF_EXIST	CTBValidationRuntimeException	Can not delete INTERACTION SCHEMA for INTERACTION ID `&INTERACTION_ID` as SENDERS are configured for this INTERACTION ID.	When MTK tries to delete the interaction schema and for that interaction ID senders are already configured then MTK cannot delete the interaction schema and throws this error. Please ensure that no senders are configured for the interaction schema which is to be deleted.

11

HDR Exception Handling

- Optimistic Locking Exceptions
- Bundled Exceptions

HDR provides a hierarchy of exception classes, which allow users to distinguish between validation errors and system errors (For example, locking errors, session timeout errors, and unexpected errors). All exception classes extend from the CommonException class.

To obtain information about as many issues as possible, exceptions may be received as a bundle with each child exception representing a different issue.

HDR translates server-side root cause exception to equivalent CommonException in the client-side EJB exception.

> **Note:** The mapping between the server-side exception to the equivalent client ETSException or its subclass is mentioned in the ExceptionConversionRules.txt file (available at <HDR_HOME>/hdr_exploded_app/oracle/apps/ctb/fwk/ex/server/ExceptionConversionRules.txt).

The code samples below help you to manage the following scenarios:

- Optimistic Locking Exception
- Bundled Exceptions
- Helper Methods to Print Details of the Exception Bundle

Optimistic Locking Exceptions

If HDR receives an OptimisticLockException from the underlying JPA framework, it wraps the exception in oracle.hsgbu.hdr.base.persist.exception.CorePersistenceException with error code HDR_OBJECT_ALREADY_MODIFIED_ERROR. The client application should attempt to query again if necessary and resubmit updates to HDR.

Example 11–1 Optimistic Locking Exception

The following code sample illustrates an optimistic locking exception:

```
// create an Act
SET_II actId = dataTypeFactory.newSET_II("9.989898.5.6.100", "OBS1001", true);
Act act = actFactory.newObservation(ActMood.EVN, null, actId);
```

```
ControlAct controlAct = actFactory.newControlActEvent(
  dataTypeFactory.nullCD(NullFlavor.NI), dataTypeFactory.nullSET_
II(NullFlavor.NI));
controlAct.addOBActRelationship(ActRelationshipType.SUBJ, act);
rimService().submit(controlAct);

// retrieve two copies of the Act
Act retrievedAct1 = retrieveAct(actId);
Act retrievedAct2 = retrieveAct(actId);

// update the Act
ControlAct controlAct2 = actFactory.newControlActEvent(
  dataTypeFactory.nullCD(NullFlavor.NI), dataTypeFactory.nullSET_
II(NullFlavor.NI));
controlAct2.addOBActRelationship(ActRelationshipType.SUBJ,
(Act)retrievedAct1.createNewVersion());
rimService().submit(controlAct2);

// try updating the same Act with the other retrieved copy
ControlAct controlAct3 = actFactory.newControlActEvent(
  dataTypeFactory.nullCD(NullFlavor.NI), dataTypeFactory.nullSET_
II(NullFlavor.NI));
controlAct3.addOBActRelationship(ActRelationshipType.SUBJ,
(Act)retrievedAct2.createNewVersion());
try
{
  rimService.submit(controlAct3);
}
catch (CTBLockingException e)
{
  // One or more objects in the submission
  // are locked by another process.
  //
  // Requery if necessary, and resubmit
  // updates to HDR.
  //
  // In this case, we know the object is
  // the Observation which was queried and
  // versioned, but this can be verified by
  // inspecting the Exception

  Act requeriedAct = retrieveAct(actId);
  ControlAct newControlAct = actFactory.newControlActEvent(
    dataTypeFactory.nullCD(NullFlavor.NI), dataTypeFactory.nullSET_
II(NullFlavor.NI));
  newControlAct.addOBActRelationship(ActRelationshipType.SUBJ,
(Act)requeriedAct.createNewVersion());
  rimService.submit(newControlAct);
}
```

Bundled Exceptions

If HDR detects multiple validation errors during processing it returns them wrapped in a CommonException or a subclass of it. These validation errors may be accessed and inspected by calling getBundledExceptions() at the top-level exception.

Example 11–2 Bundled Exceptions

The following code sample illustrates how to handle bundled exceptions:

```
Observation focalAct = mActFactory.newObservation(ActMood.EVN,
  mDataTypeFactory.nullCD(NullFlavor.NI),
  mDataTypeFactory.nullSET_II(NullFlavor.NI));

Observation obs = mActFactory.newObservation(ActMood.EVN,
  mDataTypeFactory.nullCD(NullFlavor.NI),
  mDataTypeFactory.nullSET_II(NullFlavor.NI));
obs.setStatusCode(mDataTypeFactory.newCS("INVALID_STATUS"));
obs.setMethodCode(mDataTypeFactory.newSET_CE(new CE[] {
  newCE("0251", "ObservationMethod"),
  newCE("DoesNotExist", "ObservationMethod")}));

Observation obs2 = mActFactory.newObservation(ActMood.EVN,
  mDataTypeFactory.nullCD(NullFlavor.NI),
  mDataTypeFactory.nullSET_II(NullFlavor.NI));
obs2.setStatusCode(mDataTypeFactory.newCS("A_DIFFERENT_STATUS"));

ControlAct controlAct = mActFactory.newControlActEvent(
  mDataTypeFactory.nullCD(NullFlavor.NI),
  mDataTypeFactory.nullSET_II(NullFlavor.NI));

cact.addOBActRelationship(ActRelationshipType.SUBJ, focal);
focalAct.addOBActRelationship(ActRelationshipType.COMP, obs);
focalAct.addOBActRelationship(ActRelationshipType.COMP, obs2);

try
{
  mRimService.submit(controlAct);
}
catch (CommonException e)
{
  StringBuffer sb = new StringBuffer();
  addExceptionDetailToBuffer(sb, e, 0);
  System.out.println(sb.toString());
}
```

Example 11–3 Helper Methods to Print Details of the Exception Bundle

The following code is a sample method that shows how to get exception details from bundled exceptions:

```
private StringBuffer addExceptionDetailToBuffer(StringBuffer sb,
           CommonException e, int indent) {
        sb.append(getIndentString(indent));
        sb.append(e.getClass().getName());
        sb.append(" ").append(e.getExceptionCode());
        sb.append(" ").append(e.getMessage());
        sb.append("\n");

        CommonException[] bundle = e.getBundledExceptions();
        for (int i = 0; i < bundle.length; i++) {
            sb = addExceptionDetailToBuffer(sb, bundle[i], indent + 1);
        }

        return sb;
    }
```

The following is the printed description of the exception bundle obtained from this call to RimService.submit(ControlAct):

[java] oracle.hsgbu.hdr.fwk.base.common.HDRRimException CTB_FK_MLTPL_VALIDATION_XCPTNS Multiple errors occurred processing request. [java]

oracle.hsgbu.hdr.hl7.common.RimCodedAttributeException CTB_CORE_ETS_MEM_CODE_INVALID Concept code not found in concept list. [java] oracle.hsgbu.hdr.hl7.common.RimCodedAttributeException CTB_CORE_UNKNOWN_CODE Concept Code was not found for provided Code System. [java] oracle.hsgbu.hdr.hl7.common.RimCodedAttributeException CTB_CORE_ETS_MEM_CODE_INVALID Concept code not found in concept list. [java] oracle.hsgbu.hdr.hl7.rim.types.exception.common.RimDataTypeException HDR_CORE_GTS_PARSER Invalid syntax encounterd when parsing CE literal: '"ObservationMethod"';' Unexpected character O encountered at position 0 in OID literal string "ObservationMethod".

12

Integrating the Healthcare Enterprise

Cross-Enterprise Document Sharing-b (XDS.b)

The Cross-Enterprise Document Sharing-b (XDS.b) IHE Integration Profile focuses on providing a standards-based specification for managing the sharing of XDS documents across healthcare enterprises.

An XDS document is a composition of clinical information that complies with a published standard defining the document structure, content, and encoding. As a single unit for exchange in the repository, an XDS document is assigned with a globally unique identifier and is associated with its own metadata that reflects the content of the document. The metadata is not stored in the repository along with the document, but is registered in the Document Registry for subsequent queries and retrievals of the document.

Similar to any IHE Integration Profile in the IHE IT Infrastructure Technical Framework, the XDS profile is defined by the IHE actors involved and the set of transactions performed by these actors. To support a certain integration capability, the healthcare industry must have a product implementing the IHE actors and the corresponding transactions deployed.

This section contains the following topics:

IHE Actors

The various actors in IHE XDS.b ITI integration profile are:

Actors in IHE XDS.b:

- **Document Source:** Sends documents to a Document Repository and supplies metadata for registration.
- **Document Consumer:** Queries the Document Registry for documents and retrieves documents from a document repository.
- **Document Registry:** Validates and maintains metadata for each document and responds to document queries from the Document Consumer Actor.

- **Document Repository:** Persists documents and registers the document metadata with appropriate Document Registry. It also assigns a uniqueId to documents for subsequent retrieval by a Document Consumer.
- **Patient Identity Source:** Provides a unique identifier for each patient.
- **Integrated Document Source/Repository:** Combines the functionality of the Document Source and Document Repository actors to provide and register document sets in the repository.

The HDR's IHE XDS.b solution implements the Document Repository actor and its supported transactions.

Affinity Domain

An Affinity Domain is an administrative structure containing various healthcare entities that have agreed to share clinical documents in the common infrastructure.

To ensure effective interoperability between the different entities, a Document Registry is identified, and a number of policies are established in an Affinity Domain that specify the document format, vocabulary value set, coding schemes, and the Patient Identification Domain used by the Document Registry.

To build your Affinity Domain, follow the guidelines specified in Template for XDS Affinity Domain Deployment Planning, which is available from, http://www.ihe.net/Technical_Framework/upload/IHE_ITI_White_Paper_XDS_Affinity_Domain_Template_TI_2008-12-02.pdf.

Integrating the Healthcare Enterprise Cross-Enterprise Document Sharing-b Web Service

HDR provides IHE XDS.b implementation to help healthcare practitioners and clinical vendors implement medical information integration among clinical organizations. The IHE XDS Document Repository is built on top of Oracle Healthcare Data Repository (HDR) platform.

The topics below describe:

HDR's IHE XDS.b Solution Overview

HDR implements the Document Repository Actor of the IHE XDS profile. It can be directly plugged into the IHE infrastructure and interact with the other Actors, such as the Document Registry, the Document Source, and the Document Consumer implemented by other entities.

HDR stores and manages XDS documents. The users accessing the repository can be clinical doctors, consultants, nurses, researchers, and patients.

HDR also implements the Time Client Actor in the Consistent Time (CT) profile, and the Secure Node Actor in Audit Trail and Node Authentication (ATNA) profile of IHE IT Infrastructure domain.

Supported IHE XDS.b Transactions

HDR supports the following transactions in the XDS workflow:

1. Provide And Register Document Set-b through synchronous Web services.
2. Retrieve Document Set through synchronous Web services.
3. Provide And Register Document Set-b through asynchronous Web services.
4. Retrieve Document Set through asynchronous Web services.

Synchronous Provide and Register Document Set-b

Prerequisite

Before proceeding with the document sharing features, make sure that the profile options for Document Repository actor like Document Registry server URL, Syslog server details, and TLS configuration for secure mode communication are configured as per the Installation Guide instructions.

Providing/Registering Documents

The processes of providing and registering documents are merged into a single task. This task is initiated from the Document Source Actor. It contains the following steps:

1. Generate the Patient ID for the documents. Obtain other optional entity IDs if necessary. An external partner outside of HDR implements the Patient Identity Actor.
2. Analyze the document content and generate the document metadata. For more information, refer to Section 4.2.3 Metadata Attributes in IHE_ITI_TF_Vol3.pdf.
3. Identify the URL to the HDR's Provide and Register Document Set-b. The URL is in the form: http://hostname:port/hdr/xdsrepositoryb_Soap12 or https://hostname:port/hdr/xdsrepositoryb_Soap12.
4. Wrap the document metadata and the document content in one or several HTTP/SOAP messages and submit the messages to the HDR.
5. There are many ways to implement the wrapping and submitting processes. The implementation is not mandated but the encoding and protocol must comply with the relevant specifications issued by OASIS/ebXML.
6. An acknowledgement of a successful registration and storing of the document will be received when the task is complete.

Document Storage Mode

HDR provides two behavior modes for document storage, as follows:

Store mode: If the profile option CTB_XDS_DOCUMENT_IMPORT is set to **N**.

A document received in the store mode is stored in HDR as an entire object. HDR does not parse the elements contained in the document.

Import mode: If the profile option CTB_XDS_DOCUMENT_IMPORT is set to **Y**.

A document received in the import mode is parsed by HDR, which identifies the elements inside the document and stores them separately.

Synchronous Retrieve Document Set

The retrieval of a registered document in the HDR is initiated from the Document Consumer. First query the Document Registry to get the document uniqueId, and then submit the request to the HDR to get the document.

The task generally consists of the following steps:

1. Obtain the Query Service URL and start the query for the target documents. The Query Service URL is in the form: http://hostname:port/hdr/xdsrepositoryb_Soap12 or https://hostname:port/hdr/xdsrepositoryb_Soap12.
2. Query the Document Registry for the list of documents meeting the specified criteria. The Query method varies but normally you should have the patient ID as the query input. For more information, refer to Section 3.18 Registry Stored Query in IHE_ITI_TF_Vol2a.pdf.
3. The Registry will return a list of documents along with their respective uniqueIds.
4. With the document uniqueIds of the selected documents from the list returned, the Document Consumer will send the Retrieve Document Set (ITI-43) requests to retrieve the documents. The documents are returned as entities in the response to the request. For more information, refer to Section 3.43 Retrieve Document Set in IHE_ITI_TF_Vol2b.pdf.

Asynchronous XDS.b Web Services

Prerequisites

Configure the following as per the instructions in HDR Implementation and System Administrator Guide:

1. JMS queues to address reliable messaging in asynchronous transactions.
2. The profile option for identifying the Document Registry actor's asynchronous URL.

Key Differences in the Asynchronous XDS.b Web Services

Asynchronous XDS.b (Provide and Register Document Set-b and Retrieve Document Set) transactions essentially accomplish the same set of tasks as their synchronous counterparts. However, the key aspects and differences in the asynchronous transactions are following:

Asynchronous Provide and Register Document Set-b

1. To initiate an Asynchronous Provide and Register Document Set-b transaction with HDR, the Document Source actor should identify the URL to HDR's Asynchronous Provide and Register Document Set-b. This URL is in the form: http://hostname:port/hdr/xdsrepositorybAsync_Soap12 or https://hostname:port/hdr/xdsrepositorybAsync_Soap12.
2. Document Source must specify a valid callback endpoint URL using the WS-Addressing ReplyTo property in the SOAP header of its Asynchronous Provide and Register Document Set-b request message.
3. Upon successful receipt of an Asynchronous Provide and Register Document Set-b request message, a HTTP response status code of 202, which means that request

has been accepted is sent back to the Document Source. After checking this status, the Document Source can unblock itself from the connection it initiated.

4. When an Asynchronous Provide and Register Document Set-b response is generated, HDR's Document Repository uses the callback endpoint specified in the SOAP request's ReplyTo property, to forward the response to the Document Source.
5. On receipt of an Asynchronous Provide and Register Document Set-b response message, the Document Source can use the SOAP response's RelatesTo property, to correlate the response with a request that it had sent out earlier.
6. The Document Source should finally respond with a HTTP response status code of 202, which means that request has been accepted to HDR's Doc Repository, marking the end of the transaction.

Audit Trail and Event Logs

All IHE XDS.b transactions generate audit events that are forwarded to a configured Audit Server. The location of the audit log in the Audit Server varies depending on the configuration of the Audit Server. This audit logging will be in addition and processed by the HDR's native logging mechanism.

13

HDR Clinical Document Architecture (CDA) Persistence

Overview

You can use the CDA Persistence or CDA Ingest Web service to parse and import various document types such as CDA, HITSP C32 v2.5, HITSP C32 v2.4, HITSP c37, HITSP C48, and HITSP C78 into HDR

This Web service offers two operations as follows:

1. importDocument(DocumentUniqueId) : Using the documentUniqueID provided by user, this operation will first query the document that was already persisted using IHE XDS.b ITI-41 transaction, parses the retrieved document, and imports or persists the document atomically into HDR DB.

   Also this operation throws the following SOAP faults to the user:

   - incorrectInput - When client does not specify the value for DocumentUniqueId element.
   - invalidCDADoc - When the validation of input xml against CDA.xsd fails.
   - noRootFound - When none of the templateID elements in input xml file have the root attribute value.
   - invalidDocIDFault - When there is no document with the provided uniqueID exists in HDR DB.
   - aFault - For all other erroneous conditions.

2. persistDocument(documentContent In Base64 format, documentId): This operation lets the user to provide the document in base64 format which they are interested to parse and import it in HDR DB atomically. This operation takes two parameters, a mandatory documentContent parameter, and an optional documentId. The first parameter documentContent represents the document in Base64 format. In case the user also provides the second optional parameter, documentId, then the Service will query the DOCCLIN Act using the provided documentId, and establishes a SUBJ Act relationship between the queried documentId and the newly persisted document's documentId.

   Also this operation throws the following SOAP faults to the user:

- incorrectInput - When client doesn't specify the value for Document element.
- invalidCDADoc - When the validation of input xml against CDA.xsd fails.
- noRootFound - When none of the templateID elements in input xml file have the root attribute value.
- invalidDocIDFault - When there is no document with the provided uniqueID exists in HDR DB.
- aFault - For all other erroneous conditions.

Note: This service will override the already existing document import mode feature in IHE XDS.b Web Service.

Having this Web service available in HDR, it is not recommended to use IHE XDS.b ITI-41 transaction (ProvideAndRegisterDocumentSet-B) to import the document into HDR. The IHE XDS.b ITI-41 transaction should be used to store the CDA, HITSP C32 v2.5, HITSP C32 v2.4, HITSP C37, HITSP C48, HITSP C78, and Discharge Summary documents into HDR. If customers want to import the document, they should rely on importDocument or persistDocument operations provided by this CDA Persistence Web service.

WSDL CDA Persistence Implementation

Click this link for WSDL that describes CDAPersistenceWebService with SOAP11 Binding.

Click this link for WSDL that describes CDAPersistenceWebService with SOAP12 Binding.

WS-Security for CDA Persistence Web Service

In HDR, the CDA Persistence Web service is implemented with WS-Security UserName Token profile. The Web service client has to supply UserName and Password in SOAP Header of the SOAP request that will be used to load the CDA Persistence Web service. The WebLogic server performs the user authentication using the Web service client provided UserName and Password values.

Sample CDA Persistence --> persistDocument Web Service SOAP request

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:orac="http://oracle.apps.ctb.cdapersistence.types">

   <soap:Header>

      <wsse:Security soap:mustUnderstand="true"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-sec
ext-1.0.xsd"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-util
ity-1.0.xsd">

         <wsse:UsernameToken wsu:Id="UsernameToken-10">

            <wsse:Username>IHE_XDS_USER</wsse:Username>

            <wsse:Password
Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profi
```

```
le-1.0#PasswordText"><USER_PASSWORD></wsse:Password>

         </wsse:UsernameToken>

      </wsse:Security>

   </soap:Header>

   <soap:Body>

   <orac:PersistCDADocumentRequest>

<orac:Document>PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz4NCjwhLS0gZWRpdGV
kIHdpdGggWE1MU3B5IHYyMDExIChodHRwOi8vd3d3LmFsdG92YS5jb20pIGJ5IExlbiBHYWxsYWdoZXIgK
E5JU1QpIC0tPg0KPCEtLVNhbXBsZSBYTUwgZmlsZSBnZW5lcmF0ZWQgYnkgWE1MU3B5IHYyMDA3IHNwMiA
oaHR0cDovL3d3dy5hbHRvdmEuY29tKS0tPg0KPENsaW5pY2FsRG9jdW1lbnQgeG1sbnM9InVybjpobDctb
3JnOnYzIiB4bWxuczp4c2k9Imh0dHA6Ly93d3cudzMub3JnLzIwMDEvWE1MU2NoZW1hLWluc3RhbmNlIiB
4c2k6c2NoZW1hTG9jYXRpb249InVybjpobDctb3JnOnYzIGh0dHA6Ly94cmVnMi5uaXN0Lmdvdjo4MDgwL
2hpdHNwVmFsaWRhdGlvbi9zY2hlbWEvY2RhcjJjMzIvaW5mcmFzdHJ1Y3R1cmUvY2RhL0MzMl9DREEueHN
kIj4NCgk8cmVhbG1Db2RlIGNvZGU9IlVTIi8+DQoJPHR5cGVJZCByb290PSIyLjE2Ljg0MC4xLjExMzg4M
y4xLjMiIGV4dGVuc2lvbj0iUE9DRF9IRDAwMDA0MCIvPg0KCTx0ZW1wbGF0ZUlkIHJvb3Q9IjIuMTYuODQ
wLjEuMTEzODgzLjMuMjcuMTc3NiIgYXNzaWduaW5nQXV0aG9yaXR5TmFtZT0iQ0RBL1IyIi8+DQoJPHRlb
XBsYXRlSWQgcm9vdD0iMi4xNi44NDAuMS4xMTM4ODMuMTAuMjAuMyIgYXNzaWduaW5nQXV0aG9yaXR5TmF
tZT0iSEw3L0NEVCBIZWFkZXIiLz4NCgk8dGVtcGxhdGVJZCByb290PSIxLjMuNi4xLjQuMS4xOTM3Ni4xL
jUuMy4xLjEuMSIgYXNzaWduaW5nQXV0aG9yaXR5TmFtZT0iSUhFL1BDQyIvPg0KCTx0ZW1wbGF0ZUlkIHJ
vb3Q9IjIuMTYuODQwLjEuMTEzODgzLjMuODguMTEuMzIuMSIgYXNzaWduaW5nQXV0aG9yaXR5TmFtZT0iS
ElUU1AvQzMyIi8+DQoJPGlkIHJvb3Q9IjIuMTYuMy41LjYiIGV4dGVuc2lvbj0iVGVzdDAxX0NDRDM5MCI
gYXNzaWduaW5nQXV0aG9yaXR5TmFtZT0iTklTVCBIZWFsdGhjYXJlIFByb2plY3QiLz4NCgk8IS0tVGhlI
EhMNy9DQ0Qgc3BlY2lmaWNhdGlvbiByZXF1aXJlcyB0aGF0IGV2ZXJ5IENDRCBDbGluaWNhbERvY3VtZW5
0IGNvbnRhaW4gdGhlICBzcGVjaWZpYyBMT0lOQyBjb2RlIDM0MTMzLTkuIE5vdGUgdGhhdCBNZWFuaW5nZ
nVsIFVzZSBkZWZlcnMgdG8gSUhFIHNwZWNpZmljYXRpb25zIHdoaWNoIGRvIG5vdCByZXF1aXJlIHRoZSB
DQ0QgdGVtcGxhdGVJZCBvbiB0aGUgZG9jdW1lbnQgaXRzZWxmLCB0aHVzIGFsbG93aW5nIG90aGVyIGNvZ
GVzIHRvIGJlIHVzZWQgYXQgdGhpcyBsb2NhdGlvbi4gVG8gYXZvaWQgY29uZnVzaW9uIGl0IG1heSBiZSB
iZXN0IHRvIHN0aWNrIHdpdGggdGhpcyBjb2RlIGFzIGZpeGVkIGluIHRoZSBhYnNlbmNlIG9mIGFub3RoZ
XIgcmVxdWlyZWQgc3Vic3RpdHV0ZS4tLT4NCgk8Y29kZSBjb2RlPSIzNDEzMy05IiBkaXNwbGF5TmFtZT0
iU3VtbWFyaXphdGlvbiBvZiBlcGlzb2RlIG5vdGUiIGNvZGVTeXN0ZW09IjIuMTYuODQwLjEuMTEzODgzL
jYuMSIgY29kZVN5c3RlbU5hbWU9IkxPSU5DIi8+DQoJPHRpdGxlLz4NCgk8ZWZmZWN0aXZlVGltZSB2YWx
1ZT0iMjAxMDEwMjYxMzA5NDUiLz4NCgk8Y29uZmlkZW50aWFsaXR5Q29kZS8+DQoJPGxhbmd1YWdlQ29kZ
SBjb2RlPSJlbi1VUyIvPg0KCTxyZWNvcmRUYXJnZXQ+DQoJCTxwYXRpZW50Um9sZT4NCgkJCTwhLS1UaGU
gSElUU1AvQzMyIHNwZWNpZmljYXRpb24gcmVxdWlyZXMgdGhhdCBjZXJ0YWluIHBhdGllbnQgZWxlbWVud
HMgYmUgcHJlc2VudCwgYnV0IGRvZXMgbm90IHJlcXVpcmUgdGhhdCB0aGV5IGJlIG5vbi1lbXB0eS4gQSB
tZWFuaW5nZnVsIGRvY3VtZW50IHNob3VsZCBjb250YWluIGVpdGhlciBhIGNvbXBsZXRlIHBhdGllbnRSb
2xlL0lkIG9yIGF0IGxlYXN0IGdpdmVuIGFuZCBmYW1pbHkgbmFtZXMuLS0+DQoJCQk8aWQvPg0KCQkJPGF
kZHI+DQpOb25lIG9mIHRoZSBmb2xsb3dpbmcgYWRkcmVzcyBlbGVtZW50cyBhcmUgc3BlY2lmaWNhbGx5I
HJlcXVpcmVkIGJ5IEhJVFNQIEMzMiwgc28gdGhlIGFkZHJlc3MgY291bGQgYmUganVzdCBwbGFpbiB0ZXh
0IGF0IHRoaXMgbG9jYXRpb24/IFVzZSBvZiBvbmUgb3IgbW9yZSBvZiB0aGUgZm9sbG93aW5nIGxhYmVsZ
WQgYWRkcmVzcyBsaW5lcywgd2l0aCBtZWFuaW5nZnVsIGNvbnRlbnQsIGlzIGVuY291cmFnZWQuDQoJCQk
JPHN0cmVldEFkZHJlc3NMaW5lLz4NCgkJCQk8c3RyZWV0QWRkcmVzc0xpbmUvPg0KCQkJCTxjaXR5Lz4NC
gkJCQk8c3RhdGUvPg0KCQkJCTxwb3N0YWxDb2RlLz4NCgkJCQk8Y291bnRyeS8+DQoJCQk8L2FkZHI+DQo
JCQk8dGVsZWNvbS8+DQoJCQk8cGF0aWVudD4NCgkJCQk8bmFtZT4NCgkJCQkJPGdpdmVuLz4NCgkJCQkJP
GdpdmVuLz4NCgkJCQkJPGZhbWlseS8+DQoJCQkJPC9uYW1lPg0KCQkJCTwhLS1ISVRTUC9DODMgcmVxdWl
yZXMgcGF0aWVudCBhZG1pbmlzdHJhdGl2ZSBnZW5kZXIsIGUuZy4gTSwgRiwgSSAoaW5kZXRlcm1pbmF0Z
SktLT4NCgkJCQk8YWRtaW5pc3RyYXRpdmVHZW5kZXJDb2RlIGNvZGU9IkYiIGRpc3BsYXlOYW1lPSJGZW1
hbGUiIGNvZGVTeXN0ZW09IjIuMTYuODQwLjEuMTEzODgzLjUuMSIgY29kZVN5c3RlbU5hbWU9IkhMNyBBZ
G1pbmlzdHJhdGl2ZUdlbmRlciIvPg0KCQkJCTwhLS1ISVRTUC8zMiB3aXRoIElIRSBlbmhhbmNlbWVudHM
gcmVxdWlyZXMgdGhhdCB0aGUgcGF0aWVudCBiaXJ0aFRpbWUgZWxlbWVudCBiZSBwcmVjaXNlIGF0IGxlY
XN0IHRvIHllYXIgYW5kIHN0YXRlcyB0aGF0IGl0IHNob3VsZCBiZSBwcmVjaXNlIGF0IGxlYXN0IHRvIGR
heS4gLS0+DQoJCQkJPGJpcnRoVGltZSB2YWx1ZT0iMTk4NDA3MDQiLz4NCgkJCTwvcGF0aWVudD4NCgkJP
C9wYXRpZW50Um9sZT4NCgk8L3JlY29yZFRhcmdldD4NCgk8YXV0aG9yPg0KCQk8IS0tSElUU1AvMzIgd2l
0aCBJSEUgZW5oYW5jZW1lbnRzIHJlcXVpcmVzIHRoYXQgdGhlIGF1dGhvciB0aW1lIGVsZW1lbnQgYmUgc
```

```
HJlY2lzZSBhdCBsZWFzdCB0byBkYXkgYW5kIHN0YXRlcyB0aGF0IGl0IHNob3VsZCBiZSBhIHRpbWVzdGF
tcCB3aXRoIHByZWNpc2lvbiBhdCBsZWFzdCB0byBzZWNvbmQuIC0tPg0KCQk8dGltZSB2YWx1ZT0iMjAxM
DEwMjYxNDU3MzAiLz4NCgkJPGFzc2lnbmVkQXV0aG9yPg0KCQkJPGlkLz4NCgkJCTxhZGRyLz4NCgkJCTx
0ZWxlY29tLz4NCgkJCTxhc3NpZ25lZFBlcnNvbj4NCgkJCQk8bmFtZS8+DQoJCQk8L2Fzc2lnbmVkUGVyc
29uPg0KCQkJPHJlcHJlc2VudGVkT3JnYW5pemF0aW9uPg0KCQkJCTxuYW1lPk5JU1QgSGVhbHRoY2FyZSB
UZXN0aW5nIExhYm9yYXRvcnk8L25hbWU+DQoJCQkJPHRlbGVjb20vPg0KCQkJCTxhZGRyLz4NCgkJCTwvc
mVwcmVzZW50ZWRPcmdhbml6YXRpb24+DQoJCTwvYXNzaWduZWRBdXRob3I+DQoJPC9hdXRob3I+DQoJPGN
1c3RvZGlhbj4NCgkJPGFzc2lnbmVkQ3VzdG9kaWFuPg0KCQkJPHJlcHJlc2VudGVkQ3VzdG9kaWFuT3JnY
W5pemF0aW9uPg0KCQkJCTxpZC8+DQoJCQkJPG5hbWUvPg0KCQkJCTx0ZWxlY29tLz4NCgkJCQk8YWRkci8
+DQoJCQk8L3JlcHJlc2VudGVkQ3VzdG9kaWFuT3JnYW5pemF0aW9uPg0KCQk8L2Fzc2lnbmVkQ3VzdG9ka
WFuPg0KCTwvY3VzdG9kaWFuPg0KCTxjb21wb25lbnQ+DQoJCTxub25YTUxCb2R5Pg0KCQkJPHRleHQvPg0
KCQk8L25vblhNTEJvZHk+DQoJPC9jb21wb25lbnQ+DQo8L0NsaW5pY2FsRG9jdW1lbnQ+DQo=</orac:Do
cument>

        <!--Optional:-->

        <orac:DocumentId>2.16.3.5.6^Test01_CCD390</orac:DocumentId>

     </orac:PersistCDADocumentRequest>

  </soap:Body>

</soap:Envelope>
```

where the user provided underneath the SOAP Header should have already been created under WebLogic default Security Realm myrealm.

Sample for successful CDA Persistence --> persistDocument Web Service SOAP response

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:orac="http://oracle.apps.ctb.cdapersistence.types">

    <soap:Body>

      <orac: PersistCDADocumentResponse >

         <orac:status>SUCCESS</orac:status>

      </orac: PersistCDADocumentResponse >

   </soap:Body>

</soap:Envelope>
```

Sample CDA Persistence à importDocument Web Service SOAP request

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:orac="http://oracle.apps.ctb.cdapersistence.types">

   <soap:Header>

      <wsse:Security soap:mustUnderstand="true"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-sec
ext-1.0.xsd"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-util
ity-1.0.xsd">

         <wsse:UsernameToken wsu:Id="UsernameToken-10">

            <wsse:Username>IHE_XDS_USER</wsse:Username>

            <wsse:Password
```

```
Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profi
le-1.0#PasswordText"><USER_PASSWORD></wsse:Password>

         </wsse:UsernameToken>

      </wsse:Security>

   </soap:Header>

   <soap:Body>

      <orac:ImportCDADocumentRequest>

        <orac:DocumentUniqueId>1.1.1.1.1.1.53351113</orac:DocumentUniqueId>

      </orac:ImportCDADocumentRequest>

   </soap:Body>

</soap:Envelope>
```

where the user provided under the SOAP Header should have already been created under WebLogic default Security Realm **myrealm**.

Sample for successful CDA Persistence --> importDocument Web Service SOAP response

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:orac="http://oracle.apps.ctb.cdapersistence.types">

    <soap:Body>

      <orac: ImportCDADocumentResponse >

         <orac:status>SUCCESS</orac:status>

      </orac: ImportCDADocumentResponse >

   </soap:Body>

</soap:Envelope>
```