

Oracle® Healthcare Master Person Index

Relationship Management User's Guide

Release 4.0

E71323-02

December 2016

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

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

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

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

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

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

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

Preface	vii
Audience	vii
Documentation Accessibility	vii
Related Documents	vii
Finding Information and Patches on My Oracle Support	viii
Finding Oracle Documentation	x
Conventions	x
1 Developing the Relationship Management	
1.1 Understanding Relationship Model	1-1
1.1.1 Generic Naming Standard	1-2
1.1.2 Direction of Relationship	1-2
1.1.3 Multiplicity of Relationship	1-2
1.1.4 Validity of Relationship	1-3
1.2 Understanding Relationship Rules	1-3
1.3 Understanding Relationship Policies	1-3
1.4 Understand Relationship Management Process	1-3
1.5 Understanding Relationship Management Deployment Components	1-6
1.5.1 Relationship Management Service	1-6
1.5.2 Relationship Management Data Manager	1-6
1.5.3 Relationship Management MPI Agent	1-6
1.5.4 Oracle Healthcare Master Person Index Application	1-7
1.5.5 Deployment Strategy	1-7
1.5.5.1 Standalone Deployment Strategy	1-7
1.5.5.2 Integration Deployment Strategy	1-7
1.5.5.3 MPI Agent Deployment Strategy	1-8
2 Creating the Relationship Management Project	
2.1 Creating the Relationship Management Project	2-1
2.2 Configuring the Relationship Management Applications	2-2
2.2.1 Configuring the Relationship Management Service Application	2-3
2.2.2 Configuring the Relationship Management Data Manager Application	2-3
2.2.3 Configuring the Relationship Management MPI Agent	2-4
2.3 Building the Relationship Management Applications	2-4
2.4 Creating the Relationship Management Databases	2-5

2.4.1	Creating User and Assigning Privileges	2-5
2.4.2	Creating Database Tables	2-5
2.5	Deploying JAX-RS Library	2-6
2.6	Deploying the Relationship Management Service Application.....	2-6
2.7	Deploying the Relationship Management Data Manager Application	2-6
2.8	Accessing the Relationship Management Data Manager Application	2-7
2.9	Accessing the Relationship Management REST APIs	2-7
2.9.1	Metadata Resources.....	2-8
2.9.2	Management Resources	2-8
2.9.3	Entity Resources.....	2-8
2.9.4	Relationship Resources	2-9
2.9.5	Stats Resources	2-9

3 Integrating with Master Person Index Applications

3.1	Integrating with Master Index Data Manager	3-1
3.2	Integrating with Relationship Management Master Person Index Agent.....	3-2
3.3	Integrating with Master Index Data Manager	3-2

4 Relationship Management Databases

4.1	Database Requirements.....	4-1
4.1.1	Database Platform Requirements.....	4-1
4.1.2	Operating System Requirements.....	4-1
4.1.3	Hardware Requirements	4-1
4.1.4	Oracle Database	4-1
4.2	Setting Up the Database Tables	4-2
4.3	Defining the JDBC Connection Pool	4-2
4.3.1	Creating the JDBC Connection Pool and Resource	4-2
4.4	Dropping the Database Tables.....	4-4
4.4.1	Deleting the Database Tables.....	4-4
4.4.2	Dropping the Text-Indexes.....	4-4
4.5	Database Structure	4-4
4.5.1	Understanding Database Tables Details	4-5
4.5.1.1	RM_ENTITY_TYPE	4-5
4.5.1.2	RM_RELATIONSHIP_TYPE	4-6
4.5.1.3	RM_ATTRIBUTE_TYPE	4-6
4.5.1.4	RM_ENTITY	4-7
4.5.1.5	RM_ENTITY_ATTRIBUTE	4-7
4.5.1.6	RM_RELATIONSHIP	4-7
4.5.1.7	RM_RELATIONSHIP_ATTRIBUTE	4-8
4.5.1.8	RM_DOMAIN.....	4-8
4.5.1.9	RM_RULESET	4-9
4.5.1.10	RM_TASK	4-9
4.5.1.11	RM_AUDIT_EVENT	4-9
4.5.1.12	RM_AUDIT_SOURCE	4-10
4.5.1.13	RM_AUDIT_OBJECT.....	4-10
4.5.1.14	RM_RELATIONSHIP_MERGE	4-11
4.5.1.15	RM_CATEGORY	4-11

5 Relationship Discovery and Relationship Rules

5.1	Relationship Discovery	5-1
5.1.1	Integrating with Relationship Rules	5-2
5.2	Relationship Rules	5-3
5.2.1	Rules Syntax	5-3
5.2.2	RuleSet.....	5-4
5.2.3	Condition Rules	5-5
5.2.3.1	Condition Operators	5-5
5.2.4	Relationship Management Task Interface.....	5-9
5.2.5	Registering Relationship Rulesets	5-10
5.2.5.1	Defining and Creating Rules and Ruleset.....	5-10
5.2.5.2	Designing and Implementing the Relationship Management Task Interface in Groovy Script 5-11	
5.2.5.3	Registering Ruleset.....	5-12
5.2.5.4	Registering Relationship Management Task.....	5-12

6 Relationship Update and Event Policies

6.1	Relationship Update Policy	6-1
6.1.1	Relationship Update Policy Processing Dataflow	6-2
6.1.1.1	Default Relationship Update Policy	6-2
6.1.1.2	Customizing the Relationship Update Policy	6-3
6.1.2	Relationship Update Policy Plugin Interface.....	6-3
6.2	Relationship Event Policies	6-4
6.2.1	Default MRG Event Policy	6-4
6.2.2	Default MRG Use Cases.....	6-5
6.2.2.1	MRG Use Case A	6-5
6.2.2.2	MRG Use Case B.....	6-6
6.2.2.3	MRG Use Case C.....	6-7
6.2.2.4	MRG Use Case D	6-8
6.2.3	Default UNMRG Event Policy	6-8
6.2.4	Default Add Event Policy	6-9
6.3	Understanding the Relationship Management MPI Agent.....	6-9
6.3.1	Event Policy Process Dataflow	6-10
6.3.2	Creating the Relationship Management MPI Agent	6-11
6.3.2.1	Creating and Building the Relationship Management MPI Agent.....	6-11
6.3.2.2	Configuring the Relationship Management MPI Agent.....	6-12
6.3.2.3	Deploying the Relationship Management MPI Agent.....	6-13
6.4	Customizing the Relationship Event Policy.....	6-14
6.5	Relationship Event Policy Plugin Interface.....	6-15

7 Auditing Relationship Management

7.1	Audit Event Data Model.....	7-1
7.2	Audit Event Format	7-2
7.2.1	RM_AUDIT_EVENT Table	7-2
7.2.2	RM_AUDIT_SOURCE Table.....	7-2
7.2.3	RM_AUDIT_OBJECT Table	7-3

7.3	Setting Up the Audit Database Tables	7-4
7.4	Enabling or Disabling Auditing.....	7-4
7.5	Viewing the Audit Log.....	7-5
7.6	Archiving the Audit Log.....	7-5
7.7	Dropping the Audit Database Tables	7-6

8 Relationship Management Security

8.1	Security Groups and Roles	8-1
8.1.1	Creating the Relationship Management Security Groups and Roles.....	8-2
8.2	Masking-off the Sensitive Information	8-3
8.3	Enabling SSL	8-4

9 Performance Tuning

9.1	Oracle Database Tuning.....	9-1
9.1.1	Managing the Relationship Management Database Tablespaces.....	9-1
9.1.2	Creating a BIGFILE Tablespace	9-1
9.1.3	Storing the DATAFILE on Different Storage Location	9-1
9.1.4	Increasing the Default Initial Tablespace Size	9-2
9.1.5	Managing the Relationship Management Database Indexes	9-2
9.1.6	Managing the Relationship Management Database Partition	9-2
9.2	Oracle TopLink JPA Tuning.....	9-3
9.3	Oracle WebLogic Server Tuning.....	9-3

Preface

This document provides information on the Oracle Healthcare Master Person Index (OHMPI) relationship management services that support the capability of establishing and managing healthcare master data relationships, organizing healthcare master data entities into groups and hierarchies, and creating multiple views of healthcare master data entities.

Audience

This document is intended for users who maintain a healthcare entity index and relationship among those entities.

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 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.

Related Documents

For more information and instructions for implementing and using a master person index application, see the following documents in the Oracle Healthcare Master Person Index documentation set:

- *Oracle Healthcare Master Person Index Analyzing and Cleansing Data User's Guide*
- *Oracle Healthcare Master Person Index Australia Patient Solution User's Guide*
- *Oracle Healthcare Master Person Index Command Line Reports and Database Management User's Guide*
- *Oracle Healthcare Master Person Index Configuration Guide*
- *Oracle Healthcare Master Person Index Configuration Reference*
- *Oracle Healthcare Master Person Index Data Manager User's Guide*
- *Oracle Healthcare Master Person Index Installation Guide*
- *Oracle Healthcare Master Person Index Loading the Initial Data Set User's Guide*

- *Oracle Healthcare Master Person Index Match Engine Reference*
- *Oracle Healthcare Master Person Index Message Processing Reference*
- *Oracle Healthcare Master Person Index Provider Index User's Guide*
- *Oracle Healthcare Master Person Index Real-time Loader User's Guide*
- *Oracle Healthcare Master Person Index Relationship Management Data Manager User's Guide*
- *Oracle Healthcare Master Person Index Relationship Management REST APIs Reference Guide*
- *Oracle Healthcare Master Person Index Relationship Management User's Guide*
- *Oracle Healthcare Master Person Index Release Notes*
- *Oracle Healthcare Master Person Index Security Guide*
- *Oracle Healthcare Master Person Index Standardization Engine Reference*
- *Oracle Healthcare Master Person Index Third Party Licenses and Notices*
- *Oracle Healthcare Master Person Index United Kingdom Patient Solution User's Guide*
- *Oracle Healthcare Master Person Index United States Patient Solution User's Guide*
- *Oracle Healthcare Master Person Index User's Guide*
- *Oracle Healthcare Master Person Index Working With HPD Profile Application User's Guide*
- *Oracle Healthcare Master Person Index Working With IHE Profiles User's Guide*

Note: These documents are designed to be used together when implementing a master index application.

Finding Information and Patches on My Oracle Support

Your source for the latest information about Oracle Healthcare Master Person Index is Oracle Support's self-service Web site My Oracle Support (formerly MetaLink).

Before you install and use Oracle Healthcare Master Person Index, always visit the My Oracle Support Web site for the latest information, including alerts, White Papers, installation verification (smoke) tests, bulletins, and patches.

Creating a My Oracle Support Account

You must register at My Oracle Support to obtain a user name and password account before you can enter the Web site.

To register for My Oracle Support:

1. Open a Web browser to <https://support.oracle.com>.
2. Click the **Register here** link to create a My Oracle Support account. The registration page opens.
3. Follow the instructions on the registration page.

Signing In to My Oracle Support

To sign in to My Oracle Support:

1. Open a Web browser to <https://support.oracle.com>.

2. Click **Sign In**.
3. Enter your user name and password.
4. Click **Go** to open the My Oracle Support home page.

Finding Information on My Oracle Support

There are many ways to find information on My Oracle Support.

Searching by Article ID

The fastest way to search for information, including alerts, White Papers, installation verification (smoke) tests, and bulletins is by the article ID number, if you know it.

To search by article ID:

1. Sign in to My Oracle Support at <https://support.oracle.com>.
2. Locate the Search box in the upper right corner of the My Oracle Support page.
3. Click the sources icon to the left of the search box, and then select **Article ID** from the list.
4. Enter the article ID number in the text box.
5. Click the magnifying glass icon to the right of the search box (or press the Enter key) to execute your search.

The Knowledge page displays the results of your search. If the article is found, click the link to view the abstract, text, attachments, and related products.

Searching by Product and Topic

You can use the following My Oracle Support tools to browse and search the knowledge base:

- **Product Focus** — On the Knowledge page under Select Product, type part of the product name and the system immediately filters the product list by the letters you have typed. (You do not need to type "Oracle.") Select the product you want from the filtered list and then use other search or browse tools to find the information you need.
- **Advanced Search** — You can specify one or more search criteria, such as source, exact phrase, and related product, to find information. This option is available from the **Advanced** link on almost all pages.

Finding Patches on My Oracle Support

Be sure to check My Oracle Support for the latest patches, if any, for your product. You can search for patches by patch ID or number, or by product or family.

To locate and download a patch:

1. Sign in to My Oracle Support at <https://support.oracle.com>.
2. Click the **Patches & Updates** tab. The Patches & Updates page opens and displays the Patch Search region. You have the following options:
 - In the **Patch ID or Number** field, enter the number of the patch you want. (This number is the same as the primary bug number fixed by the patch.) This option is useful if you already know the patch number.
 - To find a patch by product name, release, and platform, click the **Product or Family** link to enter one or more search criteria.

3. Click **Search** to execute your query. The Patch Search Results page opens.
4. Click the patch ID number. The system displays details about the patch. In addition, you can view the Read Me file before downloading the patch.
5. Click **Download**. Follow the instructions on the screen to download, save, and install the patch files.

Finding Oracle Documentation

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

Finding Oracle Health Sciences Documentation

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

<http://www.oracle.com/technetwork/documentation/hsgbu-154445.html>

Note: Always check the Oracle Health Sciences Documentation page to ensure you have the latest updates to the documentation.

Finding Other Oracle Documentation

To get user documentation for other Oracle products:

1. Go to the following Web page:

<http://www.oracle.com/technology/documentation/index.html>

Alternatively, you can go to <http://www.oracle.com>, point to the Support tab, and then click **Documentation**.

2. Scroll to the product you need and click the link.
3. Click the link for the documentation you need.

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.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Developing the Relationship Management

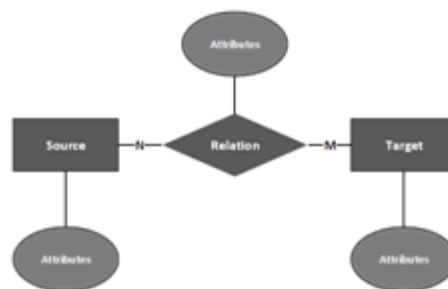
Oracle Healthcare Master Person Index (OHMPI) is used for identity resolution and creates single views of various healthcare domains, such as Patients, Individual Providers, Investigators, Payers, Organizational Providers, and so on. Within the same domain and between these domains data is highly connected and related and the volume of relationship association can be high. The association views are complicated based on many different types of relationship association.

OHMPI Relationship Management (RM) is created for managing the complex relationships as addressed above between providers and patients, among providers, and among patients based on dynamic property changes in today's healthcare setting. It also allows you to apply relationship inferring rules for advanced relationship computation and discovery. OHMPI RM is built upon metadata-driven and open architecture for offering flexible relationship data modeling and customization. OHMPI RM is based on standard open architecture for providing pluggable and extensible capability.

1.1 Understanding Relationship Model

OHMPI relationship data model is based on the standard entity-relationship (ER) data model which uses the mathematical relation construct to express relationships between entities. The OHMPI RM relationship is an explicit linkage between two master entities within a single master domain or across master domains if master entities relates. In other word, the relationships that the OHMPI RM manages are based on a single trusted view of data. The master entities are such as Patients, Individual Providers, Organizational Providers, Payers, Investigators, and so on. The OHMPI RM relationship is represented as a tuple of source, target, and relation in [Figure 1-1](#).

Figure 1-1 Relationship Data Model



The OHMPI RM is typed model. The relationship type is a type of association that can be present between two master entity types or with a single master entity type. The master entity is also known as MPI domain. It is the template for relationship which defines built-in system default relationship attributes and customer-defined relationship attributes. In process perspective there is no difference between system default relationship attributes and customer-defined attributes. The relationship types are such as Patient-of, PCP-of, Admitting-Physician-of, Emergency-Contact and more. The relationship types are divided into different categories, such as Provider and Patient relationship category, Family relationship category, and so on.

By definition, relationship implies relationship instance of a specific relationship type.

1.1.1 Generic Naming Standard

Any names for relationship type, entity type, attribute, ruleset, and task:

- Non-case sensitive
- Start with alphabets
- Mix with any letters, digits, and underscore
- Avoid one-character names
- Any names of relationship types can be mixed with hyphen

Any resources naming and query parameters of the RM REST APIs:

- Non-case sensitive

Any Ids:

- Mix with any letters and digits
- Non-case sensitive

1.1.2 Direction of Relationship

The direction of a relationship can be either bidirectional or unidirectional. A bidirectional relationship has both an owning side and an inverse side. A unidirectional relationship has only an owning side. The owning side of a relationship determines how the RM runtime makes creations to the relationship or query relationships in the database. In a unidirectional relationship, only owning side can create or navigate relationships. In other words, the navigation or creation can be done only from the source side. In a bidirectional relationship, both sides of source and target can create or navigate relationships.

1.1.3 Multiplicity of Relationship

The following are the types of multiplicities:

- **One-to-one:** One source entity instance is related to one single instance of another target entity.
- **One-to-many:** One source entity instance can be related to multiple instances of the other target entities.
- **Many-to-one:** Multiple instances of a source entity can be related to one single instance of the other target entities.
- **Many-to-many:** Multiple source entity instances can be related to multiple instances of the other targets.

1.1.4 Validity of Relationship

The validity of relationship is the state of being a valid relationship. The validity can either be:

- **Resolved:** If the relationship is valid
- OR
- **Potential:** If the validity of the relationship is not decided yet

The potential relationship is created when the relationship management performs probabilistic match on the domain MPI database for the inbound messages and could not find any assumed match. The domain MPI application responses back with a list of potential duplicates. The human review process or intelligent business workflow is required to determine any match or no match from the potential duplicates. Without the determination of match or no match, the relationship management creates the relationship with the Potential state using the potential duplicate. This relationship is named as potential relationship. The potential relationship is not actual resolved relationship. The determination of resolving the potential state is made through the post process such as the human review process or intelligent business workflow. The Resolved relationship is the actual relationship. There shall not be any ambiguity of potential entities.

For more information, see [Chapter 4, "Relationship Management Databases"](#).

1.2 Understanding Relationship Rules

The relationship rule is used for defining the business logic and process for discovering relationships. It is definitive, pluggable, and standard-based.

For more information, see [Chapter 5, "Relationship Discovery and Relationship Rules"](#).

1.3 Understanding Relationship Policies

The relationship policy is used for defining the business logic and process for processing relationships. There are two types of relationship policies: relationship update policy and relationship event policy. The relationship update policy defines the business logic and process on how to update the existing relationships. The relationship event policy defines the business logic and process on how to react with the events from the external applications. Both are definitive, pluggable, and standard-based.

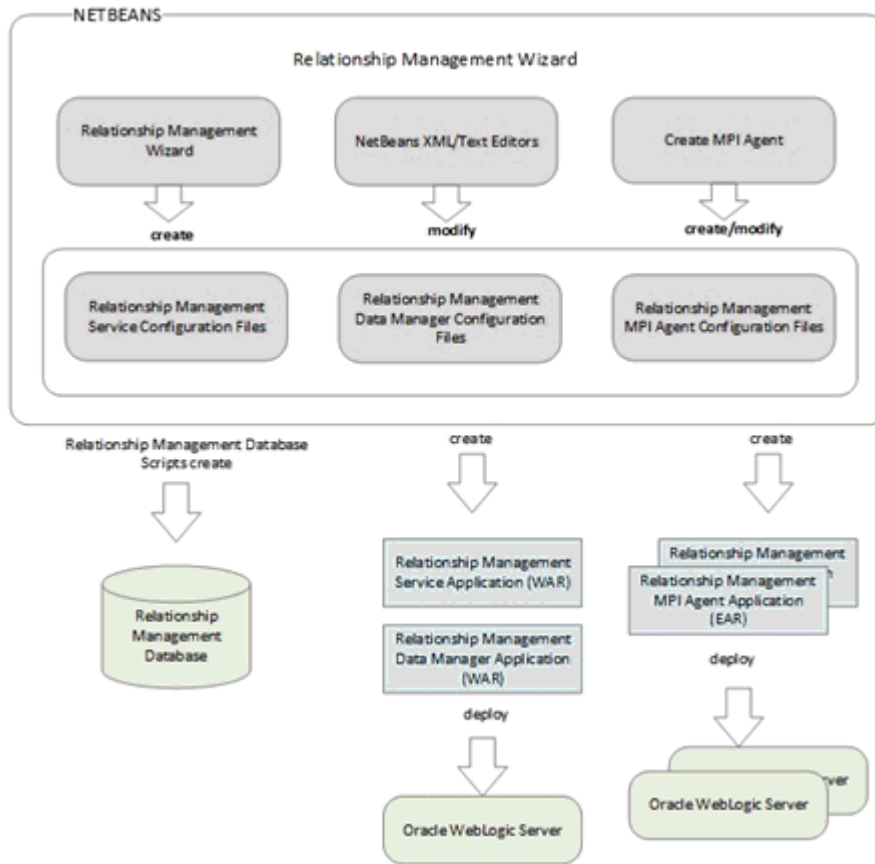
For more information, see [Chapter 6, "Relationship Update and Event Policies"](#).

1.4 Understand Relationship Management Process

This section addresses the relationship management process and architecture. The relationship management application contains the design time and run time modules. The relationship management design-time is a NetBeans IDE plug-in. To create a relationship management project, the relationship management wizard takes your input, such as application server, the application name, the application location, date format, maximum file size, maximum rows fetched, and so on.

[Figure 1–2](#) illustrates the architecture diagram of the relationship management design-time.

Figure 1–2 Relationship Management Design Time



The relationship management design time NetBeans wizard is used to:

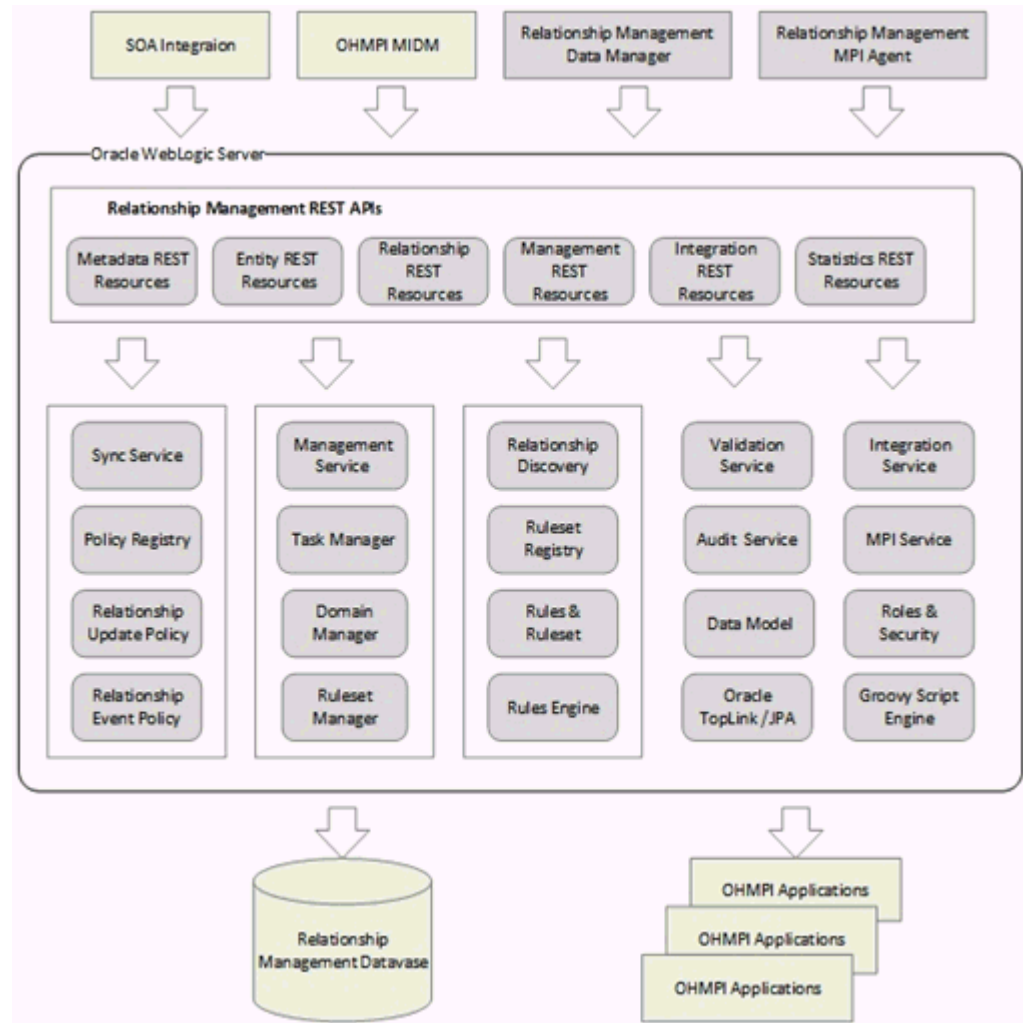
- Create the relationship management project
- Generate the relationship management service application
- Configure the relationship management service application
- Build the relationship management service application
- Generate the relationship management database scripts
- Generate the relationship management data manager application
- Configure the relationship management data manager application
- Build the relationship management data manager application
- Generate the relationship management MPI agent application
- Configure the relationship management MPI agent application
- Build the relationship management MPI agent application

Each OHMPI application requires its own dedicated relationship management MPI agent application. The wizard will generate the different MPI agent application for different MPI domain. Once you build all the relationship management applications, you deploy the relationship management service application and the relationship management data manager applications in the same Oracle WebLogic environment or the distributed environment. And deploy the relationship management MPI agent

application to the Oracle WebLogic environment where the OHMPI application runs with the outbound event notification properly configured.

Figure 1-3 illustrates the architecture diagram of the relationship management runtime.

Figure 1-3 Relationship Management Runtime



The relationship management runtime provides the core services of the relationship managements. The services are exposed as a set of REST APIs. The metadata resources REST APIs provide the capability of managing the relationship types and entity types. The service resources REST APIs provide the capability of managing the instances of relationships and entities. The integration resources REST APIs are used for integrating with the OHMPI application underlying. The management resources REST APIs provides the capability of managing the MPI domains, rulesets, and tasks. The statistics resources REST APIs provide the capability of retrieving the statistical information about the relationship management applications.

For more information on the relationship management REST APIs, see *Oracle Healthcare Master Person Index Relationship Management REST APIs Reference Guide*.

The relationship management MPI agent application is used to integrate with the OHMPI applications asynchronously to react with various MPI outbound events through the standard JMS topic. For more information, see [Section 6.2, "Relationship](#)

[Event Policies](#)".

The relationship management data manager application is the web-based relationship management data stewardship. It allows you to create or update the metadata on the fly. For more information, see *Oracle Healthcare Master Person Index Relationship Management Data Manager User's Guide*.

The OHMPI MIDM application can be integrated with the relationship management data manager directly. With the proper configuration, you can search relationships based on the result of the entity search from the OHMPI MIDM. For more information, see [Chapter 3, "Integrating with Master Person Index Applications"](#).

1.5 Understanding Relationship Management Deployment Components

The relationship management application consists of several different components. Each component can be deployed individually or optionally in a distributed fashion. The relationship management application can run without associating with any MPI domain application in a generic relationship management standalone fashion. The relationship management application also can integrate tightly with the MPI domain applications. The integration is done through the comprehensive relationship management REST APIs and the relationship management API agent. The major deployable components consist of:

- [Relationship Management Service](#)
- [Relationship Management Data Manager](#)
- [Relationship Management MPI Agent](#)
- [Oracle Healthcare Master Person Index Application](#)

1.5.1 Relationship Management Service

The relationship management service provides the core services of all the relationship managements. The services are exposed as the standard REST APIs. The relationship management service is packaged as a war application and deployed to the Oracle WebLogic server.

For more information on the RM REST APIs, see *Oracle Healthcare Master Person Index Relationship Management REST APIs Reference Guide*.

1.5.2 Relationship Management Data Manager

The relationship management data manager is a web-based data management console which provides the customers an ability to manage the relationship types, entity types, domains, rulesets, and tasks. It also allows the customers to search entities and relationships from any supported internet browsers.

For more information, see *Oracle Healthcare Master Person Index Relationship Management Data Manager User's Guide*.

1.5.3 Relationship Management MPI Agent

The relationship management MPI agent is used for integrating with OHMPI applications asynchronously through OHMPI outbound JMS-based notifications. Each OHMPI application is required to deploy one-dedicated RM MPI agent which runs in the same domain where the OHMPI executes.

For more information on integrating with OHMPI outbound events, see [Section 6.2, "Relationship Event Policies"](#).

1.5.4 Oracle Healthcare Master Person Index Application

One instance of the relationship management service is able to interact with multiple OHMPI applications. For more information, see Oracle Healthcare Master Person Index guides.

1.5.5 Deployment Strategy

The following are the deployment strategies:

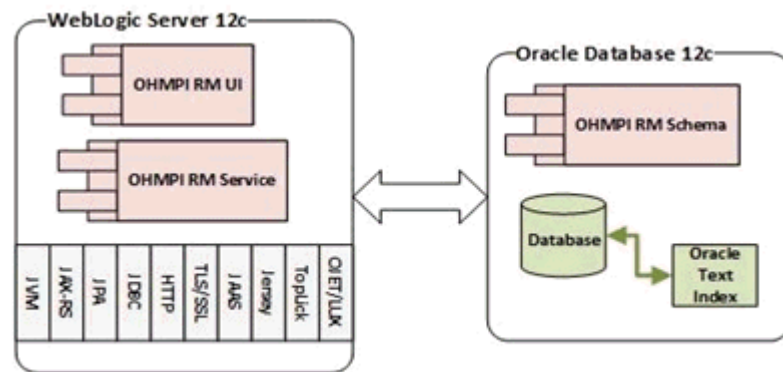
- [Standalone Deployment Strategy](#)
- [Integration Deployment Strategy](#)
- [MPI Agent Deployment Strategy](#)

1.5.5.1 Standalone Deployment Strategy

The standalone deployment strategy is generic deployment manner.

1. Deploy the relationship management database schema to the Oracle Database.
2. Deploy the relationship management service to the Oracle WebLogic server.
3. Deploy the relationship management data manager to the Oracle WebLogic server.

Figure 1–4 Relationship Management Deployment in Standalone Mode

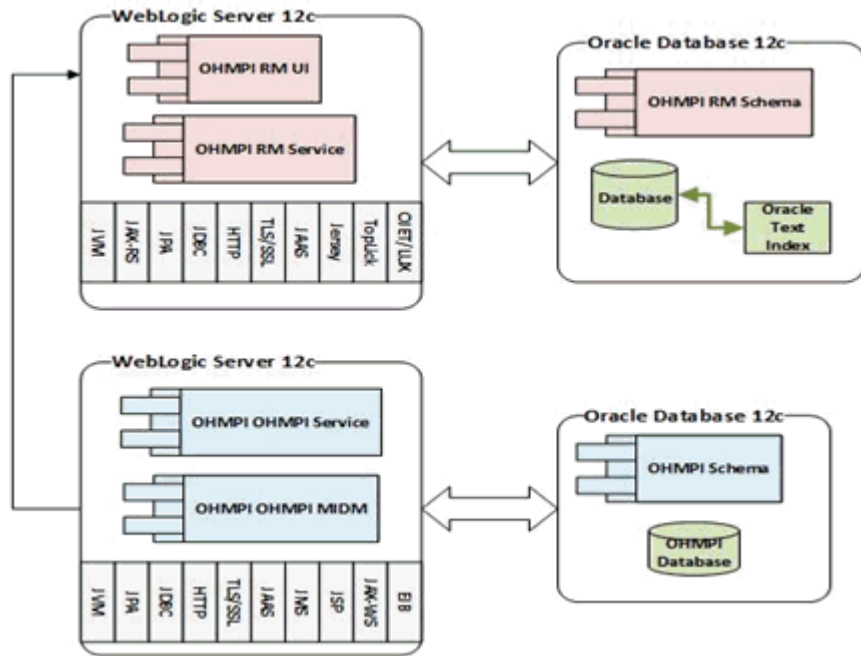


1.5.5.2 Integration Deployment Strategy

The integration deployment strategy is integration deployment manner. The relationship management is integrated with the OHMPI application(s) through MIDM and relationship management integration REST APIs.

1. Deploy the integrated OHMPI MPI application(s).
2. Deploy the relationship management database schema to the Oracle Database.
3. Deploy the relationship management service to the Oracle WebLogic server.
4. Deploy the relationship management data manager to the Oracle WebLogic server.

Figure 1–5 Relationship Management Integration Deployment

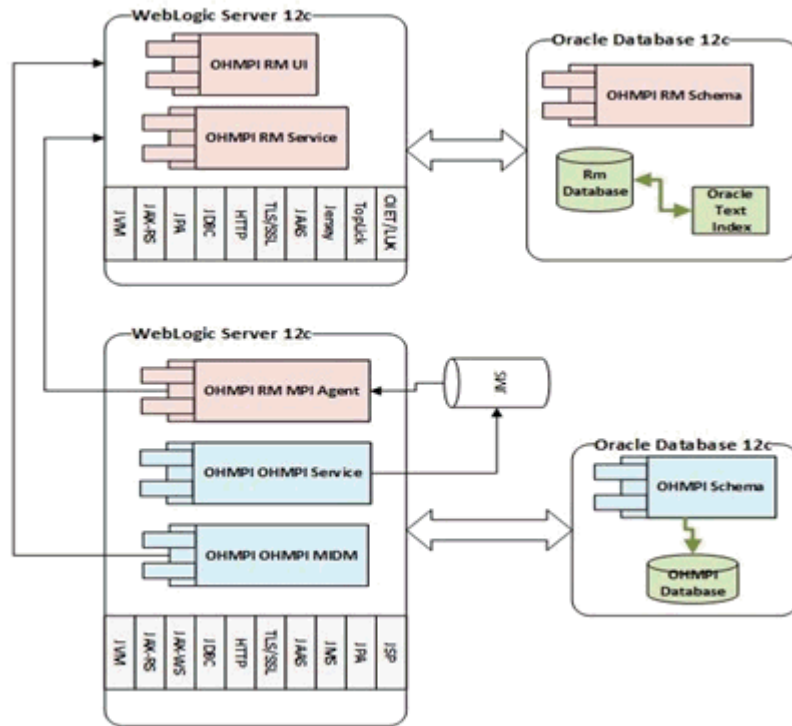


1.5.5.3 MPI Agent Deployment Strategy

The MPI agent deployment strategy is asynchronous integration deployment manner. The relationship management is integrated with the OHMPI application(s) through the MPI agent(s).

1. Deploy the integrated OHMPI MPI application(s).
2. Deploy the relationship management database schema to the Oracle Database.
3. Deploy the relationship management service to the Oracle WebLogic server.
4. Deploy the relationship management data manager to the Oracle WebLogic server.
5. Deploy the relationship management MPI agent to the Oracle WebLogic server.

Figure 1-6 Relationship Management MPI Agent Deployment



Creating the Relationship Management Project

This chapter provides information and procedure for creating, configuring, and deploying a relationship management application project. The relationship management application project contains three applications: the relationship management service application, the relationship management data manager, and the relationship management MPI agent.

Information on implementing the custom plug-ins and configuring the advanced settings is covered in the subsequent chapters.

2.1 Creating the Relationship Management Project

If you plan to implement the relationship management solution for the OHMPI applications, the prerequisite is to create and configure the OHMPI applications. Then, to create the relationship management project, perform the following steps:

1. On the NetBeans toolbar, click **New Project**.

The New Project wizard appears, and opens on the Choose Project window.

2. Under Categories, select **OHMPI**.
3. Under Projects, select **Relationship Management Application** and then click **Next**.

The Name and Location window appears with the default project name and directory paths.

4. Enter the NetBeans project name and the path where you want to store the project files, or accept the defaults.
5. Select the application server you are using from the drop-down list.
 - a. If no application servers appear in the list, click **Manage**.

The Servers window appears.

- b. Click **Add Server**.

The Add Servers Instance window appears.

- c. Select the Oracle WebLogic Server, and click **Next**.
- d. Under Server Location, accept the defaults and click **Finish**.

You are returned to the Servers window with the application server you selected listed under Servers in the left pane.

- e. Click **Close**.

The Name and Location window reappears.

6. Select the **Set as Main Project** check box to set the Relationship Management application as the main project.
7. Click **Next**.
8. On the Define Deployment Environment window, enter the following information in the Service section:
 - **Date and Time Formats:** The date and time formats for the OHMPI Relationship Management. This defines how date and time are stored.
 - **Date Format:** This is the default or application level format used by OHMPI Relationship Management backend for the date data type.
 - **Timestamp Format:** This is the default or application level format used by OHMPI Relationship Management backend for the timestamp data type.
 - **Maximum File Size:** Set the maximum size of object, rules, and groovy script files in terms of KB.
 - **Maximum Rows Fetched:** Set the maximum number of rows that can be fetched from the database. Enter the value in terms of K (1000). For example, if you set the value as 10, a maximum of 10,000 (10K) rows can be fetched.
 - **Audit Enabled:** Select **true** from the drop-down list to enable the relationship management audit.
 - **Text Index:** Select **true** from the drop-down list to enable search by Oracle text index.
9. On the Define Deployment Environment window, enter the following information in the UI section:
 - **Date and Time Formats:** The date and time formats for the OHMPI Relationship Management UI. This defines how date and time are displayed in UI.
 - **Date Format:** This is the default or application level format used by OHMPI Relationship Management UI for the date data type.
 - **Timestamp Format:** This is the default or application level format used by OHMPI Relationship Management UI for the timestamp data type.
10. Click **Finish**.

The wizard automatically creates two projects in the NetBeans Project window. The relationship management project includes the application artifacts and configuration or descriptor files of the relationship management service application, the relationship management data manager application, and the relationship management MPI agent. The relationship management custom task project provides a skeleton of the project which helps you to write the custom groovy task.

2.2 Configuring the Relationship Management Applications

After you create the OHMPI RM project, several nodes that represent the RM configuration and descriptor files are placed in the project for the RM applications. These configuration and descriptor files contain the advanced settings and parameters. If you do not plan to make any customization, skip this section and continue with [Section 2.3, "Building the Relationship Management Applications"](#). Otherwise, make sure that the configuration files are customized correctly for your implementation.

2.2.1 Configuring the Relationship Management Service Application

The configuration and descriptor files of the Relationship Management server application are located in the RM project Configuration\service folder and include the following:

- **ohmpi_rm.properties:** The properties configuration file of the relationship management service application.
- **persistence.xml:** The JPA persistence descriptor file of Oracle TopLink 12c.
- **web.xml:** The web deployment descriptor file of the relationship management service application.
- **weblogic.xml:** The WebLogic-specific deployment descriptor file of the relationship management service application.

To modify the configuration file:

1. Open the file, that you want to modify, in the Configuration\service folder.

You can modify the following files:

- ohmpi_rm.properties
- persistence.xml

To configure the amount of information that gets logged, open the persistence.xml file and change the value in the following parameter:

```
<!-- The log level: OFF/SEVERE/WARNING/INFO/CONFIG/FINE/FINER/FINEST/ALL
-->
<property name="eclipselink.logging.level.sql" value="WARNING" />
```

- web.xml
- weblogic.xml

2. Edit the file, as required.

3. Save the file.

For more information about persistence.xml, see *Oracle TopLink 12c User's Guide*. For more information about web.xml and weblogic.xml, see *Fusion Middleware Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server 12c*.

2.2.2 Configuring the Relationship Management Data Manager Application

The configuration and descriptor files of the Relationship Management data manager are located in the RM project Configuration\ui folder and include the following:

- **uiConfig.js:** The properties file of the relationship management data manager.
- **web.xml:** The web deployment descriptor file of the relationship management data manager.
- **weblogic.xml:** The WebLogic-specific deployment descriptor file of the relationship management data manager.

To modify the configuration file:

1. Open the file, that you want to modify, in the Configuration\ui folder.

You can modify the following files:

- uiConfig.js
- web.xml

- weblogic.xml
2. Edit the file, as required.
 3. Save the file.

For more information about web.xml and weblogic.xml, see *Fusion Middleware Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server 12c*.

Table 2–1 lists the properties in uiConfig.js.

Table 2–1 Properties in uiConfig.js

Property Name	Description
serviceBaseURL	Base URL for the backend REST API Services. The host name must be fully qualified name (localhost will not work).
graphPageMaxNodeCount	Maximum periphery node count per graph page.
entityTypes	These options define the node shape and color in the relationship graph view and entity summary in graph or table view. Also, list of fields that need to be masked in properties panel. For example, <pre>entityTypes: { OrgProvider: {shape: 'plus', color: 'red', summaryFieldCount: 1, sensitive_fields :[]} }</pre>
relationshipTypes	These options define the link color in the relationship graph view. List of fields that must be masked in the properties panel. <pre>relationshipTypes: { 'primary-care-physician-of': {color: '#3385b7', sensitive_fields :[]} }</pre>
dateFormat	For information, see Section 2.1, "Creating the Relationship Management Project" .
timestampFormat	For information, see Section 2.1, "Creating the Relationship Management Project" .
uiDateFormat	For information, see Section 2.1, "Creating the Relationship Management Project" .
uiTimestampFormat	For information, see Section 2.1, "Creating the Relationship Management Project" .

If you plan to integrate the relationship management service with the OHMPI service through the MPI agent, see [Section 3.2, "Integrating with Relationship Management Master Person Index Agent"](#) for creating, configuring, and deploying the MPI agent.

2.2.3 Configuring the Relationship Management MPI Agent

For information, see [Section 6.3.2.2, "Configuring the Relationship Management MPI Agent"](#).

2.3 Building the Relationship Management Applications

1. Right-click on the RM project and select **Build**.

Note: If you have already build the project previously and made the changes to the configuration, right-click on the RM project and select **Clean and Build**. This will make sure that the previous build components are cleaned up.

The following WAR files are created in the dist folder:

- ohmpi-rm-service-1.0.0.war
- ohmpi-rm-ui-1.0.0.war

If you plan to integrate the relationship management service with the OHMPI service through the MPI agent, see [Section 3.2, "Integrating with Relationship Management Master Person Index Agent"](#) for creating, configuring, and deploying the MPI agent. Otherwise, you need only the ohmpi-rm-service-1.0.0.war and ohmpi-rm-ui-1.0.0.war applications.

2.4 Creating the Relationship Management Databases

Oracle recommends that you create a separate dedicated database user for Relationship Management to handle the database tables and other database-related task.

2.4.1 Creating User and Assigning Privileges

1. Create a user.

```
CREATE USER <database_username> IDENTIFIED BY <password>;
```

2. Grant the following privileges to the user:

```
GRANT RESOURCE, CONNECT TO <database_username>;
GRANT CREATE TABLESPACE TO <database_username>;
GRANT DROP TABLESPACE TO <database_username>;
```

If you want to enable text-index search, grant the following additional privileges:

```
GRANT RESOURCE, CONNECT, CTXAPP TO <database_username>;
GRANT EXECUTE ON CTXSYS.CTX_CLS TO <database_username>;
GRANT EXECUTE ON CTXSYS.CTX_DDL TO <database_username>;
GRANT EXECUTE ON CTXSYS.CTX_DOC TO <database_username>;
GRANT EXECUTE ON CTXSYS.CTX_OUTPUT TO <database_username>;
GRANT EXECUTE ON CTXSYS.CTX_QUERY TO <database_username>;
GRANT EXECUTE ON CTXSYS.CTX_REPORT TO <database_username>;
GRANT EXECUTE ON CTXSYS.CTX_THES TO <database_username>;
GRANT EXECUTE ON CTXSYS.CTX_ULEXER TO <database_username>;
```

2.4.2 Creating Database Tables

1. In the NetBeans editor, open the following scripts from the <Project_Name>/Database Script directory:

- create-ohmpi-rm.sql
- create-ohmpi-rm-audit.sql

If you have enabled the text-index search, open the following script:

- create-ohmpi-rm-text-index.sql

2. Copy the entire text from create-ohmpi-rm.sql and paste into the SQL editor.
3. Execute the script against the database.

4. Repeat steps 2 and 3 for *create-ohmpi-rm-audit.sql* and *create-ohmpi-rm-text-index.sql* (if required).

2.5 Deploying JAX-RS Library

1. Start the Oracle WebLogic Server. For instructions, see *Oracle Fusion Middleware Online Documentation 12c Release 1 (12.1.3)*.
2. Log on to the WebLogic Server Administration Console.
3. On the left panel, under Domain Structure, select **Deployments**.
The Summary of Deployments panel appears.
4. On the right side of the panel under Deployments, click **Install**.
5. Select *jax-rs-2.0.war* from the `<weblogic_server_home>\wlserver\common\deployable-libraries` directory and click **Next**.
The Install Application Assistant panel appears.
6. Select the **Install this deployment as Library** option.
7. Click **Next**.
8. Select the server location for deployment and click **Next**.
9. Click **Finish**.

2.6 Deploying the Relationship Management Service Application

1. Start the Oracle WebLogic Server. For instructions, see *Oracle Fusion Middleware Online Documentation 12c Release 1 (12.1.3)*.
2. Log on to the WebLogic Server Administration Console.
3. On the left panel, under Domain Structure, select **Deployments**.
4. On the right side of the Summary of Deployments panel, under Deployments, click **Install**.
5. Browse and locate the folder which contains the WAR file you want to install.
6. Select the **ohmpi-rm-service-1.0.0.war** option and click **Next**.
The Install Application Assistant panel appears.
7. Select the **Install this deployment as an application** option and click **Next**.
8. Select the server location for deployment and click **Next**.
9. Click **Finish**.

2.7 Deploying the Relationship Management Data Manager Application

1. Start the Oracle WebLogic Server. For instructions, see *Oracle Fusion Middleware Online Documentation 12c Release 1 (12.1.3)*.
2. Log on to the WebLogic Server Administration Console.

3. On the left panel, under Domain Structure, select **Deployments**.
4. On the right side of the Summary of Deployments panel, under Deployments, click **Install**.
5. Browse and locate the folder which contains the WAR file you want to install.
6. Select the **ohmpi-rm-ui-1.0.0.war** option and click **Next**.
The Install Application Assistant panel appears.
7. Select the **Install this deployment as an application** option and click **Next**.
8. Select the server location for deployment and click **Next**.
9. Click **Finish**.

2.8 Accessing the Relationship Management Data Manager Application

1. Open an application server browser.
2. Enter the following URL:
`http://<Server>:<Port>/ohmpi-rm-ui/1.0.0/`
3. Enter the user name and password provided by your system administrator.
4. Click **Sign In**.

2.9 Accessing the Relationship Management REST APIs

This section describes the REST API support by Relationship Management. For information on REST APIs, their signatures, and API message body, see *Oracle Healthcare Master Person Index Relationship Management REST APIs Reference Guide*.

OHMPI RM REST API is comprehensive uniform interface for managing generic relationships and entities. The RM REST APIs are used to create, modify, and search resources.

The resources managed by the OHMPI RM application are categorized as follows:

- **Metadata Resources:** Manages entity types and relationship types
- **Management Resources:** Manages domains, tasks and rulesets
- **Entity Resources:** Manages entities
- **Relationship Resources:** Manages relationships
- **MPI Integration Resources:** Manages MPI integration between RM and MPIs
- **Stats Resources:** Views count of entity types, relationship types, entities, relationships, domains, tasks, and rulesets

All the REST APIs have security enabled using HTTP Basic Authentication. For information on security details, see [Chapter 8, "Relationship Management Security"](#).

For each REST API to complete successfully, `username:password` must be passed in the request header in base64 encoded format. For example, *Authorization: Basic d2VibG9naWM6d2VsY29tZTE=*. Where, *d2VibG9naWM6d2VsY29tZTE=* is the base64 encoded format for `weblogic:welcome1`.

The response type of each REST API is *application/json*.

For POST and PUT APIs, you must set the header Content-Type to *application/json*.

2.9.1 Metadata Resources

Metadata Resources APIs are used to:

- Create an entity type
- Update, activate, or deactivate an entity type
- Search for a specific entity type
- Search for all entity types in the system
- Create a relationship type
- Update, activate, or deactivate a relationship type
- Search for a specific relationship type
- Search for all relationship types in the system

2.9.2 Management Resources

Management Resources APIs are used to:

- Create a domain
- Upload the object.xml file for the domain
- Update, activate, or deactivate a domain
- Search for a specific domain
- Search for all domains in the system
- Create a ruleset
- Upload an xml file for the ruleset defining the ruleset
- Update, activate, or deactivate a ruleset
- Search for a specific ruleset
- Search for all rulesets in the system
- Create a task
- Upload a groovy script file for the task
- Update, activate, or deactivate a task
- Search for a specific task
- Search for all tasks in the system

2.9.3 Entity Resources

Entity Resources APIs are used to:

- Create an entity
- Update, activate, or deactivate an entity
- Search for a specific entity using its entity ID
- Search for all the entities in the system based on the entity type and entity attributes

- Search for all the entities in the system based on date attribute range and other entity attributes

2.9.4 Relationship Resources

Relationship Resources APIs are used to:

- Create a relationship using source and target entities (without rules)
- Create a relationship using single entity (using preconfigured rules)
- Create a relationship using source and target entities (using preconfigured rules)
- Update, activate, or deactivate a relationship
- Search for a specific relationship using its relationship ID
- Search for all the relationships in the system based on date attribute range and other relationship attributes
- Search for all the relationships from source and target entities, and other entity attributes
- Search for all the relationships from source or target entity ID
- Search for all relationship from source or target ID, and relationship type
- Synchronize relationships with MPI based on event type

2.9.5 Stats Resources

Stats Resources APIs are used to:

- Get information on active and inactive entity types, relationship types, domains, rulesets, and tasks
- Get information on the:
 - Number of audit events (create, update, activate, and deactivate) in last 30 days
 - Active and inactive entities
 - Active, inactive, potential, and resolved relationships

Integrating with Master Person Index Applications

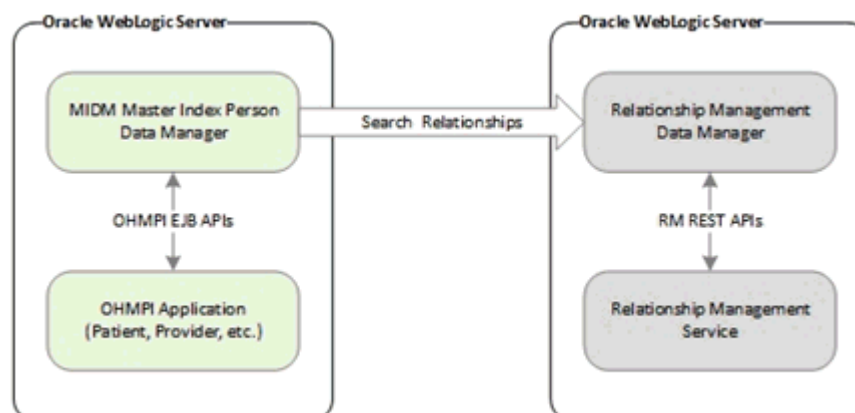
The RM application is designed to provide a flexible and open standard integration framework which allows you to integrate the RM application with OHMPI applications, various upstream or downstream enterprise applications. Through the RM REST APIs or the built-in mechanisms, the RM works with the OHMPI Master Index Data Manager (MIDM), the RM MPI agent, or any SOA applications.

OHMPI provides a flexible framework which allows you to create matching and indexing applications. MIDM allows you to view and maintain the data stored in the master person index database and cross-referenced by a master person index application. For more information, see *Oracle Healthcare Master Person Index User's Guide* and *Oracle Healthcare Master Person Index Data Manager User's Guide*.

3.1 Integrating with Master Index Data Manager

The integration of the RM data manager with the OHMPI data manager provides the integral view to search relationships from the OHMPI data manager directly. You can search domain entities from the OHMPI data manager. Then, from the domain search results view, you can select the domain entity and view relationships of the selected entity. [Figure 3-1](#) illustrates the integration data flow of integrating the MIDM data manager with the RM data manager.

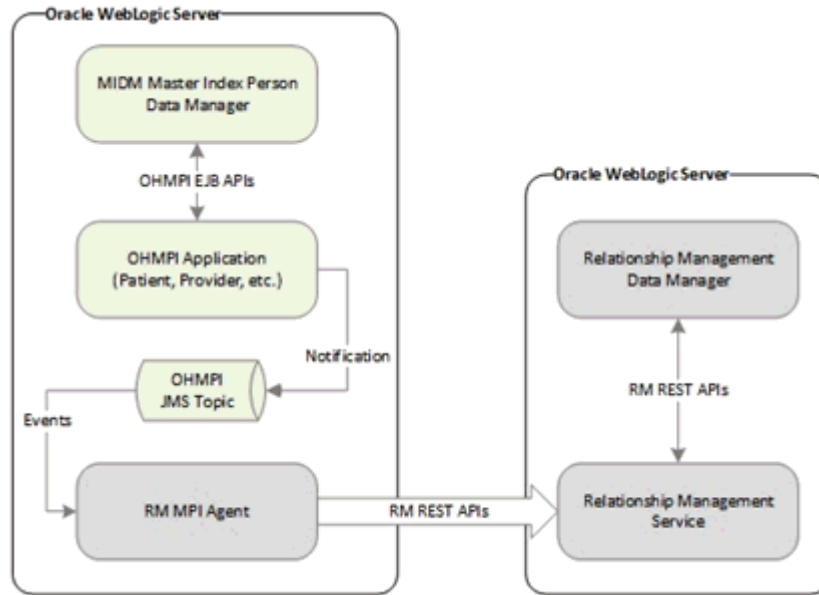
Figure 3-1 MIDM and RM Integration



3.2 Integrating with Relationship Management Master Person Index Agent

OHMPI provides outbound notification through a JMS topic to notify the external systems for the changes or updates on the any domain single best records. It synchronizes data between the OHMPI and the RM database. The RM MPI agent is designed and created for OHMPI outbound notification with the RM service to synchronize the master data. [Figure 3–2](#) illustrates the integration data flow of integrating the OHMPI and the RM through the RM MPI agent.

Figure 3–2 OHMPI and RM MPI Agent Integration



For more information, see [Section 6.2, "Relationship Event Policies"](#).

RM also provides standard-bases REST APIs which are used for programmatically integrating with various enterprise applications to provide a complete solution for various business use cases. For more information, see *Oracle Healthcare Master Person Index Relationship Management REST APIs Reference Guide*.

3.3 Integrating with Master Index Data Manager

You must create, configure, build, and deploy the RM application. For information, see [Chapter 2, "Creating the Relationship Management Project"](#). Also, you must register the entity type matching to the OHMPI domain and build up the relationship data model.

To link MIDM with the RM Data Manager:

1. Create an OHMPI project.
 - For information, see *Oracle Healthcare Master Person Index User's Guide*.
2. In the Projects panel, expand the domain-war/Web Pages/classes node in the project.
3. Double-click **ohmpi-rm.properties**.

The file opens in the NetBeans text editor.

4. Modify the following attributes:

Attribute	Default	Description
rm_base_url	http://localhost:7001/ohmpi-rm-ui/1.0.0/	The base URL of the RM data manager
rm_view_relationship_url	index.html?root=relationshipView&view=graph&query={1}~id~{0}	The relative URL of the RM view relationship page

5. Save the file.

6. Build and deploy the OHMPI application.

7. Assign the user with the proper RM roles.


For information, see [Section 8.1.1, "Creating the Relationship Management Security Groups and Roles"](#).

8. Launch and log on to MIDM.

For information on how to use MIDM, see *Oracle Healthcare Master Person Index Data Manager User's Guide*.

9. Search Record Details.

10. Select the entity from the list of search resulted records.

11. Click  (View Relationship icon).

The View Relationship panel appears in RM.

Relationship Management Databases

This release of OHMPI RM supports Oracle Database 11gR2 and later versions for backend storage of data. For the RM service API and RM UI to work properly, you must set up the database correctly.

4.1 Database Requirements

When configuring the relationship management database, there are several factors to consider, including basic software requirements, operating systems, and disk space. This section provides a summary of requirements for the database. For more detailed information about designing and implementing the database, refer to the appropriate database platform documentation. A database administrator is responsible for configuring database. Database administrator must be familiar with your data processing requirements and OHMPI RM database.

4.1.1 Database Platform Requirements

The OHMPI RM database can be run on Oracle 11gR2 or higher. You must have this software installed before beginning the database installation. Make sure you also install the latest patches for the version you are using.

4.1.2 Operating System Requirements

The database can be installed on any operating system supported by the database platform you are using. For more information, see the documentation that came with your database server.

4.1.3 Hardware Requirements

This section describes the minimum recommended hardware configuration for a database installation. These requirements are based on the minimum requirements recommended by Oracle database for a typical installation. Depending on the size of the database and expected volume, you should increase these recommendations as needed. For more information on the database and for supported operating systems, see the documentation that came with the hardware.

4.1.4 Oracle Database

For a Windows database server, Oracle recommends the following configuration as a minimal installation:

- Windows 2008, 7, Vista SP1+, XP SP2+, 2003 R2 SP2, 2008 R2 on 64 bit

- Pentium 266 or later
- 1 GB RAM (increase this based on the number of users, connections to the database, and volume)
- Virtual memory must be double the amount of RAM
- 3 GB disk space plus an additional 2 KB for each system record to be stored in the database.

Note: This is a conservative estimate per system record, assuming that most records do not contain complete data.

This depends on the Oracle environment you install. Enterprise Edition can take up to 5 GB.

- 256-color video

For a UNIX database server, Oracle recommends the following configuration as a minimal installation:

- 256 MB RAM (increase this based on the number of users and connections to the database)
- 2 GB disk space plus an additional 2 KB for each system record to be stored in the database.

Note: This is a conservative estimate per system record, assuming that most records do not contain complete data.

- Swap space must be a minimum of twice the amount of RAM

Note: Disk space recommendations do not take into account the volume and processing requirements or the number of users. These are minimal requirements to install a generic database. At a minimum, the empty database and the database software requires 2.5 GB of disk space.

4.2 Setting Up the Database Tables

For information, see [Section 2.4, "Creating the Relationship Management Databases"](#).

4.3 Defining the JDBC Connection Pool

OHMPI RM requires one database connection pool. This section provides general instructions for setting up the connection pool. For more information about the procedures, see the online help provided with your server's Administrator Console.

4.3.1 Creating the JDBC Connection Pool and Resource

The JDBC connection pools provide connections to the OHMPI-RM database. A JDBC resource (also known as a data source) gives the RM application the ability to connect to the database.

Before proceeding, make sure you have the relevant information about the OHMPI RM database (such as the database name, URL, database user credentials, and administrator login credentials).

1. Start and stop the Oracle WebLogic Server. For instructions, see *Oracle Fusion Middleware Online Documentation 12c Release 1 (12.1.3)*.
2. Launch the **Oracle WebLogic Server Administration Console**.
3. Log in using the default user name (**weblogic**) and password (**welcome1**).
The Oracle WebLogic Administration Console appears.
4. On the left panel, under Domain Structure, expand **Services**, select **JDBC > Data Sources**.
A Summary of JDBC Data Sources appears in the right panel.
5. To create a new JDBC Data Source, click **New** at the bottom of the right panel.
The Settings for a new JDBC Data Source page appears in the right panel of the page.
6. In the **Name** field, type *RMDataSource*.
7. In the **JNDI Name** field, type *jdbc/RMDataSource*.
8. Click **Save**.
A new page appears in the right panel for setting the Database Type.
9. In the **Database Type** drop-down list, select the appropriate type (for example, **Oracle**).
10. In the **Database Driver** drop-down list, select the appropriate driver. For example, **Oracle's Driver (Thin XA) for Instance Connections; Versions: 9.0.1; 9.2.0; 10, 11**.
11. Click **Next**.
12. Click **Next**.
Connection Properties appears on the Create a New JDBC Data Source panel.
13. In the **Database Name** field, type a name for the database to which you want to connect (for example, **orcl**).
14. In the **Host Name** field, type the name or the IP address of the database server (for example, **localhost**).
15. In the **Port** field, type the port on the database server that is used to connect to the database (for example, **1521**, the default for Oracle).
16. In the **Database User Name** field, type the database account user name you want to use to create database connections (for example, **root**).
17. In the **Password** field, type a password for your database account to use to create database connections.
18. In the **Confirm Password** field, re-type the password to confirm it.
19. Click **Next**.
The Settings for RMDataSource page appear in the right panel.
20. Click the **Connection Pool** tab, click **Test Configuration**, and then click **Next**.
Select Targets appears on the Create a New JDBC Data Source page in the right panel. Here you select one or more targets to deploy the new JDBC data source.

21. In the Servers check box list, select one or more target servers.

Note: If you do not select a target, the data source will be created but not deployed. You will need to deploy the data source at a later time.

22. Click **Finish**.

4.4 Dropping the Database Tables

Scripts are provided to drop the default database tables and indexes created in [Section 4.2, "Setting Up the Database Tables"](#).

4.4.1 Deleting the Database Tables

1. Open SQL editor and connect to the OHMPI RM database using the user you created in [Section 4.2, "Setting Up the Database Tables"](#).
2. Open the NetBeans editor and open the following scripts from the <Project_Name>/Database Script directory:
 - drop-ohmpi-rm-audit.sql
 - drop-ohmpi-rm.sql
3. Copy the entire text from drop-ohmpi-rm-audit.sql and paste into the SQL editor.
4. Execute the script against the database.
5. Repeat steps 3 and 4 for drop-ohmpi-rm.sql.
6. Execute the following command:

```
PURGE RECYCLEBIN;
```

4.4.2 Dropping the Text-Indexes

If you had run the text index-search related scripts in [Section 4.2, "Setting Up the Database Tables"](#), you must drop them when the OHMPI RM tables are dropped as well.

1. Open SQL editor and connect to the OHMPI RM database using the user you created in [Section 4.2, "Setting Up the Database Tables"](#).
2. Open the NetBeans editor and open the drop-ohmpi-rm-text-index.sql script from the <Project_Name>/Database Script directory.
3. Copy the entire text from drop-ohmpi-rm-text-index.sql and paste into the SQL editor.
4. Execute the script against the database.
5. Execute the following command:

```
PURGE RECYCLEBIN;
```

4.5 Database Structure

This section provides information about the OHMPI Relationship Management database, including description of each table. The relationship management database

is the metadata-driven architecture and allowed users to create and modify the relationship metadata on fly. The metadata of the OHMPI relationship management defines the underlying data tables and indexes, the abstract constructs for all the relationship types and relationships. When you create a new relationship type or a new entity type with the built-in default attributes and customer-defined attributes, the OHMPI relationship management does not need to add any additional table in the database.

The relationship management database is composed of the metadata tables, data tables and attributes extension tables. The database includes the tables listed in [Table 4-1](#).

Table 4-1 OHMPI Relationship Management Database Tables

Table Name	Description
RM_ENTITY_TYPE	Stores information about the entity types and the default system attributes of the entity types.
RM_RELATIONSHIP_TYPE	Stores information about the relationship types and the default system attributes of the relationship types.
RM_ATTRIBUTE_TYPE	Stores information about the customer-defined attributes for the entity types or the relationship types.
RM_ENTITY	Stores information about the entity instances and the default system attributes of the entity instances.
RM_ENTITY_ATTRIBUTE	Stores information about the customer-defined attributes of entity instances.
RM_RELATIONSHIP	Stores information about the relationship instances and the default system attributes of the relationship instances.
RM_RELATIONSHIP_ATTRIBUTE	Stores information about the customer-defined attributes of relationship instances.
RM_DOMAIN	Stores information about the OHMPI domains.
RM_RULESET	Stores information about the rulesets for discovering relationships.
RM_TASK	Stores information about the groovy task for the rules, update and event policies.
RM_AUDIT_EVENT	Stores information about the audit events.
RM_AUDIT_SOURCE	Stores information about the sources of the audit events.
RM_AUDIT_OBJECT	Stores information about the objects of the audit events.
RM_RELATIONSHIP_MERGE	Stores information about merging two entities.
RM_CATEGORY	Stores information about the categories of the relationship types. It is not required for this release.

4.5.1 Understanding Database Tables Details

This section describes the detail of each of the OHMPI RM database tables as listed in [Table 4-1](#).

4.5.1.1 RM_ENTITY_TYPE

Table 4-2 Relationship Management Entity Type Database Table

Column Name	Data Type	Description
ENTITY_TYPE_ID	NUMBER(10)	The primary unique key of the entity type object.
NAME	VARCHAR2(128)	The unique name of the entity type.

Table 4–2 (Cont.) Relationship Management Entity Type Database Table

Column Name	Data Type	Description
DESCRIPTION	VARCHAR2(255)	The description of the entity type.
ID	VARCHAR2(20)	The type of the global unique identifier. It can be EUID of the OHMPI, NPI, MRN, and so on. It is determined by the business needs.
STATUS	VARCHAR2(10)	The status of the entity type. It can be Active or Inactive.

4.5.1.2 RM_RELATIONSHIP_TYPE

Table 4–3 Relationship Management Relationship Type Database Table

Column Name	Data Type	Description
RELATIONSHIP_TYPE_ID	NUMBER(10)	The primary unique key of the relationship type object.
NAME	VARCHAR2(128)	The unique name of the relationship type.
CATEGORY_ID	NUMBER(10)	The foreign key of the category that this relationship type belongs.
TASK_ID	NUMBER(10)	The foreign key of the task for the relationship update policy for this relationship type.
DESCRIPTION	VARCHAR2(255)	The description of the relationship type.
SOURCE_NAME	VARCHAR2(128)	The name of the source entity type of the relationship type.
TARGET_NAME	VARCHAR2(128)	The name of the target entity type of the relationship type.
DIRECTION	VARCHAR2(20)	The direction of the relationship type. It can be UNIDIRECTIONAL or BIDIRECTIONAL.
MULTIPLICITY	VARCHAR2(20)	The multiplicity of the relationship type. It can be ONETOONE, ONETOMANY, MANYTOONE, or MANYTOMANY.
CREATED_DATE	DATE	The date when the relationship type is created. The date format is configurable.
CREATED_BY	VARCHAR2(32)	The user name that creates the relationship type. It is signed-in user.
MODIFIED_DATE	DATE	The date when the relationship type is updated. The date format is configurable.
MODIFIED_BY	VARCHAR2(32)	The user name that updates the relationship type. It is signed-in user.
EXPIRATION_DATE	DATE	The expiration date of the relationship type. If the relationship type expires, no relationship instance of the relationship type can be created.
EFFECTIVE_DATE	DATE	The effective date of the relationship type. If the relationship type is not effective, no relationship instance can be created.
STATUS	VARCHAR2(10)	The status of the relationship type. It can be Active or Inactive.

4.5.1.3 RM_ATTRIBUTE_TYPE

Table 4–4 Relationship Management Customer-Defined Attribute Type Database Table

Column Name	Data Type	Description
ATTRIBUTE_TYPE_ID	NUMBER(10)	The primary unique key of the attribute type.
RELATIONSHIP_TYPE_ID	NUMBER(10)	The foreign key of the relationship type for this attribute.
ENTITY_TYPE_ID	NUMBER(10)	The foreign key of the entity type for this attribute.
NAME	VARCHAR2(128)	The unique name of the attribute type.
DATA_TYPE	VARCHAR2(10)	The name of the data type. The supported data types are: String, Char, Float, Double, Date, Boolean, and Timestamp.
DESCRIPTION	VARCHAR2(255)	The description of the attribute type.
SIZE_VALUE	NUMBER(10)	The maximum length of the attribute.
DEFAULT_VALUE	VARCHAR2(128)	The default value of the attribute.
MIN_VALUE	VARCHAR2(128)	The minimum value of the attribute.
MAX_VALUE	VARCHAR2(128)	The maximum value of the attribute.
MANDATORY	CHAR(1)	The mandatory attribute requires value. It can be Y or N.
READ_ONLY	CHAR(1)	Once initialized, it cannot be updated. It can be Y or N.
INDEXED	CHAR(1)	Only indexed attribute is searchable. It can be Y or N.
STATUS	VARCHAR2(10)	The status of the entity type. It can be Active or Inactive.

4.5.1.4 RM_ENTITY

Table 4–5 Relationship Management Entity Database Table

Column Name	Data Type	Description
ENTITY_ID	NUMBER(20)	The primary unique key of the entity record.
ENTITY_TYPE_ID	NUMBER(10)	The foreign key of the entity type for the entity record.
ID	VARCHAR2(20)	The unique global entity identifier. The ID type is defined by the ID of the entity type.
STATUS	VARCHAR2(10)	The status of the entity type. It can be Active or Inactive.
DOCS_INDEX	VARCHAR2(1024)	The text index is used for Oracle text index internally.

4.5.1.5 RM_ENTITY_ATTRIBUTE

Table 4–6 Relationship Management Entity Customer-Defined Attribute Database Table

Column Name	Data Type	Description
ENTITY_ATTRIBUTE_ID	NUMBER(20)	The primary unique key of the entity customer-defined attribute.
ENTITY_ID	NUMBER(20)	The foreign key of the entity associated with the attribute.
NAME	VARCHAR2(128)	The name of the attribute.
VALUE	VARCHAR2(128)	The value of the attribute.

4.5.1.6 RM_RELATIONSHIP

Table 4–7 Relationship Management Relationship Database Table

Column Name	Data Type	Description
RELATIONSHIP_ID	NUMBER(20)	The primary unique key of the relationship record.
RELATIONSHIP_TYPE_ID	NUMBER(10)	The foreign key of the relationship type for the relationship record.
SOURCE_ENTITY_ID	NUMBER(20)	The foreign key of the source entity of the relationship.
TARGET_ENTITY_ID	NUMBER(20)	The foreign key of the target entity of the relationship.
CREATED_DATE	DATE	The date when the relationship is created. The date format is configurable.
CREATED_BY	VARCHAR2(32)	The user name that creates the relationship. It's signed-in user.
MODIFIED_DATE	DATE	The date when the relationship is modified. The date format is configurable.
MODIFIED_BY	VARCHAR2(32)	The user name that modifies the relationship. It's signed-in user.
EXPIRATION_DATE	DATE	The expiration date of the relationship.
EFFECTIVE_DATE	DATE	The effective date of the relationship.
STATUS	VARCHAR2(10)	The status of the entity type. It can be Active, Inactive, or Known. The only Potential relationship is Known.
VALIDITY	VARCHAR2(10)	The validity of the relationship. It can be Potential or Resolved.
DOCS_INDEX	VARCHAR2(1024)	The text index is used for Oracle text index internally.

4.5.1.7 RM_RELATIONSHIP_ATTRIBUTE

Table 4–8 Relationship Management Relationship Customer-Defined Attribute Database Table

Column Name	Data Type	Description
RELATIONSHIP_ATTRIBUTE_ID	NUMBER(20)	The primary unique key of the relationship customer-defined attribute.
RELATIONSHIP_ID	NUMBER(20)	The foreign key of the relationship associated with the attribute.
NAME	VARCHAR2(128)	The name of the attribute.
VALUE	VARCHAR2(128)	The value of the attribute.

4.5.1.8 RM_DOMAIN

Table 4–9 Relationship Management Domain Database Table

Column Name	Data Type	Description
DOMAIN_ID	NUMBER(10)	The primary unique key of the domain record.
NAME	VARCHAR2(128)	The unique name of the MPI domain.
URL	VARCHAR2(128)	The Oracle WebLogic JNDI URL of the MPI domain.
USERID	VARCHAR2(128)	The user name of accessing to the Oracle WebLogic JNDI.

Table 4–9 (Cont.) Relationship Management Domain Database Table

Column Name	Data Type	Description
PASSWORD	VARCHAR2(128)	The wallet file alias name. Currently, not in use.
JNDINAME	VARCHAR2(128)	JNDI name
OBJECT_XML	CLOB	The OHMPI domain object definition in XML.

4.5.1.9 RM_RULESET

Table 4–10 Relationship Management Ruleset Database Table

Column Name	Data Type	Description
RULESET_ID	NUMBER(10)	The primary unique key of the ruleset record.
NAME	VARCHAR2(128)	The unique name of the ruleset.
DESCRIPTION	VARCHAR2(255)	The description of the ruleset.
SOURCE	VARCHAR2(128)	The source entity name of the ruleset.
TARGET	VARCHAR2(128)	The target entity name of the ruleset
CREATED_DATE	DATE	The date when the ruleset is created. The date format is configurable.
CREATED_BY	VARCHAR2(32)	The user name that creates the ruleset. It is the signed-in user.
MODIFIED_DATE	DATE	The date when the ruleset is modified. The date format is configurable.
MODIFIED_BY	VARCHAR2(32)	The user name that modifies the relationship. It is the signed-in user.
STATUS	VARCHAR2(10)	The status of the ruleset. It can be Active or Inactive.
RULE_XML	CLOB	The ruleset file in XML format.

4.5.1.10 RM_TASK

Table 4–11 Relationship Management Task Database Table

Column Name	Data Type	Description
TASK_ID	NUMBER(10)	The primary unique key of the task record.
NAME	VARCHAR2(128)	The unique name of the task.
DESCRIPTION	VARCHAR2(255)	The description of the task.
STATUS	VARCHAR2(10)	The status of the task. It can be Active or Inactive.
TASK_SCRIPT	CLOB	The groovy task file in groovy file extension.

4.5.1.11 RM_AUDIT_EVENT

Table 4–12 Relationship Management Audit Event Database Table

Column Name	Data Type	Description
AUDIT_EVENT_ID	NUMBER(20)	The primary unique key of the audit event record.
AUDIT_SOURCE_ID	NUMBER(20)	Foreign key of the RM_AUDIT_SOURCE table linked with the audit event.
AUDIT_OBJECT_ID	NUMBER(20)	Foreign key of the RM_AUDIT_OBJECT table linked with the audit event.
EVENT_ACTION_CODE	VARCHAR2(2)	Event type of the REST API. For example, C for Create, U for Update, A for Activate, D for deactivate, R for Read.
EVENT_DATE_TIME	DATE	The date when the audit event is generated.
EVENT_OUTCOME_INDICATOR	VARCHAR2(20)	Indicates if the REST API that was invoked was SUCCESS or FAILURE.
EVENT_TYPE_CODE	VARCHAR2(256)	The type code of the audit event.

4.5.1.12 RM_AUDIT_SOURCE

Table 4–13 Relationship Management Audit Source Database Table

Column Name	Data Type	Description
AUDIT_SOURCE_ID	NUMBER(20)	The primary unique key of the audit source.
USER_ID	NUMBER(20)	User name of logged in user who invoked the REST API.
ALTERNATIVE_USER_ID	VARCHAR2(20)	The process ID of the local operating system.
ROLE_CODE	VARCHAR2(20)	The role assigned to the user who invoked the REST API.
NETWORK_ACCESS_POINT_TYPE	VARCHAR2(2)	The network access point type.
NETWORK_ACCESS_POINT_ID	VARCHAR2(128)	The network access point ID.

4.5.1.13 RM_AUDIT_OBJECT

Table 4–14 Relationship Management Audit Object Database Table

Column Name	Data Type	Description
AUDIT_OBJECT_ID	NUMBER(20)	The primary unique key of the audit object record.
OBJECT_TYPE_CODE	VARCHAR2(20)	The object type whose REST API was invoked. For example, EntityType, Domain, and so on.
OBJECTID	VARCHAR2(20)	The object ID whose REST API was invoked.
OBJECT_NAME	CLOB	The object name whose REST API was invoked.
OBJECT_QUERY	VARCHAR2(1024)	Query string of the REST API invoked.
OBJECT_DETAIL	CLOB	Details of the object provided in REST API. Currently, not in use.
SOURCE_OBJECTID	CLOB	The source object ID whose REST API was invoked. Applicable for relationship type and relationship.
SOURCE_OBJECTNAME	CLOB	The source object name whose REST API was invoked. Applicable for relationship type and relationship.
SOURCE_OBJECT_DETAIL	CLOB	Details of the source object provided in REST API. Currently, not in use.

Table 4–14 (Cont.) Relationship Management Audit Object Database Table

Column Name	Data Type	Description
TARGET_OBJECTID	CLOB	The target object ID whose REST API was invoked. Applicable for relationship type and relationship.
TARGET_OBJECTNAME	CLOB	The target object name whose REST API was invoked. Applicable for relationship type and relationship.
TARGET_OBJECT_DETAIL	CLOB	Details of the target object provided in REST API. Currently, not in use.

4.5.1.14 RM_RELATIONSHIP_MERGE

Table 4–15 Relationship Management Merge Database Table

Column Name	Data Type	Description
RELATIONSHIP_MERGE_ID	NUMBER(20)	The primary unique key of the relationship merge record.
ENTITY_ID	NUMBER(20)	The unique entity ID that merged.
RELATIONSHIP_ID	NUMBER(20)	The relationship ID caused by the merged ID.
ACTION	VARCHAR2(20)	The action of the merge. It can be Create, Deactivate, or Update.

4.5.1.15 RM_CATEGORY

Table 4–16 Relationship Management Category Database Table

Column Name	Data Type	Description
CATEGORY_ID	NUMBER(10)	The primary unique key of the category record.
NAME	VARCHAR2(128)	The unique name of the category.
DESCRIPTION	VARCHAR2(255)	The description of the category.

Relationship Discovery and Relationship Rules

This chapter introduces how to utilize the relationship rules for the relationship discovery including the relationship discovery process. It also provides the information about the relationships rules and how to define and register the relationship rules.

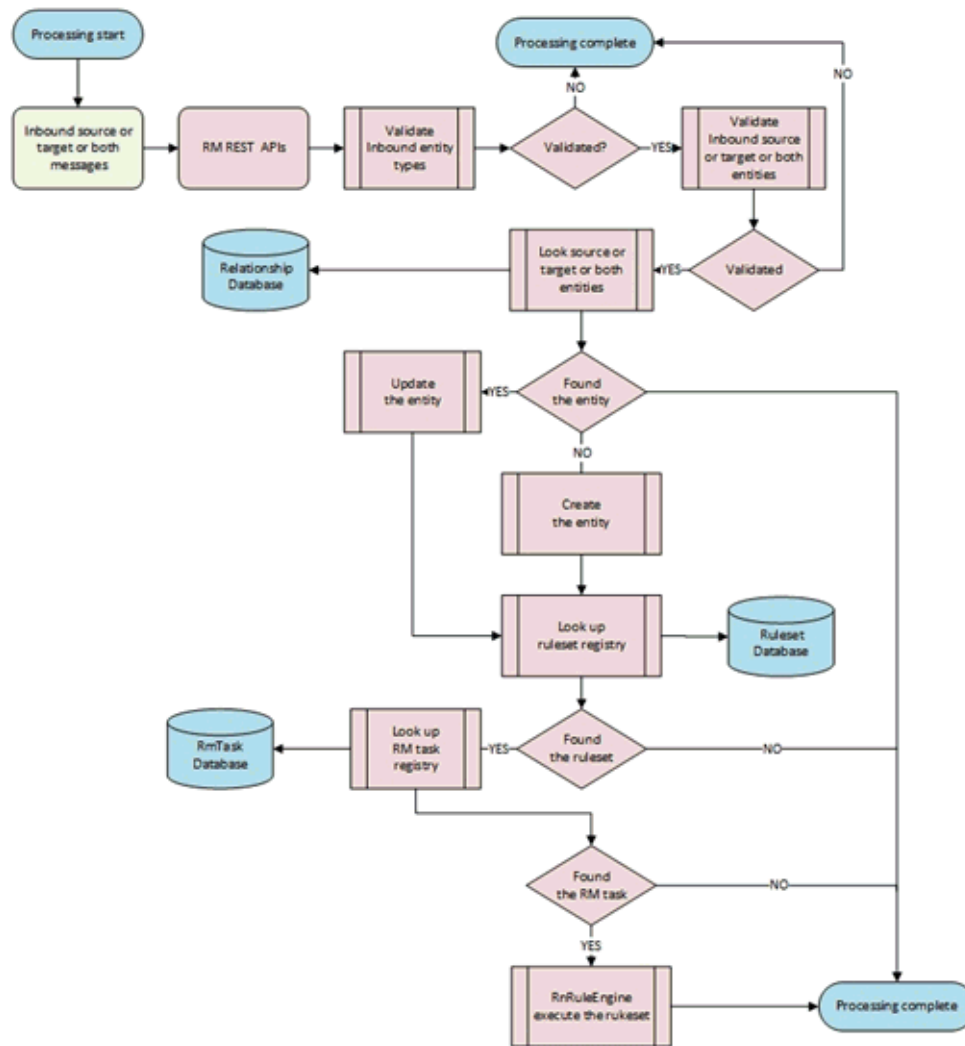
5.1 Relationship Discovery

The relationship discovery feature of the relationship management provides the ability to identify and extract key relationships in the inbound messages from the OHMPI applications or the upstream integral applications. The relationship discovery feature is built upon the standard open architecture. The business and processing logic for discovering relationships is determined by the relationship rules. The relationship rule is completely definitive and pluggable. Users can define and create custom relationship rules according to the specific business requirements.

The policies, process and business logic for automating relationship discovery are too dynamic to manage effectively as applications in many situations. The definitive relationship rules can help users develop more agile relationship management solutions.

The relationship rule is literally a business rule. The business rule as a statement that defines or constrains some aspect of the business; a business rule is intended to assert business structure or to control or influence the business's behavior. The relationship rule is to externalize the relationship management process logic. The relationship rule is executed by the standard relationship management rule engine.

[Figure 5-1](#) illustrates relationship discovery process dataflow by the relationship rules and tasks.

Figure 5–1 Relationship Discovery Process Dataflow

5.1.1 Integrating with Relationship Rules

The relationship discovery process is triggered when you execute the following relationship management REST APIs:

- Create a relationship from a single source or target entity by using configured relationship rules.
 - **entity**: The name of the source or target entity type.
 - **request body**: The attributes of the source or target entity.

```
POST /resources/relationships/entities/{entity}
{ "entity resource attributes" }
```

- Create a relationship between the source entity and the target entity by using configured relationship rules.

- **source**: The name of the source entity type.
- **target**: The name of the target entity type.
- **request body**: The attributes of the source entity and target entity.

```
POST /resources/relationships/entities/{source}/{target}
```


- ```
{ "items": [{"source entity attributes"}, {"target entity attributes"}] }
```
- Create a relationship from a single source or target entity by using configured relationship rules.

- **entity:** The name of the source or target entity type.
- **request body:** The attributes of the source or target entity.

```
POST /integration /relationships/entities/{entity}
{ "entity resource attributes" }
```

- Create a relationship between the source entity and the target entity by using configured relationship rules.

- **source:** The name of the source entity type.
- **target:** The name of the target entity type.
- **request body:** The attributes of the source entity and target entity.

```
POST /integration/relationships/entities/{source}/{target}
{ "items": [{"source entity attributes"}, {"target entity attributes"}] }
```

For more information, see *Oracle Healthcare Master Person Index Relationship Management REST APIs Reference Guide*.

## 5.2 Relationship Rules

A relationship rule is as a set of the *if-then* statements. The *if-then* statements are the rules. A rule is composed of two parts: a condition and an action. When the condition is met, the action is executed. The *if* portion contains conditions. The inputs to a relationship rule engine are a collection of rules called a rule execution set and data objects. The outputs are determined by the inputs and may include the relationship and entity data objects with modifications.

Use relationship rules when the business logic and process workflow for creating relationships are:

- Too dynamic to be managed at the pre-defined applications
- Cannot be predefined and require customization

### 5.2.1 Rules Syntax

A rule is composed of the following elements:

- A unique name
- A description
- A list of conditions
  - A condition having the format: "source" ["operator" "target"]
  - An operator between source and target can be: =, <>, <, >, <=, >=, in, like, between, exist, or between

---

**Note:** =, <>, and exist are supported in this release.

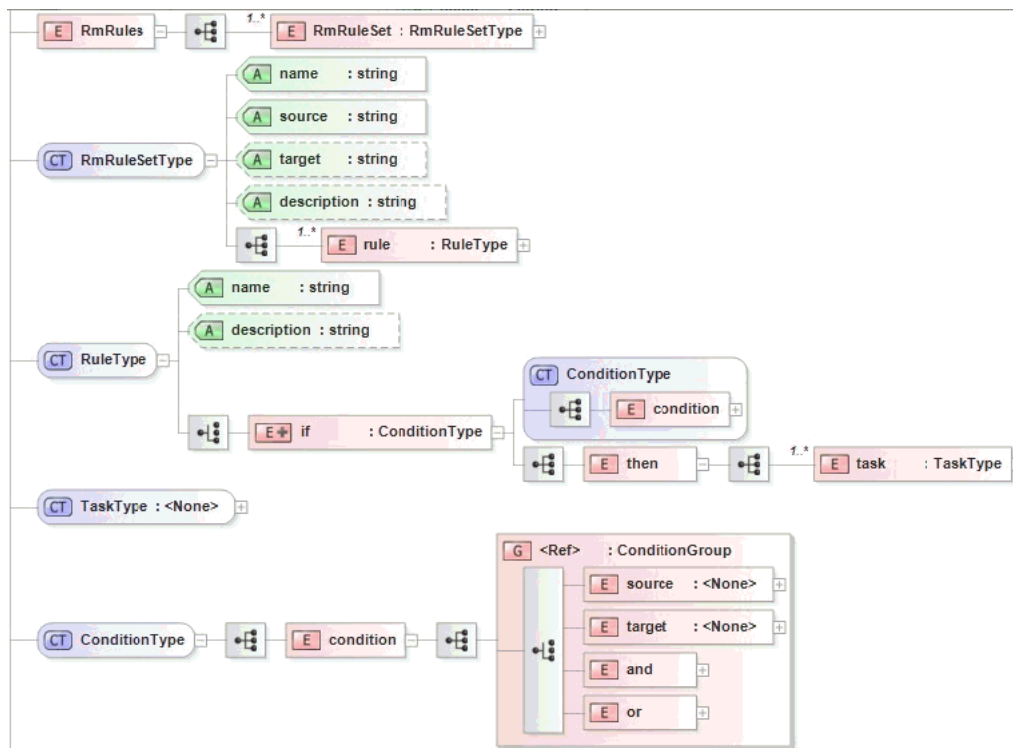
---

- If a condition has only *source*, then operator is *exists*
- The value of *source* is the entity attribute in EPATH starting with the domain attribute name. For example:
  - \* "source" = "FirstName\_St"

- \* "source" = "Address.AddressLine1\_StName"
- The value of *target* is also the entity attribute in EPATH starting with the domain attribute name or can be constant value of (text or numeric value) or an enumeration of values expressed as [value1, value2, and so on].
- AND or OR group condition
- A list of tasks
  - \* A task having the format: "TaskName" ["arg-1" "arg-2" ... "arg-N"]
  - \* TaskName is unique and has one-to-one map to groovy script class name

Figure 5–2 illustrates the ruleset XSD schema.

**Figure 5–2 RuleSet XSD Schema**



## 5.2.2 RuleSet

A ruleset is a collection of rules. Each rule set can contain one rule or combine multiple rules. Each rule can have a single condition or multiple conditions grouped, each rule can include a single task or multiple tasks if the condition is satisfied.

The ruleset starts with RmRuleSet element in the ruleset XML file. Each ruleset XML file contains one or multiple rulesets. The following are the RmRuleSet attributes:

- **name:** The unique name of the ruleset, the name needs to comply with the relationship management generic naming standard.
- **source:** The name of the source entity type for the relationship type that this ruleset processes. It has to match the name of the source entity type in the RM ruleset table.

- **target:** The name of the target entity type for the relationship type that this ruleset processes. It has to match the name of the target entity type in the RM ruleset table.
- **description:** The description of the ruleset.

The rule starts with the rule element. The rule element contains the following attributes:

- **name:** The unique name of the rule, the name needs to comply with the relationship management generic naming standard.
- **description:** The description of the rule.

## 5.2.3 Condition Rules

You can define condition rules to use in conjunction with the operation rules described in [Section 5.2.3.1, "Condition Operators"](#). Conditional rules return either true or false. They only define a condition and not an action, so they must be used with other types of rules. Conditional rules use *if*, *condition*, and *then* statements in the following format:

```
<rule name="rule name" description="rule description" >
 <if>
 <condition>
 ...
 </condition>
 <then>
 ...
 </then>
 </if>
</rule>
```

### 5.2.3.1 Condition Operators

The condition operators are used to evaluate values for condition rules. The following condition operators are predefined:

- equal
- not equal
- exist
- not exist
- conditional OR
- conditional AND

#### 5.2.3.1.1 Equal

This operator evaluates whether the value of the specific attribute of the source entity is equal to the value of the specific attribute of the target source. This operator returns *true* if the conditions are matched; otherwise it returns *false*. The syntax for equal is:

```
<source name="attribute name of the source entity" op="=" target=" attribute name
of the target entity" />
```

The parameters for equal are:

- **name:** The qualified attribute name of the source entity in e-path
- **op:** The predefined operator indicator (=)
- **target:** The qualified attribute name of the target entity in e-path

The e-path starts with the attribute name of the primary object or the child object name following by a dot and the attribute name of the child object. For example, `FirstName` indicates the `FirstName` attribute of `Patient`, `Address.AddressLine1` indicates the `AddressLine` attribute of `Patient's Address` child object. The attributes of the entity type and the name of the child object have to match what is defined in the metadata.

The following rule checks whether the value of the `Id` of the `Patient's Provider` equals to the value of the `NPI Provider`. They match, the operator returns `true`, and then the task of `CreatePatientOfTask` gets executed; otherwise the rule does nothing.

```
<rule name="rule name" description="rule description" >
 <if>
 <condition>
 <source name="Provider.Id" op="=" target="NPI" />
 </condition>
 <then>
 <task name="CreatePatientOfTask" arg1="Patient-Of " />
 </then>
 </if>
</rule>
```

### 5.2.3.1.2 Not Equal

This operator evaluates whether the value of the specific attribute of the source entity is not equal to the value of the specific attribute of the target source. This operator returns `true` if the conditions are matched; otherwise it returns `false`. The syntax for not equal is:

```
<source name="attribute name of the source entity" op="<>" target=" attribute name
of the target entity" />
```

The parameters for not equal are:

- **name:** The qualified attribute name of the source entity in e-path
- **op:** The predefined operator indicator (<>)
- **target:** The qualified attribute name of the target entity in e-path

The following rule checks whether the value of the `Id` of the `Patient's Provider` does not equal to the value of the `NPI Provider`. They do not match, the operator returns `true`, then the task of `DeactivatePatientOfTask` gets executed; otherwise the rule does nothing.

```
<rule name="rule name" description="rule description" >
 <if>
 <condition>
 <source name="Provider.Id" op="<>" target="NPI" />
 </condition>
 <then>
 <task name="DeactivatePatientOfTask" arg1="Patient-Of " />
 </then>
 </if>
</rule>
```

### 5.2.3.1.3 Exist

This operator evaluates whether the value of the specific attribute of the source entity or the target entity exists. This operator returns `true` if the conditions are matched; otherwise it returns `false`. The syntax for exist is:

```
<source name="attribute name of the source entity" />
```

OR

```
<target name="attribute name of the target entity" />
```

The parameter for exist is:

- **name:** The qualified attribute name of the source entity or the target entity in e-path.

The following rule checks whether the incoming message contains the value of the Doctor attribute of the Patient, the operator returns true, and then the task of PatientOfTask gets executed; otherwise the rule does nothing.

```
<rule name="rule name" description="rule description" >
 <if>
 <condition>
 <source name="Doctor" />
 </condition>
 <then>
 <task name=" CreatePatientOfTaskByDoctorNode" arg1="Patient-Of" />
 </then>
</if>
</rule>
```

#### 5.2.3.1.4 Not Exist

This operator evaluates whether the value of the specific attribute of the source entity or the target entity does not exist. This operator returns *true* if the conditions are matched; otherwise it returns *false*. The syntax for not exist is:

```
<source name="attribute name of the source entity" op="not exists" />
```

OR

```
<target name="attribute name of the target entity" op="not exists" />
```

The parameter for not exist is:

- **name:** The qualified attribute name of the source entity or the target entity in e-path.

The following rule checks whether the incoming message does not contain the value of the Doctor attribute of the Patient, the operator returns true, and then the task of "PatientOfTask" gets executed; otherwise the rule does nothing.

```
<rule name="rule name" description="rule description" >
 <if>
 <condition>
 <source name="Doctor" op="not exists" />
 </condition>
 <then>
 <task name=" CreatePatientOfTaskByDoctorNode" arg1="Patient-Of" />
 </then>
</if>
</rule>
```

#### 5.2.3.1.5 Conditional OR

The conditional OR operation is performed on multiple condition operators. The multiple condition operators are grouped by *OR* and the condition operators are evaluated in order. This operator returns *true* if any of the condition operators results in true; otherwise it returns *false*. The syntax for conditional OR is:

```

<or>
 <!-- Condition Operator -->
 <source name="attribute name of the source entity" op="=" target=" attribute
name of the target entity" />
 <source name="attribute name of the source entity" op="<" target=" attribute
name of the target entity" />
 <source name="attribute name of the source entity" />
 <target name="attribute name of the target entity" />
 ...
</or>

```

The following rule checks whether the value of the NPI of the Patient's Provider equals to the value of the NPI Provider or whether the value of the Id of the Patient's Provider equals to the value of the NPI Provider either of them matches, the operator returns true, then the task of CreatePatientOfTask gets executed; otherwise the rule does nothing.

```

<rule name="rule name" description="rule description" >
 <if>
 <condition>
 <or>
 <source name="Provider.NPI" op="=" target="NPI" />
 <source name="ProviderId" op="=" target="NPI" />
 </or>
 </condition>
 <then>
 <task name="CreatePatientOfTask" arg1="Patient-Of " />
 </then>
 </if>
</rule>

```

### 5.2.3.1.6 Conditional AND

The conditional AND operation is performed on multiple condition operators. The multiple operators are grouped by *AND* and the condition operators are evaluated in order. This operator returns *true* if all the condition operators result in true; otherwise it returns *false*. The syntax for conditional AND is:

```

<and>
 <!-- Condition Operator -->
 <source name="attribute name of the source entity" op="=" target=" attribute
name of the target entity" />
 <source name="attribute name of the source entity" op="<" target=" attribute
name of the target entity" />
 <source name="attribute name of the source entity" />
 <target name="attribute name of the target entity" />
 ...
</and>

```

The following rule checks whether the value of the NPI of the Patient's Provider equals to the value of the NPI Provider and whether the value of the Id of the Patient's Provider equals to the value of the NPI Provider both of them matches, the operator returns true, then the task of CreatePatientOfTask gets executed; otherwise the rule does nothing.

```

<rule name="rule name" description="rule description" >
 <if>
 <condition>
 <and>
 <source name="Provider.NPI" op="=" target="NPI" />
 <source name="ProviderId" op="=" target="NPI" />
 </and>
 </condition>
 <then>
 <task name="CreatePatientOfTask" arg1="Patient-Of " />
 </then>
 </if>
</rule>

```

```

 </and>
 </condition>
 <then>
 <task name="CreatePatientOfTask" arg1="Patient-Of " />
 </then>
</if>
</rule>

```

The conditional AND and conditional OR operators can be combined together and create more complicated conditions. The two samples of syntax for combining conditional AND and conditional OR operators are listed below:

```

<rule name="rule name" description="rule description" >
 <if>
 <condition>
 <and>
 ...
 <or>
 ...
 </or>
 ...
 </and>
 </condition>
 <then>
 <task name="CreatePatientOfTask" arg1="Patient-Of " />
 </then>
 </if>
</rule>

```

```

<rule name="rule name" description="rule description" >
 <if>
 <condition>
 <or>
 ...
 <and>
 ...
 </and>
 ...
 </or>
 </condition>
 <then>
 <task name="CreatePatientOfTask" arg1="Patient-Of " />
 </then>
 </if>
</rule>

```

## 5.2.4 Relationship Management Task Interface

```

package oracle.hsgbu.ohmpi.rm.rules.task;

import oracle.hsgbu.ohmpi.rm.rules.service.RmTaskException;

/**
 * RmTaskInterface Interface
 * @author <author_name>
 */public interface RmTaskInterface {
/**

```

```

* Execute the task
* @param context RmTaskContext
* @throws RmTaskException
*/
void execute(RmTaskContext context) throws RmTaskException;
}

```

## 5.2.5 Registering Relationship Rulesets

This section explains how to register the relationship rulesets and the relationship management tasks. To do so, perform the following steps:

1. Define and create rules and ruleset. For information, see [Section 5.2.5.1, "Defining and Creating Rules and Ruleset"](#).
2. Design and implement the relationship management task interface in groovy script. For information, see [Section 5.2.5.2, "Designing and Implementing the Relationship Management Task Interface in Groovy Script"](#).
3. Register the ruleset. For information, see [Section 5.2.5.3, "Registering Ruleset"](#).
4. Register the relationship management task. For information, see [Section 5.2.5.4, "Registering Relationship Management Task"](#).

### 5.2.5.1 Defining and Creating Rules and Ruleset

1. Create a rule and ruleset, in a text or XML editor, based on the rule and ruleset syntaxes respectively.

The following is a sample CreatePatientOfRuleset.xml ruleset:

```

<?xml version="1.0" encoding="UTF-8"?>
<RmRules xmlns="http://www.w3.org"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.w3.org rules.xsd" >
 <RmRuleSet name=" CreatePatientOfRuleset "
 source="Patient"
 target="Provider "
 description="a sample ruleset" >
 <rule name="CreatePatientOfRule" description="a sample rule" >
 <if>
 <condition>
 <source name="Provider.Id" op="=" target="NPI" />
 </condition>
 <then>
 <task name="CreatePatientOfTask" arg1="Patient-Of " />
 </then>
 </if>
 </rule>
 </RmRuleSet>
</RmRules>

```

2. Register the ruleset.

For information, see [Section 5.2.5.3, "Registering Ruleset"](#).

Registering the ruleset using the management REST APIs or the RM data manager validates the syntax of the ruleset.



### 5.2.5.2 Designing and Implementing the Relationship Management Task Interface in Groovy Script

After you create the ruleset, you must design and implement the RM task which is referred by the ruleset. The RM task is the implementation of `RmTaskInterface` in groovy. The RM project creates a skeleton custom RM task. You can implement your business logic based on the business use cases. Perform the following steps to implement the RM Task interface:

1. In the Projects panel, expand the Source Packages of OHMPI Relationship Management Custom Rm Task.
2. Open the `CustomRmTaskImpl.groovy` file in the NetBeans editor.

The following appears:

```

/*-----*
 * Copyright (c) 2010 Oracle Corporation, Redwood Shores, CA, USA *
 * All rights reserved. *
-----/
package oracle.hsgbu.ohmpi.rm.rules.task;

import oracle.hsgbu.ohmpi.rm.data.AttributeType;
import oracle.hsgbu.ohmpi.rm.data.EntityType;
import oracle.hsgbu.ohmpi.rm.data.RelationshipType;
import oracle.hsgbu.ohmpi.rm.data.KeyValue;
import oracle.hsgbu.ohmpi.rm.data.EntityAttribute;
import oracle.hsgbu.ohmpi.rm.data.Entity;
import oracle.hsgbu.ohmpi.rm.data.RelationshipAttribute;
import oracle.hsgbu.ohmpi.rm.data.Relationship;
import oracle.hsgbu.ohmpi.rm.rules.task.RmTaskContext;
import oracle.hsgbu.ohmpi.rm.rules.task.RmTaskInterface;
import oracle.hsgbu.ohmpi.rm.rules.task.RmTaskMetaData;
import oracle.hsgbu.ohmpi.rm.rules.task.RmTaskService;
import oracle.hsgbu.ohmpi.rm.rules.service.data.DataObject;
import oracle.hsgbu.ohmpi.rm.rules.service.data.RmDataObject;
import oracle.hsgbu.ohmpi.rm.rules.service.data.EntityDataObject;
import oracle.hsgbu.ohmpi.rm.rules.service.data.LiteralDataObject;
import oracle.hsgbu.ohmpi.rm.rules.service.RmTaskException;

/**
 * Custom Implementation of RmTaskInterface
 * @author oracle
 */
class CustomRmTaskImpl implements RmTaskInterface {

public void execute(RmTaskContext context)
 throws RmTaskException {
 RmTaskMetaData rmMetadata = context.getRmTaskMetaData();
 RmTaskService rmService = context.getRmTaskService();
 EntityDataObject sourceData = context.getSourceData();
 EntityDataObject targetData = context.getTargetData();
 RmDataObject rmData = context.getRmData();

print "Custom implementation of groovy task"
 }
}

```

3. Implement your business logic.

---

---

**Note:** Make sure that there is no syntax error.

---

---

4. Save the file.
5. Register the Rm groovy task.  
For information, see [Section 5.2.5.4, "Registering Relationship Management Task"](#).

### 5.2.5.3 Registering Ruleset

1. Define the relationship task business and processing logic.
2. Define the relationship rules and combine rules in a ruleset.  
You create the relationship ruleset xml file. For example, CreatePatientOfRuleset.xml.
3. Register the relationship ruleset either using the relationship management REST API or the relationship management data manager by assigning the unique ruleset name.

To use the relationship management REST API, follow these steps:

- a. Create the relationship ruleset using the management REST API:

```
POST /management/rulesets/{ruleset}
```

For example,

```
POST /management/rulesets/CreatePatientOfRuleset
{"name": " CreatePatientOfRuleset",
 "description":"ruleset for creating Patient-Of relationship",
 "source":"Patient"
 "target": "IndProvider"}
```

- b. Upload the relationship ruleset XML file using the management REST API:

```
PUT /management/rulesets/{ruleset}files
```

For example,

```
PUT/management/rulesets/CreatePatientOfRuleset/files
CreatePatientOfRuleset.xml
```

### 5.2.5.4 Registering Relationship Management Task

1. Define the relationship task business and processing logic.
2. Implement the relationship task interface in groovy script.  
Create the relationship task interface implementation file name with *.groovy* extension file name. For example, CreatePatientOfTaskImpl.groovy.
3. Register the relationship task interface implementation either using the relationship management REST API or the relationship management data manager by assigning the unique task name for the relationship task.

To use the relationship management REST API, perform the following steps:

- a. Register the custom relationship event policy using the management REST API:

```
POST /management/tasks/{task}
```

For example,

---

```
POST /management/tasks/CreatePatientOfTask
{"name": "CreatePatientOfTask "}
```

- b.** Upload the relationship task implementation script groovy file using the management REST API:

```
PUT /management/tasks/{task}files
```

For example,

```
PUT /management/tasks/CreatePatientOfTask/files
CreatePatientOfTaskImpl.groovy
```



---

## Relationship Update and Event Policies

The relationship management policies are used to specify custom business logic and process logic for a variety of relationship management transactions and events. The relationship policy is pluggable and written in standard groovy script. The custom relationship policy is loaded in the relationship management task table and identified by the reserved system task names. By default, the relationship management contains the default relationship policies which define the default business logic and process logic. The default business logic and process logic satisfy common use cases. You must understand the supported common use cases to see if your business cases are covered during the solution design phase. You must define and create your relationship policy for your business logic and process logic.

The relationship management supports the following types of relationship policies:

- **Relationship Update Policy:** The relationship update policy defines processing logic on how to update or merge the attributes of two relationships when two relationships get updated or merged. The default strategy is 'most recent modified'. This strategy ranks the field values from the incoming message in descending order according to the time that the relationship was last modified. The value populated in the relationship comes from the most recently modified relationship. Each relationship type can define its own update policy. The policy is triggered by the relationship management when the relationship management creates a relationship between two entities while the two entities already have the same relationship.
- **Relationship Event Policy:** The relationship event policy is a set of relevant policies and defines processing logic on how to process events from the incoming message. ADD event is for adding a new entity. MRG event is for merging two entities to be one. UNMRG event is for splitting one entity to two entities. These events are coming from the integrated MPI applications or integration tier application. The event policy is triggered by the relationship management when the user invokes the event synchronization REST API.

### 6.1 Relationship Update Policy

For any inbound message of requesting to establish a relationship of a specific relationship type between the source entity and the target entity, the same relationship between these two entities might exist already in the relationship management database, the difference between the newly attempted relationship and the existing relationship can be only with changes of the attribute values such as expiration date or adding new attributes. Therefore, how to update the existing relationships is the business decision. For example,

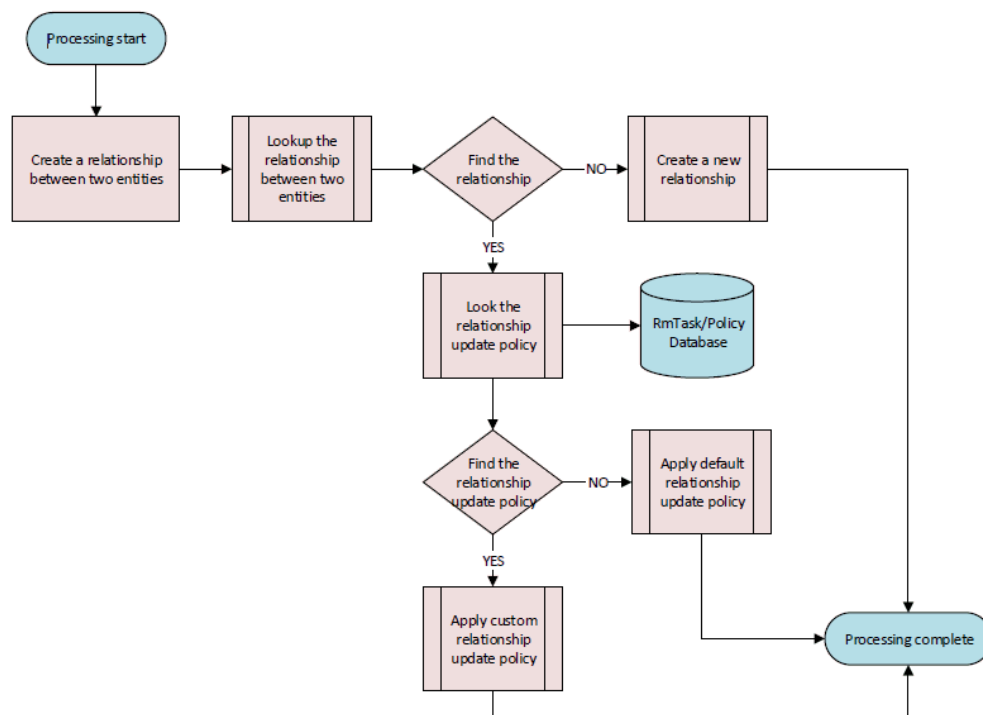
- Allow one relationship instance of a specific relationship type between the two entities, merge the new relationship and the old relationship based on the business use cases
- Allow multiple relationship instances of a specific relationship type exist between the two entities such as Doctor A is PCP of Patient B at the different clinics.

A pluggable relationship update policy of a relationship type manages how to update the existing relationships for various use cases in the relationship management database. This provides flexibility to customize the update relationship business logic and process for satisfying different use cases.

### 6.1.1 Relationship Update Policy Processing Dataflow

Figure 6–1 illustrates the process steps, and includes the triggering process of the default or custom relationship update policy.

**Figure 6–1 Relationship Update Policy Processing Dataflow**



#### 6.1.1.1 Default Relationship Update Policy

The following are the strategies of the default relationship update policy:

- You can create only one singular relationship between two entities
- Update the existing attributes of existing relationship based on the most recent modified date
- Add the new attributes of the existing relationship based on the relationship type metadata

### 6.1.1.2 Customizing the Relationship Update Policy

Each relationship type can be associated with a specific custom update policy. This section explains how to customize the relationship update policy. To customize the update policy, perform the following steps:

1. Define the custom update business and process logic
2. Implement the custom update policy interface in groovy script.  
Create the policy interface implementation file name with *.groovy* extension file name. For example, *CustomPatientOfUpdatePolicyImpl.groovy*.
3. Load the custom update policy interface implementation either using the relationship management Metadata REST API or the relationship management data manager by assigning a unique policy task name.

- a. Register a new custom relationship policy using the management REST API:

```
POST /management/tasks/{task}
```

For example,

```
POST /management/tasks/CustomPatientOfUpdatePolicyImpl
{"name": "CustomPatientOfUpdatePolicyImpl"}
```

- b. Upload the custom relationship update policy script groovy file using the management REST API:

```
PUT /management/tasks/{task}files
```

For example,

```
PUT/management/tasks/CustomPatientOfUpdatePolicyImpl/files
CustomPatientOfUpdatePolicyImpl.groovy
```

4. Update the update policy task of the relationship type using metadata REST API:

```
PUT /metadata/relationships/{name}
```

For example,

```
PUT /metadata/relationships/patient-of
{"name": "patient-of",
"task": {"name": "CustomPatientOfUpda
"description": "customized update policy for updating patient-of
relationship"}
```

Now, you can create a new relationship of the same relationship type between two entities that are already linked by the existing relationship. The result will be a single instance of the relationship for the same relationship type.

## 6.1.2 Relationship Update Policy Plugin Interface

```
package oracle.hsgbu.ohmpi.rm.policy;

import java.util.List;
import oracle.hsgbu.ohmpi.rm.data.Relationship;

/**
 * Relationship UpdatePolicy interface
 * @author <author_name>
 */
public interface UpdatePolicy {
/**
 * Apply business and process logic of updating the existing relationship
```

```

* @param oldRelationship The existing relationship
* @param newRelationship The newly attempted relationship
* @return List<Relationship> A list of updated relationships
* @throws RelationshipPolicyException Throw exception if any error occurs
*/
List<Relationship> apply(Relationship oldRelationship, Relationship
newRelationship)
 throws RelationshipPolicyException;
}

```

To implement the relationship update policy interface, you must have the *ohmpi-rm-client.jar* file under your relationship management project folder.

## 6.2 Relationship Event Policies

The OHMPI is setup to deliver JMS outbound notification for operations such as an update or a merge and a un-merge. When an operation occurs, the OHMPI creates an event message in XML and sends the event message to a JMS topic to notify external integrated applications of changes in data. The RM is integrated with the OHMPI applications and reacted to the OHMPI notification events by subscribing the JMS topic. The events are broadcasted from the OHMPI applications through the JMS topic to the RM MPI agent. And the RM MPI agent transforms the XML events to the messages in the RM format and sends to the RM by invoking the RM REST API. The RM takes the expected actions according the events. The following are the six events: ADD, MRG, UNMRG, DEA, REA, and UPD. This release provides the default business and processing logic for the following events:

- **ADD:** When a new entity, such as Patient or IndProvider is added and a new enterprise object is added in the OHMPI, the OHMPI generates ADD event and publishes it to the MPI JMS topic. The ADD event contains the event type, the domain type, the EUID, the SBR and the all the necessary information.
- **MRG:** When two enterprise objects, such as Patient or IndProvider are merged in the OHMPI, the OHMPI generates MRG event and publishes it to the MPI JMS topic. The MRG event contains the event type, the domain type, the survived EUID, the survived SBR, the merged EUID, the merged SBR and the all the necessary information.
- **UNMRG:** When an enterprise object, such as Patient or IndProvider gets unmerged in the OHMPI, the OHMPI generates UNMRG event and publishes it to the MPI JMS topic. The UNMRG event contains the event type, the domain type, the survived EUID, the survived SBR, the merged EUID, the merged SBR and the all the necessary information.

The event policy defines and implements the business and processing logic for the event. The event policy is written in groovy script and completely pluggable. The relationship management contains the out-of-box default event policies for processing ADD, MRG, and UNMRG events. The relationship management does not provide the default relationship event policies for UPD, DEA, and REA events. You must define and implement the relationship event policy in order to process these events.

### 6.2.1 Default MRG Event Policy

This relationship event policy defines and implements the default business and processing logic for processing MRG event. The policy performs the following steps to process MRG event:

1. Deactivate the merged entity.
2. Deactivate all the relationships associated with the merged entity.



These relationships are named by the merged relationships.

3. Replicate the merged relationships to the surviving entity if the relationships associated with surviving entity do not have the merged relationships.
4. Update the relationships associated with the surviving entity if the relationships associated with surviving entity already contain the merged relationship.
5. Record the merge information in the relationship management merge table to keep track of the merge process.

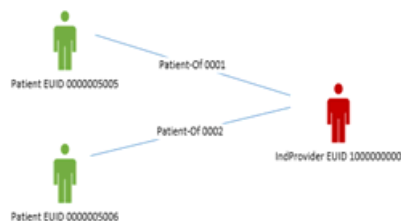
## 6.2.2 Default MRG Use Cases

This section explains the default MRG use cases which are handled by the default MRG event policies. The more complicated MRG use cases can be built upon of them and can be also handled by default MRG event policies or the custom event policies. To understand the use cases better, the use cases are addressed based on an example that the user has Patient MPI and IndProvider MPI. The user also has the corresponding relationship date model including Patient source entity and IndProvider target entity and the relationship types between these two entity types, such as Patient-Of and Admitted-By.

### 6.2.2.1 MRG Use Case A

Two MPI entities from one MPI domain have the relationships of the same relationship type with the same entity from other MPI domain. [Figure 6–2](#) illustrates a sample MRG use case A diagram.

**Figure 6–2 Sample MRG Use Case A**



**Figure 6–3 Merge Result of Sample MRG Use Case A**



After the default MRG event policy processes the MRG event, the result is illustrated in [Figure 6–3](#). The Patient EUID 0000005005 is merged to the Patient EUID 0000005006. The Patient EUID 0000005006 is the merged entity and the Patient EUID 0000005005 is the surviving entity. The following are the steps performed by the default MRG event policy:

1. Deactivate the merged entity.
  - a. Deactivate Patient EUID 0000005005.
2. Deactivate the relationship associated with the merged entity.

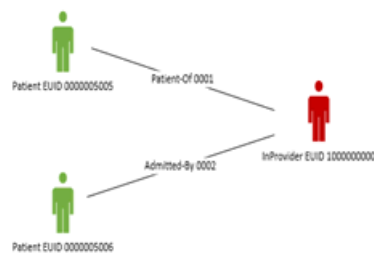
This relationship is the merged relationship.

- a. Deactivate the merged relationship Patient-Of 0001.
3. Update the attributes of the surviving relationship using from the merged relationship.
  - a. If the surviving relationship has the attribute from the merged relationship, the surviving relationship wins the value of the attribute.
  - b. If the surviving relationship does not have the attribute from the merged relationship, the surviving relationship replicates the attribute from the merged relationship.
  - c. Update Patient-Of 0001 to Patient-Of 0002.
4. Record the merge information in the relationship merge table to keep track of the merge.

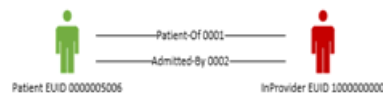
### 6.2.2.2 MRG Use Case B

Each of two MPI entities from one MPI domain has the different relationship of the different relationship type with the same entity from other MPI domain. [Figure 6-4](#) illustrates a sample MRG use case B diagram.

**Figure 6-4 Sample MRG Use Case B**



**Figure 6-5 Merge Result of Sample MRG Use Case B**



After the default MRG event policy processes the MRG event, the result is illustrated in [Figure 6-5](#). The Patient EUID 0000005005 is merged to the Patient EUID 0000005006. The Patient EUID 0000005005 is the merged entity and the Patient EUID 0000005006 is the surviving entity. The following are the steps performed by the default MRG event policy:

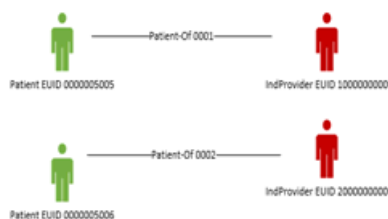
1. Deactivate the merged entity.
  - a. Deactivate Patient EUID 0000005005.
2. Deactivate the relationship associated with the merged entity.
  - a. Deactivate the merged relationship Patient-Of 0001.
3. Create a new relationship which to the surviving entity.

- a. Replicate the merged relationship Patient-Of 0001 between the surviving Patient EUID 0000005006 and IndProvider 1000000000.
4. Record the merge information in the relationship merge table to keep track of the merge.

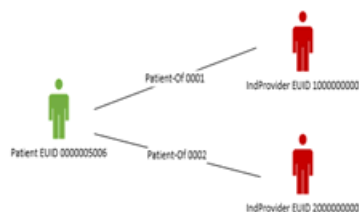
### 6.2.2.3 MRG Use Case C

Each of two MPI entities from one MPI domain has the relationships of the same relationship type with the different entities from other MPI domain. [Figure 6-6](#) illustrates a sample MRG use case C diagram.

**Figure 6-6 Sample MRG Use Case C**



**Figure 6-7 Merge Result of Sample MRG Use Case C**



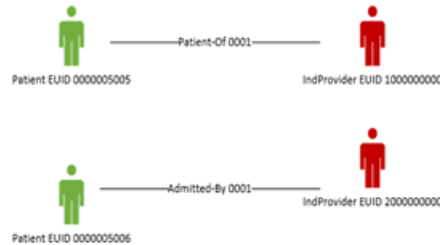
After the default MRG event policy processes the MRG event, the result is illustrated in [Figure 6-7](#). The Patient EUID 0000005005 is merged to the Patient EUID 0000005006. The Patient EUID 0000005005 is the merged entity and the Patient EUID 0000005006 is the surviving entity. The following are the steps performed by the default MRG event policy:

1. Deactivate the merged entity.
  - a. Deactivate Patient EUID 0000005005.
2. Deactivate the relationship associated with the merged entity.
  - a. Deactivate the merged relationship Patient-Of 0001.
3. Replicate the merged relationship to the surviving entity.
  - a. Replicate the merged relationship Patient-Of 0001 between the surviving Patient EUID 0000005006 and IndProvider 2000000000.
4. Record the merge information in the relationship merge table to keep track of the merge.

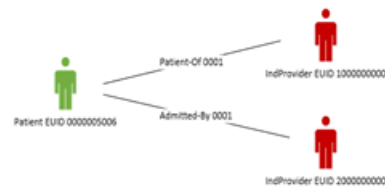
### 6.2.2.4 MRG Use Case D

Each of two MPI entities from one MPI domain has the different relationship of the different relationship type with the different entities from other MPI domain. [Figure 6–8](#) illustrates a sample MRG use case D diagram.

**Figure 6–8 Sample MRG Use Case D**



**Figure 6–9 Merge Result of Sample MRG Use Case D**



After the default MRG event policy processes the MRG event, the result is illustrated in [Figure 6–9](#). The Patient EUID 0000005005 is merged to the Patient EUID 0000005006. The Patient EUID 0000005005 is the merged entity and the Patient EUID 0000005006 is the surviving entity. The following are the steps performed by the default MRG event policy:

1. Deactivate the merged entity.
  - a. Deactivate Patient EUID 0000005005.
2. Deactivate the relationship associated with the merged entity.
  - a. Deactivate the merged relationship Patient-Of 0001.
3. Create a new relationship which to the surviving entity.
  - a. Replicate the merged relationship Patient-Of 0001 between the surviving Patient EUID 0000005006 and IndProvider 1000000000.
4. Record the merge information in the relationship merge table to keep track of the merge.

## 6.2.3 Default UNMRG Event Policy

This relationship event policy defines and implements the default business and processing logic for processing UNMRG event.

Each MRG use case has its corresponding UNMRG use case logically. The UNMRG process is a reverse process of the MRG to roll back the changes resulted from the MRG.

The processing of the above use cases is about changing source entity of a relationship. The same logic can apply to the changing of target entity of a relationship and multiple relationships.

### 6.2.4 Default Add Event Policy

This relationship event policy defines and implements the default business and processing logic for processing ADD event. The policy performs the following steps to process ADD event:

1. Validate the entity
2. Look up the entity
  - a. Update the attributes of the entity based on the entity type metadata if the entity exists in the relationship management database
  - b. Create a new entity if the entity does not exist in the relationship management database
3. Look up the Ruleset registry
  - a. Execute the ruleset to create a relationship if the ruleset is found for the entity
  - b. Process completes if the ruleset is not found

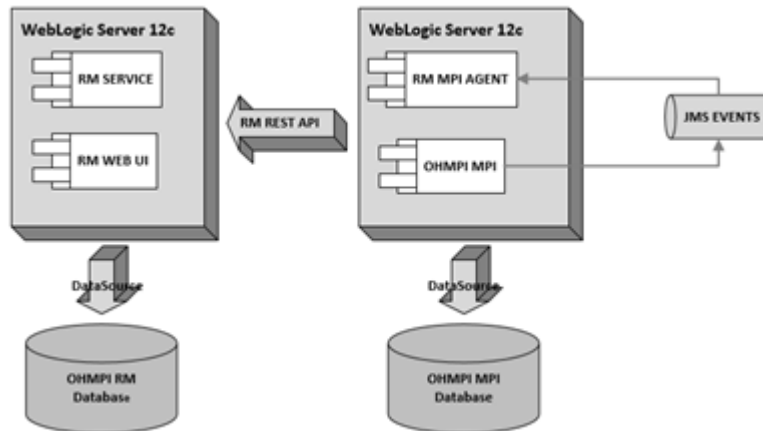
By default, no ruleset is configured in the ruleset registry. Therefore, the default Add event policy creates or updates the entity in the relationship management database.

## 6.3 Understanding the Relationship Management MPI Agent

The relationship management MPI agent is used for subscribing to the OHMPI JMS outbound topic and receiving all the OHMPI events from the OHMPI applications. The relationship management MPI agent transforms the XML event into the relationship management message, and sends it to the relationship management service by invoking the relationship management synchronization REST API. The relationship management MPI agent can filter out the unwanted events to the relationship management service. The filter is configurable.

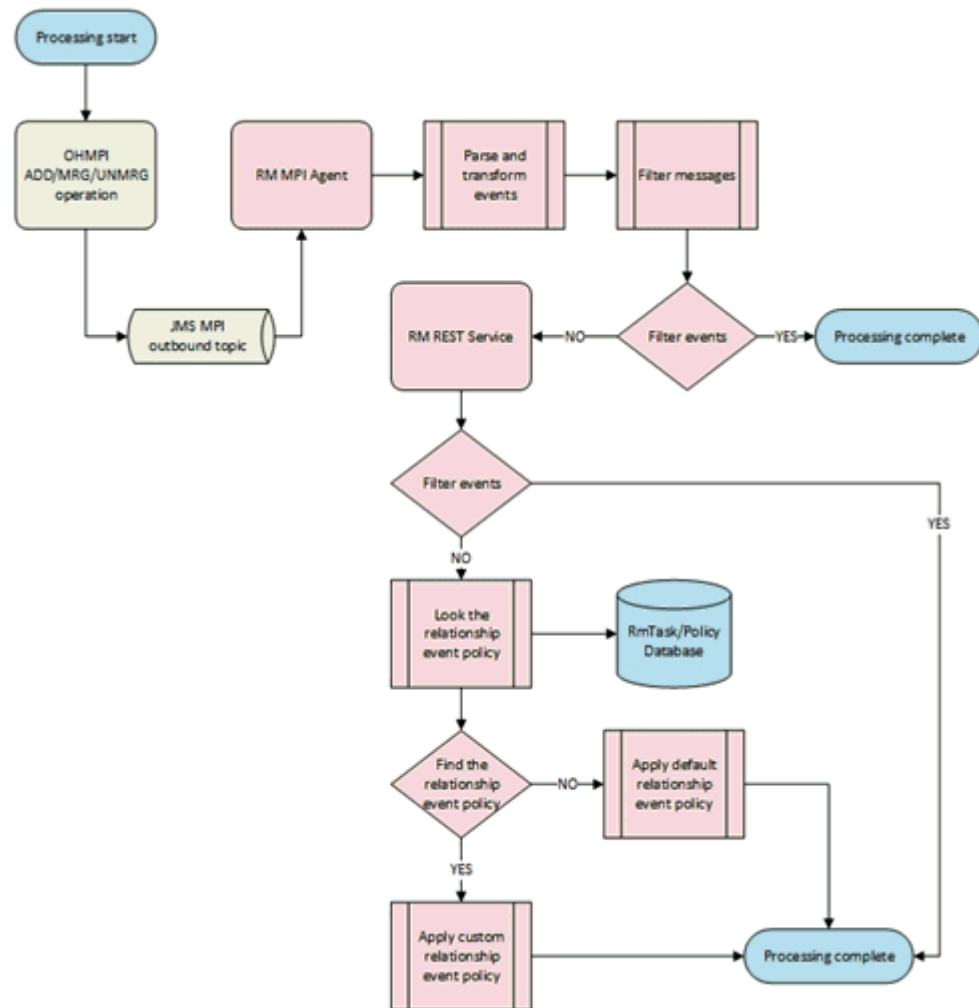
In order for the relationship management to synchronize the MPI events from the MPI applications, several applications are required to be configured and deployed. [Figure 6–10](#) illustrates the required software and applications.

**Figure 6–10** Deployment Architecture of OHMPI and MPI Agent, and the Relationship Management



### 6.3.1 Event Policy Process Dataflow

Figure 6–11 illustrates the process of the OHMPI JMS outbound events, the relationship management MPI agent, and includes the triggering process of the default or custom relationship event policy.

**Figure 6–11 Relationship Event Policy Processing Dataflow**

### 6.3.2 Creating the Relationship Management MPI Agent

To create a relationship management MPI agent application, perform the following steps:

1. Create the relationship management MPI application. For information, see [Chapter 2, "Creating the Relationship Management Project"](#).
2. Create and build the relationship management MPI agent application. For information, see [Section 6.3.2.1, "Creating and Building the Relationship Management MPI Agent"](#).
3. Configure the relationship management MPI agent application. For information, see [Section 6.3.2.2, "Configuring the Relationship Management MPI Agent"](#).
4. Deploy the relationship management MPI agent application. For information, see [Section 6.3.2.3, "Deploying the Relationship Management MPI Agent"](#).

#### 6.3.2.1 Creating and Building the Relationship Management MPI Agent

1. In the Projects window, right-click on the RM project and select **Create MPI Agent**.

The Create MPI Agent dialog box appears.

2. Enter values for the following:

- MPI application name
- Base Url
- Wallet File Path
- RM Date Format
- MPI Date Format
- MPI Events

3. Click **Create**.

The relationship management wizard creates the MPI agent and builds the EAR file of the MPI agent in the dist folder of the project. The file name is *<MPI application name>-ohmpi-rm-agent-1.0.0.ear*.

4. To create a different MPI agent, repeat steps 1 through 3.

### 6.3.2.2 Configuring the Relationship Management MPI Agent

To open the existing relationship management project:

1. From the NetBeans IDE, click **File** and then select **Open Project**.
2. In the Open Project panel, browse to the path where you have created the relationship management project.
3. Select the relationship management project and click **Open Project**.
4. In the Projects window, expand the relationship management project > configuration node > Agent node.
5. Right-click the *rm\_agent.properties* file and select **Edit**.

The MPI agent properties appear in the right panel.

6. Edit the following properties in the *rm\_agent.properties* file:

- Edit the base URL (relationship.management.base.url) of the relationship management REST APIs
- Edit the wallet file path (relationship.management.wallet.file) of the relationship management MPI agent
- Edit the relationship management supported MPI events (relationship.management.events)

You can define multiple events delimited by a comma. The default events are ADD, MRG, and UNMRG. You can add UPD, DEA, and REA events. The relationship management provides out-of-box default event policies for processing ADD, MRG, and UNMRG events, but does not provide any default event policies for UPD, DEA, and REA events.

- Edit the date format (relationship.management.date.format) of the relationship management service
- Edit the date format (relationship.management.mpi.date.format) of the MPI domain application
- Edit the MPI domain application name (relationship.management.mpi.application.name)



To configure the RM MPI agent:

1. Open the file that you want to modify.
  - From the Configuration\agent folder, you can modify the following file:
    - rm\_agent.properties
  - From the Configuration\agent\META-INF folder, you can modify the following files:
    - application.xml
    - web.xml
    - weblogic.xml
2. Edit the file, as required.
3. Save the file.

### 6.3.2.3 Deploying the Relationship Management MPI Agent

The relationship management MPI agent is always deployed to the same WebLogic Server domain where the OHMMPI application runs. Each OHMMPI application requires its own specific MPI agent. To deploy the MPI agent, you must first configure the OHMMPI outbound JMS topic and deploy the OHMMPI application, and then perform the following steps:

1. Create and install the MPI agent wallet file. For information, see [Section 6.3.2.3.1, "Creating the RM MPI Agent Wallet Files"](#).
2. Deploy and run the MPI agent application. For information, see [Section 6.3.2.3.2, "Deploying and Running the MPI Agent Application on Oracle WebLogic Server"](#).

#### 6.3.2.3.1 Creating the RM MPI Agent Wallet Files

---

**Note:** Make sure that JVM 1.7.0\_67 or higher version is set in the system execution path.

---

1. Go to the relationship management project folder on the command console.
2. Execute the following command in the wallet creation utility:

```
run generate-wallet.bat/generate-wallet.sh [userName] [password] [walletpwd]
where,
```

- **username:** Username for the RM agent connecting or executing to the RM APIs
- **password:** Password for the RM connecting or executing to the RM APIs
- **walletpwd:** Wallet password for securing the wallet content

The ohmpiRMWallet folder is created.

3. Copy the ohmpiRMWallet folder to the MPI domain config folder of your WebLogic Server.
4. Expand your relationship management project in the Projects windows and open the rm\_agent.properties file.
5. Edit relationship.management.wallet.file in the rm\_agent.properties file and set it to be ./config/ohmpiRMWallet.

6. Build and deploy the RM MPI agent. For information, see [Section 6.3.2, "Creating the Relationship Management MPI Agent"](#).

#### 6.3.2.3.2 Deploying and Running the MPI Agent Application on Oracle WebLogic Server

1. On the left panel of the WebLogic Server Administration Console, under Domain Structure, select **Environment > Deployments**.

The Summary of Deployments panel appears.

2. Click **Install**.

The Summary of Deployments panel with a Deployments table containing a list of EAR files appears.

3. Locate your application EAR and click **Next**.

The Install Application Assistant section appears in the right panel.

4. Locate the deployment file you want to install and prepare for deployment.

**Tip:** Select the file path that represent the application root directory, archive file, exploded archive directory, or application module descriptor that you want to install. You can also enter the path of the application directory or file in the Path field.

---

---

**Note:** Only valid file paths are displayed. If you cannot find your deployment files, upload your file(s) and/or confirm that your application contains the required deployment descriptors.

---

---

5. Click **Next**.

---

---

**Note:** When deploying an MPI EAR file through the WebLogic Administrator Console, under Security, make sure that you select the **DD Only** option.

---

---

6. Click **Finish**.

## 6.4 Customizing the Relationship Event Policy

This section explains how to customize the relationship event policy. To customize the relationship event policy, you must implement the Event Policy interface using groovy script, and load it to the relationship management task registry using the reserved task names for the relationship event policies.

- AddEventPolicy is the reserved task name for the custom relationship ADD event policy
- MrgEventPolicy is the reserved task name for custom relationship MRG event policy
- UnmrgEventPolicy is the reserved task name for custom relationship UNMRG event policy

To customize the relationship event policy:

1. Determine which event needs to be customized.

2. Define the custom event business and processing logic.
3. Implement the custom relationship event policy interface in groovy script.  
You create the event policy interface implementation file name with the *.groovy* extension file name. For example, *CustomAddEventPolicyImpl.groovy*.
4. Load the custom relationship event policy interface implementation either using the relationship management REST API or the relationship management data manager by assigning the reserved task name for the relationship event policy.

To use the relationship management REST API:

- a. Register the custom relationship event policy using the management REST API:

```
POST /management/tasks/{task}
```

For example,

```
POST /management/tasks/MrgEventPolicy
{"name": "MrgEventPolicy"}
```

- b. Upload the custom relationship update policy script groovy file using the management REST API:

```
PUT /management/tasks/{task}files
```

For example,

```
PUT/management/tasks/MrgEventPolicy/files
CustomAddEventPolicyImpl.groovy
```

## 6.5 Relationship Event Policy Plugin Interface

```
package oracle.hsgbu.ohmpi.rm.policy;

/**
 * EventPolicy Interface
 * @author <author_name>
 */
public interface EventPolicy {
 /* The reserved task name for custom relationship ADD event policy*/
 public static final String ADD_EVENT_POLICY = "AddEventPolicy";
 /* The reserved task name for custom relationship UPD event policy*/
 public static final String UPD_EVENT_POLICY = "UpdEventPolicy";
 /* The reserved task name for custom relationship MRG event policy*/
 public static final String MRG_EVENT_POLICY = "MrgEventPolicy";
 /* The reserved task name for custom relationship UNMRG event policy*/
 public static final String UNMRG_EVENT_POLICY = "UnmrgEventPolicy";
 /* The reserved task name for custom relationship DEA event policy*/
 public static final String DEA_EVENT_POLICY = "DeaEventPolicy";
 /* The reserved task name for custom relationship REA event policy*/
 public static final String REA_EVENT_POLICY = "ReaEventPolicy";

 /**
 * callback method to execute the event policy
 * @param context EventPolicy context
 * @throws EventPolicyException
 */
 void apply(EventPolicyContext context)
 throws EventPolicyException;
}
```



---

---

## Auditing Relationship Management

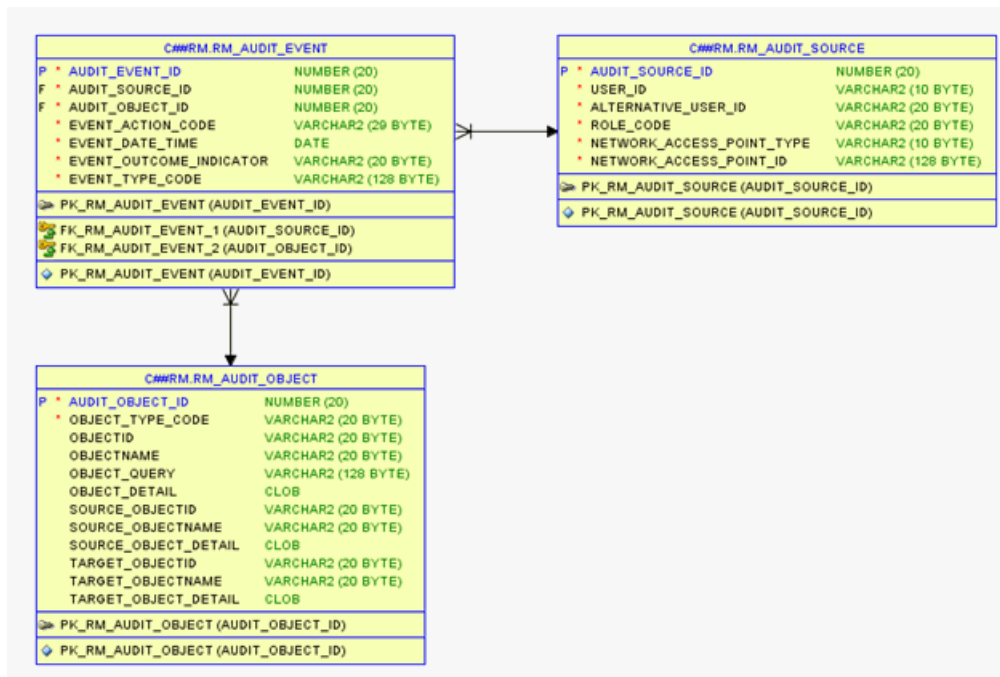
The OHMPI RM auditing service meets the HIPAA compliance statement in the application level for monitoring and auditing the relationship management backend service activities. The following are the types of auditing event that the relationship management auditing service creates:

- Object Access Events
  - Entity Type Access
  - Relationship Type Access
  - Entity Access
  - Relationship Access
  - Domain Access
  - Task Access
  - Ruleset Access
- Operation Events
  - Create
  - Update
  - Search
  - Activate and Deactivate
- MPI Inbound Notification Events

### 7.1 Audit Event Data Model

Figure 7-1 illustrates the audit event data model.

Figure 7-1 Audit Event Data Model



## 7.2 Audit Event Format

### 7.2.1 RM\_AUDIT\_EVENT Table

Field Name	Optional (O) or Mandatory (M)	Value Constraints
EventId	M	Indicates the system generated ID
EventActionCode	M	Possible values are: <ul style="list-style-type: none"> <li>▪ C: create</li> <li>▪ U: update</li> <li>▪ R: read</li> <li>▪ D: deactivate</li> <li>▪ A: activate</li> </ul>
EventDateTime	M	Displays the date and time when the event was logged
EventOutcomeIndicator	M	Possible values are SUCCESS and FAILURE
EventTypeCode	M	Indicates the REST Resource URI

### 7.2.2 RM\_AUDIT\_SOURCE Table

Field Name	Optional (O) or Mandatory (M)	Value Constraints
UserId	M	Indicates the user who initiated the transaction

Field Name	Optional (O) or Mandatory (M)	Value Constraints
AlternativeUserId	M	Indicates the process ID as used within the local operating system in the local system logs
RoleCode	M	Indicates the role of the user ID
NetworkAccessPointType	M	Possible values are: <ul style="list-style-type: none"> <li>■ 1: Machine (DNS) name</li> <li>■ 2: IP address</li> </ul>
NetworkAccessPointId	M	Indicates the machine name or IP address, as specified in RFC 3881

### 7.2.3 RM\_AUDIT\_OBJECT Table

The RM\_AUDIT\_OBJECT table stores information based on the OBJECT\_TYPE\_CODE.

Field Name	Optional (O) or Mandatory (M)	Value Constraints
ObjectTypeCode	M	EntityType
ObjectId	O	Indicates the entity type ID
ObjectName	O	Indicates the entity type name
ObjectQuery	O	Query string base64 encoded
ObjectDetail	O	Not supported

Field Name	Optional (O) or Mandatory (M)	Value Constraints
ObjectTypeCode	M	RelationshipType
ObjectId	O	Indicates the relationship type ID
ObjectName	O	Indicates the relationship type name
ObjectQuery	O	Query string base64 encoded
ObjectDetail	O	Not supported
SourceName	O	Indicates the source entity type name of the relationship type.
TargetName	O	Indicates the target entity type name of the relationship type.

Field Name	Optional (O) or Mandatory (M)	Value Constraints
ObjectTypeCode	M	Entity
ObjectId	O	Indicates the entity ID
ObjectName	O	Indicates the entity type name
ObjectQuery	O	Query string base64 encoded
ObjectDetail	O	Not supported

Field Name	Optional (O) or Mandatory (M)	Value Constraints
ObjectTypeCode	M	Entity
ObjectId	O	Indicates the entity ID
ObjectName	O	Indicates the entity type name

Field Name	Optional (O) or Mandatory (M)	Value Constraints
ObjectQuery	O	Query string base64 encoded
ObjectDetail	O	Not supported
SourceObjectId	O	Indicates the source entity ID
SourceObjectName	O	Indicates the source entity name
SourceObjectDetail	O	Not supported
TargetObjectId	O	Indicates the target entity ID
TargetObjectName	O	Indicates the target entity name
TargetObjectDetail	O	Not supported

Field Name	Optional (O) or Mandatory (M)	Value Constraints
ObjectTypeCode	M	Domain
ObjectId	O	Indicates the domain ID
ObjectName	O	Indicates the domain name
ObjectQuery	O	Query string base64 encoded
ObjectDetail	O	Not supported

Field Name	Optional (O) or Mandatory (M)	Value Constraints
ObjectTypeCode	M	RmTask
ObjectId	O	Indicates the RmTask ID
ObjectName	O	Indicates the RmTask name
ObjectQuery	O	Query string base64 encoded
ObjectDetail	O	Not supported

Field Name	Optional (O) or Mandatory (M)	Value Constraints
ObjectTypeCode	M	RmRuleSet
ObjectId	O	Indicates the RmRuleSet ID
ObjectName	O	Indicates the RmRuleSet name
ObjectQuery	O	Query string base64 encoded
ObjectDetail	O	Not supported

### 7.3 Setting Up the Audit Database Tables

You must create the audit tables when the application is deployed. For information on how to execute the database create scripts, see [Section 2.4.2, "Creating Database Tables"](#).

### 7.4 Enabling or Disabling Auditing

---



---

**Note:** By default, auditing is enabled.

---



---



While creating the RM project in NetBeans, you can enable or disable auditing. For information, see [Section 2.1, "Creating the Relationship Management Project"](#).

To modify the audit configuration after a project is created:

1. In NetBeans, navigate to the RM project Configuration\service folder.
2. Open the ohmpi-rm.properties file.
3. Configure the desired value for the *relationship.management.audit.enabled* property.
  - To enable auditing, set the value to true.
  - To disable auditing, set the value to false.
4. Right-click on the RM project and select **Clean and Build**.

---

---

**Note:** Even if the auditing is disabled, you must create the audit tables. For information, see [Section 7.3, "Setting Up the Audit Database Tables"](#).

---

---

## 7.5 Viewing the Audit Log

1. Open an SQL editor and connect to the OHMPI RM database using the user with read access to the audit tables.
2. In the NetBeans editor, open the following script from the <Project\_Name>/samples directory:
  - search-ohmpi-rm-audit.sql

---

---

**Note:** You can modify the script as required. By default, all the audit rows are sorted in descending order of date-time.

---

---

3. Copy the entire text from search-ohmpi-rm-audit.sql and paste into the SQL editor.
4. Execute the script against the database.

## 7.6 Archiving the Audit Log

Auditing service (if it is enabled) logs each API call made to the Relationship Management Service. So, the audit entries can become huge over a period of time. You can archive the audit entries before a specific period of time.

1. Open an SQL editor and connect to the OHMPI RM database using the user with delete access to the audit tables.
2. In the NetBeans editor, open the following script from the <Project\_Name>/samples directory:
  - delete-ohmpi-rm-audit.sql

---

---

**Note:** You can modify the end\_date in the script as the date-time until when the audit logs need to be deleted from each auditing table.

---

---

3. Copy the entire text from delete-ohmpi-rm-audit.sql and paste into the SQL editor.
4. Execute the script against the database.

## 7.7 Dropping the Audit Database Tables

For information, see [Section 4.4, "Dropping the Database Tables"](#).

---



---

## Relationship Management Security

This chapter explains how to secure the RM application. RM REST APIs are authenticated using Basic Authentication.

### 8.1 Security Groups and Roles

RM defines the users, groups, and security roles based on the WebLogic users, groups, and security roles defined in the *Fusion Middleware Securing Resources Using Roles and Policies for Oracle WebLogic Server Guide*. This release defines the following three security roles:

[Table 8–1](#) describes the security roles defined in this release.

**Table 8–1 Security Roles**

Security Role	Name	Privilege	Description	Operating Resources
RmIntegrator	System Integrator	search, create, and update	Used for creating and updating relationships, registering MPI domains, and resolving potential relationships	metadata, relationships, entities, and application configuration
RmAnalyst	Business Analyst	search	Used for searching and viewing the relationships	metadata, relationships, and entities
RmAdministrator	System Administrator	search, create, and update	Used for creating and updating relationships, registering MPI domains, and resolving potential relationships	metadata, relationships, entities, and application configuration

---



---

**Note:** In this release, there is no difference between the system administrator role and system integrator role at the application level.

---



---

[Table 8–2](#) describes the relationship security groups defined in this release.

**Table 8–2 Relationship Security Group**

Security Group	RM Security Role	WebLogic Role
RmAdministrator	RmAdministrator	Admin
RmAnalyst	RmAnalyst	Not Applicable
RmIntegrator	RmIntegrator	AdminChannelUser, Deployer, Operator, Monitor, and AppTester

### 8.1.1 Creating the Relationship Management Security Groups and Roles

Before deploying the RM application, you must create security groups, roles, and users using the WebLogic Server Administration Console.

To create roles and groups:

1. Log on to the Oracle WebLogic Server Administration Console using the credentials for administrator.  
The Oracle WebLogic Administration Console appears.
2. On the left panel, under Domain Structure, select **Security Realms**.
3. In the table on the Summary of Security Realms panel, click **myrealm**, that is the name of the realm.  
The Settings for myrealm panel appears.
4. Click the **Users and Groups** tab and then click the **Groups** tab.  
The Groups table appears.
5. Click **New** to add a new RM group.  
The Create a New Group panel appears.
6. In the **Name** field, type *RmAnalyst*.
7. Click **OK**.
8. Repeat steps 6 through 8 to add the following RM groups:
  - RmAdministrator
  - RmIntegrator
9. Click the **Roles and Policies** tab and then click the **Realm Roles** tab.  
The Roles table appears.
10. Expand **Global Roles** and click **Roles**.  
The Global Roles table appears.
11. Click **New** to add a new RM role.  
The Create a New Role for this Realm panel appears.
12. In the **Name** field, type *RmAnalyst*.
13. Click **OK**.
14. Repeat steps 12 through 14 to add the following RM roles:
  - RmAdministrator
  - RmIntegrator
15. Click **RmAnalyst** in the Global Roles table.

The Edit Global Role panel appears.

16. Click **Add Conditions**.
17. Select **Group** from the **Predicate List** drop-down list.
18. Click **Next**.
19. In the **Group Argument Name** field, type **RmAnalyst**.
20. Click **Add**.
21. Click **Finish**.
22. Click **Save**.
23. Repeat steps 16 through 23 for the following RM roles by selecting the corresponding group:
  - RmAdministrator
  - RmIntegrator
24. Click the **Users and Groups** tab and then click the **Users** tab.
 

The Users table appears.
25. Click **New** to add a new RM user.
 

The Create a New User panel appears.
26. In the **Name** field, type *RmAnalyst*.
27. Enter a password for the user in the **Password** field and reconfirm the password in the **Confirm Password** field.
28. Click **OK**.
29. Click **RmAnalyst** in the Users table.
 

The Settings for RmAnalyst panel appears.
30. Click the **Groups** tab.
31. Double-click the **RmAnalyst** group under the **Available** section.
 

The RmAnalyst group moves under the Chosen section.
32. Click **Save**.
 

The RmAnalyst group is assigned to the RmAnalyst user.
33. Repeat steps 25 through 33 for the following users:
  - RmAdministrator
  - RmIntegrator

## 8.2 Masking-off the Sensitive Information

You can mask the custom attributes of relationship and entity instances including ID field of the entity.

To mask the sensitive information, configure the sensitive fields in <Relationship Management project>\config\ui\uiConfig.js.

The following example shows masking configuration for entity type:

```
entityTypes: {
 USPatient: {shape: 'human', color: 'green', sensitive_fields ['SSN' , 'ID']},
```

```
 ...
 }
```

The following example shows masking configuration for relationship type:

```
relationshipTypes: {
 'primary-care-physician-of': {color: '#3385b7', sensitive_fields :['<attribute
name>']},
 ...
}
```

The masked value for a field configured as sensitive is displayed as \*\*\*\*\*.

## 8.3 Enabling SSL

Oracle recommends that you secure RM APIs and application using SSL. To do so, you must configure SSL in WebLogic server. For more information on how to configure WebLogic SSL, see *Oracle Fusion Middleware Administering Security for Oracle WebLogic Server 12.1.3*.

---

---

## Performance Tuning

This chapter provides information about the relationship management performance tuning. The relationship management application project automatically creates the default Oracle database table spaces, database indexes, and Oracle TopLink cache configuration. The settings service as base for you to tune the relationship management application based on the actual resources and environments.

### 9.1 Oracle Database Tuning

For more information on how to manage and tune database tablespaces, see *Managing Tablespaces of Oracle Database 12c Administrator's Guide*.

#### 9.1.1 Managing the Relationship Management Database Tablespaces

The RM project automatically creates the default three tablespaces with the following settings:

- SMALLFILE tablespace
- Datafile is created at \$ORACLE\_HOME
- Tablespace size is 100M and is auto extendable

The OHMPI\_RM tablespace is created for all the relationship management database tables. The OHMPI\_RM\_TEMP tablespace is created for the temporary database tables. The OHMPI\_RM\_AUDIT tablespace is created for the audit database tables.

To customize these settings according to the actual requirements and platforms, modify the *create-ohmpi-rm.sql* or *create-ohmpi-rm-audit.sql* script directly using a text editor.

#### 9.1.2 Creating a BIGFILE Tablespace

Modify the table creation statement by specifying the BIGFILE keyword. For example:

```
CREATE BIGFILE TABLESPACE OHMPI_RM
DATAFILE 'OHMPI_RM.DBF'
SIZE 100M REUSE
AUTOEXTEND ON;
```

#### 9.1.3 Storing the DATAFILE on Different Storage Location

Modify the tablespace creation statement by specifying different DATAFILE path. For example:

- For Linux:

```
CREATE TABLESPACE OHMPI_RM
DATAFILE '/users/tablespace/OHMPI_RM.DBF'
SIZE 100M REUSE
AUTOEXTEND ON;
```

- For Windows:

```
CREATE TABLESPACE OHMPI_RM
DATAFILE 'C:\users\tablespace\OHMPI_RM.DBF'
SIZE 100M REUSE
AUTOEXTEND ON;
```

### 9.1.4 Increasing the Default Initial Tablespace Size

Modify the tablespace creation statement by changing the default initial size of the tablespace. For example:

```
CREATE TABLESPACE OHMPI_RM
DATAFILE 'OHMPI_RM.DBF'
SIZE 200M REUSE
AUTOEXTEND ON;
```

### 9.1.5 Managing the Relationship Management Database Indexes

The RM project automatically creates the database script for creating default database indexes and constraints on all the RM database tables to provide faster access to data for operations. You can create application-specific indexes based on the business use cases by modifying the *create-ohmpi-rm.sql* script. For more information, see *Using Indexes in Database Applications of Oracle Database 12c Advanced Application Developer's Guide*.

The RM project also creates the default database text indexes. The text indexes are only enabled for customer-defined attributes of entity type and relationship type. By default, the text index is disabled. To enable the text index:

1. Define the user-defined attribute of entity type or relationship type indexable using the RM data manager or the RM metadata REST APIs.
2. Set `relationship.management.text.index` property to `true` in `ohmpi_rm.properties`.
3. Rebuild and redeploy the RM application.

For information on how to build and deploy the RM application, see [Chapter 2, "Creating the Relationship Management Project"](#).

For information on text index, see *Text Application Developer's Guide of Oracle Database 12c*.

### 9.1.6 Managing the Relationship Management Database Partition

The database partitioning enhances the performance, manageability, and availability of the RM applications. Partitioning is entirely transparent to the application. Partitioned tables are identical to non-partitioned tables in perspective of the SQL queries and the DML statements. The RM follows the generic Oracle Database partitioning best practice and rules. For example, partition a table when:

- Tables which are greater than 2 GB should always be considered as candidates for partitioning.



- Tables containing historical data, in which new data is added into the newest partition. For example, a historical table where you can only update the current month's data and the other 11 months are read only.

The supported partitioning strategies include Range partitioning, Hash partitioning, List partitioning, Range-Range composite partitioning, and Range-Hash composite partitioning.

Partitioning is use cases and data related. You must analyze and understand the data and the use cases as well to determine what partitioning strategies are used. Generally, you can use the hash partitioning for the RM\_ENTITY table to distribute the entity rows among partitions by the ENTITY\_TYPE\_ID partitioning key.

You can use the hash partitioning and range partitioning for the RM\_RELATIONSHIP table. RELATIONSHIP\_TYPE\_ID is the candidate partitioning key for the hash partitioning or composite partitioning to distribute relationships rows in its own partitions. Based on the search criteria of the use cases, CREATED\_DATE, MODIFIED\_DATE, EXPIRATION\_DATE, and EFFECTIVE\_DATE are the candidate partitioning keys for the date range partitioning or composite partitioning. The date range partitioning can be by months or by weeks based on the data.

For more information, see *Partitioning Guide of Oracle Database 12c*.

## 9.2 Oracle TopLink JPA Tuning

The RM project creates the default JPA tuning parameters in *persistence.xml*. By default,

- Session cache is enabled.

```
<property name="eclipselink.cache.size.default" value="5000" />
```

- Maximum number cached objects are set to 5000.

```
<property name="eclipselink.cache.shared.default" value="true" />
```

You can add more JPA performance tuning parameters in *persistence.xml*, and then redeploy the RM application.

For more information, see *Oracle TopLink (EclipseLink) JPA Performance Tuning*.

## 9.3 Oracle WebLogic Server Tuning

The RM application is Java EE standard-based and execute in the WebLogic server. The RM application also supports clustering. The tuning techniques of the WebLogic application are applicable to the RM application, such as:

- Tune pool sizes
- Use the prepared statement cache
- Use logging last resource optimization
- Tune connection backlog buffering
- Tune the chunk size
- Use optimistic or read-only concurrency
- Use local interfaces
- Use eager-relationship-caching
- Tune HTTP sessions
- Tune messaging applications

For more information, see *Performance Tuning for WebLogic Server 12c*.