Oracle® Retail

Payload Mapper Guide
Release 15.0
E66509-01

December 2015

ORACLE

Oracle® Retail Payload Mapper Guide, Release 15.0

E66509-01

Primary Author: Sanal Parameswaran

Contributing Author : Maria Andrew

Value-Added Reseller (VAR) Language

Oracle Retail VAR Applications

The following restrictions and provisions only apply to the programs referred to in this section and licensed to you. You acknowledge that the programs may contain third party software (VAR applications) licensed to Oracle. Depending upon your product and its version number, the VAR applications may include:

(i) the **MicroStrategy** Components developed and licensed by MicroStrategy Services Corporation (MicroStrategy) of McLean, Virginia to Oracle and imbedded in the MicroStrategy for Oracle Retail Data Warehouse and MicroStrategy for Oracle Retail Planning & Optimization applications.

(ii) the **Wavelink** component developed and licensed by Wavelink Corporation (Wavelink) of Kirkland, Washington, to Oracle and imbedded in Oracle Retail Mobile Store Inventory Management.

(iii) the software component known as **Access Via™** licensed by Access Via of Seattle, Washington, and imbedded in Oracle Retail Signs and Oracle Retail Labels and Tags.

(iv) the software component known as **Adobe Flex™** licensed by Adobe Systems Incorporated of San Jose, California, and imbedded in Oracle Retail Promotion Planning & Optimization application.

You acknowledge and confirm that Oracle grants you use of only the object code of the VAR Applications. Oracle will not deliver source code to the VAR Applications to you. Notwithstanding any other term or condition of the agreement and this ordering document, you shall not cause or permit alteration of any VAR Applications. For purposes of this section, "alteration" refers to all alterations, translations, upgrades, enhancements, customizations or modifications of all or any portion of the VAR Applications including all reconfigurations, reassembly or reverse assembly, re-engineering or reverse engineering and recompilations or reverse compilations of the VAR Applications or any derivatives of the VAR Applications. You acknowledge that it shall be a breach of the agreement to utilize the relationship, and/or confidential information of the VAR Applications for purposes of competitive discovery.

The VAR Applications contain trade secrets of Oracle and Oracle's licensors and Customer shall not attempt, cause, or permit the alteration, decompilation, reverse engineering, disassembly or other reduction of the VAR Applications to a human perceivable form. Oracle reserves the right to replace, with functional equivalent software, any of the VAR Applications in future releases of the applicable program.

Contents

Send Us Your Comments

Oracle Retail Payload Mapper Guide, Release 15.0

Oracle welcomes customers' comments and suggestions on the quality and usefulness of this document.

Your feedback is important, and helps us to best meet your needs as a user of our products. For example:

- Are the implementation steps correct and complete?
- Did you understand the context of the procedures?
- Did you find any errors in the information?
- Does the structure of the information help you with your tasks?
- Do you need different information or graphics? If so, where, and in what format?
- Are the examples correct? Do you need more examples?

If you find any errors or have any other suggestions for improvement, then please tell us your name, the name of the company who has licensed our products, the title and part number of the documentation and the chapter, section, and page number (if available).

> **Note:** Before sending us your comments, you might like to check that you have the latest version of the document and if any concerns are already addressed. To do this, access the Online Documentation available on the Oracle Technology Network Web site. It contains the most current Documentation Library plus all documents revised or released recently.

Send your comments to us using the electronic mail address: retail-doc_us@oracle.com

Please give your name, address, electronic mail address, and telephone number (optional).

If you need assistance with Oracle software, then please contact your support representative or Oracle Support Services.

If you require training or instruction in using Oracle software, then please contact your Oracle local office and inquire about our Oracle University offerings. A list of Oracle offices is available on our Web site at www.oracle.com.

Preface

This *Oracle Retail Payload Mapper Guide* is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Audience

This guide is for:

- Systems administration and operations personnel
- Database administrators
- System analysts and programmers
- Integrators and implementation staff personnel

Documentation Accessibility

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

Access to Oracle Support

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

Related Documents

For more information, see the following:

- *Oracle Retail Integration Bus Implementation Guide*
- *Oracle Retail Integration Bus Installation Guide*
- *Oracle Retail Integration Bus Operations Guide*
- *Oracle Retail Integration Bus Release Notes*
- *Oracle Retail Integration Bus Hospital Administration Guide*
- *Oracle Retail Integration Bus Security Guide*
- *Oracle Retail Integration Bus Support Tools Guide*
- *Oracle Retail Enterprise Integration Guide*
- *Oracle Retail Integration Bus Integration Gateway Services Guide*
- *Oracle Retail Functional Artifact Guide*
- *Oracle Retail Functional Artifact Generator Guide*
- *Oracle Retail Service-Oriented Architecture Enabler Tool Guide*
- *Oracle Retail Integration Bus Java Messaging Service (JMS) Console Guide*
- *Oracle Retail Integration Bus Data Model*

Customer Support

To contact Oracle Customer Support, access My Oracle Support at the following URL:

https://support.oracle.com

When contacting Customer Support, please provide the following:

- Product version and program/module name
- Functional and technical description of the problem (include business impact)
- Detailed step-by-step instructions to re-create
- Exact error message received
- Screen shots of each step you take

Review Patch Documentation

When you install the application for the first time, you install either a base release (for example, 15.0) or a later patch release (for example, 15.0.1). If you are installing the base release or additional patch releases, read the documentation for all releases that have occurred since the base release before you begin installation. Documentation for patch releases can contain critical information related to the base release, as well as information about code changes since the base release.

Improved Process for Oracle Retail Documentation Corrections

To more quickly address critical corrections to Oracle Retail documentation content, Oracle Retail documentation may be republished whenever a critical correction is needed. For critical corrections, the republication of an Oracle Retail document may at times **not** be attached to a numbered software release; instead, the Oracle Retail document will simply be replaced on the Oracle Technology Network Web site, or, in the case of Data Models, to the applicable My Oracle Support Documentation container where they reside.

This process will prevent delays in making critical corrections available to customers. For the customer, it means that before you begin installation, you must verify that you have the most recent version of the Oracle Retail documentation set. Oracle Retail documentation is available on the Oracle Technology Network at the following URL:

`http://www.oracle.com/technetwork/documentation/oracle-retail-100266.html`

An updated version of the applicable Oracle Retail document is indicated by Oracle part number, as well as print date (month and year). An updated version uses the same part number, with a higher-numbered suffix. For example, part number E123456-**02** is an updated version of a document with part number E123456-**01**.

If a more recent version of a document is available, that version supersedes all previous versions.

Oracle Retail Documentation on the Oracle Technology Network

Oracle Retail product documentation is available on the following web site:

`http://www.oracle.com/technetwork/documentation/oracle-retail-100266.html`

(Data Model documents are not available through Oracle Technology Network. You can obtain them through My Oracle Support.)

Conventions

Navigate: This is a navigate statement. It tells you how to get to the start of the procedure and ends with a screen shot of the starting point and the statement "the Window Name window opens."

```
This is a code sample
    It is used to display examples of code
```

1

Support Policy Disclaimer

The PL/SQL Payload Mapper tool is complex and should be used carefully to avoid breaking functionality with Oracle Retail Integration Bus products—and to avoid making future generally available (GA) releases difficult or impossible to accept. Such predicaments are not supported by Oracle Retail, including My Oracle Support.

This guide aims to mitigate the risks of unsupported architectural revamping by providing guidance and references for how to attempt mixed version integration safely and effectively. The sample code and the approaches described in this guide are complex and require a high level of skill and knowledge of the products involved. Any issues that may arise with mixed version integration are considered part of a customization and, therefore, are the responsibility of the customer, not Oracle Retail.

The PL/SQL Payload Mapper tools and examples are designed and tested to work with GA RIB Business Objects.

The sample code referenced throughout in this guide is a non-GA deliverable intended to complement this document and provide proof of concepts useful in a mixed version integration solution.

The PL/SQL Payload Mapper tool support does not extend to the use of the output. Support does not extend beyond the normal support that any tool has; it must conform to its own specifications and documentation.

2

PL/SQL Application Mixed Version Support Use Cases

The following are known problems concerning PL/SQL Application Mixed Version Support:

- Customer deployments of the Oracle Retail applications are Enterprise Release specific (for example, 13.0.2, 13.1.1, 14.1.0, and 15.0.0).
- Between any given enterprise release (and often point releases) there are functional and technical changes to the RIB integration layer that can make them incompatible. These functional gaps must be analyzed and resolved specifically for each base release and for each customer's business needs. Often there are technology stack gaps as well, such as the transition from Oracle Application servers to WebLogic.
- With few exceptions, every Oracle Retail deployment has customer specific modifications to the retail applications and the RIB integration flows.
- Enterprise deployment is expensive, as is customization. Reapplying customizations to upgrade an application or RIB can be cost prohibitive.

Use Cases

The following are the primary mixed version support use cases:

- Upgrade one application but not all. For example, upgrade RWMS to 15.0.0 but do not upgrade RMS 12.x.
- Add a new application from a newer release. For example, add 15.0.0 RWMS to a RMS/RPM 10, 11, 12.x, or 13.x deployment.
- Gradually upgrade all Oracle Retail applications in the entire deployment, when the goal is for a phased approach.

Adaptation should be applied closest to the application end points as possible— and as close to the end point that causes least disruption to the existing deployment and still satisfy the customer's long-term objective.

The customer's long-term objective determines the best mixed version approach. Each case has either a forward or backward adaptation strategy.

Forward and Backward Mapping Use Case Solutions

Forward adaptation is done at the lowest version application. Backward adaptation is done at the higher version application. Regardless of whether forward or backward adaptation is chosen, the solution may require custom code involving mapping payloads in both directions, publication (PUB) and subscription (SUB).

For example, if the strategy is to keep RMS at a fixed version (say, 12.0.x). But to add multiple new Oracle Retail applications that integrate through RIB (say, RWMS 15.0.0 and SIM 15.0.0), forward adaptation at the RMS end point is recommended.

Conversely, if the strategy is to keep RMS at a fixed version (say 12.0.x) and there are numerous third party integrations through that RIB version and only a use case to add a single newer version of RWMS (say 15.0.0), backward adaptation at the RWMS end point is recommended.

You must consider mixes of use cases and strategies.

3

PL/SQL Solution Approach and Concepts

This section describes the general approach for adapting a PL/SQL application.

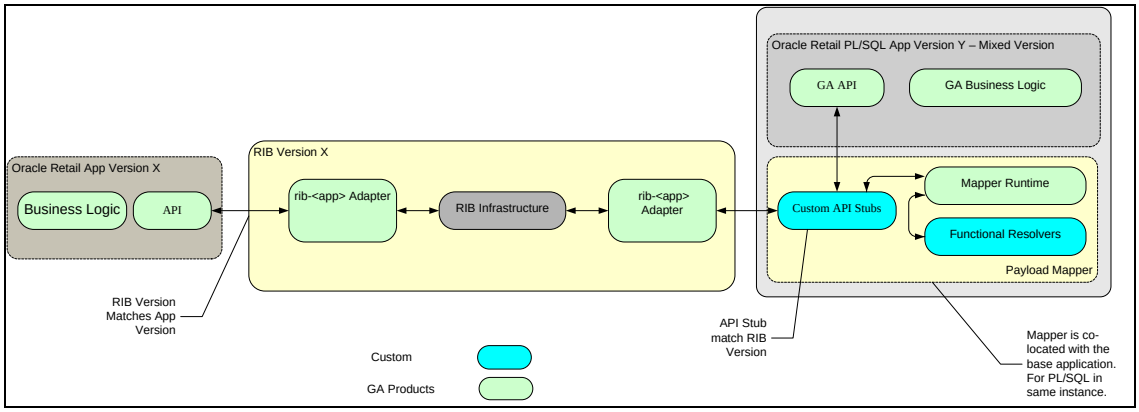
The Adapter Pattern

In programming, the adapter pattern is a design pattern that translates one interface into a compatible interface. An *adapter* allows applications to work together, when normally they cannot because of incompatible interfaces. It provides its interface to clients while using the original interface. The adapter translates calls to its interface into calls to the original interface. Typically, the amount of code necessary to do this is small. The adapter also transforms data into appropriate forms.

The Payload Mapper tools, of which the PL/SQL Payload Mapper tool set is a part, implement the adaptive pattern between two versions of RIB.

Concepts

The following diagram illustrates the general approach for adapting a PL/SQL application into an existing deployment topology of a differing version:



Adapting a PL/SQL Application

The PL/SQL solution approach is to perform the adaptation at the PL/SQL application end point using PL/SQL code. The choice of which PL/SQL application end point in the deployment the adaptation will co-locate with and adapt to depends on the customer's use case and long term strategy.

The general approach is to create an adapting database schema that contains a set of adapting application API's to intercept the deployed RIB's PUB and SUB calls (GETNXT and CONSUME). The adapting APIs call the Mapper Runtime procedure. Using pre-created mapping metadata, the Mapper Runtime procedure performs mapping from the source to the target payload, inserts default values or calls a Functional Resolver as needed, and returns the converted object. The adapting API either calls the GA CONSUME API for SUB calls, or it returns the converted object to the RIB adapter making the GETNXT call.

Benefits

The following are the benefits of adapting a PL/SQL application:

- No changes are required to the existing RIB objects, RIB infrastructure, or base application APIs.
- Both versions of RIB can co-exist and be fully functional, with some additional customization logic for GETNXT, thus allowing for a phased upgrade path.
- Mapping is isolated from the base products and follows the principle of separation of concerns.
- Upgrades to any of the mixed versions can be applied without modification, so the GA support is not broken.
- Existing application and RIB customizations remain untouched.

4

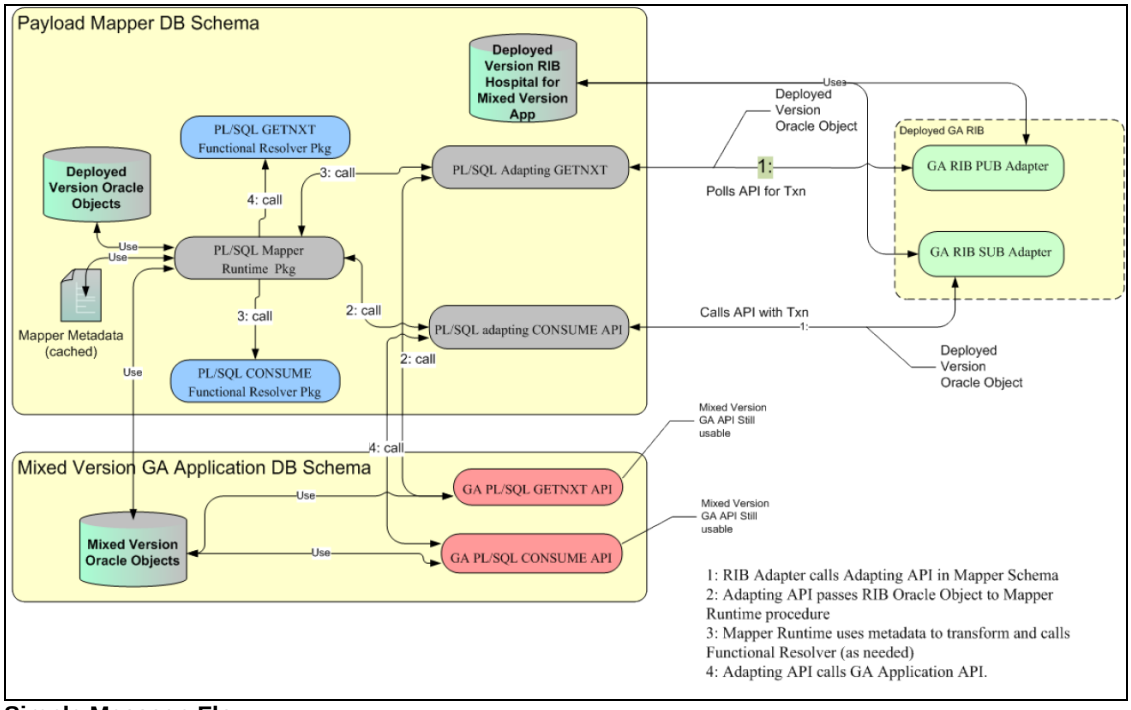
Introduction to the PL/SQL Payload Mapper

The PL/SQL Payload Mapper tools provide a standard way to assist in the custom development of PL/SQL side solutions to mixed version integrations that use RIB, while avoiding situations that break functionality of the base RIB and Oracle application products. A primary design goal is to avoid making future GA releases of Oracle Retail base products difficult or impossible to accept.

Simple Message Flow and Processes

The following illustrates the message flow:

Note: The numbered list in the right corner corresponds to the numbers noted within the flow.



Simple Message Flow

Note: See the *Oracle Retail Enterprise Integration Guide* for information.

Mixed Version Subscription Process

The following steps describe the mixed version subscription process:

1. The deployed version application's publishing adapter creates the message. The event that triggers the message creation may be a polling operation (for PL/SQL applications) or a synchronous invocation (for of Java EE applications). The message is published to a pre-determined JMS topic and is available for all registered subscribers to the JMS topic for pick up.
2. The deployed version RIB SUB adapter on the Mixed Version end picks up the message from the JMS and converts the deployed version RIB XML message to a corresponding deployed version RIB object.
3. In the GA RIB message flow process, the deployed version RIB adapter invokes the subscribe PL/SQL API in the application database schema. This is the point in the flow that is altered for the Mixed Version Support.
4. PL/SQL Payload Mapper Runtime procedures are deployed in a database schema separate from the Mixed Version application database schema. It has all the required grants from the Mixed Version application database schema.
5. Mapper Runtime includes the adapting CONSUME APIs; each one has the same signature as the base APIs. The RIB deployed version adapter is configured to connect to these APIs. The RIB object is passed from the deployed version RIB adapter to the adapting CONSUME APIs (see **1** in the Simple Message Flow above). The RIB Object is the deployed version RIB Object.
6. The adapting CONSUME APIs in the Mapper Runtime schema invoke the Mapper Runtime procedure to convert the RIB object to and from the Mixed Version RIB object (see **2** in the Simple Message Flow above).
7. The Mapper Runtime procedure invokes the appropriate Functional Resolver if needed, as defined in the mapping metadata created by the Mapper UI tool (see **3** in the Simple Message Flow above). The Functional Resolvers reside in the Mapper Runtime database schema and resolve differences in mapping.
8. The Adapting CONSUME API invokes the Mixed Version CONSUME API with the converted object. The process continues as in the base release (see **4** in the Simple Message Flow above).

Mixed Version Publication Process

The following steps describe the Mixed Version Publication Process:

1. For PL/SQL Applications, publication is handled by a polling RIB adapter.
2. The deployed version RIB Adapter polls the Mapping Runtime adapting GETNXT API, each having the same signature as the base API's (see **1** in the Simple Message Flow above).
3. The adapting GETNXT API invokes the Mixed Version application's GETNXT API (see **2** in the Simple Message Flow above).
4. The Mixed Version GETNXT returns a RIB Oracle Object.
5. The Mapper Runtime adapting GETNXT passes the Object to the Mapper Runtime procedure for conversion (see **3** in the Simple Message Flow above).
6. The Mapper Runtime procedure invokes the appropriate Functional Resolver as needed, as defined in the mapping metadata created by the Mapper UI tool (see **4** in the Simple Message Flow above), and hands the converted object back to the Mapper Runtime adapting GETNXT.

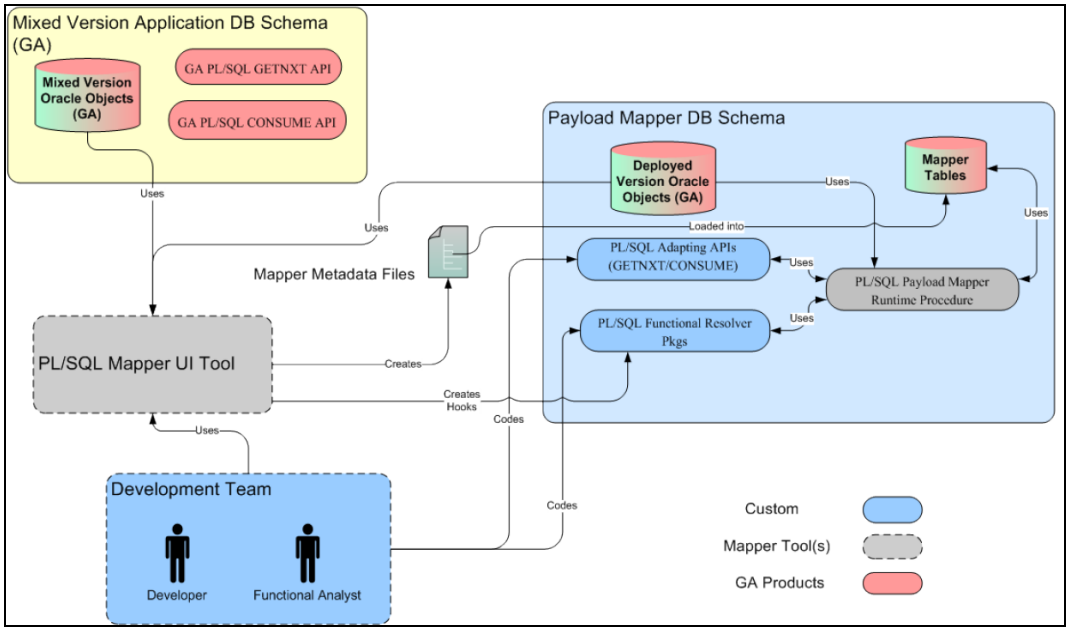
7. The Mapper Runtime Adapter GETNXT returns the deployed version RIB object to the calling deployed version RIB Adapter for publication to the JMS.

Tool Overview and Relationships

The PL/SQL Payload Mapper tool set comprises a design time and a runtime component. The design time tool is the PL/SQL Payload Mapper UI (or Mapper UI), and the runtime tool is the PL/SQL Payload Mapper Runtime (or Mapper Runtime). The Mapper Runtime uses the metadata output of the Mapper UI and the custom PL/SQL code required to complete the payload adaptation.

The Mapper UI assists developers in defining the mappings between RIB objects for the different versions and to create the metadata XML file. The metadata includes the rules to map the source payload to the target payload. It is then deployed with and used by the Mapper Runtime in adapting the source version of a payload to the target version payload. Because the Oracle Retail RIB is bi-directional (PUB and SUB), adaptation mappings are required where source and target are for both directions.

During the design activity, elements of the target database type with no direct mapping to source elements are identified by the Mapper UI. The tool provides the facility to create hooks for custom components to resolve the differences in the payloads. The hooks created by the Mapper UI are called Functional Resolvers. They are the custom PL/SQL code units designed and coded by the developer, based on business needs and the functional analysis of the application's use of the unmapped elements.



Tool Overview and Relationships

5

PL/SQL Payload Mapper Technical Specifications

This section describes technical specifications for the PL/SQL Payload Mapper.

Supported Database Versions

The following database version is supported:

Supported On	Version Supported
Oracle Database	12c Enterprise Edition

Supported RIB Versions

The PL/SQL Payload Mapper is supported for the following versions of RIB Objects:

Supported On	Version Supported
Oracle Retail Integration Bus	12.x, 13.x , 14.x and 15.0.0

6

PL/SQL Payload Mapper User Interface Tool

The PL/SQL Payload Mapping UI (Mapper UI) is designed and built as a stand-alone application. This section provides an overview of the Mapper UI and how to install and use it.

Prerequisites

The PL/SQL Mixed Version Application and PL/SQL Payload Mapper Runtime database schemas must exist and have the RIB Oracle Objects installed.

Because the mapping is at the Oracle Object level from the database schemas, it is critical to understand the RIB Objects and the relationship to RIB messages and message types.

> **Note:** See the *Oracle Retail Enterprise Integration Guide* for information about message families and types.
> Also see Appendix E: Examples of APIs and Objects (RWMS).

Task List

Task	Notes
Create the Payload Mapper database schema and install per the instructions in this guide.	See the section, Installation.
Have connection information available for the two schemas available, including the following: ▪ Machine details of the database, such as hostname, database port, and database SID. ▪ Application database schema details (user ID and password). ▪ Payload Mapping schema (user ID and password)	See the section, Recommended Deployment Topology.
Prepare a directory structure to save the metadata XML files created during the Mapping UI sessions.	PUB and SUB for both directions. See Mapping Metadata XML File.
Make sure that the JAVA_HOME environment variable is set and in the path for the user performing these tasks.	Java version 1.7.0_x with latest security updates.

Installation

The PL/SQL Payload Mapper UI (Mapper UI) is created as an Eclipse RCP plug-in. Installation of Eclipse is not required; the Mapper UI is a standalone tool distributed as a zip archive:

PlsqlPayloadMappingDesignUi15.0.0ForAllx.x.xApps_eng_ga.tar

Installation of the UI is an extraction of the tar file to a desired location, as follows.

1. Create a folder where the UI will reside (for example, mapperUI).
2. Copy the tar file to the new folder.
3. Extract the tar file.
4. The extraction creates a folder called PlsqlPayloadMappingDesignUi.
5. Browse to the directory PlsqlPayloadMappingDesignUi.
6. Extract PlsqlPayloadMappingDesignUi.jar.
7. The extraction creates a folder called PlsqlPayloadMappingDesignUi.
8. Under PlsqlPayloadMappingDesignUi, browse to the appropriate OS directory (Run the command "uname -a" to identify the appropriate operating system).
9. Go to the folder, PlsqlPayloadMappingDesignUi.
10. Launch the Mapper UI.

Configuration

To configure the properties for the Mapper UI, open the oracle.retail.rib.mapper_xxx.jar and edit the application.properties file.

The application.properties file contains the following editable configurations:

- newmapping.wizard.entertext, which denotes whether the PL/SQL Payload Mapper UI takes its source and target configuration values from the application.properties file or prompts the user for them. Valid values are True and False.
  - True disables the splash screens for source/target prompts from being shown.
  - False enables the splash screen for source/target to enter them.
- The source and target schema details.

> **Note:** The source and target depend on the direction of the message flow (PUB/SUB). Mappings are required in both directions.

```
# New and Open mapping wizard default values (used if
newmapping.wizard.entertext=TRUE)

source.driver=oracle.jdbc.driver.OracleDriver
source.dbhost=localhost
source.dbport=1521
source.dbsid=orcl
source.dbuser=<source db user>
source.version=12.0.9


target.driver=oracle.jdbc.driver.OracleDriver
target.dbhost=localhost
target.dbport=1521
```

```
target.dbsid=orcl
target.dbuser=<target db user>
target.version=15.0.0
```

> **Note:** The source and target depend on the direction of the message flow. Flows are in both directions, from the Deployed Version to the Mixed Version, and from the Mixed Version to the Deployed Version.

- application.default.locale, which is the locale of the design time tool.

Internationalization

The Mapper UI messages and window attributes, such as the title messages and buttons, are internationalized using the standard Eclipse framework.

Internationalization is at the following levels:

- Messages, prompts, and dialog texts.
- Buttons and menu provided by the Workbench.

In the tool plug-in executable (oracle.retail.rib.mapper_xxx.jar) is a folder within the jar file called locale. It contains the localized messages for various languages.

The class LocalePropertiesLoader refers to the application.properties file setting of application.default.locale for the default locale and loads the language bundle of the mentioned locale. The locale specific file contains localized text for messages, prompts, and dialogs.

As shipped, the MapperUI includes the following:

- application.default.locale = en
- file in locale folder = applicationText_en.properties

The buttons and menu contributions are localized using the properties file, plugin_<Locale>.properties. The file is located in the root level of the plug-in executable (oracle.retail.rib.mapper_xxx.jar). The Mapper UI has shipped the locale *en* properties.

To add a new language bundle so that it can be used by the Mapper UI, do the following:

- Add the new bundle into the locale directory in the MapperUI plug-in jar file.
- Add the new bundle plugin.properties to the root level in the MapperUI plug-in jar file.
- Set the value of the default locale in the application.properties file.

> **Note:** See Eclipse documentation for addition details.

PL/SQL Payload Mapper UI Usage

This section describes how to use the Mapper UI.

Launch the Mapper UI

To launch the Mapper UI, move to the directory where the Mapper UI was installed and to the subdirectory appropriate for the Operating System (OS) on which the Mapper UI resides.

For example, cd ~ /PlsqlPayloadMappingDesignUi/<OS>/PlsqlPayloadMappingDesignUi.

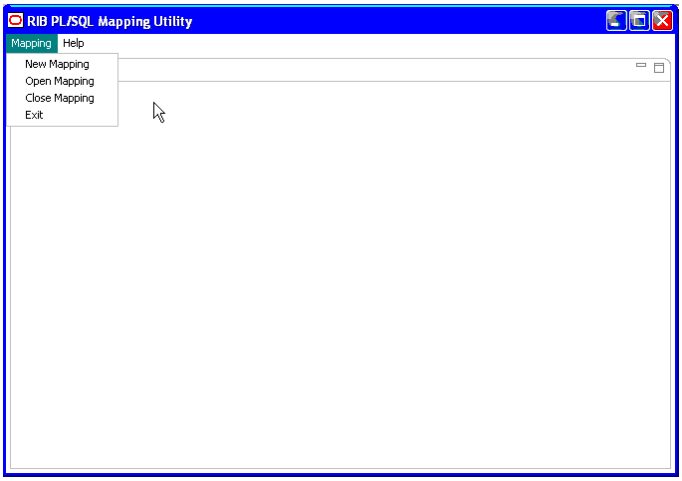
For Windows, click PlsqlPayloadMappingDesignUi.exe.

For Linux, sh PlsqlPayloadMappingDesignUi.

Note: Java must be installed and in the user path.

A splash screen is displayed.

The tool opens with a blank screen. From this blank screen, select the **Mapping** menu.



Create a New Mapping

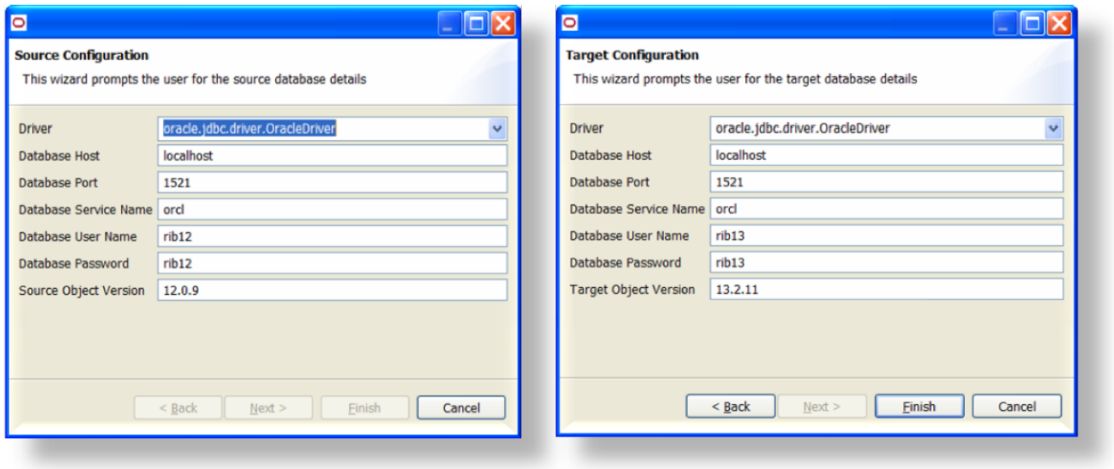
To create a new mapping, complete the following steps:

1. From the Mapping menu, select **New Mapping**.

   The Source and Target Configuration splash screen shows when the launcher is invoked, unless the application.properties file property newmapping.wizard.entertext is set to True.

   When the property is set toTrue, the wizards are not displayed. Instead, the user is prompted for the source and target database schema passwords.

> **Note:** See the section, Configuration.



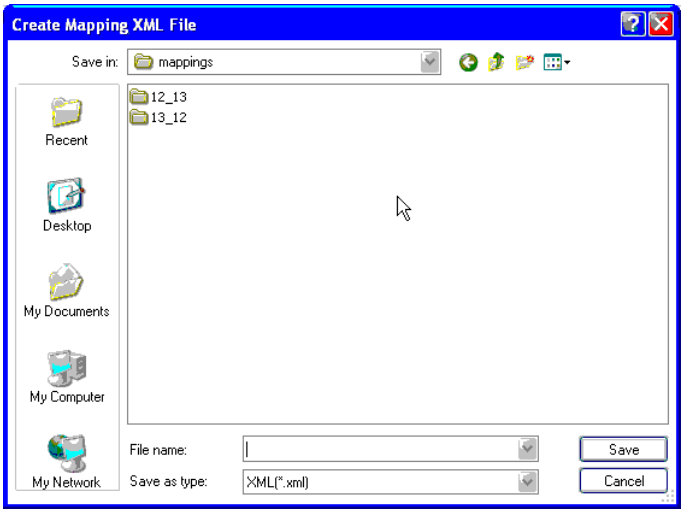
Source and Target Configuration

The source and target depend on the direction of the flow. Usually, flows are in both directions, from the Deployed Version to the Mixed Version and from the Mixed Version to the Deployed Version.

2. Click **Finish**.
3. Enter the name and location of the metadata XML file created as a result of the mapping.

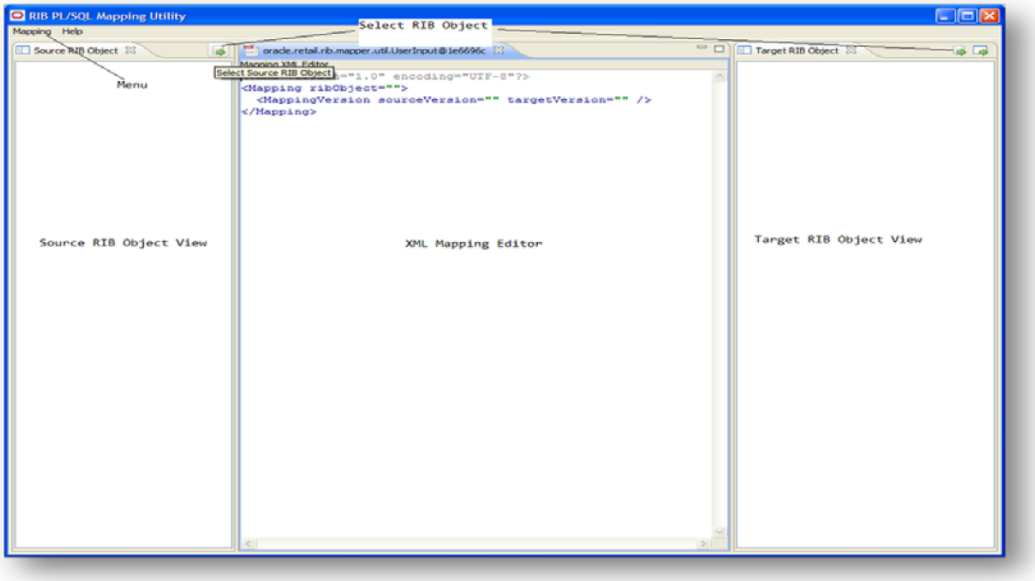
For example, RIB_AllocDesc_REC_15.0.0_13.2.0.xml.

> **Note:** See the section, Mapping Metadata XML File.
> See also the examples in these appendices:
> Appendix F: Examples of RWMS PUB Files
> Appendix G: Examples of RWMS SUB Files



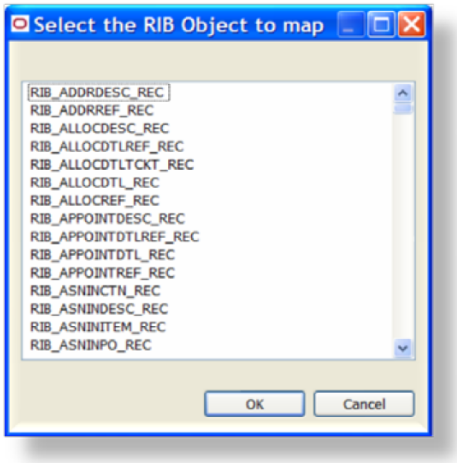
Note: After the file name is specified and the location specified, click Save to create the file. From that point, saving the XML file is handled programmatically based on user action throughout the mapping session (for example, automap, or drag and drop).

In the Workbench screen shown, type has not been selected. The left side view is for the source type; the right side is for target type. The various options and menu buttons are shown.



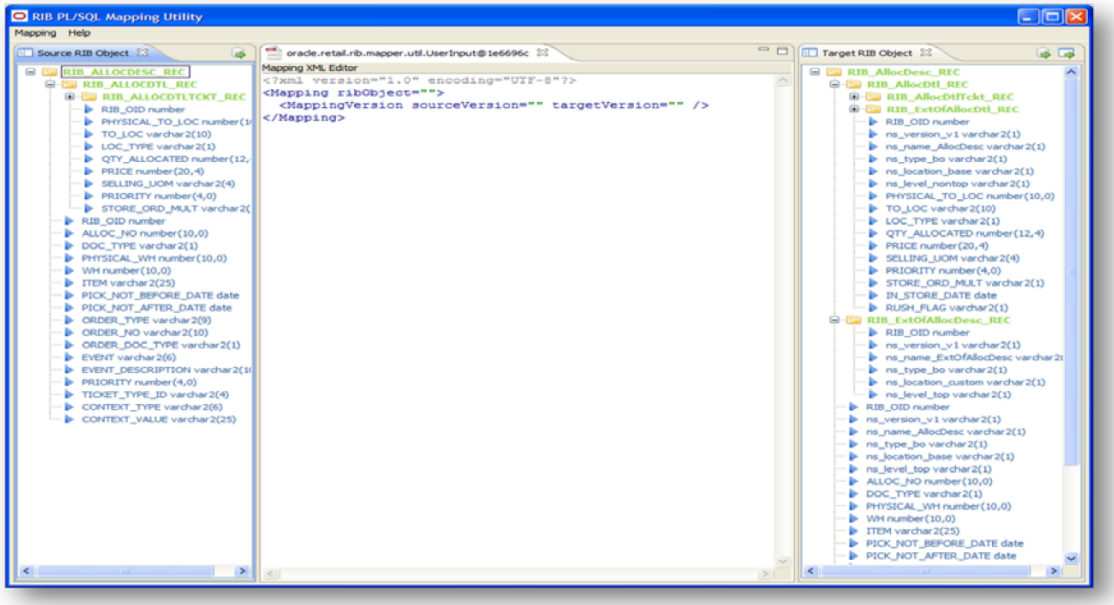
The Mapper UI Workbench

4. The button shown on the view toolbar includes database icon with the Select Source RIB Object tooltip. Click this button to open a selection dialog with database types from the source/target database. The selection screen is as shown below.



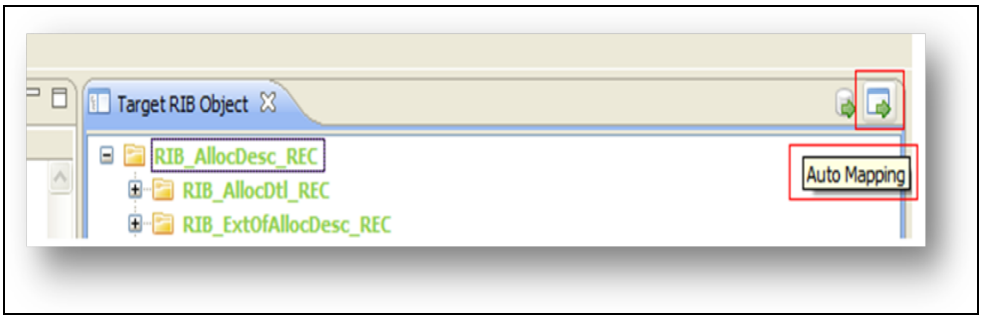
Type Selection Dialog

5. Database types are obtained by reading the data dictionary of the target or source schema. Select the source and target database type to map. Once these values are selected, the Workbench is shown as follows.



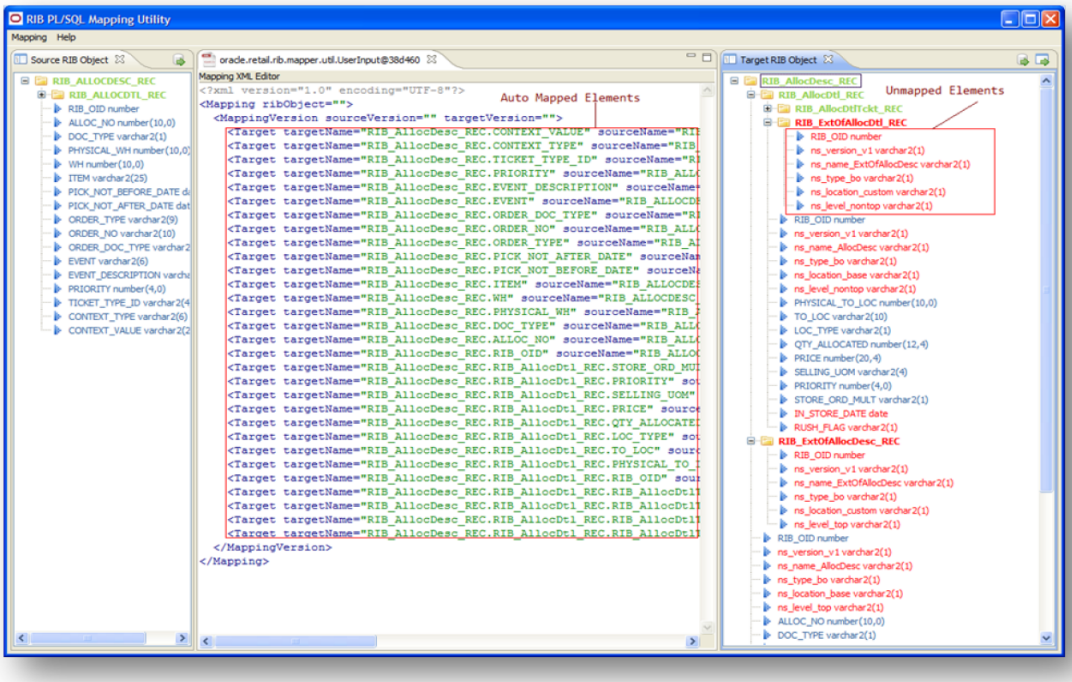
Mapper UI Workbench

6. The selected source and target type is shown in the views as a tree structure. From the target object view toolbar, select **Auto Map Action** to create a best fit map.



Auto Map Action

Auto mapping compares the target object with the selected source object and creates the best fit map. The mapping editor is refreshed to show the best fit map. The elements not mapped on the target view are changed to a different color.



Result of Auto Map Action

Once Auto Map is compete, the drag and drop feature is provided to map the elements that do not map.

> **Note:** The drag and drop feature is enabled only on an element-to-element basis.

7. Drag the source attributes to the target attributes. Mappings are created as the result of the action. Mapping should be between only elements belonging to the same hierarchy in source and target Objects.

For example:

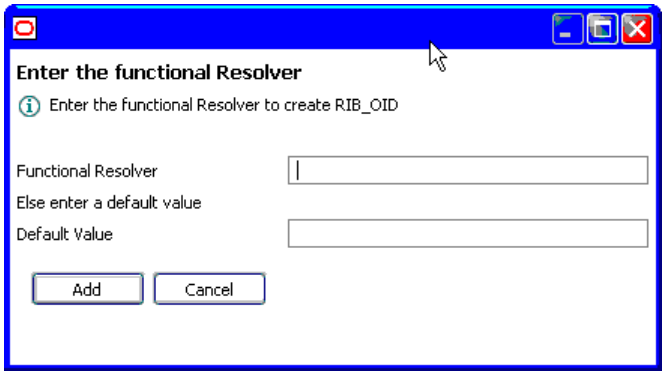
RIB_ASNInDesc_REC has an element ASNInPO_TBL(collection), and ASNInPO_TBL has an ASNInItem_TBL(collection).

In this case, elements of source ASNInItem_TBL should be mapped to elements of target ASNInItem_TBL only.

The viewer in the middle of the source and target view is an editor that displays the metadata XML as it is created.

8. After dragging and dropping, if the drop target does not match the dragged source (data type mismatch), the developer is prompted to enter Functional Resolver or a default value.

   Data type and Data length are validated on the value entered for the default value.



Note: See the section, Functional Resolvers.

Edit an Existing Mapping

To edit an existing mapping, do the following:

1. From the Mapping menu, select the Open Mapping. You will be prompted to select the mapping file to edit. Once the mapping file is selected, the source and target objects open automatically. The object names are picked up from the mapping xml file.
2. Edit the mapping using the drag-drop or the double click feature, as described in the section, Create a New Mapping.

Note: Re-selecting auto map is not advised. Doing so erases all previously created mappings from the file.

Mapping Metadata XML File

The metadata XML file defines the rules for creating a version of database type from another version of the same type. The XML file is used by the Mapper Runtime to create the target database type from a source type.

> **Note:** Manually editing metadata XML files is strongly discouraged. Elements must be in the same order as in the Oracle object. The Mapper UI maintains that order and adds all elements of the target object, mapped and unmapped. To manually edit the files, you must ensure all elements are present and accurately defined.

The XML file is formatted as follows:

Mapping XML Format

There is one XML file for each RIB object database type. For ease of use and clarity, naming the file name to match the RIB object as well as the version is recommended. This allows unambiguous naming for source code control and moving to the Runtime Host system.

For example, RIB_AllocDesc_REC_15.0.0_13.2.1.xml.

> **Note:** Object names are maintained in the XML file and used for processing. The name of the file is not used for processing.

The XML file content is the metadata loaded into the Mapper Runtime. This identifies the Oracle object for source and destination:

```
<Mapping targetRibObject="RIB_AllocDesc_REC" sourceRibObject="RIB_ALLOCDESC_REC">
```

The source and destination vary, depending on whether the mapping is for a PUB or a SUB. The source and target versions:

```
<MappingVersion sourceVersion="13.2.1" targetVersion="15.0.0">
```

The following identifies the element (node) of the Oracle Object being adapted:

```
<Element name="RIB_AllocDesc_REC" tableType="" recordType="RIB_AllocDesc_REC"
source="RIB_ALLOCDESC_REC" functionalResolver="" unmapped="false">
```

For each attribute in the node, there is a line identifying the element, the source element mapped to it, the functional resolver to be called at runtime (if there is one), a default (if specified), and the identifier set by the Mapper UI to indicate that it has been mapped (true or false):

```
<Attribute name="ALLOC_NO" source="RIB_ALLOCDESC_REC.ALLOC_NO"
functionalResolver="" default="”” unmapped="false" />
<Attribute name="RIB_OID" source="" functionalResolver="" defaultValue="10" />
```

Note: Mapper Runtime checks for the default value and inserts it. The default value takes precedence over the call to a Functional Resolver.

7

PL/SQL Payload Mapper Runtime Tool

The PL/SQL Mapper Runtime uses payload (Oracle object) as source values and maps them to a target payload (Oracle object). The source and target depend on the direction of the message flow (PUB/SUB).

Mapping is based on the Mapping XML, which is created using the mapping utility discussed in the previous section. Mapping may be direct, one for one (like to like). Or gaps may be detected in the design phase that either adds a default value or identifies a Functional Resolver to be called.

> **Note:** See the section, Simple Message Flow and Processes.

Mapper Runtime Components

The Runtime Mapper comprises the following components:

- Adaptive APIs (custom)
- Runtime Mapper (base tool)
- Functional Resolvers (custom)
- Support scripts and PL/SQL packages (examples provided)

Mapper Runtime Adaptive APIs

The custom adapting PL/SQL APIs, CONSUME and GETNXT (which correspond to the Mixed Version schema), are installed into the Mapper Runtime schema. These adapting APIs (also known as pseudo APIs) must be created by the developer. The signature on each API must be identical to the signature on the corresponding Mixed Version API. These APIs make the call to the Mapper Runtime and invoke the actual CONSUME /GETNXT APIs in the application database schema.

> **Note:** See the section, Simple Message Flow and Processes. See also the appendices at the end of this guide.

Mapper Runtime Procedure

The Mapper Runtime procedure is the mapping engine and a PL/SQL package installed into the Mapper Runtime Schema. The transformations it performs are controlled by the metadata mappings between specific source and target objects. The metadata is created by the Mapper UI and stored in files that are loaded into the MAPPING_DETAIL table.

> **Note:** See the section, PAYLOAD_MAPPER PL/SQL Procedure.

Functional Resolvers

Functional Resolvers are custom program units that may have to be created during the mapping process to resolve a specific mapping difference that functional analysis has determined as a gap that must be handled by something more than a default.

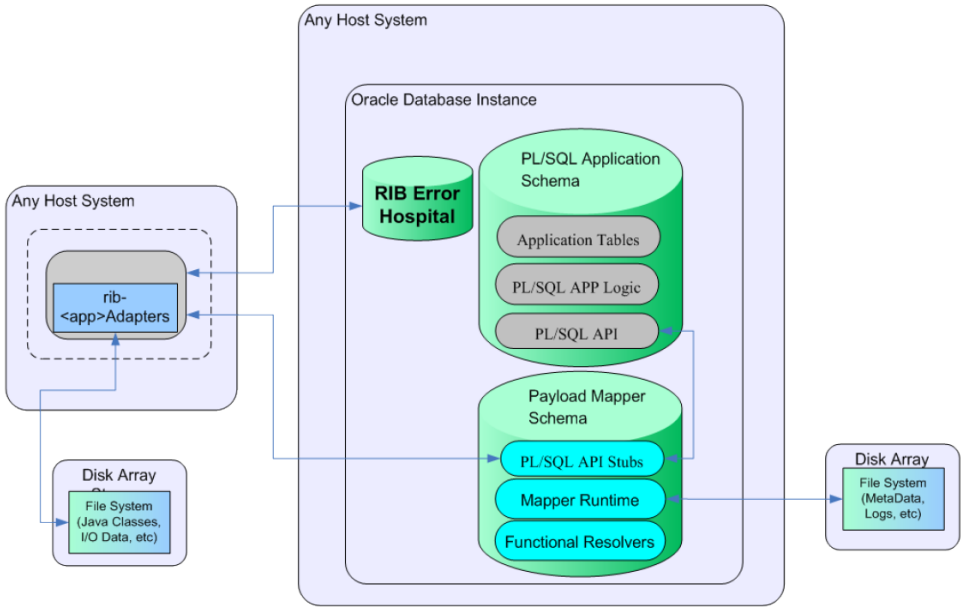
Note: See the section, Functional Resolvers.

Support Scripts

Support scripts are sample scripts for performing many of the basic tasks required to install and maintain Mapper Runtime.

Note: See the section, Packaged PL/SQL Payload Mapper Runtime PL/SQL Procedures

Recommended Deployment Topology



The Runtime Mapper is co-located in the same database instance as the installed GA Mixed Version of the application mapped to the deployed RIB version.

- Mixed Version Main Schema -- Contains the GA application. Grants from this schema for all objects are given to Mapper Runtime schema.
- Mapper Runtime Schema -- Contains all deployed RIB Version RIB object types, as well as tables, packages, and procedures for PL/SQL runtime, including the adapting (pseudo) APIs for GETNXT and CONSUME calls.

PL/SQL Payload Mapper Runtime Installation

Payload Mapper installation has many pre-requisites and dependencies that must be understood by the customer's database administrators, system administrators, application server administrators, and Oracle Retail application teams.

Prerequisite Tasks

Task	Notes
Create the Mapper Runtime database schema	*<mapper runtime schema>*
Grant the Mapper Runtime schema ▪ Connect ▪ Resource ▪ Create Any Directory ▪ Grant Select on v_$session	
Install the Mixed Version Application.	Follow installation and deployment instructions for the application.
Create or identify a user that can hold the directory structure on the database host system to place the Mapper Runtime metadata, adapting APIs, and logging files.	The user location must have appropriate read and write permissions.
In the Mapper Runtime Schema, install the version of RIB Oracle Objects that matches the version of RIB that was deployed.	The Mapper Schema also requires that the version of the installed RIB Objects match the version of the deployed RIB. The grants scripts give permission on the Mixed Version Objects to the Mapper Runtime.

Installation Process

The following tables describe the tasks required to complete the installation process:

Initial Installation Summarized Task List

Task	Notes
Place the distribution into the directory structure created in the pre-requisites and untar the contents.	> tar xvf PlsqlPayloadMapperEngine15.0.0ForAllx.x.xApps_eng_ga.tar
Create grants on the Mixed Version application schema objects to the Mapper Runtime schema owner.	An example script is provided in the distribution, GrantPrivileges.sql. **Note:** It is important to re-run the grants whenever there is a change to the Objects in the Mixed Version schema; otherwise, the Runtime fails with errors when it attempts to use the base RIB Objects. See the section, Packaged Support Scripts and PL/SQL Procedures.
Create the database entry for the external file system location to hold logs and metadata.	An example script is provided in the distribution, CreateDirectory.sql. See the section, Packaged Support Scripts and PL/SQL Procedures.
Create the tables used by the Mapper Runtime.	An example script is provided in the distribution, CreateTables.sql.
Create the triggers needed.	An example script is provided in the distribution, CreateTriggers.sql

Task	Notes
Run the CreateMappingErrorLog script to create the logger.	The package is provided in CreateMappingErrorLog.sql.
Create the Mapper Runtime procedure.	The package is provided in Create_Payload_Mapper.sql. **Note:** The custom Adapter APIs must be compiled into the Mapper schema before the Create_Payload_Mapper.sql and ConvertRoutinginfo.sql can be run.
Run the ConvertRoutingInfo procedure.	The package is provided in ConvertRoutingInfo.sql.

Mapping Runtime Files and Procedures

Task	Notes
Move the custom Adapting APIs to the directory structure created in the prerequisites. Install and compile them in the Mapper Runtime schema.	These are the custom APIs that must be created. You must create a custom adaptive API in the Mapper schema for every RIB API (SUB or PUB) used by the installation (for example, RWMS). See Appendix A: Examples of Adapting APIs
Move the custom Functional Resolvers to the directory structure created in the prerequisites. Install and compile them in the Mapper Runtime schema.	
Move the metadata XML files created by the Mapper UI to the directory structure created in the prerequisites. Execute the cachemapping tool to install them into the Mapper Runtime schema.	See the section, Cachemapping Usage.

Configure RIB to the Mapper Runtime

Task	Notes
Install the deployed RIB error hospital for the Mixed Version application.	This must be the hospital for the deployed release of RIB, not the Mixed Version.
Configure the deployed RIB adapters to point to the Mapper Runtime Adapting APIs and to the deployed version of RIB's error hospital installed in the Mapper Runtime schema.	(See documentation for the deployed release of the Oracle Retail Integration Bus.) This may require a new installation of the deployed RIB adapters (ribforxxx) that correspond to the name of the mixed version applications but match the version of the deployed RIB. If a deployed version of the ribforxxx exists, make a simple configuration change in rib-home to point to the Adapting APIs in the Mapper schema.

Packaged PL/SQL Payload Mapper Runtime PL/SQL Procedures

Task	Notes
Create_Payload_Mapper.sql For example, >sqlplus <*mapper runtime schema*> SQL> @Create_Payload_Mapper.sql <*mixed version schema*> <*mapper runtime schema*> 15.0.0 12.0.9	The PL/SQL Mapper Runtime PL/SQL package. The arguments are Mixed Version schema name, Runtime Mapper schema name, Mixed Version and deployed version. These versions should be same as what has been provided while creating the XML Mappings (for example, 15.0.0 and 12.0.9). **Warning:** If at runtime the release numbers do not match, this error is logged: "ORA-01403: no data found."
ConvertRoutingInfo.sql/ For example, >sqlplus <*mapper runtime schema*> SQL> @ConvertRoutingInfo.sql <*mixed version schema*> <*mapper runtime schema*>	The procedure to map the RIB Routing Information Objects in the Payload Mapper database schema. The arguments are Mixed Version schema name and Runtime Mapper schema name.

Packaged Support Scripts and PL/SQL Procedures

Task	Notes
GrantPrivileges.sql For example, >sqlplus <*mixed version schema*> SQL> @ GrantPrivileges.sql <*mapper runtime schema*>	This example script grants permissions to the Payload Mapper database schema in the RIB Oracle Objects of the Application database schema. **Note:** It is important to re-run the grants whenever there is a change to the Objects in the Mixed Version schema; otherwise, the Runtime fails with errors when it attempts to use the base RIB Objects.
CreateTables.sql For example, >sqlplus <*mapper runtime schema*> SQL> @CreateTables.sql	The script creates the required Mapper Runtime tables in the Payload Mapper database schema.
CreateTriggers.sql For example, >sqlplus <*mapper runtime schema*> SQL> @CreateTriggers.sql	The script creates the required Mapper Runtime triggers in the Payload Mapper database schema.
CreateSequences.sql For example, >sqlplus <*mapper runtime schema*> SQL> @CreateSequences.sql	The script creates the required Mapper Runtime sequences in the Payload Mapper DB Schema.
InsertSeedData.sql For example, >sqlplus <*mapper runtime schema*> SQL> @InsertSeedData.sql	The script inserts into the OBJECT_FILE_MAPPING table name/value pairs to associate a RIB object to its logging file. **Note:** This script may require adjustments to support RMS versus RWMS, as well as any customizations.

Task	Notes
CreateDirectory.sql For example, >sqlplus <*mapper runtime schema*> SQL> @CreateDirectory.sql /stage/Payload_Mapper/mapper_dir	The script creates a directory in Payload Mapper database schema that points to an external file system location. This directory is where the Mapping metadata XML file resides. The directory must be passed as a parameter, and it must have appropriate read and write privileges.
CreateMappingErrorLog.sql For example, >sqlplus <*mapper runtime schema*> SQL> @CreateMappingErrorLog.sql	The script compiles the procedure to log errors.

Installation and Usage: Database Caching of Mapping Metadata

To improve performance, Mapping Runtime must not access the file system for every message conversion. A tool has been developed to read the file system for the Mapping UI metadata XML files, parse the files, and cache the mappings into database tables.

The Mapping Runtime reads the database table to obtain the mappings. This prevents the mapping utility from accessing the file system again and again. The same PL/SQL tool can be used to reload/refresh a mapping XML.

Task	Notes
The java executable should be in the PATH environment variable of the user. CLASSPATH is set by the script.	Java 1.7.x with latest security updates.
Execute cachemapping.sh/cachemapping.bat.	The script to cache the PL/SQL Payload Mapping metadata XML in the Payload Mapper database schema.

Cachemapping Usage

The script requires the following input:

- JDBC URL to connect to the utility schema. For example, jdbc.oracle.thin:@<DB machine name>:<DB PORT>:<DB SID>.
- Mapper Runtime schema user name.
- Mapper Runtime schema password. (The user is prompted for it.)
- User name or login ID (also prompted). The operator ID is used for audit columns, such as Created By and Modified By in Mapper Runtime tables.
- Directory where the metadata XMLs are located.
- Mapping XML name: This is an optional argument. If the name is provided, only that file is uploaded and refreshed in the database. This argument can be used to refresh selective mappings if a mapping XML is later changed.

Database Tables

This section generally describes the database tables used by the Mapper Runtime. The tables are shown here for representation; actual structure may change with releases.

MAPPING_HEADER: This table is the master table for the mappings. It stores the Object Name along with the Source and Target versions. MAPPING_ID is the primary key.

Column Name	Description
MAPPING_ID	Primary Key. This is just a sequential number
OBJECT_NAME	The name of the message type.
SOURCE_VERSION	The version of the message that needs to be converted.
TARGET_VERSION	The version of the message that needs to be created.
CREATE_BY	Audit column
CREATE_DATE	Audit column
MODIFY_BY	Audit column
MODIFY_DATE	Audit column

MAPPING_DETAIL: This table stores all the mappings for all the necessary objects. MAPPING_ID is the foreign key that refers to the MAPPING_HEADER table.

Column Name	Description
MAPPING_DETAIL_NO	Primary Key. Populated by sequence SEQ_MAPPING_DETAIL
MAPPING_ID	References Mapping_Header.
ELEMENT_TYPE	Indicates whether it is an attribute or element from XML. This is for future use.
TARGET	The target flattened element. For example, RIB_ItemDesc_REC. ITEMUPCDESC_TBL. ITEM_NUMBER_TYPE.
SOURCE	The source element. For example, RIB_ITEMDESC_REC.RIB_OID.
FUNC_RES	If any functional resolver is suggested in the mapping, the same is stored here.
SUP_TYPE	The base object from which the target child collection is made.
NEST_TABLE	The actual collection for the element.
DEFAULT_VALUE	Default value, if provided in the Mapping.
CREATE_BY	Audit column
CREATE_DATE	Audit column
MODIFY_BY	Audit column
MODIFY_DATE	Audit column

MAPPING_ERROR_LOG: This table stores all the unhandled errors.

Column Name	Description
SEQ_NO	Primary Key. This is a sequential number
ERROR_TS	Date when the error occurred
ERROR_DESC	Description of the error. Most likely this stores SQLERRM

UTILITY_SYSTEM_OPTIONS: This table stores Runtime name/value options.

Column Name	Description
CONFIG_ID	Option Name
CONFIG_VALUE	Option Value

OBJECT_FILE_MAPPING: This table stores lookup values for files used by the Mapper Runtime Logger.

Column Name	Description
OBJECT_NAME	RIB Object Name
FILE_NAME	External logging file name

Purging

The **MAPPING_ERROR_LOG** table stores only the errors related to file handling. Once the Mapper directory where the log files and the other artifacts are stored is in place and permissions are granted, the growth of this table is not expected to substantial.

Oracle Retail does recommend periodic checking of the table deleting all rows which are more than 90 days old.

PAYLOAD_MAPPER PL/SQL Procedure

PL/SQL Mapper Runtime transforms an object from one version to another. It transforms the object of source version as input and returns the transformed object of the target version as output. Transformation is based on mappings between source and target object stored in the MAPPING_DETAIL table. The target object element value could be the source object element itself; or it could be a default value or a functional resolver.

The PAYLOAD_MAPPER procedure is the Mapper Runtime engine and contains one overloaded function, MAP_OBJECT_SOURCE_TO_TARGET. This function is called from each of the Adapter APIs created in the Mapper Runtime schema. MAP_OBJECT_SOURCE_TO_TARGET uses the source RIB_OBJECT as input and transforms it into target RIB_OBJECT.

For subscriptions, the MAP_OBJECT_SOURCE_TO_TARGET is called first in the Adapter API. Then the actual CONSUME call is made, using the transformed object. For publishing, the actual GETNXT call is made first; then MAP_OBJECT_SOURCE_TO_TARGET is called to transform the object.

During transformation, the target object element value can be the source object element; or, if it is missing, MAP_OBJECT_SOURCE_TO_TARGET inserts a default value or a call to a Functional Resolver. These decisions are made in the Mapper UI and in the metadata store in the MAPPING_DETAIL table.

MAP_OBJECT_SOURCE_TO_TARGET Execution Flow

The following steps comprise the MAP_OBJECT_SOURCE_TO_TARGET execution flow:

Note: See Appendix D: Mapper Runtime Sequence Diagram.

1. Call GetRibObjectType to get the actual underlying object type. For example, RIB_OBJECT may store RIB_AllocDesc_Rec.

2. Open the log file defined in OBJECT_FILE_MAPPING for logging.
3. Call GetXMLMappings to get the mappings defined in MAPPING_DETAIL table. If no mappings are defined, "No Data Found" is logged in the log file. The mappings are populated in a collection defined at the Package Level (tbMap).
4. Convert the RIB_OBJECT into the actual underlying object type.
5. Log the values of the Source Object into the log file (only if LOGGING = Y in the UTILITY_SYSTEM_OPTIONS table).

   Call RECUR, passing in the first element position in Mapping_Detail table. It is always 1. Also pass the hierarchy, which is also 1.

   a. Get the values of the object at first level and log it into a file.
   b. Get the position of the first element of the object or collection at the next level.
   c. Call RECUR again, passing in the position and level of hierarchy.
   d. Get the values of the object at this level and log it into a file.
   e. Get the position of the first element of object or collection at the next level or the position of the first element of a different object or collection at the same level.
   f. Call RECUR again, passing in the position and level of hierarchy.
   g. Get the values of the object at this level and log it into a file.
   h. Perform the above two steps repeatedly until all child objects are covered and values are logged in the file.
6. Call InitObj to start transformation of the object.
   a. Using a CLOB, start creating constructor for the target object.
   b. Start traversing through tbMap to get mappings.
   c. If the element is at level 1, enter the value in the CLOB based on mappings.
   d. If it is a direct mapping, call ConstructSourceValue to get the source column.
   e. To use a default value, enter the default value.
   f. To use a functional resolver, call GetFunctionalResolver to get the actual function with parameters.
   g. If the element is at a level higher than 1, call CreateChildObject. (This element will be a part of the collection within the main object.)
      i. Call RecursiveChildObjects passing the current position of tbMap.
      ii. Get the count of the collection and get the number of elements in the collection.
      iii. Create two loops, where the outer one is for count of the collection and the inner one is for the number of elements in the collection.
      iv. For each element of target object find the source per mapping and enter the correct value in the CLOB.

         If it is a direct mapping, use the source element directly.

         To use a default value, enter the default value.

         To use a functional resolver, call GetFunctionalResolver to get the actual function with parameters.

         If an element is found at a higher level, or if the element is at a lower level than the current level, execute the steps in Item "g" until all the elements are mapped.
   h. Run the constructor so that we get the transformed object.
7. Log the values of the Target Object into the log file (only if LOGGING = Y in the UTILITY_SYSTEM_OPTIONS table).

Call RECUR, passing in the first element position in Mapping_Detail table. It always will be 1. Also pass the hierarchy, which is also 1.

a. Get the values of the object at first level and log it into file.
b. Get the position of first element of object or collection at the next level
c. Call RECUR again, passing in the position and level of hierarchy.
d. Get the values of the object at this level and log it into file.
e. Get the position of first element of object or collection at the next level, or the position of the first element of a different object or collection at the same level.
f. Call RECUR again, passing in the position and level of hierarchy.
g. Get the values of the object at this level and log it into file.
h. Perform the above two steps repeatedly until all child objects are covered and the values are logged in the file.

8. Close the log file.
9. Return the target object.

Error Handling

The Adapting (pseudo) APIs are the point of invocation from the RIB Adapter. These APIs interact with the CONSUME and GETNXT APIs.

Exceptions raised by the actual APIs are forwarded to the RIB Adapter accordingly.

Any fatal exceptions raised by the PL/SQL Mapping Runtime are mapped to the errors already handled by the RIB adapter. Unless a fatal exception is encountered, the conversion of the RIB object from one version to the next is not stopped.

Errors related to file handling are logged in the MAPPING_ERROR_LOG table; conversion of the RIB object continues.

Logging

The flow of execution and errors are logged in the file, rib_mapper_<RIB MessageFamily.log>. There is one log file for each RIB message family.

> **Note:** Log File maintenance is not automated. A customer process to watch, archive, and purge must be built.

Logging is globally controlled by a setting a name/value pair in the Mapper Runtime UTILITY_SYSTEM_OPTIONS table in the Mapper Runtime schema.

During installation, the name/value pair setting for logging is enabled, where LOGGING = Y and a seeding script is run to associate the RIB_OBJECT with the logging file:

```
Insert into OBJECT_FILE_MAPPING (OBJECT_NAME,FILE_NAME) values
('RIB_ALLOCDESC_REC','rib_mapper_Alloc.log');
Insert into OBJECT_FILE_MAPPING (OBJECT_NAME,FILE_NAME) values
('RIB_ALLOCREF_REC','rib_mapper_Alloc.log');
Insert into OBJECT_FILE_MAPPING (OBJECT_NAME,FILE_NAME) values
('RIB_APPOINTDESC_REC','rib_mapper_Receiving.log');
Insert into OBJECT_FILE_MAPPING (OBJECT_NAME,FILE_NAME) values
('RIB_APPOINTREF_REC','rib_mapper_Receiving.log');
```

> **Note:** See Appendix B: Sample Log File.

8

Functional Resolvers

Functional Resolvers are custom program units that must be created during the mapping process to resolve a specific mapping difference. The PL/SQL Payload Mapper does not have pre-built functional resolvers.

The functional gaps among Oracle Retail Enterprise releases can vary widely and must be determined through careful impact analysis at functional and operational levels.

Simple element to element differences are only part of the analysis required. Changes behind the messages and elements to support new or enhanced Enterprise functionality among the releases also require impact analysis. Keep in mind that the more complex the Functional Resolver becomes, the larger the impact on message throughput when calling it for every message.

Use Cases Examples

The following are use case examples:

- The source version of an Object does not have simple attribute X, which exists on the Y table. The target version of the object does have attribute X. The Functional Resolver will enrich the information by selecting values from other sources (such as other tables in the application schema) and adding them to the target version of the object.
- The target version of the Object uses XYZ information to support new functionality. The source version of the object does not have XYZ information. There is no convenient base place from where to select this information in the source system, and a simple default value is not adequate. The Functional Resolver can either create or look-up a simple value for XYZ, or it can look up values for XYZ from a custom implemented table in the source system.

General Recommendations

Carefully consider the value in adding a functional resolver: Is the new information really necessary, and will it drive value in the target system? Or is inserting a default value or null value valid (and not too functionally limiting) for the target system?

Because Functional Resolvers exist at the element level, every message will have a call to the Functional Resolver package, which obviously introduces additional performance impact to the message flow.

RIB Object Customization

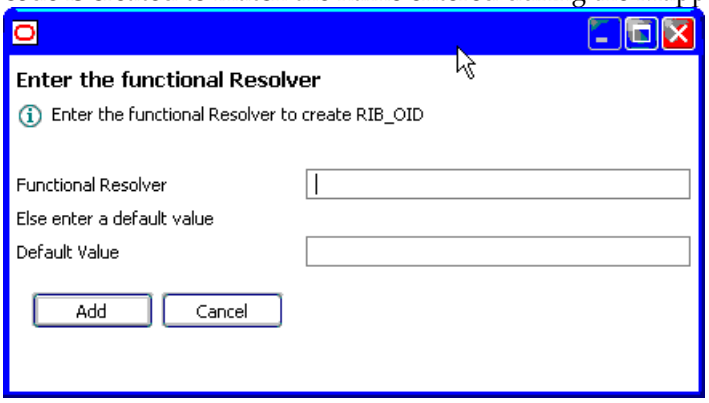
The PL/SQL Payload Mapper Runtime supports the customization and extension of RIB as outlined in the *Oracle Retail Integration Bus Implementation Guide*.

The customization process must follow the guidelines for the deployed version of RIB and the Mixed Version of the RIB Objects. When the Deployed Objects are installed in the Mapper Runtime schema, the corresponding Mixed Version RIB objects must be installed in the Mixed Version base schema.

> **Note:** It is important to re-run the grants whenever there is a change to the Objects in the Mixed Version schema; otherwise, the Runtime fails with errors when it attempts to use the base RIB Objects.

How to Create a Functional Resolver

The tool designed with functional resolution at the element level. So for an unmapped element between source and target, the Mapper Runtime expects that the metadata created by the Mapper UI contains a default, or a Functional Resolver package to call. The developer and analyst must work together on requirements and to make sure the code is created to match the name entered during the Mapper UI session.



Enter the Functional Resolver screen

Naming Convention

For clarity, it is recommended that a common naming convention for all Functional Resolvers is used.

For example, GET_<element_name>(param1, param2,…) RETURN <data type>.

Basic Concepts

The following are basic concepts concerning Functional Resolvers:

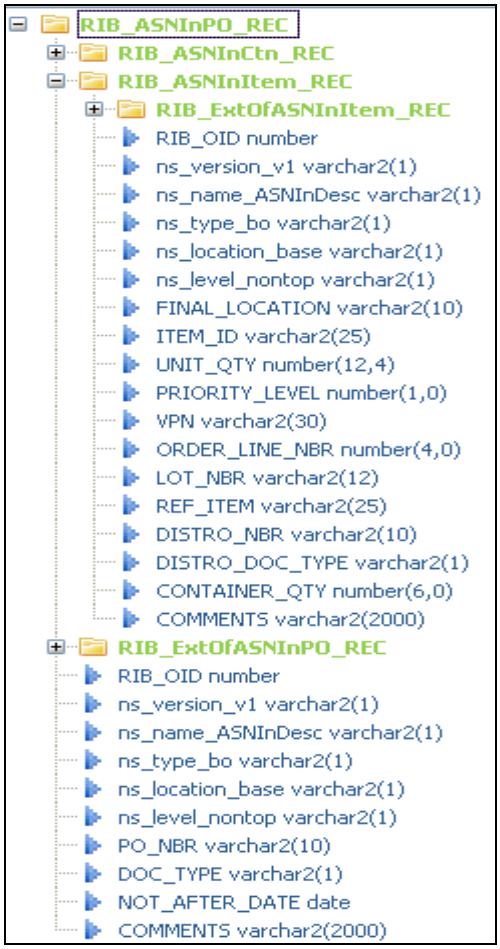
- They are PL/SQL functions that may or may not have input parameters.
- They should have built in exception handlers so that all exceptions are caught, as well as an acceptable value that can be substituted when an object or collection element is returned.
- NULL can be returned for any exceptions.
- Parameters containing the full hierarchy should be from the same collection or object for which the Functional Resolver is written.

Examples

RIB Object Types may not be simple objects. They may contain collections as elements, and these collections may contain collections within them. So the developer must take care to create a Functional Resolvers with parameters from the same hierarchy level.

Collections

If a Functional Resolver is needed for the Object RIB_ASNInPO_REC for the FINAL_LOCATION element in the collection RIB_ASNIn_Item_Rec, the parameters used should be from the same collection. See the collection example below.



Collection Example

Using the naming convention suggested, a sample Functional Resolver created in the Mapper UI is as follows:
Get_FinalLocation(RIB_ASNInPO_REC. RIB_ASNInItem_Rec.DISTRO_NBR)
Constants can also be passed with this function:

```
Get_FinalLocation(RIB_ASNInPO_REC.RIB_ASNInItem_Rec.DISTRO_NBR, 100)

A source element (p_distro_nbr) is used to resolve another element
(FINAL_LOCATION)

FUNCTION Get_FinalLocation(p_distro_nbr VARCHAR2) RETURN VARCHAR2

FUNCTION Get_FinalLocation(p_distro_nbr VARCHAR2, p_const NUMBER default 100)
RETURN VARCHAR2
```

Note: See Appendix C: Examples of Functional Resolvers.

9

Integration Testing

The use of the Payload Mapper means that an additional sub-system is inserted into every RIB message family. For every message type new logic is executed. This custom sub system has not been tested by Oracle Retail.

Oracle Retail always recommends that every deployment plan includes Integration and Performance testing. With the addition of the Payload Mapper to a deployment, it is critical that full integration testing is part of the deployment planning.

Example Compatibility Testing Outline

For inbound messages (those coming from the Deployed RIB), build and execute a test suite of messages of all message families and message types. Verify that the Deployed RIB Version (such as 12.0.9) messages are successfully converted to the Mixed Version (such as 15.0.0) RIB Objects and are consumed by the Mixed Version application successfully.

This testing can be done through the RIB tools that allow messages to be injected directly to the various JMS topics.

> **Note:** See "Testing the RIB" in the *Oracle Retail Integration Bus Operations Guide*.

To test the outbound messages (those that originate with the Mixed Version application), testing requires using the application to generate business events that will be published to RIB. As the messages are published, use RIB tools to capture the messages at the topics and verify them. Then allow the messages to flow to a test installation of the deployed applications to validate that functional gaps have been addressed.

Performance Considerations

The impact on performance must be considered when deciding whether to use the Payload Mapper. Flows perform at something less than a non-mapped flow.

Therefore, Oracle Retail recommends applying the transformation/mapping as close to application end points as possible. The location of the PL/SQL Mapper Runtime within the same Oracle database instance as the Mixed Version application will help to mitigate the impact.

Oracle Retail also recommends a Performance Test as part of every deployment plan. Even if formal testing is not planned, the use of RIB tools and processes discussed in the *Oracle Retail Integration Bus Operations Guide* can measure the relative performance of the RIB subsystem and diagnose bottlenecks. For a deployment with a Mapper approach, performance testing is especially encouraged.

For the PL/SQL approach, mapping occurs in the Oracle database instance. Testing the performance of the RIB message flows adds a measurement component that is neither obvious nor distinct when using the tools and methods described in the *Oracle Retail Integration Bus Operations Guide.* Mapping activities take place as the application is in use, so database side tools and measures common to PL/SQL application tuning must be used.

Note: See "Performance Considerations and Testing the RIB" in the *Oracle Retail Integration Bus Operations Guide.*

10

Java Payload Mapper

The purpose of the Java Payload Mapper is to provide a standard, consistent way to assist in the custom development of Java side solutions to mixed version integrations that use RIB, and do it in a way that avoids situations that break functionality of the base RIB and the Oracle Application products. One of the primary goals is to avoid making future releases of Oracle Retail based products difficult or impossible to accept.

RIB Payload Versions

The Java Payload Mapper is tested with the following versions of RIB payloads:

- 11.1.0
- 12.0.9
- 13.0.2
- 13.1.0
- 13.2.0
- 14.1.0
- 15.0.0

Install a Web Service in WebLogic

Take the following steps to deploy the Payload Mapper utility to an Oracle WebLogic Application Server as a Web service.

Prerequisites

1. Download `JavaPayloadMapper15.0.0ForAllx.x.xApps_eng_ga.tar.`
2. Extract the downloaded archive.
   `tar –xvf JavaPayloadMapper15.0.0ForAllx.x.xApps_eng_ga.tar`
3. The `payloadmapper-service-<version>.ear` is located in `payload-mapper/webservice` directory.
4. The installation and base configuration of the Oracle WebLogic Server is beyond the scope of this document. Work with the Application Server Administration team to determine the physical and logical placement of the Payload Mapper component within the WebLogic Server deployment.

Deploy Payload Mapper Web Service

Using the WebLogic Server Administration Console, complete the following steps:

1. Login to WebLogic Administration Console.
2. Navigate to the **Deployments** page.
3. Click **Install**. Click **Lock & Edit**, if needed.

   Note: If the application has already been installed, see "Redeploy the Application".

   The Locate deployment to install and prepare for deployment page is displayed.

Follow the instructions to locate the `payloadmapper-service-<version>.ear` file.

4. Select upload your file(s).
5. On the Upload a Deployment to the admin server page, use **Browse** to locate the `payloadmapper-service-<version>.ear` file in the `<PM_HOME>/payload-mapper/webservice` directory.
6. Select the `payloadmapper-service-<version>.ear`.
7. Click **Next** and move to Choose targeting style page.
8. Select **Install this deployment as an application**.
9. Select a target to deploy the application (for example, payloadmapper-server).
10. Click **Next** and move to **Optional Settings** page.
11. Click **Next** and move to **Review Your Choices** page.
12. Select **No, I will review the configuration later** and click **Finish** to deploy the application.
13. For production mode, click **Activate Changes**.
14. Select the `payloadmapper-service` and click **Start --> Servicing all requests**.

Verify Payload Mapper Web Service

To verify the Payload Mapper Web service, complete the following steps:

1. Navigate to the **Deployments** page.
2. Locate the **payloadmapper-service** on the Summary of Deployments page.
3. Click **payloadmapper-service**, to move to the Settings for payloadmapper-service.
4. Select the **Testing** tab.
5. Expand the PayloadMapperService.
6. Click **?WSDL** in the Test Point column.
7. You can see the WSDL. You can use this WSDL to test the service (using a testing tool like SoapUI).

The installation is complete. For more information, see the Java Payload Mapper Web Service Usage section.

Redeploy the Application

If the payloadmapper-service application has already been deployed, follow these steps:

1. If payloadmapper-service application is running, select **Stop** and **When Work Completes or Force Stop Now**, depending on the environment.

   Note: The recommended option always is **When Work Completes**.

2. Select **Delete**.

   The payloadmapper-service should now not display on the Summary of Deployment page.

Once the application is redeployed, you may return to the appropriate step in "Deploy the Payload Mapper".

Post Deployment Activity

Once the deployment is successful, copy the directory **payloadmapperjars** from the location where `JavaPayloadMapper15.0.0ForAllx.x.xApps_eng_ga.tar` is

extracted, to the Domain Home. Domain Home is the directory of the WebLogic domain of the server to which you deploy the Web service. Restart WebLogic server, if needed.

Custom Payload Mapper for payloads that are not Packaged

To use the Java Payload Mapper for versions of RIB Payloads those are not packaged as part of JavaPayloadMapper15.0.0ForAllx.x.xApps_eng_ga.tar, you need to download the Payload libraries for the versions you want to map. RIB payload libraries for all available versions of RIB can be obtained from corresponding RIB Functional Artifacts.

For example: To use the payload version 14.1.0, which is not packaged in the payloadmapperjars by default.

1. Create a folder "14.1.0" under `$DOMAIN_HOME/payloadmapperjars/`
2. Download the `RibFuncArtifact14.1.0ForAll14.1.0Apps_eng_ga.tar`
3. Extract the `RibFuncArtifact14.1.0ForAll14.1.0Apps_eng_ga.tar` and copy these below jars to `payloadmapperjars/14.1.0`

   `retail-public-payload-java-beans-14.1.0.jar`

   `retail-public-payload-java-beans-base-14.1.0.jar`

   Download and Extract `RetailSOAEnabler14.1.0ForAll14.1.0Apps_eng_ga.tar`
4. Locate and copy over the below jars from `/retail-soa-enabler/integration-lib/` to `$DOMAIN_HOME/payloadmapperjars/14.1.0`

   `jaxb-api-2.2.9.jar`

   `jaxb-impl-2.2.6.jar`

   `jaxb-xjc-2.2.6.jar`

   `jsr173_1.0_api.jar`
5. Copy the `$DOMAIN_HOME/payloadmapperjars/15.0.0/payloadmapper-util-15.0.0.jar` to `$DOMAIN_HOME/payloadmapperjars/14.1.0`

Directory Structure

The following directory structure is created on extracting the Java Payload Mapper. The location where this tar is extracted is PM_HOME.

```
payload-mapper/
|-- java-docs
|-- payloadmapperjars
|   |-- 11.1.0
|   |   |-- castor-0.9.4.3.jar
|   |   |-- retek-payload-typed.jar
|   |   `-- retek-rib-support.jar
|   |-- 12.0.9
|   |   |-- castor-0.9.4.3.jar
|   |   |-- retek-payload-typed.jar
|   |   `-- retek-rib-support.jar
|   |-- 13.0.2
|   |   |-- castor-1.0.5-xml.jar
|   |   |-- rib-public-api.jar
|   |   |-- rib-public-payload-java-beans.jar
|   |   `-- xercesImpl-2.10.0.jar
|   |-- 13.1.0
|   |   |-- jaxb-api.jar
|   |   |-- jaxb-impl.jar
|   |   |-- jaxb-xjc.jar
|   |   |-- jsr173_1.0_api.jar
|   |   |-- payloadmapper-util.jar
|   |   |-- retail-public-payload-java-beans-base.jar
|   |   `-- retail-public-payload-java-beans.jar
|   |-- 13.2.0
```

```
|   |   |-- jaxb-api.jar
|   |   |-- jaxb-impl.jar
|   |   |-- jaxb-xjc.jar
|   |   |-- jsr173_1.0_api.jar
|   |   |-- payloadmapper-util.jar
|   |   |-- retail-public-payload-java-beans-base.jar
|   |   `-- retail-public-payload-java-beans.jar
|   `-- 15.0.0
|       |-- jaxb-api-2.2.9.jar
|       |-- jaxb-impl-2.2.6.jar
|       |-- jaxb-xjc-2.2.6.jar
|       |-- jsr173_1.0_api.jar
|       |-- payloadmapper-util-15.0.0.jar
|       |-- retail-public-payload-java-beans-15.0.0.jar
|                `--  retail-public-payload-java-beans-base-
15.0.0.jar|
`-- webservice
    `-- payloadmapper-service-<version>.ear
```

Java Payload Mapper Web Service Usage

The Java Payload Mapper dynamically maps the fields of one version of the payload to that of another version of the payload, on a best effort basis.

Format of the input to Payload Mapper Web Service

```
<v1:convertMappingData>
        <!--Optional:-->
         <v11:MappingData>
            <sourcePayloadReleaseVersion></sourcePayloadReleaseVersion>
            <destinationPayloadReleaseVersion></destinationPayloadReleaseVersion>
            <!--Optional:-->
            <v11:DataToMap>
               <payload></payload>
            </v11:DataToMap>
            <!--Optional:-->
            <v11:MappingFunction>
               <!--1 or more repetitions:-->
               <v11:Function>
                  <name></name>
                  <needExecutionLog>true</needExecutionLog>
               </v11:Function>
            </v11:MappingFunction>
            <!--Optional:-->
            <v11:MappingFunctionExecutionLog>
               <v11:MappedElements>
                  <!--Zero or more repetitions:-->
                  <v11:sourcePayloadElementName></v11:sourcePayloadElementName>
               </v11:MappedElements>
               <v11:UnableToMapElements>
                  <!--Zero or more repetitions:-->
                  <v11:UnableToMapElement>
                     <v11:sourcePayloadElementName></v11:sourcePayloadElementName>
                     <v11:Why>
                        <issueCode></issueCode>
                        <description></description>
                     </v11:Why>
                  </v11:UnableToMapElement>
               </v11:UnableToMapElements>
            </v11:MappingFunctionExecutionLog>
            <!--Optional:-->
            <v11:DataAfterApplyingMappingFunction>
               <payload></payload>
            </v11:DataAfterApplyingMappingFunction>
         </v11:MappingData>
</v1:convertMappingData>
```

The Java Payload Mapper Web service exposes an operation called **convertMappingData** that accepts xml as input.

The inputs required are:

- **sourcePayloadReleaseVersion**
- **destinationPayloadReleaseVersion**
- **DataToMap** (This is the source payload)

- **needExecutionLog** (Optional)

  The possible values for **needExecutionLog** are true or false.

  If **needExecutionLog** is set to true, then Payload Mapper populates the **MappingFunctionExecutionLog** of the above XML with the list of elements that were mapped and the list of elements that were not mapped with the reason code (issue code) and reason.

The **DataAfterApplyingMappingFunction** section of the above XML is populated with the converted destination payload.

Payload Mapper as a Service

Java Payload Mapper can be used as a service to perform the payload translation. In this scenario, the payload mapper is hosted as a Web service. The service is called from an orchestration layer (for example, SOA, Service Bus, etc.) for the payload conversion. The service does its best effort in conversion. Since the service will report the list of fields that it cannot convert, the orchestration layer can enrich the conversion by adding custom conversion logic to the fields that were not converted.

A

Appendix: Examples of Adapting APIs

```
RIB Publication API (GETNXT)
create_<rms,rdm>mfm_<message family>.sql

BEGIN
EXECUTE IMMEDIATE '
CREATE OR REPLACE PACKAGE <API_NAME> AS

PROCEDURE getnxt (o_status_code OUT VARCHAR2,
                      o_error_msg OUT VARCHAR2,
                        o_message_type OUT VARCHAR2,
                        o_message OUT '||'&2'||'.RIB_OBJECT,
                        o_bus_obj_id OUT '||'&2'||'.RIB_BUSOBJID_TBL,
                        o_routing_info OUT '||'&2'||'.RIB_ROUTINGINFO_TBL,
                        i_num_threads IN NUMBER DEFAULT 1,
                        i_thread_val IN NUMBER DEFAULT 1,
                        i_facility_type IN VARCHAR2,
                        i_dc_dest_id IN VARCHAR2);
END;';

EXECUTE IMMEDIATE'
CREATE OR REPLACE PACKAGE BODY <API_NAME> AS

PROCEDURE getnxt (o_status_code OUT VARCHAR2,
                  o_error_msg OUT VARCHAR2,
                  o_message_type OUT VARCHAR2,
                  o_message OUT '||'&2'||'.RIB_OBJECT,
                  o_bus_obj_id OUT '||'&2'||'.RIB_BUSOBJID_TBL,
                  o_routing_info OUT '||'&2'||'.RIB_ROUTINGINFO_TBL,
                  i_num_threads IN NUMBER DEFAULT 1,
                  i_thread_val IN NUMBER DEFAULT 1,
                  i_facility_type IN VARCHAR2,
                  i_dc_dest_id IN VARCHAR2) IS
   lo_util_message        '||'&1'||'.RIB_OBJECT;
   lo_util_bus_obj_id     '||'&1'||'.RIB_BUSOBJID_TBL;
   lo_util_routing_info   '||'&1'||'.RIB_ROUTINGINFO_TBL;
BEGIN

    '||'&1'||'.<API_NAME>.GETNXT(o_status_code, o_error_msg, o_message_type,
lo_util_message, lo_util_bus_obj_id, lo_util_routing_info, i_num_threads,
i_thread_val, i_facility_type, i_dc_dest_id);
    IF lo_util_message IS NOT NULL AND o_status_code =
'||'&1'||'.RIB_CODES.MFM_SUCCESS THEN
       o_message := Payload_Mapper.Map_Object_Source_To_Target(lo_util_message,
o_error_msg);
       IF o_error_msg IS NULL THEN
          o_routing_info := Convert_RIB_ROUTINGINFO_TBL(lo_util_routing_info,
o_error_msg);
          IF o_error_msg IS NOT NULL THEN
             o_status_code := '||'&1'||'.RIB_CODES.MFM_FATAL_ERROR;
          END IF;
          SELECT  cast(lo_util_bus_obj_id as '||'&2'||'.RIB_BUSOBJID_TBL)
          INTO    o_bus_obj_id
          FROM    dual;
       END IF;
    END IF;
EXCEPTION
```

```
   WHEN OTHERS THEN
      o_status_code := '||'&1'||'.RIB_CODES.MFM_FATAL_ERROR;
      o_error_msg   := Substr(''Error converting RIB_ROUTINGINFO_TBL
''||sqlcode||''  ''||sqlerrm, 1, 2000);
END getnxt;

END;';
EXCEPTION
WHEN OTHERS THEN
   NULL;
END;
/

RIB Subscription API (CONSUME)
create_<rms,rdm>rdmsub_<message family>.sql

BEGIN
EXECUTE IMMEDIATE '
CREATE OR REPLACE PACKAGE <API_NAME> IS

   PROCEDURE consume (O_STATUS_CODE OUT VARCHAR2,
                      O_ERROR_MESSAGE OUT VARCHAR2,
                      I_MESSAGE IN '||'&2'||'.RIB_OBJECT,
                      I_MESSAGE_TYPE IN VARCHAR2,
                      I_FACILITY_TYPE IN VARCHAR2);
END RDMSUB_ASNIN;';

EXECUTE IMMEDIATE '
CREATE OR REPLACE PACKAGE BODY <API_NAME> IS

   PROCEDURE consume (O_STATUS_CODE OUT VARCHAR2,
                      O_ERROR_MESSAGE OUT VARCHAR2,
                      I_MESSAGE IN '||'&2'||'.RIB_OBJECT,
                      I_MESSAGE_TYPE IN VARCHAR2,
                      I_FACILITY_TYPE IN VARCHAR2) AS
      lo_Util_Message '||'&1'||'.RIB_OBJECT;
   BEGIN
      lo_Util_Message  := Payload_Mapper.Map_Object_Source_To_Target(I_MESSAGE,
O_ERROR_MESSAGE);
      IF O_ERROR_MESSAGE IS NULL THEN
         '||'&1'||'.RDMSUB_ASNIN.consume(O_STATUS_CODE, O_ERROR_MESSAGE,
lo_Util_Message,
                                         I_MESSAGE_TYPE, I_FACILITY_TYPE);
      ELSE
         o_status_code := '||'&1'||'.RIB_CODES.MFM_FATAL_ERROR;
      END IF;
   EXCEPTION
      WHEN OTHERS THEN
         O_STATUS_CODE     := '||'&1'||'.RIB_CODES.MFM_FATAL_ERROR;
         O_ERROR_MESSAGE   := Substr(''Error converting object ''||sqlcode||''
''||sqlerrm, 1, 2000);
   END;
END RDMSUB_ASNIN;';
EXCEPTION
WHEN OTHERS THEN
   NULL;
END;
/
```

B

Appendix: Sample Log File

```
4741008  21-OCT-10-06-20-44: Starting conversion of RIB_ASNINDESC_REC from version
12.0.9 to 13.2.1
4741008  21-OCT-10-06-20-44: Fetching the mappings from database
4741008  21-OCT-10-06-20-44: Starting conversion of object
4741008  21-OCT-10-06-20-44: Source Object to be converted is:
4741008  21-OCT-10-06-20-44: RIB_ASNInDesc_REC: RIB_OID, TO_LOCATION,
FROM_LOCATION, ASN_NBR, ASN_TYPE, CONTAINER_QTY, BOL_NBR, SHIPMENT_DATE,
EST_ARR_DATE, SHIP_ADDRESS1, SHIP_ADDRESS2, SHIP_ADDRESS3, SHIP_ADDRESS4,
SHIP_ADDRESS5, SHIP_CITY, SHIP_STATE, SHIP_ZIP, SHIP_COUNTRY_ID, TRAILER_NBR,
SEAL_NBR, CARRIER_CODE, VENDOR_NBR, SHIP_PAY_METHOD, <ASNINPO_TBL>, COMMENTS,
4741008  21-OCT-10-06-20-44: 1, A, A, A, A, 1, A, 13-OCT-10, 13-OCT-10, A, A, A,
A, A, A, A, A, A, A, A, A, A, A, <ASNINPO_TBL>, A,
4741008  21-OCT-10-06-20-44:     RIB_ASNInPO_REC: RIB_OID, PO_NBR, DOC_TYPE,
NOT_AFTER_DATE, COMMENTS, <ASNINCTN_TBL>, <ASNINITEM_TBL>,
4741008  21-OCT-10-06-20-44:     1, A, A, 13-OCT-10, A, <ASNINCTN_TBL>,
<ASNINITEM_TBL>,
4741008  21-OCT-10-06-20-44:         RIB_ASNInCtn_REC: RIB_OID, FINAL_LOCATION,
CONTAINER_ID, CONTAINER_WEIGHT, CONTAINER_LENGTH, CONTAINER_WIDTH,
CONTAINER_HEIGHT, CONTAINER_CUBE, EXPEDITE_FLAG, IN_STORE_DATE, RMA_NBR,
TRACKING_NBR, FREIGHT_CHARGE, MASTER_CONTAINER_ID, <ASNINITEM_TBL>, COMMENTS,
4741008  21-OCT-10-06-20-44:         1, A, A, 1, 1, 1, 1, 1, A, 13-OCT-10, A, A,
1, A, <ASNINITEM_TBL>, A,
4741008  21-OCT-10-06-20-44:             RIB_ASNInItem_REC: RIB_OID,
FINAL_LOCATION, ITEM_ID, UNIT_QTY, PRIORITY_LEVEL, VPN, ORDER_LINE_NBR, LOT_NBR,
REF_ITEM, DISTRO_NBR, DISTRO_DOC_TYPE, CONTAINER_QTY, COMMENTS,
4741008  21-OCT-10-06-20-44:             1, A, A, 1, 1, A, 1, A, A, A, A, 1, A,
4741008  21-OCT-10-06-20-44:         RIB_ASNInItem_REC: RIB_OID, FINAL_LOCATION,
ITEM_ID, UNIT_QTY, PRIORITY_LEVEL, VPN, ORDER_LINE_NBR, LOT_NBR, REF_ITEM,
DISTRO_NBR, DISTRO_DOC_TYPE, CONTAINER_QTY, COMMENTS,
4741008  21-OCT-10-06-20-44:         1, A, A, 1, 1, A, 1, A, A, A, A, 1, A,
4741008  21-OCT-10-06-20-44: Creating target object
4741008  21-OCT-10-06-20-44: Number of elements in Object: 128
4741008  21-OCT-10-06-20-44: Assigning value for RIB_OID
4741008  21-OCT-10-06-20-44: Using Source column: PayloadSourceObj.RIB_OID
4741008  21-OCT-10-06-20-44: Assigning value for ns_version_v1
4741008  21-OCT-10-06-20-44: Assigning value for ns_name_ASNInDesc
4741008  21-OCT-10-06-20-44: Assigning value for ns_type_bo
4741008  21-OCT-10-06-20-44: Assigning value for ns_location_base
4741008  21-OCT-10-06-20-44: Assigning value for ns_level_top
4741008  21-OCT-10-06-20-44: Assigning value for SCHEDULE_NBR
4741008  21-OCT-10-06-20-44: Assigning value for AUTO_RECEIVE
4741008  21-OCT-10-06-20-44: Assigning value for TO_LOCATION
4741008  21-OCT-10-06-20-44: Using Source column: PayloadSourceObj.TO_LOCATION
4741008  21-OCT-10-06-20-44: Assigning value for FROM_LOCATION
4741008  21-OCT-10-06-20-44: Using Source column: PayloadSourceObj.FROM_LOCATION
4741008  21-OCT-10-06-20-44: Assigning value for ASN_NBR
4741008  21-OCT-10-06-20-44: Using Source column: PayloadSourceObj.ASN_NBR
4741008  21-OCT-10-06-20-44: Assigning value for ASN_TYPE
4741008  21-OCT-10-06-20-44: Using Source column: PayloadSourceObj.ASN_TYPE
4741008  21-OCT-10-06-20-44: Assigning value for CONTAINER_QTY
4741008  21-OCT-10-06-20-44: Using Source column: PayloadSourceObj.CONTAINER_QTY
4741008  21-OCT-10-06-20-44: Assigning value for BOL_NBR
4741008  21-OCT-10-06-20-44: Using Source column: PayloadSourceObj.BOL_NBR
4741008  21-OCT-10-06-20-44: Assigning value for SHIPMENT_DATE
4741008  21-OCT-10-06-20-44: Using Source column: PayloadSourceObj.SHIPMENT_DATE
4741008  21-OCT-10-06-20-44: Assigning value for EST_ARR_DATE
```

```
4741008  21-OCT-10-06-20-44: Using Source column: PayloadSourceObj.EST_ARR_DATE
4741008  21-OCT-10-06-20-44: Assigning value for SHIP_ADDRESS1
4741008  21-OCT-10-06-20-44: Using Source column: PayloadSourceObj.SHIP_ADDRESS1
4741008  21-OCT-10-06-20-44: Assigning value for SHIP_ADDRESS2
4741008  21-OCT-10-06-20-44: Using Source column: PayloadSourceObj.SHIP_ADDRESS2
4741008  21-OCT-10-06-20-44: Assigning value for SHIP_ADDRESS3
4741008  21-OCT-10-06-20-44: Using Source column: PayloadSourceObj.SHIP_ADDRESS3
4741008  21-OCT-10-06-20-44: Assigning value for SHIP_ADDRESS4
4741008  21-OCT-10-06-20-44: Using Source column: PayloadSourceObj.SHIP_ADDRESS4
4741008  21-OCT-10-06-20-44: Assigning value for SHIP_ADDRESS5
4741008  21-OCT-10-06-20-44: Using Source column: PayloadSourceObj.SHIP_ADDRESS5
4741008  21-OCT-10-06-20-44: Assigning value for SHIP_CITY
4741008  21-OCT-10-06-20-44: Using Source column: PayloadSourceObj.SHIP_CITY
4741008  21-OCT-10-06-20-44: Assigning value for SHIP_STATE
4741008  21-OCT-10-06-20-44: Using Source column: PayloadSourceObj.SHIP_STATE
4741008  21-OCT-10-06-20-44: Assigning value for SHIP_ZIP
4741008  21-OCT-10-06-20-44: Using Source column: PayloadSourceObj.SHIP_ZIP
4741008  21-OCT-10-06-20-44: Assigning value for SHIP_COUNTRY_ID
4741008  21-OCT-10-06-20-44: Using Source column: PayloadSourceObj.SHIP_COUNTRY_ID
4741008  21-OCT-10-06-20-44: Assigning value for TRAILER_NBR
4741008  21-OCT-10-06-20-44: Using Source column: PayloadSourceObj.TRAILER_NBR
4741008  21-OCT-10-06-20-44: Assigning value for SEAL_NBR
4741008  21-OCT-10-06-20-44: Using Source column: PayloadSourceObj.SEAL_NBR
4741008  21-OCT-10-06-20-44: Assigning value for CARRIER_CODE
4741008  21-OCT-10-06-20-44: Using Source column: PayloadSourceObj.CARRIER_CODE
4741008  21-OCT-10-06-20-44: Assigning value for VENDOR_NBR
4741008  21-OCT-10-06-20-44: Using Source column: PayloadSourceObj.VENDOR_NBR
4741008  21-OCT-10-06-20-44: Assigning value for SHIP_PAY_METHOD
4741008  21-OCT-10-06-20-44: Using Source column: PayloadSourceObj.SHIP_PAY_METHOD
4741008  21-OCT-10-06-20-44:     Creating collection RIB_ASNInPO_TBL
4741008  21-OCT-10-06-20-44:     Count of collection is 1
4741008  21-OCT-10-06-20-44:     Processing Record 1
4741008  21-OCT-10-06-20-44:     Assigning Value for RIB_OID
4741008  21-OCT-10-06-20-44:     Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).RIB_OID
4741008  21-OCT-10-06-20-44:     Assigning Value for ns_version_v1
4741008  21-OCT-10-06-20-44:     Assigning Value for ns_name_ASNInDesc
4741008  21-OCT-10-06-20-44:     Assigning Value for ns_type_bo
4741008  21-OCT-10-06-20-44:     Assigning Value for ns_location_base
4741008  21-OCT-10-06-20-44:     Assigning Value for ns_level_nontop
4741008  21-OCT-10-06-20-44:     Assigning Value for PO_NBR
4741008  21-OCT-10-06-20-44:     Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).PO_NBR
4741008  21-OCT-10-06-20-44:     Assigning Value for DOC_TYPE
4741008  21-OCT-10-06-20-44:     Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).DOC_TYPE
4741008  21-OCT-10-06-20-44:     Assigning Value for NOT_AFTER_DATE
4741008  21-OCT-10-06-20-44:     Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).NOT_AFTER_DATE
4741008  21-OCT-10-06-20-44:     Assigning Value for COMMENTS
4741008  21-OCT-10-06-20-44:     Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).COMMENTS
4741008  21-OCT-10-06-20-44:         Creating collection RIB_ASNInCtn_TBL
4741008  21-OCT-10-06-20-44:         Count of collection is 1
4741008  21-OCT-10-06-20-44:         Processing Record 1
4741008  21-OCT-10-06-20-44:         Assigning Value for RIB_OID
4741008  21-OCT-10-06-20-44:         Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINCTN_TBL(1).RIB_OID
4741008  21-OCT-10-06-20-44:         Assigning Value for ns_version_v1
4741008  21-OCT-10-06-20-44:         Assigning Value for ns_name_ASNInDesc
4741008  21-OCT-10-06-20-44:         Assigning Value for ns_type_bo
4741008  21-OCT-10-06-20-44:         Assigning Value for ns_location_base
4741008  21-OCT-10-06-20-44:         Assigning Value for ns_level_nontop
```

```
4741008  21-OCT-10-06-20-44:         Assigning Value for FINAL_LOCATION
4741008  21-OCT-10-06-20-44:         Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINCTN_TBL(1).FINAL_LOCATION
4741008  21-OCT-10-06-20-44:         Assigning Value for CONTAINER_ID
4741008  21-OCT-10-06-20-44:         Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINCTN_TBL(1).CONTAINER_ID
4741008  21-OCT-10-06-20-44:         Assigning Value for CONTAINER_WEIGHT
4741008  21-OCT-10-06-20-44:         Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINCTN_TBL(1).CONTAINER_WEIGHT
4741008  21-OCT-10-06-20-44:         Assigning Value for CONTAINER_LENGTH
4741008  21-OCT-10-06-20-44:         Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINCTN_TBL(1).CONTAINER_LENGTH
4741008  21-OCT-10-06-20-44:         Assigning Value for CONTAINER_WIDTH
4741008  21-OCT-10-06-20-44:         Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINCTN_TBL(1).CONTAINER_WIDTH
4741008  21-OCT-10-06-20-44:         Assigning Value for CONTAINER_HEIGHT
4741008  21-OCT-10-06-20-44:         Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINCTN_TBL(1).CONTAINER_HEIGHT
4741008  21-OCT-10-06-20-44:         Assigning Value for CONTAINER_CUBE
4741008  21-OCT-10-06-20-44:         Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINCTN_TBL(1).CONTAINER_CUBE
4741008  21-OCT-10-06-20-44:         Assigning Value for EXPEDITE_FLAG
4741008  21-OCT-10-06-20-44:         Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINCTN_TBL(1).EXPEDITE_FLAG
4741008  21-OCT-10-06-20-44:         Assigning Value for IN_STORE_DATE
4741008  21-OCT-10-06-20-44:         Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINCTN_TBL(1).IN_STORE_DATE
4741008  21-OCT-10-06-20-44:         Assigning Value for RMA_NBR
4741008  21-OCT-10-06-20-44:         Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINCTN_TBL(1).RMA_NBR
4741008  21-OCT-10-06-20-44:         Assigning Value for TRACKING_NBR
4741008  21-OCT-10-06-20-44:         Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINCTN_TBL(1).TRACKING_NBR
4741008  21-OCT-10-06-20-44:         Assigning Value for FREIGHT_CHARGE
4741008  21-OCT-10-06-20-44:         Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINCTN_TBL(1).FREIGHT_CHARGE
4741008  21-OCT-10-06-20-44:         Assigning Value for MASTER_CONTAINER_ID
4741008  21-OCT-10-06-20-44:         Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINCTN_TBL(1).MASTER_CONTAINER_ID
4741008  21-OCT-10-06-20-44:             Creating collection RIB_ASNInItem_TBL
4741008  21-OCT-10-06-20-44:             Count of collection is 1
4741008  21-OCT-10-06-20-44:             Processing Record 1
4741008  21-OCT-10-06-20-44:             Assigning Value for RIB_OID
4741008  21-OCT-10-06-20-44:             Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINCTN_TBL(1).ASNINITEM_TBL(1).RIB_OID
4741008  21-OCT-10-06-20-44:             Assigning Value for ns_version_v1
4741008  21-OCT-10-06-20-44:             Assigning Value for ns_name_ASNInDesc
4741008  21-OCT-10-06-20-44:             Assigning Value for ns_type_bo
4741008  21-OCT-10-06-20-44:             Assigning Value for ns_location_base
4741008  21-OCT-10-06-20-44:             Assigning Value for ns_level_nontop
4741008  21-OCT-10-06-20-44:             Assigning Value for FINAL_LOCATION
4741008  21-OCT-10-06-20-44:             Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINCTN_TBL(1).ASNINITEM_TBL(1).FINAL_LOCATION
4741008  21-OCT-10-06-20-44:             Assigning Value for ITEM_ID
4741008  21-OCT-10-06-20-44:             Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINCTN_TBL(1).ASNINITEM_TBL(1).ITEM_ID
4741008  21-OCT-10-06-20-44:             Assigning Value for UNIT_QTY
4741008  21-OCT-10-06-20-44:             Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINCTN_TBL(1).ASNINITEM_TBL(1).UNIT_QTY
4741008  21-OCT-10-06-20-44:             Assigning Value for PRIORITY_LEVEL
4741008  21-OCT-10-06-20-44:             Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINCTN_TBL(1).ASNINITEM_TBL(1).PRIORITY_LEVEL
4741008  21-OCT-10-06-20-44:             Assigning Value for VPN
```

```
4741008  21-OCT-10-06-20-44:             Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINCTN_TBL(1).ASNINITEM_TBL(1).VPN
4741008  21-OCT-10-06-20-44:             Assigning Value for ORDER_LINE_NBR
4741008  21-OCT-10-06-20-44:             Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINCTN_TBL(1).ASNINITEM_TBL(1).ORDER_LINE_NBR
4741008  21-OCT-10-06-20-44:             Assigning Value for LOT_NBR
4741008  21-OCT-10-06-20-44:             Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINCTN_TBL(1).ASNINITEM_TBL(1).LOT_NBR
4741008  21-OCT-10-06-20-44:             Assigning Value for REF_ITEM
4741008  21-OCT-10-06-20-44:             Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINCTN_TBL(1).ASNINITEM_TBL(1).REF_ITEM
4741008  21-OCT-10-06-20-44:             Assigning Value for DISTRO_NBR
4741008  21-OCT-10-06-20-44:             Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINCTN_TBL(1).ASNINITEM_TBL(1).DISTRO_NBR
4741008  21-OCT-10-06-20-44:             Assigning Value for DISTRO_DOC_TYPE
4741008  21-OCT-10-06-20-44:             Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINCTN_TBL(1).ASNINITEM_TBL(1).DISTRO_DOC_TYPE
4741008  21-OCT-10-06-20-44:             Assigning Value for CONTAINER_QTY
4741008  21-OCT-10-06-20-44:             Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINCTN_TBL(1).ASNINITEM_TBL(1).CONTAINER_QTY
4741008  21-OCT-10-06-20-44:             Assigning Value for COMMENTS
4741008  21-OCT-10-06-20-44:             Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINCTN_TBL(1).ASNINITEM_TBL(1).COMMENTS
4741008  21-OCT-10-06-20-44:                 Creating collection
RIB_ExtOfASNInItem_TBL
4741008  21-OCT-10-06-20-44:                 Count of collection is 0
4741008  21-OCT-10-06-20-44:         Assigning Value for COMMENTS
4741008  21-OCT-10-06-20-44:         Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINCTN_TBL(1).COMMENTS
4741008  21-OCT-10-06-20-44:             Creating collection RIB_ExtOfASNInCtn_TBL
4741008  21-OCT-10-06-20-44:             Count of collection is 0
4741008  21-OCT-10-06-20-44:         Creating collection RIB_ASNInItem_TBL
4741008  21-OCT-10-06-20-44:         Count of collection is 1
4741008  21-OCT-10-06-20-44:         Processing Record 1
4741008  21-OCT-10-06-20-44:         Assigning Value for RIB_OID
4741008  21-OCT-10-06-20-44:         Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINITEM_TBL(1).RIB_OID
4741008  21-OCT-10-06-20-44:         Assigning Value for ns_version_v1
4741008  21-OCT-10-06-20-44:         Assigning Value for ns_name_ASNInDesc
4741008  21-OCT-10-06-20-44:         Assigning Value for ns_type_bo
4741008  21-OCT-10-06-20-44:         Assigning Value for ns_location_base
4741008  21-OCT-10-06-20-44:         Assigning Value for ns_level_nontop
4741008  21-OCT-10-06-20-44:         Assigning Value for FINAL_LOCATION
4741008  21-OCT-10-06-20-44:         Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINITEM_TBL(1).FINAL_LOCATION
4741008  21-OCT-10-06-20-44:         Assigning Value for ITEM_ID
4741008  21-OCT-10-06-20-44:         Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINITEM_TBL(1).ITEM_ID
4741008  21-OCT-10-06-20-44:         Assigning Value for UNIT_QTY
4741008  21-OCT-10-06-20-44:         Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINITEM_TBL(1).UNIT_QTY
4741008  21-OCT-10-06-20-44:         Assigning Value for PRIORITY_LEVEL
4741008  21-OCT-10-06-20-44:         Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINITEM_TBL(1).PRIORITY_LEVEL
4741008  21-OCT-10-06-20-44:         Assigning Value for VPN
4741008  21-OCT-10-06-20-44:         Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINITEM_TBL(1).VPN
4741008  21-OCT-10-06-20-44:         Assigning Value for ORDER_LINE_NBR
4741008  21-OCT-10-06-20-44:         Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINITEM_TBL(1).ORDER_LINE_NBR
4741008  21-OCT-10-06-20-44:         Assigning Value for LOT_NBR
4741008  21-OCT-10-06-20-44:         Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINITEM_TBL(1).LOT_NBR
```

```
4741008  21-OCT-10-06-20-44:         Assigning Value for REF_ITEM
4741008  21-OCT-10-06-20-44:         Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINITEM_TBL(1).REF_ITEM
4741008  21-OCT-10-06-20-44:         Assigning Value for DISTRO_NBR
4741008  21-OCT-10-06-20-44:         Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINITEM_TBL(1).DISTRO_NBR
4741008  21-OCT-10-06-20-44:         Assigning Value for DISTRO_DOC_TYPE
4741008  21-OCT-10-06-20-44:         Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINITEM_TBL(1).DISTRO_DOC_TYPE
4741008  21-OCT-10-06-20-44:         Assigning Value for CONTAINER_QTY
4741008  21-OCT-10-06-20-44:         Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINITEM_TBL(1).CONTAINER_QTY
4741008  21-OCT-10-06-20-44:         Assigning Value for COMMENTS
4741008  21-OCT-10-06-20-44:         Using Source column:
PayloadSourceObj.ASNINPO_TBL(1).ASNINITEM_TBL(1).COMMENTS
4741008  21-OCT-10-06-20-44:             Creating collection
RIB_ExtOfASNInItem_TBL
4741008  21-OCT-10-06-20-44:             Count of collection is 0
4741008  21-OCT-10-06-20-44:         Creating collection RIB_ExtOfASNInPO_TBL
4741008  21-OCT-10-06-20-44:         Count of collection is 0
4741008  21-OCT-10-06-20-44: Assigning value for COMMENTS
4741008  21-OCT-10-06-20-44: Using Source column: PayloadSourceObj.COMMENTS
4741008  21-OCT-10-06-20-44:     Creating collection RIB_ExtOfASNInDesc_TBL
4741008  21-OCT-10-06-20-44:     Count of collection is 0
4741008  21-OCT-10-06-20-44: Converted Target Object is:
4741008  21-OCT-10-06-20-44: RIB_ASNInDesc_REC: RIB_OID, ns_version_v1,
ns_name_ASNInDesc, ns_type_bo, ns_location_base, ns_level_top, SCHEDULE_NBR,
AUTO_RECEIVE, TO_LOCATION, FROM_LOCATION, ASN_NBR, ASN_TYPE, CONTAINER_QTY,
BOL_NBR, SHIPMENT_DATE, EST_ARR_DATE, SHIP_ADDRESS1, SHIP_ADDRESS2, SHIP_ADDRESS3,
SHIP_ADDRESS4, SHIP_ADDRESS5, SHIP_CITY, SHIP_STATE, SHIP_ZIP, SHIP_COUNTRY_ID,
TRAILER_NBR, SEAL_NBR, CARRIER_CODE, VENDOR_NBR, SHIP_PAY_METHOD, <ASNINPO_TBL>,
COMMENTS, <EXTOFASNINDESC_TBL>,
4741008  21-OCT-10-06-20-44: 1, , , , , , , , A, A, A, A, 1, A, 13-OCT-10, 13-OCT-
10, A, A, A, A, A, A, A, A, A, A, A, A, A, A, <ASNINPO_TBL>, A,
<EXTOFASNINDESC_TBL>,
4741008  21-OCT-10-06-20-44:     RIB_ASNInPO_REC: RIB_OID, ns_version_v1,
ns_name_ASNInDesc, ns_type_bo, ns_location_base, ns_level_nontop, PO_NBR,
DOC_TYPE, NOT_AFTER_DATE, COMMENTS, <ASNINCTN_TBL>, <ASNINITEM_TBL>,
<EXTOFASNINPO_TBL>,
4741008  21-OCT-10-06-20-44:     1, , , , , , A, A, 13-OCT-10, A, <ASNINCTN_TBL>,
<ASNINITEM_TBL>, <EXTOFASNINPO_TBL>,
4741008  21-OCT-10-06-20-44:         RIB_ASNInCtn_REC: RIB_OID, ns_version_v1,
ns_name_ASNInDesc, ns_type_bo, ns_location_base, ns_level_nontop, FINAL_LOCATION,
CONTAINER_ID, CONTAINER_WEIGHT, CONTAINER_LENGTH, CONTAINER_WIDTH,
CONTAINER_HEIGHT, CONTAINER_CUBE, EXPEDITE_FLAG, IN_STORE_DATE, RMA_NBR,
TRACKING_NBR, FREIGHT_CHARGE, MASTER_CONTAINER_ID, <ASNINITEM_TBL>, COMMENTS,
<EXTOFASNINCTN_TBL>,
4741008  21-OCT-10-06-20-44:         1, , , , , , A, A, 1, 1, 1, 1, 1, A, 13-OCT-
10, A, A, 1, A, <ASNINITEM_TBL>, A, <EXTOFASNINCTN_TBL>,
4741008  21-OCT-10-06-20-44:             RIB_ASNInItem_REC: RIB_OID,
ns_version_v1, ns_name_ASNInDesc, ns_type_bo, ns_location_base, ns_level_nontop,
FINAL_LOCATION, ITEM_ID, UNIT_QTY, PRIORITY_LEVEL, VPN, ORDER_LINE_NBR, LOT_NBR,
REF_ITEM, DISTRO_NBR, DISTRO_DOC_TYPE, CONTAINER_QTY, COMMENTS,
<EXTOFASNINITEM_TBL>,
4741008  21-OCT-10-06-20-44:             1, , , , , , A, A, 1, 1, A, 1, A, A, A,
A, 1, A, <EXTOFASNINITEM_TBL>,
4741008  21-OCT-10-06-20-44:                 RIB_ExtOfASNInItem_REC:
4741008  21-OCT-10-06-20-44:                 <NULL>
4741008  21-OCT-10-06-20-45:         RIB_ASNInItem_REC: RIB_OID, ns_version_v1,
ns_name_ASNInDesc, ns_type_bo, ns_location_base, ns_level_nontop, FINAL_LOCATION,
ITEM_ID, UNIT_QTY, PRIORITY_LEVEL, VPN, ORDER_LINE_NBR, LOT_NBR, REF_ITEM,
DISTRO_NBR, DISTRO_DOC_TYPE, CONTAINER_QTY, COMMENTS, <EXTOFASNINITEM_TBL>,
```

```
4741008  21-OCT-10-06-20-45:         1, , , , , , A, A, 1, 1, A, 1, A, A, A, A, 1,
A, <EXTOFASNINITEM_TBL>,
4741008  21-OCT-10-06-20-45:             RIB_ExtOfASNInItem_REC:
4741008  21-OCT-10-06-20-45:             <NULL>
4741008  21-OCT-10-06-20-45:         RIB_ExtOfASNInPO_REC:
4741008  21-OCT-10-06-20-45:         <NULL>
4741008  21-OCT-10-06-20-45: Successfully converted.
```

C

Appendix: Examples of Functional Resolvers

This appendix includes two example scenarios.

Example Scenario 1

RIB_LocationDesc_REC is used by RWMS to subscribe to Locations from RMS. In this object, **org_unit_id** is a column that is not mapped. In the Mapper UI, when prompted, the Functional Resolver is specified as follows:

```
Get_Unit_Id(RIB_LOCATIONDESC_REC.DEST_ID).
```

In this example, the parameter passed is the column from the same object in the same hierarchy.

```
Create or replace
    FUNCTION GET_UNIT_ID
        (p_dest_id IN VARCHAR2)
        RETURN VARCHAR2 AS lnUnit NUMBER;
    BEGIN
        SELECT ORG_UNIT_ID
        INTO      lnUnit
        FROM    RWMS01.WH
        WHERE  wh  = p_dest_id
        AND ROWNUM = 1;
        RETURN lnUnit;
    EXCEPTION
        RETURN NULL;
    END GET_UNIT_ID;
```

> **Note**: All exceptions must be handled, and a value must be returned.

In this example the message is enriched by selecting against the base application tables. As discussed, doing this has a serious impact on performance and should be carefully considered.

Example Scenario 2

RIB_ItemDesc_REC is used by RWMS to subscribe to Items from RMS.

RIB_ItemDesc_REC contains an object type of RIB_ItemHdrDesc_REC called ItemHdrDesc as one of its columns. In RIB_ItemDesc_REC, **NOTIONAL_PACK_IND** is a column that is not mapped.

In the Mapper UI, when prompted, the Functional Resolver is specified as follows: Get_Indicator(RIB_ItemDesc_REC.ITEMHDRDESC.ITEM).

The parameter selected is the column from the same object in the same hierarchy.

```
Create or replace
FUNCTION GET_ Indicator
(p_item_id IN VARCHAR2)
RETURN VARCHAR2 AS    lvInd   VARCHAR2(1);
BEGIN
SELECT PACK_IND
```

```
INTO      lvInd
FROM    RMS01.ITEM_MASTER
WHERE  item = p_item_id
 AND ROWNUM = 1;
RETURN lvInd;
EXCEPTION
           RETURN NULL;
END GET_INDICATOR;
```

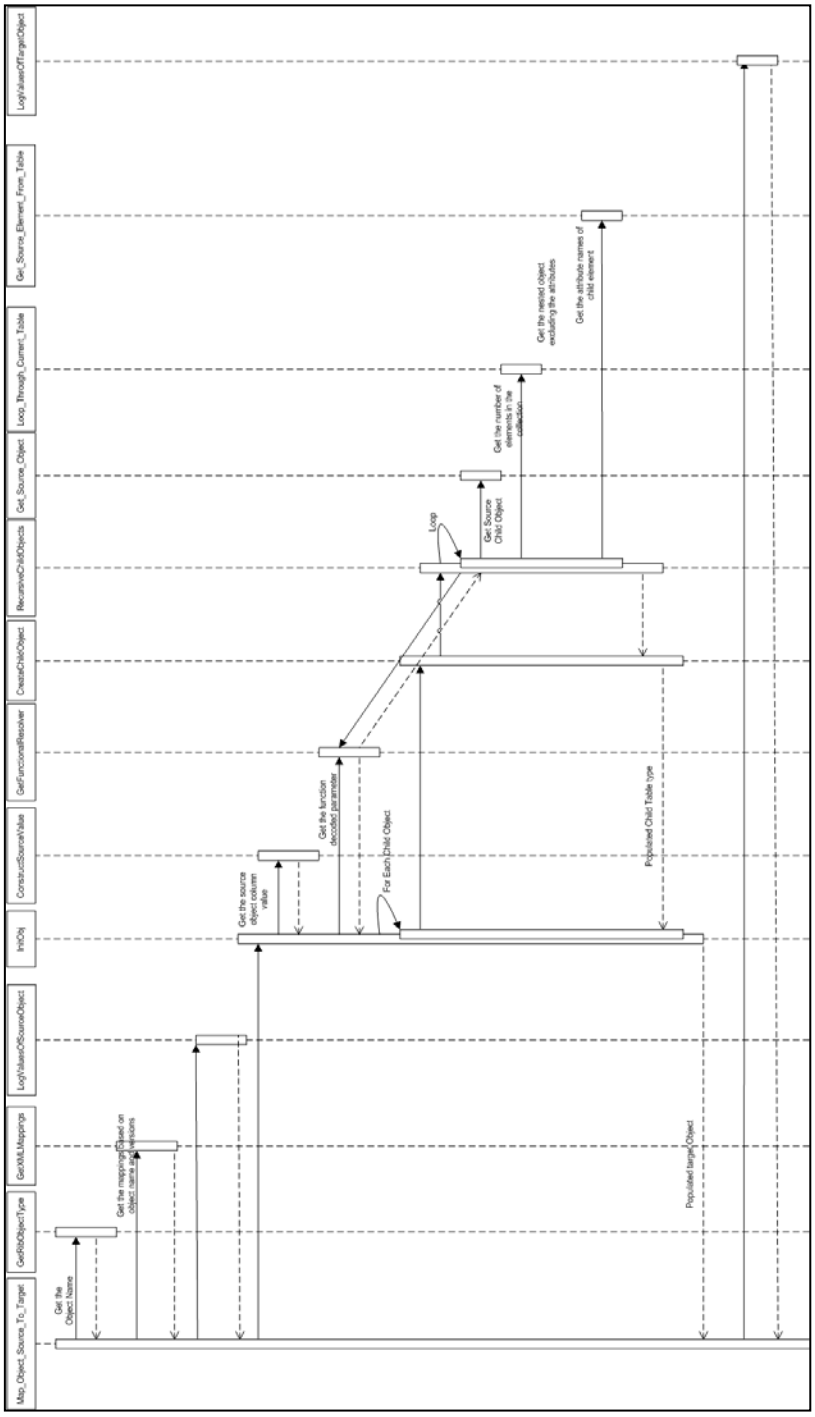
Note: All exceptions must be handled, and a value must be returned.

In this example the message is enriched by selecting against another base application table. As discussed, doing this has a serious impact on performance and should be carefully considered.

D

Appendix: Mapper Runtime Sequence Diagram

The following diagram illustrates the mapper runtime sequence:



E

Appendix: Examples of APIs and Objects (RWMS)

Publish API	RIB Oracle Objects
RDMMFM_ASNIN.GETNXT	RIB_ASNInDesc_REC, RIB_ASNInRef_REC
RDMMFM_ASNOUT.GETNXT	RIB_ASNOutDesc_REC
RDMMFM_CUSTRETURN.GETNXT	RIB_CustRetDesc_REC
RDMMFM_INVADJUST.GETNXT	RIB_InvAdjustDesc_REC
RDMMFM_ITEMWH.GETNXT	RIB_ItemWHDesc_REC, RIB_ItemWHRef_REC
RDMMFM_RTV.GETNXT	RIB_RTVDesc_REC
RDMMFM_RECEIVING.GETNXT	RIB_ReceiptDesc_REC
RDMMFM_SOSTATUS.GETNXT	RIB_SOStatusDesc_REC
RDMMFM_SPACELOCS.GETNXT	RIB_SpaceLocsDesc_REC, RIB_SpaceLocsRef_REC
RDMMFM_WHEQUIPCLS.GETNXT	RIB_WHEquipClsDesc_REC, RIB_WHEquipClsRef_REC
RDMMFM_WHEQUIP.GETNXT	RIB_WHEquipDesc_REC, RIB_WHEquipRef_REC
RDMMFM_WHSPACELOCS.GETNXT	RIB_WHSpaceLocsDesc_REC, RIB_WHSpaceLocsRef_REC
RDMMFM_WHTRANS.GETNXT	RIB_WHTransDesc_REC

Subscribe API	RIB Oracle Objects
RDMSUB_ASNIN.CONSUME	RIB_ASNInDesc_REC, RIB_ASNInRef_REC
RDMSUB_DIFFGRP.CONSUME	RIB_DiffGrpRef_REC, RIB_DiffGrpDtlRef_REC, RIB_DiffGrpHdrDesc_REC, RIB_DiffGrpDtlDesc_REC
RDMSUB_DIFFS.CONSUME	RIB_DiffRef_REC, RIB_DiffDesc_REC
RDMSUB_DlvySlt.CONSUME	RIB_DeliverySlotRef_REC, RIB_DeliverySlotDesc_REC
RDMSUB_ITEMS.CONSUME	RIB_ItemDesc_REC, RIB_ISCDimRef_REC, RIB_ItemBOMRef_REC, RIB_ItemRef_REC, RIB_ItemSupCtyRef_REC, RIB_ItemSupRef_REC, RIB_ItemUDALOVRef_REC, RIB_ItemUPCRef_REC
RDMSUB_LOC.CONSUME	RIB_LocationDesc_REC, RIB_LocationRef_REC
RDMSUB_ORDER.CONSUME	RIB_PODesc_REC, RIB_PORef_REC
RDMSUB_PENDRETURN.CONSUME	RIB_PendRtrnRef_REC, RIB_PendRtrnDtlRef_REC, RIB_PendRtrnDesc_REC
RDMSUB_SKUOPTM.CONSUME	RIB_SKUOptmDesc_REC
RDMSUB_SO.CONSUME	RIB_SODesc_REC, RIB_SORef_REC

Subscribe API	RIB Oracle Objects
RDMSUB_UDAS.CONSUME	RIB_UDARef_REC, RIB_UDAValRef_REC, RIB_UDADesc_REC, RIB_UDAValDesc_REC
RDMSUB_VENDOR.CONSUME	RIB_VendorRef_REC, RIB_VendorAddrRef_REC, RIB_VendorDesc_REC
RDMSUB_WOIN.CONSUME	RIB_WOInDesc_REC, RIB_WOInRef_REC
RDMSUB_WOOUT.CONSUME	RIB_WOOutDesc_REC, RIB_WOOutRef_REC

F

Appendix: Examples of RWMS PUB Files

The following is a list of RWMS PUB files:

RIB_AppointDesc_REC _13.2.0.xml
RIB_AppointRef_REC _13.2.0.xml
RIB_ASNInDesc_REC _13.2.0.xml
RIB_ASNInRef_REC _13.2.0.xml
RIB_ASNOutDesc_REC _13.2.0.xml
RIB_CustRetDesc_REC _13.2.0.xml
RIB_InvAdjustDesc_REC _13.2.0.xml
RIB_ItemWHDesc_REC _13.2.0.xml
RIB_ItemWHRef_REC _13.2.0.xml
RIB_ReceiptDesc_REC _13.2.0.xml
RIB_RTVDesc_REC _13.2.0.xml
RIB_RTVReqDesc_REC _13.2.0.xml
RIB_RTVReqRef_REC _13.2.0.xml
RIB_SOStatusDesc_REC _13.2.0.xml
RIB_SpaceLocsDesc_REC _13.2.0.xml
RIB_SpaceLocsRef_REC _13.2.0.xml
RIB_WHEquipClsDesc_REC _13.2.0.xml
RIB_WHEquipClsRef_REC _13.2.0.xml
RIB_WHEquipDesc_REC _13.2.0.xml
RIB_WHEquipRef_REC _13.2.0.xml
RIB_WHSpaceLocsDesc_REC _13.2.0.xml
RIB_WHSpaceLocsRef_REC _13.2.0.xml
RIB_WHTransDesc_REC _13.2.0.xml

G

Appendix: Examples of RWMS SUB Files

The following is a list of RWMS SUB files:

RIB_AllocDesc_REC_12.0.9.xml	RIB_StoreDesc_REC _12.0.9.xml
RIB_AllocRef_REC_12.0.9.xml	RIB_StoreRef_REC _12.0.9.xml
RIB_ASNInDesc_REC_12.0.9.xml	RIB_UDADesc_REC _12.0.9.xml
RIB_ASNInRef_REC_12.0.9.xml	RIB_UDARef_REC _12.0.9.xml
RIB_DiffDesc_REC_12.0.9.xml	RIB_UDAValDesc_REC _12.0.9.xml
RIB_DiffGrpDtlDesc_REC_12.0.9.xml	RIB_UDAValRef_REC _12.0.9.xml
RIB_DiffGrpDtlRef_REC_12.0.9.xml	RIB_VendorAddrRef_REC _12.0.9.xml
RIB_DiffGrpHdrDesc_REC_12.0.9.xml	RIB_VendorDesc_REC _12.0.9.xml
RIB_DiffGrpRef_REC_12.0.9.xml	RIB_VendorRef_REC _12.0.9.xml
RIB_DiffRef_REC_12.0.9.xml	RIB_WOInDesc_REC _12.0.9.xml
RIB_ISCDimRef_REC_12.0.9.xml	RIB_WOInRef_REC _12.0.9.xml
RIB_ItemBOMRef_REC_12.0.9.xml	RIB_WOOutDesc_REC _12.0.9.xml
RIB_ItemDesc_REC _12.0.9.xml	RIB_WOOutRef_REC _12.0.9.xml
RIB_ItemRef_REC _12.0.9.xml	
RIB_ItemSupCtyRef_REC _12.0.9.xml	
RIB_ItemSupRef_REC _12.0.9.xml	
RIB_ItemTcktRef_REC _12.0.9.xml	
RIB_ItemUDADateRef_REC _12.0.9.xml	
RIB_ItemUDAFFRef_REC _12.0.9.xml	
RIB_ItemUDALOVRef_REC _12.0.9.xml	
RIB_ItemUPCRef_REC _12.0.9.xml	
RIB_LocationDesc_REC _12.0.9.xml	
RIB_LocationRef_REC _12.0.9.xml	
RIB_PendRtrnDesc_REC _12.0.9.xml	
RIB_PendRtrnDtlRef_REC _12.0.9.xml	
RIB_PendRtrnRef_REC _12.0.9.xml	
RIB_PODesc_REC _12.0.9.xml	
RIB_PORef_REC _12.0.9.xml	
RIB_SODesc_REC _12.0.9.xml	
RIB_SORef_REC _12.0.9.xml	