Oracle® Commerce Retail Extension Module

Merchandising Implementation Guide
Release 16.0.2
E90907-01

January 2018

ORACLE

Oracle Commerce Retail Extension Module Merchandising Implementation Guide, Release 16.0.2

E90907-01

Primary Author: Bernadette Goodman

Contributors: Aarti Iyer, Rajeev Jayappa, Chad Timm

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 ..... **vii**

Preface ..... **ix**

Documentation Accessibility ix
Customer Support ix
Review Patch Documentation ix
Improved Process for Oracle Retail Documentation Corrections ix
Oracle Retail Documentation on the Oracle Technology Network x
Conventions x

1 Introduction **1**

2 Architecture **1**

3 Prerequisite Data **1**

4 Bulk Data Integration (BDI) **1**

Bulk Feeds 1
Importer Jobs 1

5 Incremental Data Integration (RIB) **1**

Message Families 1
Overview of Message Flow 4
RIB XSD to RXMDI Staging Scheme Mapping 22
RXMDI EAR 31
Extensibility 32

6 Java Batch **35**

Overview 35
RXM Batch Job Admin 35
Job XMLs 36
Batchlets and Deciders 37
RXMDI Job Admin WAR 37
Extensibility 37

7 Oracle Data Integrator **41**

Components 41
Packaging 41
Extensibility 42
Transformations 42
Database Tables to Database Tables 42
Database Tables to XML Files 42
Flat Files to Database tables 43

8 Promotion and Pricing Integration **45**

Promotion Integration 45
Promotion Commerce Enhancements 46

Pricing Integration 46
Pricing Commerce Enhancements 47
9 Job Mapping 49
BDI/RIB 49
Pricing and Promotions 51
10 Packaging and Deployment 53

Send Us Your Comments

Oracle Commerce Retail Extension Module Merchandising Implementation Guide, Release 16.0.2

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 guide provides information needed for an implementation of Oracle Commerce Retail Extension Module with Oracle Retail Merchandising Operations Management.

Documentation Accessibility

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

Access to Oracle Support

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

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, 16.0) or a later patch release (for example, 16.0.2). 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

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

Introduction

RXMDI and ODI run on the RXM Staging Server deployed at the Corporate Data Center. The RXM Staging server has access to two dedicated Oracle databases: BDI Interface Inbound Schema and RXMDI Staging Schema. RXMDI and ODI run on the Oracle Retail release 16.0 supported stack with Fusion Middleware 12.2.1.2.

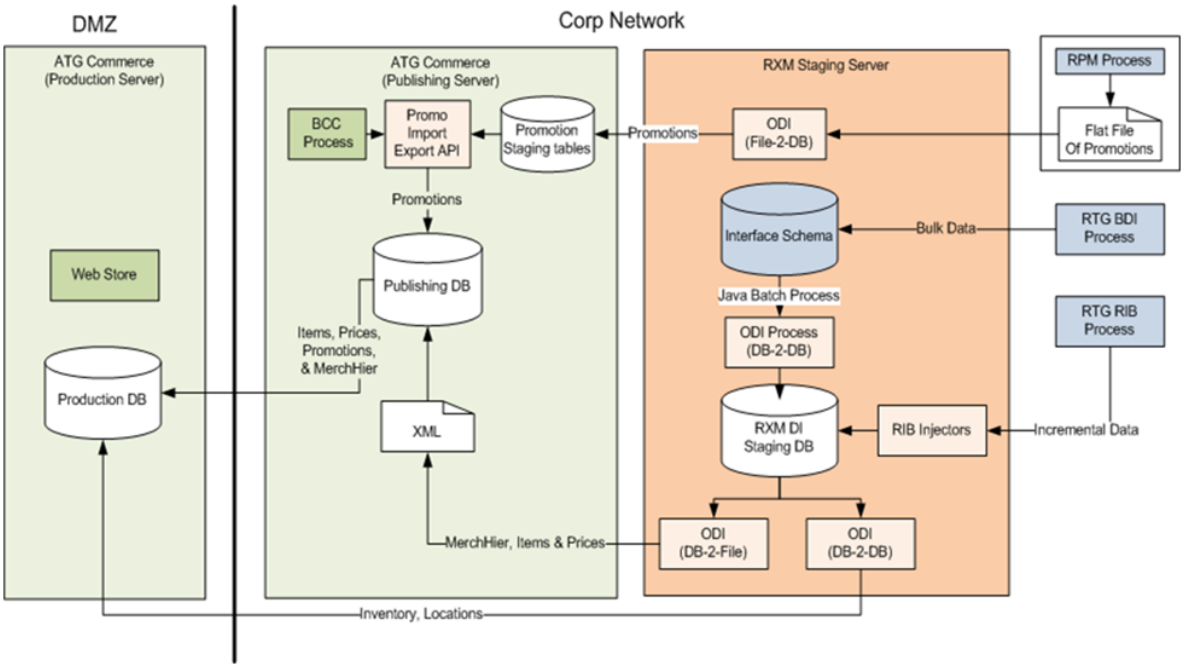
The incremental and bulk data is staged and primarily mapped into XML files for feeding to Oracle Commerce's SQLImport program running on Oracle Commerce Publishing Server, or placed into the Oracle Commerce Production repository directly. The promotional data is mapped to staging tables in Oracle Commerce Publishing for further transformation and import by an RXM scheduled service.

Technologies used for data integration:

- Oracle Data Integrator Enterprise (ODI) 12c (12.2.1.2)
- Oracle WebLogic 12c (12.2.1.2)
- Oracle Database 12c (12.1.0.2.0)
- Java 8

2

Architecture



This diagram shows the BDI, RIB, and RPM processes. MOM and RXMDI are on the right, RXM Publishing in the middle, and RXM Production on the left.

Oracle Retail Merchandising System

- Bulk data moves from RMS to RXMDI through BDI Process flows. They move the data into the BDI Input (that is, BDI Inbound Interface) Schema. They are then transformed by ODI into the RXMDI Staging Schema using the RXM Batch jobs which invoke the appropriate ODI Scenarios.
- Incremental data moves from RMS to RXMDI through RIB. RIB data is processed by implementing RIB message injectors and persisting the data directly into the RXMDI Staging Schema using JPA.
- Merchandise Hierarchy and Item data from the RXMDI Staging Schema is transformed by ODI into XML files which are then imported into the RXM publishing server's Business Control Center (BCC) from where it is published to RXM's production server.
- Inventory, Store, and Warehouse data from the RXMDI Staging Schema is transformed by ODI and inserted into RXM's production server through Direct SQL Load.

Oracle Retail Price Management

RPM Regular and Clearance Price change data arrive as flat files into a directory read by ODI, and are transformed by ODI into XML files which are then imported into the RXM publishing server's Business Control Center (BCC) from where it is published to RXM's production server.

RPM Promotion data also arrives as flat files into a directory read by ODI, and are transformed by ODI and inserted directly into RXM's publishing server through Direct SQL Load. Using a timed Scheduler, they are then processed through the PromotionImportExport API provided by Oracle Commerce and imported into the BCC from where they are published to RXM's production server.

3

Prerequisite Data

Prior to loading data from RXMDI Staging tables to RXM, the following tables need to be populated with appropriate data.

Based on the values obtained from RMS and set up in RXM Publishing's Business Control Center (BCC) insert the following values into RXMDI_STORE_SITE:

- store_id: Obtained from RMS
- site id: Created in BCC
- catalog id: Created in BCC
- price_list_id: Created in BCC
- sale_list_id: Created in BCC
- file_name: Leave blank
- folder_id: Created in BCC

Based on the representation of unit of measure's (UOM) in RMS and ATG, map the UOMs in the RXMDI_LOOKUP_UOM.

RXMDI_LOOKUP_UOM Sample Data

```
INSERT INTO rxmdi_lookup_uom (mom_uom, atg_uom) VALUES ('EA', 'units');
INSERT INTO rxmdi_lookup_uom (mom_uom, atg_uom) VALUES ('LB', 'pounds');
INSERT INTO rxmdi_lookup_uom (mom_uom, atg_uom) VALUES ('LBS', 'pounds');
INSERT INTO rxmdi_lookup_uom (mom_uom, atg_uom) VALUES ('KG', 'kilograms');
```

- RXM requires latitude and longitude to be populated for a given address. If they are not available, latitude and longitude values are defaulted to zero in the DCS_LOCATION RXM table.
- RXMDI_LOCATION_GEO, located in the RXMDI staging schema, could be pre-populated with this data. ODI procedure GetGeoCode could also be customized to call an external geo location service to populate this data.

RXMDI_LOCATION_GEO Sample Data

```
Insert into RXMDI_LOCATION_GEO
(LOCATION_ID,LATITUDE,LONGITUDE,FORMATTED_ADDRESS,ERROR) values
(5251,'44.6496868','-93.2427200','Lakeville, MN, USA','0');
Insert into RXMDI_LOCATION_GEO
(LOCATION_ID,LATITUDE,LONGITUDE,FORMATTED_ADDRESS,ERROR) values
(4241,'44.8407980','-93.2982799','Bloomington, MN, USA','0');
Insert into RXMDI_LOCATION_GEO
(LOCATION_ID,LATITUDE,LONGITUDE,FORMATTED_ADDRESS,ERROR) values
(3231,'34.9353693','-83.3890464','York, GA 30568, USA','0');
Insert into RXMDI_LOCATION_GEO
(LOCATION_ID,LATITUDE,LONGITUDE,FORMATTED_ADDRESS,ERROR) values
(6261,'45.1607987','-93.2349489','Blaine, MN, USA','0');
```

4

Bulk Data Integration (BDI)

Oracle Bulk Data Integration (BDI) is a product that defines the architecture and infrastructure used to move bulk data between Oracle Retail applications. BDI sits in the middle of Oracle Retail Merchandising System (RMS) and other applications, and it is built on top of Java EE and Java Batch platform. In a Bulk Data Integration system, Message Families are represented as interface modules. Each interface module (such as DiffGrp_Fnd) contains an RMS component that takes care of pulling and staging data for publication to the External BDI system. Interface modules are divided by functional entity (such as Item Master, Stores, and Diffs).

RMS publishes Bulk Data through the Bulk Data Infrastructure (BDI) Process Flows into the BDI Interface Inbound Schema. The RXM Importer Jobs are part of the BDI Process flows and, through RXM Batch Jobs, invoke the corresponding ODI Scenario. This data is extracted by ODI, transformed, and then persisted into the Retail Extension Module Data Integration (RXMDI) Staging schema.

All the mappings in tables are same between RXMDI Staging schema and BDI Interface Inbound Schema. On reclassification of item, it is not deleted from the previously linked Merchandise Hierarchy.

Bulk Feeds

RXMDI receives Bulk data for the following feeds from RMS BDI:

- InvAvailWh_Tx (Inventory)
- Store_Fnd and StoreAddr_Fnd (Store and Store Address)
- MerchHier_Fnd (Merchandise Hierarchy)
- ItemHdr_Fnd (Item Header)
- ItemLoc_Fnd (Item Location)
- RelatedItem_Fnd (Related Items)
- Diff_Fnd and DiffGrp_Fnd (Diff and Diff Group)

Warehouse, warehouse address, and item image are definition only, no data flow.

Importer Jobs

If data is known to be present in the BDI Interface Schema, he RXM Importer jobs can be directly invoked using the RXM Batch Job Admin through Importer jobs. Each interface module has its own Importer Job. For example, there is one for Store, one for Store Address, one for Inventory, and so on. Each Importer Job invokes the corresponding ODI Scenario which extracts the data from the BDI Interface Schema, transforms it, and persists it into the RXMDI Staging Schema.

This is how each Importer Job functions:

1. RMS publishes data with a unique data set ID for a particular interface module. This data set ID maps to a set of sequence numbers representing the records for that interface module. It is cumulative (that is, includes data from previous data sets).
2. When the Importer Job is invoked, the oracle.retail.commerce.batch.DataSetBatchlet determines the latest dataset ID and also queries the RXMDI_ODI_JOB_AUDIT table

in the RXMDI Staging schema to determine if this dataset has already been successfully processed.

3. If the dataset has not been processed, it computes the sequence numbers that belong to that data set and pass along that information to the oracle.retail.commerce.batch.ODIBatchlet.
4. The oracle.retail.commerce.batch.ODIBatchlet invokes the ODI Agent after setting relevant data retrieved from the Job XML as well as the sequence numbers obtained from the oracle.retail.commerce.batch.DataSetBatchlet.
5. Once the Job has been processed by ODI, the oracle.retail.commerce.batch.AuditBatchlet writes a record in the RXMDI_ODI_JOB_AUDIT table in the RXMDI Staging schema with a status of Processed or Failed.
6. An oracle.retail.commerce.batch.StepDecider works in conjunction with the logic in the Job Specification Language (JSL) to allow the Batchlets to decide the next step in the process or whether the process should be Stopped.

RXMDI_ODI_JOB_AUDIT

```
  data_set_id                       DECIMAL(19,0) NOT NULL, -- Data Set Id
provided as part of the Bulk data set
  data_set_begin_seq_num            DECIMAL(19,0) NOT NULL, -- Beginning sequence
number of dataset in inbound interface table
  data_set_end_seq_num              DECIMAL(19,0) NOT NULL, -- Ending sequence
number of dataset in inbound interface table
  src_sys_data_set_ready_time       TIMESTAMP NOT NULL,     -- Time when source
system provided data in outbound tables
  job_complete_status               VARCHAR2(20) NOT NULL,  -- Status after Job
has completed
  interface_module_name             VARCHAR2(20) NOT NULL,  -- Name of the
Interface Module
  create_time                       TIMESTAMP NOT NULL      -- Time when record
was created
);

-- Add primary key constraint

ALTER TABLE RXMDI_ODI_JOB_AUDIT ADD CONSTRAINT pk_rxmdi_odi_job_audit PRIMARY KEY
(data_set_id, create_time);
```

5

Incremental Data Integration (RIB)

RMS publishes create, modify, and delete messages to the Oracle Retail Integration Bus (RIB) for various Message Families through the RIB-RXM adapter. The messages from the adapter are consumed by RXMD) and persisted in the RXMDI Staging schema using the Java Persistence API (JPA).

Technologies

- EclipseLink JPA provider
- Apache Camel
- JPA (through xml)

Message Families

Stores

Stores can have multiple addresses one-to-many mapping using JPA.

Message Types:

- storecre: Message gets persisted in RXMDI_STORE, RXMDI_STORE_ADDRESS tables creating a new row with rxmdi_control as N (New) in both the RXMDI_STORE and RXMDI_STORE_ADDRESS (list of addresses) tables.
- storemod: Existing row in the database is updated and rxmdi_control is set to N (New) in both the RXMDI_STORE and RXMDI_STORE_ADDRESS (list of addresses) tables.
- storedel: Row with corresponding storeId is first retrieved from the database and then updated with corresponding values from StoreRef; rxmdi_control is set to DN (Delete) in both RXMDI_STORE, and RXMDI_STORE_ADDRESS (list of addresses) tables.
- storedtlcre: Same as storecre
- storedtlmod: Same as storemod
- storedtldel: It is just address delete, so store's rxmdi_control will remain as only the corresponding addresses' rxmdi_control is set to DN (Delete). Store changes rxmdi_control to DN (Delete) only when there is a storedel message.

Item Locations

Message Types:

- ItemLocCre: Record is persisted in rxmdi_item_loc with rxmdi_control as N (New).
- ItemLocMod: Record is modified in rxmdi_item_loc with rxmdi_control as N (New).
- ItemLocDel: Record is persisted in rxmdi_item_loc with rxmdi_control as DN (Delete).
- ItemLocReplMod: Not supported for RXMDI.

Warehouses

Message Types:

- whcre: Message gets persisted in the RXMDI_WH and RXMDI_WH_ADDR tables creating a new row with rxmdi_control as N (New) in both the RXMDI_WH and RXMDI_WH_ADDR (list of addresses) tables.
- whmod: Existing row in the database is updated and rxmdi_control is set to N (New)in both the RXMDI_WH and RXMDI_WH_ADDR (list of addresses) tables.
- whdel: Row with corresponding wh_Id is first retrieved from the database and then updated with the corresponding values from WHRef; rxmdi_control is set to DN (Delete) in both the RXMDI_WH and RXMDI_WH_ADDR (list of addresses) tables.
- whdtlcre: Same as whcre.
- whdtlmod: Same as whmod.
- whdtldel: It is just address delete, so the WH's rxmdi_control will remain as is and only the corresponding addresses' rxmdi_control is set to DN (Delete). WH changes rxmdi_control to DN (Delete) only when there is a whdel message.
- whaddcre: Message to handle internal RMS processing, but it still comes out of RMS. Not supported by RXMDI.
- whaddmod: Message to handle internal RMS processing, but it still comes out of RMS. Not supported by RXMDI.

Merchandise Hierarchy

This table lists the attributes being statically set in the staging table, as these are not provided by the RIB.

Attribute in RXMDI_MERCH_HIER	Valid Values
HierarchyLevel	For department: Department For Class: CLASS For Subclass: SUBCLASS
HierarchyId	Field introduced by us for creating primary key. It is a unique identifier comprising all three display id plus prepended prefix. Example: For department 3: d3 For department 3, class 1: d3c1 For department 3, class 1 and subclass 4: d3c1s
Parent Level	For department: GROUP For class: DEPARTMENT For Subclass: CLASS

The rest of the attributes are obtained from RIB messages. RIB does not provide the hierarchy node ID and parent node ID.

Message Types:

- Record is persisted in the RXMDI_MERC_HIER table in the RXMDI staging schema.
- deptcre: New record is persisted and rxmdi_control is set to N (New).
- deptmod: Modifies record persisted in deptcre; rxmdi_control is set to N (New).
- deptdel: Modifies record persisted in above two; rxmdi_control is set to DN (Delete).
- subclasscre: Creates new record is persisted; rxmdi_control is set to N (New).

- subclassmod: Modifies record persisted in subclasscre; rxmdi_control is set to N (New).
- subclassdel: Modifies record persisted in deptdel and subclasscre; rxmdi_control is set to DN (Delete).
- classcre: Creates new record is persisted; rxmdi_control is set to N (New).
- classmod: Modifies record persisted in classcre; rxmdi_control is set to N (New).
- classdel: Modifies record persisted in subclassdel and classcre; rxmdi_control is set to DN (Delete).

Items

The Item in RMS maps to either Product or SKU in RXM. On reclassification of item, item is not deleted from previously linked Merchandise Hierarchy.

Message Type:

- Items: Items and item components make up what is called the Items message family:
  - ItemCre: Comprised of several other messages, but RXMDI subscribes only to relatedItem, itemHeader, and itemImage. Record is persisted in the RXMDI_ITEM_HDR, RXMDI_ITEM_IMAGE, RXMDI_RELATED_ITEM, and RXMDI_RELATED_ITEM_DTL tables in RXMDI staging.
  - ItemHdrMod: Modifies record persisted in itemCre in the RXMDI_ITEM_HDR table with rxmdi_control as N (New).
  - ItemDel: Modifies record persisted in itemCre in RXMDI_ITEM_HDR table with rxmdi_control as DN (Delete).
  - ItemImageCre: New record is persisted to the RXMDI_ITEM_IMAGE table with rxmdi_control as N (New).
  - ItemImageMod: Modify record is persisted to the RXMDI_ITEM_IMAGE table with rxmdi_control as N (New).
  - ItemImageDel: Modify record is persisted to the RXMDI_ITEM_IMAGE table with rxmdi_control as DN (Delete).
  - RelItemHeadCre: New record is persisted to the RXMDI_RELATED_ITEM and RXMDI_RELATED_ITEM_DTL tables with rxmdi_control as N (New) in each.
  - RelItemHeadMod: Modify record is persisted to the RXMDI_RELATED_ITEM and RXMDI_RELATED_ITEM_DTL tables with rxmdi_control as N (New) in each.
  - RelItemHeadDel: Modify record is persisted to the RXMDI_RELATED_ITEM and RXMDI_RELATED_ITEM_DTL tables with rxmdi_control as DN (Delete) in each.
  - ItemBOMCre: New record is persisted to the RXMDI_PACK_ITEM table with rxmdi_control as N (New).
  - ItemBOMMod: Modify record is persisted to the RXMDI_PACK_ITEM table with rxmdi_control as N (New).
- Differentiator Groups: Differentiator Groups allow clients to group differentiator identifiers (Diff IDs) into logical groupings (for example: pant sizes, shirt colors, or flavors).
- DiffGrpHdrCre: Record is persisted in rxmdi_diff_grp with rxmdi_control as N (New). BDI provides diff_type_desc attribute which is available in RIB in another message family called Seed Data message family: DiffTypeDesc.xsd to which RXM is not subscribing, so this attribute is changed to nullable in the schema:

  - DiffGrpHdrMod: Record is modified in rxmdi_diff_grp with rxmdi_control as N (New).
  - DiffGrpDel: Record is persisted in rxmdi_diff_grp with rxmdi_control as DN (Delete).
    DiffGrpDtl comprised of mapping between diff and diff group.
  - DiffGrpDtlCre: Record is persisted in rxmdi_diff_grp_dtl with rxmdi_control as N (New).
  - DiffGrpDtlMod: Record is modified in rxmdi_diff_grp_dtl with rxmdi_control as N(New).
  - DiffGrpDtlDel: Record is persisted in rxmdi_diff_grp_dtl with rxmdi_control as DN (Delete).
- Differentiator Identifiers: Differentiators (Diffs, as they are commonly called) allow users to further distinguish items:
  - DiffCre: Record is persisted in rxmdi_diff with rxmdi_control as N (New).
  - DiffMod: Record is modified in rxmdi_diff with rxmdi_control as N (New).
  - DiffDel: Record is persisted in rxmdi_diff with rxmdi_control as DN (Delete).

Overview of Message Flow

These are the steps in the message flow:

1. RIB-RXM adapter publishes messages which are consumed by RXMDI. The message types and families which can be consumed are referenced in injectors.xml. Injectors.xml can be modified to add or remove additional Message Families and Message Types. Following is a snippet of injectors.xml:
   - Message Family: STORES
   - Message Types: STORECRE, STOREDEL, STOREMOD

injectors.xml

```
<injector_config>
    <family name="STORES">
        <injector

class="oracle.retail.commerce.incremental.location.LocationMessageInjector">
            <type>STORECRE</type>
        </injector>
        <injector

class="oracle.retail.commerce.incremental.location.LocationMessageInjector">
            <type>STOREDEL</type>
        </injector>
        <injector

class="oracle.retail.commerce.incremental.location.LocationMessageInjector">
            <type>STOREMOD</type>
        </injector>
```

2. Any Message Type that is out of scope for RXM or is not in use is filtered out using a Message Filter. Following is a snippet of a Message Filter:

Message Filter

```
        <injector

class="oracle.retail.commerce.incremental.filter.MessageFilteringInjector">
```

```
            <type>DIVISIONCRE</type>
        </injector>
        <injector

class="oracle.retail.commerce.incremental.filter.MessageFilteringInjector">
            <type>DIVISIONMOD</type>
        </injector>
        <injector

class="oracle.retail.commerce.incremental.filter.MessageFilteringInjector">
            <type>DIVISIONDEL</type>
        </injector>
```

3. The nessage injected by RIB is mapped to the appropriate Apache Camel route. The payload will be wrapped into a new Apache Camel integration framework message and sent to the mapped Camel route. The Camel route will be responsible for persistence and any other business logic. Following is a snippet of the camel route for STORECRE message:

Camel Route

```
<!-- CamelContext is the Camel runtime, where Camel routes are defined. -->
    <camel:camelContext id="LocationCamelContext" trace="false"
xmlns="http://camel.apache.org/schema/spring">

   <!-- Route to create data received from StoreDesc RIB Message for message type:
STORECRE or STOREDTLCRE -->
     <route id="createStoreData">
            <from uri="direct:createStoreData"/>
            <bean ref="Store" method="createStore"/>
            <to uri="jpa://oracle.retail.commerce.incremental.location.Store"/>
            <log message="Store created with id: ${body.store}"/>
      </route>
```

4. There is an Entity for every message. It has setters and getters along with the business logic which gets mapped using entity mapping to the respective tables in the RXMDI schema. JPA uses XML to persist data to database. Following is a snippet of entity mapping for Store:

Store Entity Mapping

```
<entity-mappings
    xmlns="http://www.eclipse.org/eclipselink/xsds/persistence/orm"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.eclipse.org/eclipselink/xsds/persistence/orm
    http://www.eclipse.org/eclipselink/xsds/eclipselink_orm_2_4.xsd"
    version="2.4">

    <!-- Entities -->

     <entity name="Store"
class="oracle.retail.commerce.incremental.location.Store" access="PROPERTY">
       <table name="rxmdi_store"/>
        <attributes>
           <id name="store"  access="PROPERTY">
                <column name="store_id" length="10"/>
           </id>
            <basic name="storeType"  access="PROPERTY">
                <column name="store_type" length="6"/>
            </basic>
            <basic name="storeName"  access="PROPERTY">
                <column name="store_name" length="150"/>
            </basic>
            <basic name="storeName10"  access="PROPERTY">
```

```
                <column name="store_name_10" length="10"/>
            </basic>
            <basic name="storeName3"  access="PROPERTY">
                <column name="store_name_abbr" length="3"/>
            </basic>
           .....
      ...
            <basic name="timezone"  access="PROPERTY">
                <column name="timezone" length="64"/>
            </basic>
            <basic name="rxmdiControl">
                <column name="rxmdi_control" length="20"/>
</basic>
            <basic name="lastUpdate">
                <column name="last_update" length="6"/>
            </basic>
            <!-- unidirectional one-to-many -->
            <one-to-many name="storeAddress" target-
entity="oracle.retail.commerce.incremental.location.StoreAddress">
                <join-column name="store_id" referenced-column-name="store_id"
nullable="false" />
                 <cascade>
                   <cascade-all/>
                 </cascade>
            </one-to-many>
        </attributes>
      </entity>
</entity-mappings>
```

5. On delete, RXM gets a sparse RIB message, for delete to get a hold of the JPA object from the schema. Make changes based on the new RIB message and then persist. Similarly for modify (one-to-many objects), RXM retrieves the row from the database, updates the respective columns, and then persists it. This logic of getting a hold of the JPA object and then modifying it is based on requirements for implementation as a Camel Processor.
6. The Processor implements a camel processor to process message exchanges from the route. The processor gets the primary key from the message exchange, then reads the JPA object from the staging schema, updates the object with latest RIB message, and persists it. Following is a snippet of the StoreDeleteProcessor.java:

Camel Processor

```
  @Override
    public void process(Exchange exchange) throws Exception
    {
        EntityManager em = exchange.getContext().getEndpoint("jpa:" +
Store.class.getName(), JpaEndpoint.class)
                .getEntityManagerFactory().createEntityManager();
        try
        {
            em.getTransaction().begin();
            StoreRef storeRef = (StoreRef)exchange.getIn().getBody();
            if (storeRef != null)
            {
                Query q =
em.createQuery(ProcessorQuery.selectStore).setParameter("store",
storeRef.getStore());
                if (q.getSingleResult() != null)
                {
                    Store store = (Store)q.getSingleResult();
                    store.setStockholdingInd(storeRef.getStockholdingInd());
                    store.setStoreType(storeRef.getStoreType());
```

```
                List<StoreAddress> storeAddressList = store.getStoreAddress();
                for (StoreAddress storeAddress : storeAddressList)
                {
                    storeAddress.setRxmdiControl(IncrementalConstants.DELETE);
                    storeAddress.setLastUpdate(new Date());
                }
                store.setRxmdiControl(IncrementalConstants.DELETE);
                store.setLastUpdate(new Date());
                logger.info("Store id:" + store.getStore() + " deleted");
                exchange.getOut().setBody(store);
            }
            em.getTransaction().commit();
        }
    }
    catch (Exception e)
```

7. After the message is persisted in the staging schema, it is picked up by ODI Scenario and processed, transformed, and persisted in either of the following two forms:
   - The Merchandise Hierarchy and Item data from the RXMDI Staging Schema is transformed by ODI into XML files, which is then imported into the RXM publishing server's Business Control Center (BCC) from where it is published to RXM's production server.
   - Store and Warehouse data from the RXMDI Staging Schema is transformed by ODI and inserted into RXM's production server through Direct SQL Load.

Schema

The following tables are in RXMDI Staging:

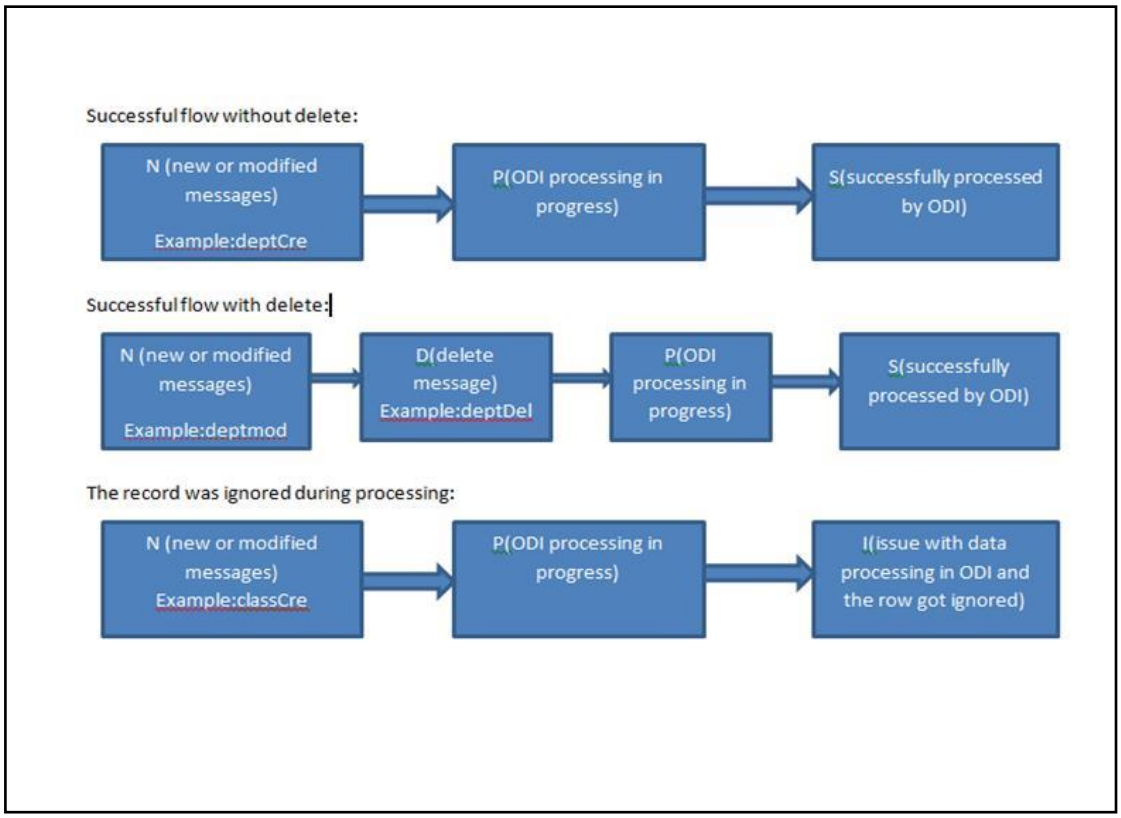
- Store: RXMDI_STORE, RXMDI_STORE_ADDRESS
- Merchandise Hierarchy: RXMDI_MERCH_HIER
- Items: RXMDI_ITEM_HDR, RXMDI_ITEM_IMAGE, RXMDI_ITEM_LOC, RXMDI_RELATED_ITEM, RXMDI_RELATED_ITEM_DTL, RXMDI_DIFF, RXMDI_DIFF_GRP, RXMDI_DIFF_GRP_DTL
- Warehouse: RXMDI_WH, RXMDI_WH_ADDR

The tables in staging closely resemble the BDI Interface Inbound tables. The staging tables have two additional control columns (rxmdi_control and last_update). The Controller/Flag in rxmdi_control is used to determine the life-cycle of a record.

The following are the different states for rxmdi_control:

- N: New record for created/modified message
- DN: Record to be Deleted
- DP: Delete being processed
- D: Record deleted from RXM but still remains in rxmdi staging
- RP: Referenced parent for merchandise hierarchy
- P: Record being processed by ODI
- S: Record successfully processed by ODI
- I: Record being ignored by ODI

The possible life-cycles for the rxmdi_control states are shown in the following figure:



Note: It is the responsibility of the retailer to establish the business process for handling the purge procedure.

Life-cycle rxmdi_control of delete message:

1. RP(referenced parent of merchandise hierarchy) ->S(After processing successfully)
2. DN(delete RIB message)->DP(processing delete message)->D(once deleted from RXM schema/productCatalog.xml generated for delete)

Delete

Merchandise Hierarchy :

1. If delete d4, then have to delete d4c41, d4c41s411. If do not, then d4c41, d4c41s411 will become un-categorized.
2. If a subclass is deleted, the parent reference should be updated to not refer to the deleted child anymore in prodcutCatalogxml.
3. On category delete, the associated item will not be deleted, as items can be associated to several categories.

Item:

1. If an item is deleted, it could have several related items. If an item is deleted, its related item will not be deleted. Only the item which gets a delete message will be deleted; related item will not be deleted.
2. If a product is deleted which has several SKUs, all the SKUs will also get deleted.

Reclassification:

Item reclassification allows moving an item from one department/class/subclass to another.

1. RXM handles RIB reclassification only. BDI reclassification will be handled by implementers. One of the ways implementers can handle BDI reclassification is through BCC.
2. It occurs in the case of ItemHdrMod RIB message. Both old and new subclass gets persisted in to productCatalog.xml using ODI as mentioned below:
   - Old subclass will not have association to item.
   - New subclass will have association to item.

Staging DDL

rxmdi_store

```
CREATE TABLE rxmdi_store (
  store_id                  NUMBER(10,0)        NOT NULL,  -- Unique ID of the
store.
  store_type                VARCHAR2(6),  -- This will indicate whether a
particular store is a franchise or company store.
  store_name                VARCHAR2(150),  -- Contains the name of the store
which, along with the store number, identifies the store.
  store_name_10             VARCHAR2(10),  -- Contains a ten character
abbreviation of the store name.
  store_name_abbr           VARCHAR2(3),  -- Contains a three character
abbreviation of the store name.
  store_name_sec            VARCHAR2(150),  -- Secondary name of the store.
  store_class_id            VARCHAR2(1),  -- Contains the code letter indicating
the class of which the store is a member. Valid values are A through E.
  store_class_desc          VARCHAR2(250),  -- Contains the name of the store
class.
  manager                   VARCHAR2(120),  -- Contains the name of the store
manager.
  open_date                 TIMESTAMP,  -- Contains the date on which the store
opened.
  close_date                TIMESTAMP,  -- Contains the date on which the store
closed.
  acquire_date              TIMESTAMP,  -- Contains the date on which the store
was acquired.
  remodel_date              TIMESTAMP,  -- Contains the date on which the store
was last remodeled.
  fax_number                VARCHAR2(20),  -- Contains the fax number for the
store.
  phone_number              VARCHAR2(20),  -- Contains the phone number for the
store.
  email                     VARCHAR2(100),  -- Holds the email address for the
location.
  total_sq_feet             NUMBER(8,0),  -- Contains the total square footage
of the store.
  selling_sq_feet           NUMBER(8,0),  -- Contains the total square footage
of the stores selling area.
  linear_distance           NUMBER(8,0),  -- Holds the total merchandisable
space of the location.
  vat_region                NUMBER(4,0),  -- Contains the number of the Value
Added Tax region in which this store is contained.
  vat_incl_ind              VARCHAR2(1),  -- Indicates whether or not Value
Added Tax will be included in the retail prices for the store. Valid values are Y
or N.
  stock_holding_ind         VARCHAR2(1),  -- This column indicates whether the
store can hold stock. In a non-multichannel environment this will always be Y.
  channel_id                NUMBER(4,0),  -- In a multichannel environment this
will contain the channel with which the store is associated. Valid values can be
found on the channels table.
  channel_name              VARCHAR2(120),  -- Contains the name of the channel.
```

```
  store_format_id             NUMBER(4,0),  -- Contains the number indicating the
format of the store. Valid values are found on the store format table.
  store_format_name           VARCHAR2(60),  -- Contains the name of the store
format.
  mall_name                   VARCHAR2(120),  -- Contains the name of the mall in
which the store is located.
  district                    NUMBER(10,0),  -- Contains the number of the
district in which the store is a member.
  transfer_zone               NUMBER(4,0),  -- Contains the transfer zone in which
the store is located. Valid values are located on the tsfzone table.
  transfer_zone_desc          VARCHAR2(120),  -- Contains the name of the Transfer
Zone.
  default_wh                  NUMBER(10,0),  -- Contains the number of the
warehouse that may be used as the default for creating crossdock masks. This
determines which stores are associated with or sourced from a warehouse. Will hold
only virtual warehouses in a multi-channel environment.
  stop_order_days             NUMBER(3,0),  -- Contains the number of days before
a store closing that the store will stop accepting orders.
  start_order_days            NUMBER(3,0),  -- Contains the number of days before
the store_open_date that the store will begin accepting orders.
  currency_code               VARCHAR2(3),  -- This field contains the currency
code under which the store operates.
  store_lang_iso_code         VARCHAR2(6),  -- This column identifies the language
to be used for the given store.
  tran_no_generate            VARCHAR2(6),  -- Contains the level at which unique
POS transaction numbers are generated. If the store has one sequence
number that is used for all registers, then the value in this columnwill be S
(Store) otherwise it will be R (Register).
  duns_number                 VARCHAR2(9),  -- This field holds the Dun and
Bradstreet number to identify the store.
  sister_store                NUMBER(10,0),  -- This field will hold a store
number which will be used to relate the current store to the historical data of an
existing store.
  tsf_entity_id               NUMBER(10,0),  -- This is the Id of the transfer
entity this store belongs to. A transfer entity is a group of locations that are a
part of single legal entity and share same accounting set of books.
  org_unit_id                 NUMBER(15,0),  -- Column will contain the
organizational unit ID value.
  auto_rcv                    VARCHAR2(1),  -- This column will indicate whether
the client is allowing automatic receipt for the store. Valid Values are Y (Yes),
N (No), D (System Default). Default value should be D.


  remerch_ind                 VARCHAR2(1),  -- Identifies stores that are
undergoing a significant remerchandising effort. Used only when AIP is integrated
with RMS.
  wf_customer                 NUMBER(10,0),  -- Numeric Id of the customer.
  timezone                    VARCHAR2(64),  -- Indicates the time zone of the
store. For example, America/New_York.
  customer_order_loc_ind      VARCHAR2(1),  -- This Column determines whether the
location is customer order location or not.
  rxmdi_control               VARCHAR2(20),
  last_update                 TIMESTAMP,
  CONSTRAINT rxmdi_store_p PRIMARY KEY (store_id)
);
```

rxmdi_store_addr

```
CREATE TABLE rxmdi_store_addr (
  store_id                     NUMBER(10,0)         NOT NULL DEFERRABLE INITIALLY
DEFERRED,  -- Contains the unique ID of the store.  Data will always be present in
this field.
  addr                         NUMBER(11)           NOT NULL,-- This column
contains a unique number used to distinguish between different addresses.
```

```
  addr_type                    VARCHAR2(2)          NOT NULL,  -- Contains the
code used to identify the address type.  Common examples include 01 (Business), 02
(Postal), 03 (Returns), 04 (Order), 05 (Invoice) and 06 (Remittance).  Data will
always be present in this field.
  addr_type_desc               VARCHAR2(20),  -- Contains the code used to
identify the address type.  Common Common examples include Business, Postal,
Returns, Order, Invoice and Remittance.  Description data is only sent in the
primary integration language of the system.
  primary_addr_ind             VARCHAR2(1)          NOT NULL,  -- Indicates
whether the address is the primary address for the address type.  Valid values are
Y and N.
  add_1                        VARCHAR2(240)        NOT NULL,  -- Contains the
first line of the address.  This information is required.
  add_2                        VARCHAR2(240),  -- This column contains the second
line of the address.
  add_3                        VARCHAR2(240),  -- This column contains the third
line of the address.
  city                         VARCHAR2(120)        NOT NULL,  -- Contains the
name of the city that is associated with the address.  This information is
required.
  county                       VARCHAR2(250),  -- This column holds the county
name for the location.
  state                        VARCHAR2(3),  -- This column contains the state
abbreviation for the address.
  country                      VARCHAR2(3)          NOT NULL,  -- Contains the ISO
3166-1 country code associated with the address.  This information is required.
  post_code                    VARCHAR2(30),  -- This column contains the zip code
for the address.
  jurisdiction_code            VARCHAR2(10),  -- This column contains the name of
the contact for the supplier at this address.
  contact_name                 VARCHAR2(120),  -- This column contains the name of
the contact for the supplier at this address.
  contact_phone                VARCHAR2(20),  -- This column contains the phone
number of the contact person at this address.
  contact_fax                  VARCHAR2(20),  -- This column contains the fax
number of the contact person at this address.
  contact_email                VARCHAR2(100),  -- This column contains the email
address of the partner or suppliers representative contact.
  rxmdi_control                VARCHAR2(20),
  last_update                  TIMESTAMP,
  CONSTRAINT rxmdi_store_addr_p PRIMARY KEY (store_id, addr) DEFERRABLE INITIALLY
DEFERRED
);
```

rxmdi_store_site

```
CREATE TABLE rxmdi_store_site(
  store_id           NUMBER(10) NOT NULL,
  site_id            VARCHAR2(40),
  catalog_id         VARCHAR2(40),
  price_list_id      VARCHAR2(40),
  sale_list_id       VARCHAR2(40),
  file_name          VARCHAR2(40),
  folder_id          VARCHAR2(40),
  CONSTRAINT rxmdi_store_site_p PRIMARY KEY(store_id)
);
```

rxmdi_wh

```
CREATE TABLE rxmdi_wh (
  wh_id                    NUMBER(10,0)       NOT NULL,--
```

```
Contains the number which uniquely identifies the warehouse.The wh table stores
all warehouses in the system.Both virtual and physical warehouses will be stored
on this table.The addition of the new column, physical_wh, helps determine which
warehouses are physical and which are virtual.All physical warehouses will have a
physical_wh column value equal to their wh number.Virtual warehouses will have a
valid physical warehouse in this column.
  wh_name                   VARCHAR2(150)      NOT NULL,
Contains the name of the warehouse which, along with the warehouse number,
identifies the warehouse.
  wh_name_secondary         VARCHAR2(150),-- Secondary name of the warehouse.
  email                     VARCHAR2(100),-- Holds the email address for the
location
  vat_region                NUMBER(4,0),-- warehouse is located.
  org_hier_type             NUMBER(4,0),-- Contains the organization type that
will be used in reporting purposes for the warehouse.  The type comes from the
organizational hierarchy.Valid values are:1 = Company 10 = Chain 20 = Area 30 =
Region 40 = District 50 = Store
  org_hier_value            NUMBER(10,0),-- Contains the code associated with the
specific organizational hierarchy type.  Valid values include the company number,
chain number, area number, etc.
  currency_code             VARCHAR2(3)        NOT NULL,  -- This field contains
the currency code under which the warehouse operates.
  physical_wh               NUMBER(10,0), -- This column will contain the number
of the physical warehouse that is assigned to the virtual warehouse.
  primary_vwh               NUMBER(10,0), -- This field holds the virtual
warehouse that will used as the basis for all transactions for which only a
physical warehouse and not a virtual warehouse has not been specified.
  channel_id                NUMBER(4,0), -- This column will contain the channel
for which the virtual warehouse will be assigned.
  stockholding_ind          VARCHAR2(1)        NOT NULL,-- This column will
indicate if the warehouse is a stock holding location.  In a non-multichannel
environment, this will always be Y.  In a multichannel environment it will be N
for a physical warehouse and Y for a virtual warehouse.
  break_pack_ind            VARCHAR2(1), -- Indicates whether or not the warehouse
is capable of distributing less than the supplier case quantity.   Valid values
are : Y or N.
  redist_wh_ind             VARCHAR2(1), -- Indicates that the warehouse is a Re-
Distribution warehouse.  Used as a location on Purchase Orders in place of actual
locations that are unknown at the time of Purchase Order creation and approval.
This value allows the Redistribution Report to identify orders requiring
redistribution.  A Warehouse with this indicator will not be limited in any RMS
transactions.   Valid values are Y or N.
  delivery_policy           VARCHAR2(6),-- Contains the delivery policy of the
warehouse.  Next Day indicates that the if a location is closed, the warehouse
will deliver on the next day.  Next Valid Delivery Day indicates that the
warehouse will wait until the next scheduled delivery day before delivering. Valid
values come from the DLVY code on code_head/code_detail.
  restricted_ind            VARCHAR2(1) , -- Indicator used to restrict virtual
warehouses from receiving stock during an inbound type transaction (ex. positive
SOH inventory adjustment, PO over-receipt) when stock needs to be prorated across
virtual warehouses within a physical warehouse because a virtual warehouse in the
physical warehouse has not been identified for the transaction.  The indicator
will restrict the virtual warehouse from receiving stock unless all the valid
virtual warehouses determined by the system are restricted, then the stock will be
distributed across those restricted virtual warehouses. This indicator will only
be used in a multi-channel environment.  It is always set to No in a single
channel environment.
```

```
  protected_ind             VARCHAR2(1),  -- Indicator used to determine if the
virtual warehouse is affected last in transactions where inventory is removed or
affected first in short-shipment type transactions where inventory is being added.
The indicator will be used in any outbound or inventory removal type transactions
(ex. RTVs, negative SOH inventory adjustments, etc.) when the system has to
distribute the transaction quantity across virtual warehouses within a physical
warehouse either because a virtual warehouse has not been specified or couldnt be
derived or if a virtual warehouse doesnt have enough stock to cover the
transaction quantity and stock needs to be pulled from other virtual warehouse
within the physical warehouse.  The indicator will also be used for inbound type
transactions where there is some sort of short-shipment ex. a short-shipment for a
PO).  The indicator will determine which virtual warehouses will have their order
quantity fulfilled first with the receipt quantity.  Note that this indicator does
not guarantee that stock will not be pulled from the virtual warehouse, it is only
used to ensure that the virtual warehouse is affected last. This indicator will
only be used in a multi-channel environment.  It is always set to No in a single
channel environment.
  forecast_wh_ind          VARCHAR2(1) , -- This indicator determines if a
warehouse is forecastable. The intent of this indicator is to restrict the
information being sent to RDF against which to generate forecasts.
  rounding_seq              NUMBER(10,0), -- This column determines which virtual
warehouses within a physical warehouse should be rounded together as well as
determining which virtual warehouse receives the additional stock or decreased
stock due to rounding. This value will be a virtual warehouse number. All
warehouses will the same rounding seq number will be rounded together with the
warehouse that equals the rounding seq receiving any extra stock.
  repl_ind                  VARCHAR2(1) , -- This indicator determines if a
warehouse is replenishable.
  repl_wh_link              NUMBER(10,0),-- This field holds the replenishable
warehouse that is linked to this virtual warehouse. This link implies that the
virtual warehouse is included in the net inventory calculations for the
replenishable warehouse.
  repl_src_ord              NUMBER(1,0), -- This field contains the order from
which the inventory is sourced from the linked warehouses.
  ib_ind                    VARCHAR2(1) ,-- This field indicates if the warehouse
is an investment buy warehouse.
  ib_wh_link                NUMBER(10,0), -- This field contains the investment
buy warehouse that is linked to the virtual warehouse. This link implies that the
virtual warehouse is included in the net inventory calculations for the investment
buy warehouse.
  auto_ib_clear             VARCHAR2(1) ,-- This indicator determines if the
investment buys inventory should be automatically transferred to the turn
(replenishable) warehouse when an order is received by the turn warehouse.
  duns_number               VARCHAR2(9),-- This field holds the Dun and Bradstreet
number to identify the warehouse
  duns_loc                  VARCHAR2(4), -- This field holds the Dun and
Bradstreet number to identify the location
  tsf_entity_id             NUMBER(10,0), -- ID of the transfer entity with which
this warehouse is associated. Valid values are found on the TSF_ENTITY table. A
transfer entity is a group of locations that share legal requirements around
product management.
  finisher_ind              VARCHAR2(1),-- Yes/No value which indicates if this
virtual warehouse is an internal finisher.
  inbound_handling_days     NUMBER(2,0) , -- Warehouse inbound handling days are
defined as the number of days that the warehouse requires to receive any item and
get it to the shelf so that it is ready to pick.
  org_unit_id               NUMBER(15,0), -- this column will hold the oracle
oraganizational unit id value.
  vwh_type                  VARCHAR2(6), -- This attribute will be included in the
location downloads to AIP.
```

```
  org_entity_type           VARCHAR2(1), -- This is the new column that will
specify if the warehouse is a legal entity  (Importer, Exporter) or a regular
warehouse. Valid values are: R - regular warehouse (including finisher); M -
importer; X - exporter. Default value is R.
  rxmdi_control             VARCHAR2(20),  -- This column indicates the status of
the current row data. Possible values are 'N'- New, 'P' - In Process, 'E' - Error,
S - Success, I- Ignored.
  last_update               TIMESTAMP, -- This column contains the timestamp of
when the data in the row has been updated.
  CONSTRAINT rxmdi_wh_p PRIMARY KEY (wh_id)
);
```

rxmdi_wh_addr

```
CREATE TABLE rxmdi_wh_addr (
  wh_id                 NUMBER(10,0)        NOT NULL DEFERRABLE INITIALLY
DEFERRED,-- Contains the unique ID of the wh.  Data will always be present in this
field.
  addr                  NUMBER(11)          NOT NULL, -- This column contains a
unique number used to distinguish between different addresses.
  addr_type             VARCHAR2(2)         NOT NULL,  -- Contains the code used to
identify the address type.  Common examples include 01 (Business), 02 (Postal), 03
(Returns), 04 (Order), 05 (Invoice) and 06 (Remittance).  Data will always be
present in this field.
  addr_type_desc        VARCHAR2(20),-- Contains the code used to identify the
address type.  Common Common examples include Business, Postal, Returns, Order,
Invoice and Remittance.  Description data is only sent in the primary integration
language of the system.
  primary_addr_ind      VARCHAR2(1)         NOT NULL, -- Indicates whether the
address is the primary address for the address type.  Valid values are Y and N.
  add_1                 VARCHAR2(240)       NOT NULL,-- Contains the first line of
the address.  This information is required.
  add_2                 VARCHAR2(240),-- This column contains the second line of
the address.
  add_3                 VARCHAR2(240), -- This column contains the third line of
the address.
  city                  VARCHAR2(120)      NOT NULL, -- Contains the name of the
city that is associated with the address.  This information is required.
  county                VARCHAR2(250),-- This column holds the county name for the
location.
  state                 VARCHAR2(3),-- This column contains the state abbreviation
for the address.
  country               VARCHAR2(3)        NOT NULL,-- Contains the ISO 3166-1
country code associated with the address.  This information is required.
  post_code             VARCHAR2(30), -- This column contains the zip code for the
address.
  jurisdiction_code     VARCHAR2(10), -- Identifies the jurisdiction code for the
country-state relationship.
  contact_name          VARCHAR2(120),-- This column contains the name of the
contact for the supplier at this address.
contact_phone         VARCHAR2(20), -- This column contains the phone number of the
contact person at this address.
  contact_fax           VARCHAR2(20),-- This column contains the fax number of the
contact person at this address.
  contact_email         VARCHAR2(100),-- This column contains the email address of
the partner or suppliers representative contact.
  rxmdi_control         VARCHAR2(20), -- This column indicates the status of the
current row data. Possible values are 'N'- New, 'P' - In Process, 'E' - Error, S -
Success, I- Ignored.
  last_update           TIMESTAMP, -- This column contains the timestamp of when
the data in the row has been updated.
  CONSTRAINT rxmdi_wh_addr_p PRIMARY KEY (wh_id, addr)DEFERRABLE INITIALLY
DEFERRED
```

```
);
```

rxmdi_diff

```
CREATE TABLE rxmdi_diff(
    diff_id            VARCHAR2(10)        NOT NULL, -- Contains the unique ID of
the diff. Data will always be present in this field.
    diff_desc          VARCHAR2(120)       NOT NULL,-- Contains the type code for
the diff. All diffs belong to one and only one type.
    diff_type          VARCHAR2(6)         NOT NULL, -- This field will hold a
value of the types of differentiators contained in this differentiator group,such
as S -size, C -color, F -flavor, E -scent, P -pattern. Valid values are stored in
the DIFF_TYPE table.
    diff_type_desc     VARCHAR2(120)       NULL, -- Contains the description of
the diff.  Data will always exist in this field.
    industry_code      VARCHAR2(10)        NULL,-- Optionally can hold the unique
code used by industry standards to identify the differentiator. For example, in
the US, the National Retail Federation defines a standard Color and Size Codes
that gives retailers, vendors and manufacturers a common language for product
color and size identification for EDI purposes. For example, mens pants size
combination 32x32 has a NRF code number 10492.
    industry_subgroup  VARCHAR2(10)        NULL,-- Optionally can hold a sub-
grouping code used by industry standards to further identify the differentiator.
For example, in the US, the National uses a subgroup for colors (e.g. purple is
defined as 500; dark purple represents a range from 501 - 509, medium purple
represents a range from 510 - 519, bright purple represents a range from 520 -
529, etc.).
    rxmdi_control      VARCHAR2(20)        NULL,-- Controller/Flag for RXMDI to
determine state of record
    last_update        TIMESTAMP           NULL,--Timestamp of when the row is
updated
    CONSTRAINT rxmdi_diff_p PRIMARY KEY(diff_id)
);
```

rxmdi_diff_grp

```
CREATE TABLE rxmdi_diff_grp (
    diff_group_id        VARCHAR2(10)          NOT NULL,-- This field will hold a
unique number id for the differentiator group.  Data will always be present in
this field.
    diff_group_desc      VARCHAR2(120)         NOT NULL, -- Description of the
differentiator group (for example: Mens Shirt Sizes, Womens Shoe Sizes, Girls
Dress Sizes, Shower Gel Scents, Yogurt Flavors, etc.).  Description data is only
sent in the primary integration language of the system.
    diff_type_id         VARCHAR2(6)           NOT NULL,-- This field will hold a
value of the types of differentiators contained in this differentiator group,
including but not limited to: S (size), C (color), F (flavor), E (scent), P
(pattern).
    diff_type_desc       VARCHAR2(120)         NULL, -- Contains the description
of the differentiator type.  For Example:  Size, Color, Flavor, Scent, Pattern.
Description data is only sent in the primary integration language of the system.
rxmdi_control        VARCHAR2(20)          NULL,-- Controller/Flag for RXMDI to
determine state of record
    last_update          TIMESTAMP             NULL,--Timestamp of when the row is
updated
    CONSTRAINT diff_grp_p PRIMARY KEY(diff_group_id)
);
```

rxmdi_diff_grp_dtl

```
CREATE TABLE rxmdi_diff_grp_dtl(
    diff_group_id          VARCHAR2(10)        NOT NULL,-- This field will hold a
unique id for the differentiator group.  Data will always be present in this
field.
```

```
    diff_id               VARCHAR2(10)        NOT NULL,-- This field will hold a
unique id for the diff that is a member of this diff group.  Data will always be
present in this field.
    rxmdi_control         VARCHAR2(20)        NULL,-- Controller/Flag for RXMDI
to determine state of record
    last_update           TIMESTAMP           NULL,--Timestamp of when the row is
updated
    CONSTRAINT pk_diff_grp_dtl PRIMARY KEY(diff_group_id,diff_id)
);
```

rxmdi_item_hdr

```
CREATE TABLE rxmdi_item_hdr(
    item                        VARCHAR2(25)     NOT NULL,
    item_parent                 VARCHAR2(25)     NULL,
    item_grandparent            VARCHAR2(25)     NULL,
    pack_ind                    VARCHAR2(1)      NULL,
    simple_pack_ind             VARCHAR2(1)      NULL,
    item_level                  NUMBER(1,0)      NULL,
    tran_level                  NUMBER(1,0)      NULL,
    inventory_ind               VARCHAR2(1)      NULL,
    diff_1_level                VARCHAR2(6)      NULL,
    diff_1_type                 VARCHAR2(6)      NULL,
    diff_1                      VARCHAR2(10)     NULL,
    diff_2_level                VARCHAR2(6)      NULL,
    diff_2_type                 VARCHAR2(6)      NULL,
    diff_2                      VARCHAR2(10)     NULL,
    diff_3_level                VARCHAR2(6)      NULL,
    diff_3_type                 VARCHAR2(6)      NULL,
    diff_3                      VARCHAR2(10)     NULL,
    diff_4_level                VARCHAR2(6)      NULL,
    diff_4_type                 VARCHAR2(6)      NULL,
    diff_4                      VARCHAR2(10)     NULL,
    dept                        NUMBER(4,0)      NULL,
    unique_class                NUMBER(10,0)     NULL,
    class                       NUMBER(4,0)      NULL,
    unique_subclass             NUMBER(10,0)     NULL,
    subclass                    NUMBER(4,0)      NULL,
    status                      VARCHAR2(1)      NULL,
    description                 VARCHAR2(250),
    secondary_item_desc         VARCHAR2(250),
    short_desc                  VARCHAR2(120),
    brand_name                  VARCHAR2(30),
    merchandise_ind             VARCHAR2(1),
    primary_ref_item_ind        VARCHAR2(1) ,
    cost_zone_group_id          NUMBER(4,0),
    standard_uom                VARCHAR2(4),
    uom_conv_factor             NUMBER(20,4),
    package_size                NUMBER(12,4),
    package_uom                 VARCHAR2(4),
    store_order_multiple        VARCHAR2(1),
forecast_ind                VARCHAR2(1),
    currency_code               VARCHAR2(3),
    original_unit_retail        NUMBER(20,4),
    mfg_rec_retail              NUMBER(20,4),
    retail_label_type           VARCHAR2(6),
    retail_label_value          NUMBER(20,4),
    item_aggregate_ind          VARCHAR2(1) ,
    diff_1_aggregate_ind        VARCHAR2(1) ,
    diff_2_aggregate_ind        VARCHAR2(1) ,
    diff_3_aggregate_ind        VARCHAR2(1) ,
    diff_4_aggregate_ind        VARCHAR2(1) ,
    item_number_type            VARCHAR2(6) NOT NULL,
```

```
    format_id                   VARCHAR2(6),
    prefix                      NUMBER(2,0),
    rec_handling_temp           VARCHAR2(6),
    rec_handling_sens           VARCHAR2(6),
    perishable_ind              VARCHAR2(1),
    waste_type                  VARCHAR2(6),
    waste_pct                   VARCHAR2(6),
    default_waste_pct           NUMBER(12,4),
    constant_dim_ind            VARCHAR2(1),
    contains_inner_ind          VARCHAR2(1),
    sellable_ind                VARCHAR2(1) ,
    orderable_ind               VARCHAR2(1) ,
    pack_type                   VARCHAR2(1),
    order_as_type               VARCHAR2(1),
    item_service_level          VARCHAR2(6),
    gift_wrap_ind               VARCHAR2(1) ,
    ship_alone_ind              VARCHAR2(1) ,
    item_form_ind               VARCHAR2(1),
    catch_weight_ind            VARCHAR2(1) ,
    catch_weight_type           VARCHAR2(1),
    catch_weight_order_type     VARCHAR2(6),
    catch_weight_sale_type      VARCHAR2(6),
    catch_weight_uom            VARCHAR2(4),
    deposit_item_type           VARCHAR2(6),
    container_item              VARCHAR2(25),
    deposit_in_price_per_uom    VARCHAR2(6),
    soh_inquiry_at_pack_ind     VARCHAR2(1),
    notional_pack_ind           VARCHAR2(1),
    comments                    VARCHAR2(2000),
    rxmdi_control               VARCHAR2(20),
    last_update                 TIMESTAMP,
    CONSTRAINT rxmdi_item_hdr_p PRIMARY KEY (item)
);
```

rxmdi_item_image

```
CREATE TABLE rxmdi_item_image (
    item            VARCHAR2(25)         NOT NULL,-- This field contains the
unique alphanumeric identifier for the item, the image is for.
    image_name      VARCHAR2(120)        NOT NULL,-- This field contains the name
of the image of the item.
    image_addr      VARCHAR2(255)        NOT NULL,  -- This field contains the
actual path where the file of the image of the item is stored.
    image_desc      VARCHAR2(40)         NULL,-- This field contains the
description associated with the image of the item.
    image_type      VARCHAR2(6)          NULL, -- This field contains the type of
the image of the item.  Valid values are defined as member of IITD code type.
    primary_ind     VARCHAR2(1)          NULL, -- This field will indicate
whether this record is the primary image of the item or not.  Valid values are
Y(es) and N(o) only.Default to N value if left blank or set as NULL.
display_priority NUMBER(4,0)          NULL, -- This field will specify the display
sequence order of images associated to the item per priority.
    rxmdi_control   VARCHAR2(20)         NULL,
    last_update     TIMESTAMP            NULL,
    CONSTRAINT rxmdi_item_image_p PRIMARY KEY (item,image_name)
);
```

rxmdi_item_loc

```
CREATE TABLE RXMDI_ITEM_LOC (
  loc_type                    VARCHAR2(1)             NOT NULL,  -- Describes the
type of location.  Valid values include S (store), W (warehouse) and E (external
finisher).  Data will always be present in this field.
```

```
  location                    NUMBER(10,0)           NOT NULL,  -- Numeric ID of
location.  The intersection of location and item is a distinct entity.  Data will
always be present in this field.
  item                        VARCHAR2(25)           NOT NULL,  -- ID of item.
The intersection of location and item is a distinct entity.  Data will always be
present in this field.
  item_parent                 VARCHAR2(25),  -- ID identifies the item/group at
the level above the item. This value must exist as an item in another row on the
item_master table.
  item_grandparent            VARCHAR2(25),  -- identifies the item/group two
levels above the item. This value must exist as both an item and an item parent in
another row on the item_master table.
  currency_code               VARCHAR2(3),  -- This field contains the currency
code under which the store/wh operates.
  initial_unit_retail         NUMBER(20,4),  -- Contains the unit retail price in
the standard unit of measure for the item/location combination. This field is
stored in the local currency.
  selling_unit_retail         NUMBER(20,4),  -- Contains the unit retail price in
the selling unit of measure for the item/location combination. This field is
stored in the local currency.
  selling_uom                 VARCHAR2(4),  -- Contains the selling unit of
measure for an items single-unit retail.
  taxable_ind                 VARCHAR2(1),  -- Indicates if item is taxable at
the store.
  local_item_desc             VARCHAR2(250)          NOT NULL,  -- Contains the
local description of the item.  This may be the same as the primary description of
the item, a regional description of the item (e.g. jimmies vs sprinkles in the US
or roll vs bap vs cob vs bun in the UK), or a value in a local language (e.g.
Overlay dress true black knit at US stores vs Lagenkleid - Strick, tiefschwarz at
stores in Germany).  The intent is that this string is appropriate to print
description on signage/receipts at this location.
  local_short_desc            VARCHAR2(120),  -- Contains the local short
description of the item.
  ti                          NUMBER(12,4),  -- Number of shipping units (cases)
that make up one tier of a pallet.
  hi                          NUMBER(12,4),  -- Number of tiers that make up a
complete pallet (height).
  store_order_multiple        VARCHAR2(1),  -- This column contains the multiple
in which the item needs to be shipped from a warehouse to the location.
  status                      VARCHAR2(1),  -- Current status of item at the
store.
  daily_waste_pct             NUMBER(12,4),  -- Average percentage lost from
inventory on a daily basis due to natural wastage.
  measure_of_each             NUMBER(12,4),  -- Size of an each in terms of the
uom_of_price. For example 12 oz. Used in ticketing.
  measure_of_price            NUMBER(12,4),  -- Size to be used on the ticket in
terms of the uom_of_price.
  uom_of_price                VARCHAR2(4),  -- Unit of measure that will be used
on the ticket for this item.
  primary_variant             VARCHAR2(25),  -- This field is used to address
sales of PLUs (i.e. above transaction level items) when inventory is tracked at a
lower level (i.e. UPC). This field will only contain a value for items one level
higher than the transaction level.
  primary_cost_pack           VARCHAR2(25),  -- This field contains an item
number that is a simple pack containing the item in the item column for this
record.
  primary_supplier            NUMBER(10,0),  -- Numeric identifier of the
supplier who will be considered the primary supplier for the specified item/loc.
  primary_origin_country      VARCHAR2(3),  -- Contains the identifier of the
origin country which will be considered the primary country for the specified
item/location.
```

```
  receive_as_type              VARCHAR2(2),  -- This column determines whether the
stock on hand for a pack component item or the buyer pack itself will be updated
when a buyer pack is received at a warehouse.
  inbound_handling_days        NUMBER(2,0),  -- This field indicates the number of
inbound handling days for an item at a warehouse type location.
  source_method                VARCHAR2(1),  -- This value will be used to specify
how the adhoc PO/TSF creation process should source the item/location request.
  source_wh                    NUMBER(10,0),  -- This value will be used by the
ad-hoc PO/Transfer creation process to determine which warehouse to fill the
stores request from.
  uin_type                     VARCHAR2(6),  -- This column will contain the
unique identification number (UIN) used to identify the instances of the item at
the location.
  uin_label                    VARCHAR2(6),  -- This column will contain the label
for the UIN when displayed in SIM.
  capture_time_in_proc         VARCHAR2(6),  -- This column will indicate when the
UIN should be captured for an item during transaction processing.
  ext_uin_ind                  VARCHAR2(1),  -- EXT_UIN_IND This Yes/No indicator
indicates if UIN is being generated in the external system.
  intentionally_range_ind      VARCHAR2(1),  -- This column determines if the
location is ranged intentionally by the user for replenishment/selling or
incidentally ranged by the RMS programs when item is not ranged to a specific
location on the transaction.
  costing_location             NUMBER(10,0),  -- Numeric identifier of the costing
location for the franchise store. This field may contain a store or a warehouse.
  costing_loc_type             VARCHAR2(1),  -- This field holds the type of
costing location in the costing location field.
  launch_date                  TIMESTAMP,  -- Holds the date that they item should
first be sold at the location.
  qty_key_options              VARCHAR2(6),  -- Determines whether the qty key on
a POS should be used for this item at the location.
  manual_price_entry           VARCHAR2(6),  -- Determines whether the price
can/should be entered manually on a POS for this item at the location.
  deposit_code                 VARCHAR2(6),  -- Indicates whether a deposit is
associated with this item at the location.
  food_stamp_ind               VARCHAR2(1),  -- Indicates whether the item is
approved for food stamps at the location. This value will be downloaded to the
POS.
  wic_ind                      VARCHAR2(1),  -- Indicates whether the item is
approved for WIC at the location. This value will be downloaded to the POS.
  proportional_tare_pct        NUMBER(12,4),  -- Holds the value associated of the
packaging in items sold by weight at the location.
  fixed_tare_value             NUMBER(12,4),  -- Holds the value associated of the
packaging in items sold by weight at the location.
  fixed_tare_uom               VARCHAR2(4),  -- Holds the unit of measure value
associated with the tare value. The only processing RMS does involving the fixed
tare value and UOM is downloading it to the POS.
  reward_eligible_ind          VARCHAR2(1),  -- Holds whether the item is legally
valid for various types of bonus point/award programs at the location.
  natl_brand_comp_item         VARCHAR2(25),  -- Holds the nationally branded item
to which you would like to compare the current item.
return_policy                VARCHAR2(6),  -- Holds the return policy for the item
at the location. Valid values for this field belong to the code_type RETP.
  stop_sale_ind                VARCHAR2(1),  -- Indicates that sale of the item
should be stopped immediately at the location (i.e. in case of recall etc).
  elect_mtk_club               VARCHAR2(6),  -- Holds the code that represents the
marketing clubs to which the item belongs at the location.
  report_code                  VARCHAR2(6),  -- Code to determine which reports
the location should run.
  req_shelf_life_on_selection  NUMBER(4,0),  -- Holds the required shelf life for
an item on receipt in days.
  ib_shelf_life                NUMBER(4,0),  -- This column will hold the
Investment Buyspecific shelf life for the item/location
```

```
  store_orderable_ind          VARCHAR2(1),  -- STORE_REORDERABLE_IND Indicates
whether the store may re-order the item. This field is required to be either= Y -
yes or N - no. The field will default to N. No RMS processing is based on the
value in this field.
  rack_size                    VARCHAR2(6),  -- Indicates the rack size that
should be used for the item. This field is not required. Valid values for the
field can be found and defined in the code_type RACK.
  full_pallet_item             VARCHAR2(1),  -- Indicates whether a store must
reorder an item in full pallets only.
  in_store_market_basket       VARCHAR2(6),  -- Holds the in store market basket
code for this item/location combination. Valid values for the field can be found
in the code_type STMB.
  storage_location             VARCHAR2(7),  -- Holds the current storage location
or bin number for the item at the location. No RMS processing is based on the
value in this field.
  alt_storage_location         VARCHAR2(7),  -- Holds the preferred alternate
storage location or bin number for the item at the location.
  returnable_ind               VARCHAR2(1),  -- This field will contain a value of
Yes when the item can be returned to the location
  refundable_ind               VARCHAR2(1),  -- This field will contain a value of
Yes when the item is refundable at the location.
  backorder_ind                VARCHAR2(1),  -- This field will contain a value of
Yes when the item can be back ordered to the location
  merchandise_ind              VARCHAR2(1),  -- Indicates if the item is a
merchandise item (Y, N).
  rxmdi_control                VARCHAR2(20),
  last_update                  TIMESTAMP,
  CONSTRAINT rxmdi_item_loc_p PRIMARY KEY (item, location)
);
```

rxmdi_merch_hier

```
 CREATE TABLE rxmdi_merch_hier(
  hierarchy_level                  VARCHAR2(10)            NOT NULL,  -- This
information identifies the level of the merchandise hierarchy that is described by
this record.  Value is always DIVISION, GROUP, DEPT, CLASS, SUBCLASS. This field
can not be null.
  hierarchy_id                     VARCHAR2(20)            NOT NULL, --Prefix plus
Id. Example for department 3 hierarchyId will be d3 . For department 1 and class 2
hierarchyId will be d1c2
  hierarchy_node_id                NUMBER(10,0)            NULL,  -- This
information identifies the the node of the merchandise hierarchy that is described
by this record. This field can not be null.  HierarchyNodeId is only unique within
an HierarchyLevel (meaning it is possible, for example, that there is both a
DIVISION 1 and a GROUP 1 in the full merchandise hierarchy).
  hierarchy_node_name              VARCHAR2(150)           NULL,  -- Name of the
merchandise hierarchy entity.  Description data is only sent in the primary
integration language of the system.
parent_level                      VARCHAR2(10)           NOT NULL,  -- Level of
the merchandise hierarchy above the current node. Both ParentLevel and ParentId
are should be evaluated to correctly traverse the hierarchy
  parent_node_id                   NUMBER(10,0)            NULL,  -- Id of the
level of the merchandise hierarchy above the current node. Both ParentLevel and
ParentNodeId are should be evaluated to correctly traverse the hierarchy.
  grandparent_merch_display_id     VARCHAR2(20)            NULL,  -- Only
populated for SUBCLASS entities.  For subclasses, this column will hold the
department ID used for display purposes in RMS (department is the grandparent of
subclass).  Note that in RMS, dept, class and subclass display IDs are combined to
form a composite unique key.  Every department can have a class 1.  Every class in
Department 1000 can have a subclass 1.  Looking only at the display ids, all three
values are required for uniqueness.  Node that for subclasses, the HierarchyNodeId
is unique.  It is a non-displayed, unique value that emilinates the need for a
composite key.
```

```
  parent_merch_display_id          VARCHAR2(20)            NULL,  -- Only
populated for CLASS and SUBCLASS entities.  For classes, this column holds the
department ID used for display purposes in RMS (department is the parent of class)
For subclasses, this column holds the class ID used for display.  Note that in
RMS, dept, class and subclass display IDs are combined to form a composite unique
key.  Every department can have a class 1.  Every class in Department 1000 can
have a subclass 1.  Looking only at the display ids, all three values are required
for uniqueness.  Node that for subclasses and classes, the HierarchyNodeId is
unique.  It is a non-displayed, unique value that emilinates the need for a
composite key.
  merch_display_id                 VARCHAR2(20)            NOT NULL,  -- Only
populated for DEPARTMENT, CLASS and SUBCLASS entities.  For departments, this
column holds the department display id.  For subclasses, this column holds the
subclass display id.  Note that in RMS, dept, class and subclass display IDs are
combined to form a composite unique key.  Every department can have a class 1.
Every class in Department 1000 can have a subclass 1.  Looking only at the display
ids, all three values are required for uniqueness.
  purchase_type                    NUMBER(1)               NULL,  -- Contains a
code which indicates whether items in this department are normal merchandise,
consignment stock or concession items. Valid values are: 0 = Normal Merchandise, 1
= Consignment Stock, 2 = Concession Items
  rxmdi_control                    VARCHAR2(20)            NULL,
  last_update                      TIMESTAMP               NULL,
  CONSTRAINT rxmdi_merch_hier_p PRIMARY KEY (hierarchy_level, hierarchy_id)
);
```

rxmdi_rltd_itm

```
CREATE TABLE rxmdi_rltd_itm (
  relationship_id              NUMBER(20,0)           NOT NULL,  -- Unique
identifier for each relationship header.  Data will always exist in this field.
  item                         VARCHAR2(25)           NOT NULL,  -- Item for
which the relationships are defined.  Data will always exist in this field.
  relationship_name            VARCHAR2(255)          NOT NULL,  -- Description
of the relationship.  Data will always exist in this field.
  relationship_type            VARCHAR2(6)            NOT NULL,  -- Describes the
type of relationship.Valid values include: CRSL (Cross Sell), SUBS (Substitution),
UPSL (Up-sell).
  mandatory_ind                VARCHAR2(1)            NOT NULL,  -- This field
indicates whether the relationship should be mandatory.  For example, an item like
a laptop may have a mandatory cross sell relationship.  The related items could be
different power cords for the US, UK, Mainland Europe, India, etc. When the laptop
is sold, it should be mandatory that one of the related power cords also be
selected.   Generally, only cross sell relationships are mandatory.  Substitution
and upsell relationships can be defined as mandatory, but in those cases, the
definition of mandatory is at the discretion of the client and generally means
that substitution or upsell must, as business process, be offered to consumers.
  rxmdi_control                VARCHAR2(20),
last_update                  TIMESTAMP,
  CONSTRAINT rxmdi_rltd_itm_p PRIMARY KEY (relationship_id)
);
```

rxmdi_rltd_itm_dtl

```
CREATE TABLE rxmdi_rltd_itm_dtl (
  relationship_id               NUMBER(20,0)              NOT NULL DEFERRABLE
INITIALLY DEFERRED,  -- Unique identifier for each relationship header.  Data will
always exist in this field.
  related_item                  VARCHAR2(25)              NOT NULL,  -- Item id
of the related item.  This is the item that should be Cross Sold, Substituted, or
Up Sold when the item on the parent record is sold.
  priority                      NUMBER(4,0),  -- Applicable only in case of
relationship type SUBS. In case of multiple related substitute items, this column
could be used (optional) to define relative priority.
```

```
  start_date                      TIMESTAMP,  -- From this date related item can
be used on transactions.
  end_date                        TIMESTAMP,  -- Till this date related item can
be used on transactions. A value of null means that it is effective forever.
  rxmdi_control                   VARCHAR2(20),
  last_update                     TIMESTAMP,
  CONSTRAINT rxmdi_rltd_itm_dtl_p PRIMARY KEY (relationship_id,
related_item)DEFERRABLE INITIALLY DEFERRED
);
```

rxmdi_pack_item

```
CREATE TABLE rxmdi_pack_item (
  pack_no                     VARCHAR2(25)            NOT NULL,  -- Alphanumeric
value that uniquely identifies the pack for which details are held in this table.
  seq_no                      NUMBER(4,0)             NULL,      -- Contains a
sequence number used to uniquely identify a row in the PACKITEM table.
  item                        VARCHAR2(25),                      -- Alphanumeric
value that identifies the component item within the pack. If pack item is created
using pack template then the component items are stored in the PACKITEM_BREAKOUT
table and this field is null.
  item_parent                 VARCHAR2(25),                      -- This field
contains the parent item (if any) associated  with the component item of the pack.
  pack_tmpl_id                NUMBER(8,0),                       -- Contains the
pack template ID associated with the pack item.
  pack_qty                    NUMBER(12,4)            NOT NULL,  -- Contains the
quantity of component items within the pack. If the pack item is created using a
pack template then the quantity specified here is 1 and the actual component item
quantities in the pack are stored in PACKITEM_BREAKOUT table.
  rxmdi_control               VARCHAR2(20),
  last_update                 TIMESTAMP,
  CONSTRAINT rxmdi_pack_item_p PRIMARY KEY (pack_no, item)
);
```

RIB XSD to RXMDI Staging Scheme Mapping

In the following tables, RIB XSD is in the left column and the RXMDI Staging schema is in the right column.

StoreDesc	rxmdi_store
store	store_id
store_type	store_type
store_name	store_name
store_name10	store_name_10
store_name3	store_name_abbr
store_class	store_class_id
store_mgr_name	manager
store_open_date	open_date
store_close_date	close_date
acquired_date	acquire_date
fax_number	fax_number
email	email

StoreDesc	rxmdi_store
total_square_ft	total_sq_feet
selling_square_ft	selling_sq_feet
linear_distance	linear_distance
stockholding_ind	stock_holding_ind
channel_id	channel_id
store_format	store_format_id
mall_name	mall_name
district	district
transfer_zone	transfer_zone
description	transfer_zone_desc
default_wh	default_Wh
stop_order_days	stop_order_days
start_order_days	start_order_days
currency_code	currency_code
lang	store_lang_iso_code
duns_number	duns_number
org_unit_id	org_unit_id
timezone_name	timezone
	rxmdi_control
	last_update

AddrDesc	rxmdi_store_addr
addr	addr
addr_type	addr_type
primary_addr_ind	primary_addr_ind
add_1	add_1
add_2	add_2
add_3	add_3
city_id	city
country_id	county
state_name	state
country_name	country
post	post_code
jurisdiction_code	jurisdiction_code

AddrDesc	rxmdi_store_addr
contact_name	contact_name
contact_phone	contact_phone
contact_fax	contact_fax
contact_email	contact_email
	rxmdi_control
	last_update

WhDesc	rxmdi_wh
wh	wh_id
wh_name	wh_name
email	email
currency_code	currency_code
physical_wh	physical_wh
channel_id	channel_id
stockholding_ind	stockholding_ind
break_pack_ind	break_pack_ind
redist_wh_ind	redist_wh_ind
delivery_policy	delivery_policy
duns_number	duns_number
duns_loc	duns_loc
org_unit_id	org_unit_id
	rxmdi_control
	last_update

AddrDesc	rxmdi_wh_addr
addr	addr
addr_type	addr_type
primary_addr_ind	primary_addr_ind
add_1	add_1
add_2	add_2
add_3	add_3
city_id	city
country_id	county

AddrDesc	rxmdi_wh_addr
state_name	state
country_name	country
post	post_code
jurisdiction_code	jurisdiction_code
contact_name	contact_name
contact_phone	contact_phone
contact_fax	contact_fax
contact_email	contact_email
	rxmdi_control
	last_update

ItemHdrDesc	rxmdi_item_hdr
item	item
item_parent	item_parent
item_grandparent	item_grandparent
pack_ind	pack_ind
simple_pack_ind	simple_pack_ind
item_level	item_level
tran_level	tran_level
inventory_ind	inventory_ind
diff_1	diff_1
diff_1_type	diff_1_type
diff_2	diff_2
diff_2_type	diff_2_type
diff_3	diff_3
diff_3_type	diff_3_type
diff_4	diff_4
diff_4_type	diff_4_type
dept	dept
class	Class
subclass	subclass
status	status
item_desc	description
short_desc	short_desc

ItemHdrDesc	rxmdi_item_hdr
brand	brand_name
merchandise_ind	merchandise_ind
primary_ref_item_ind	primary_ref_item_ind
cost_zone_group_id	cost_zone_group_id
standard_uom	standard_uom
uom_conv_factor	uom_conv_factor
package_size	package_size
package_uom	package_uom
store_ord_mult	store_order_multiple
forecast_ind	forecast_ind
mfg_rec_retail	mfg_rec_retail
retail_label_type	retail_label_type
retail_label_value	retail_label_value
item_number_type	item_number_type
format_id	format_id
prefix	prefix
handling_temp	rec_handling_temp
handling_sensitivity	rec_handling_sens
perishable_ind	perishable_ind
waste_type	waste_type
waste_pct	waste_pct
default_waste_pct	default_waste_pct
const_dimen_ind	constant_dim_ind
contains_inner_ind	contains_inner_ind
sellable_ind	sellable_ind
orderable_ind	orderable_ind
pack_type	pack_type
order_as_type	order_as_type
item_service_level	item_service_level
gift_wrap_ind	gift_wrap_ind
ship_alone_ind	ship_alone_ind
item_xform_ind	item_form_ind
catch_weight_ind	catch_weight_ind
deposit_item_type	deposit_item_type

ItemHdrDesc	rxmdi_item_hdr
container_item	container_item
deposit_in_price_per_uom	deposit_in_price_per_uom
soh_inquiry_at_pack_ind	soh_inquiry_at_pack_ind
notional_pack_ind	notional_pack_ind
comments	comments
Not available	rxmdi_control
Not available	last_update

ItemLocDesc	rxmdi_item_loc
loc_type	loc_type
loc	location
item	item
	item_parent
	item_grandparent
	currency_code
unit_retail	initial_unit_retail
selling_unit_retail	selling_unit_retail
selling_uom	selling_uom
taxable_ind	taxable_ind
local_item_desc	local_item_desc
local_short_desc	local_short_desc
	ti
	hi
	store_order_multiple
status	status
	daily_waste_pct
	measure_of_each
	measure_of_price
	uom_of_price
	primary_variant
	primary_cost_pack
primary_supp	primary_supplier
primary_cntry	primary_origin_country
receive_as_type	receive_as_type

ItemLocDesc	rxmdi_item_loc
	inbound_handling_days
source_method	source_method
source_wh	source_wh
uin_type	uin_type
uin_label	uin_label
capture_time	capture_time_in_proc
ext_uin_ind	ext_uin_ind
ranged_ind	intentionally_range_ind
	costing_location
	costing_loc_type
	launch_date
	qty_key_options
	manual_price_entry
	deposit_code
	food_stamp_ind
	wic_ind
	proportional_tare_pct
	fixed_tare_value
	fixed_tare_uom
	reward_eligible_ind
	natl_brand_comp_item
	return_policy
	stop_sale_ind
	elect_mtk_club
	report_code
	req_shelf_life_on_selection
	ib_shelf_life
	store_orderable_ind
	rack_size
	full_pallet_item
	in_store_market_basket
	storage_location
	alt_storage_location
returnable_ind	returnable_ind

ItemLocDesc	rxmdi_item_loc
	refundable_ind
	backorder_ind
	merchandise_ind
	rxmdi_control
	last_update

ItemImageDesc	rxmdi_item_image
item	item
image_name	image_name
image_addr	image_addr
	image_desc
image_type	image_type
primary_ind	primary_ind
display_priority	display_priority
	rxmdi_control
	last_update

DiffDesc	rxmdi_diff
diff_id	diff_id
diff_desc	diff_desc
diff_type	diff_type
	diff_type_desc
industry_code	industry_code
industry_subcode	industry_subgroup
	rxmdi_control
	last_update

DiffGrpHdrDesc	rxmdi_diff_grp
diff_group_id	diff_group_id
diff_group_desc	diff_group_desc
diff_group_type	diff_type_id
	diff_type_desc
	rxmdi_control

DiffGrpHdrDesc	rxmdi_diff_grp
	last_update

DiffGrpDtlDesc	rxmdi_diff_grp_dtl
diff_group_id	diff_group_id
diff_id	diff_id
	rxmdi_control
	last_update

RelatedItemDesc	rxmdi_rltd_itm
relationship_id	relationship_id
item	item
relationship_name	relationship_name
relationship_type	relationship_type
mandatory_ind	mandatory_ind
	rxmdi_control
	last_update

RelatedItemDtl	rxmdi_rltd_itm_dtl
related_item	related_item
priority	priority
effective_date	start_date
end_date	end_date
	rxmdi_control
	last_update

ItemBOMDesc	rxmdi_pack_item
pack_n	pack_no
item	item
pack_qty	pack_qty
	rxmdi_control
	last_update

department:MrchHrDeptDesc **class:MrchHrClsDesc** **subclass:MrchHrSclsDesc**	**rxmdi_merch_hier**
For department: DEPARTMENT For class: CLASS For subclass: SUBCLASS	hierarchy_level
Field introduced by RXM for creating primary key. It is a unique identifier comprising all three display IDs plus prepended prefix. Example: For department 3: d3 For department 3, class 1: d3c1 For department 3, class 1 & subclass 4: d3c1s4	hierarchy_id
	hierarchy_node_id
	hierarchy_node_name
For department: GROUP For class: DEPARTMENT For subclass: CLASS	parent_level
	parent_node_id
For subclass: dept (example 123)	grandparent_merch_display_id
For class: dept For subclass: class	parent_merch_display_id
For department: dept For class: class For subclass: subclass	merch_display_id
purchase_type	purchase_type
	rxmdi_control
	last_update

The fields not mapped and left empty are not available in the RIB XSD.

The following two attributes are created in the RXMDI Staging Schema tables for managing the life-cycle of a record:

- rxmdi_control: Controller/Flag
- last_update: Timestamp at which the record was last updated

RXMDI EAR

The RXMDI EAR includes the Incremental binaries containing Camel Processors (JPA) to process and persist incoming RIB Messages. It is included in the RXMDI release package and consists of the following components:

- RIB Application Plugin JAR: This is the entry point for incoming RIB Messages from the RIB-RXM adapter.
- RXMDI WAR: This consists of the web application components like web.xml and applicationContext.xml.

- JPA Artifacts: This includes the Entity definition XMLs and is included in the RXMDI WAR.
- Incremental JAR: This contains all the Camel Processors and other Business Logic required for processing and persisting incoming RIB Messages.

Extensibility

- SQL queries are externalized. The queries can be customized.
- In the Camel Context XML, the existing route can be changed, new routes can be added and processors can be updated.
- Additional Message Families and Message Types can be subscribed to by adding them to injectors.xml, persistence.xml, and web.xml.
- Processors can be customized with different criteria.
  - New Entities can be created to extend already existing Entities.

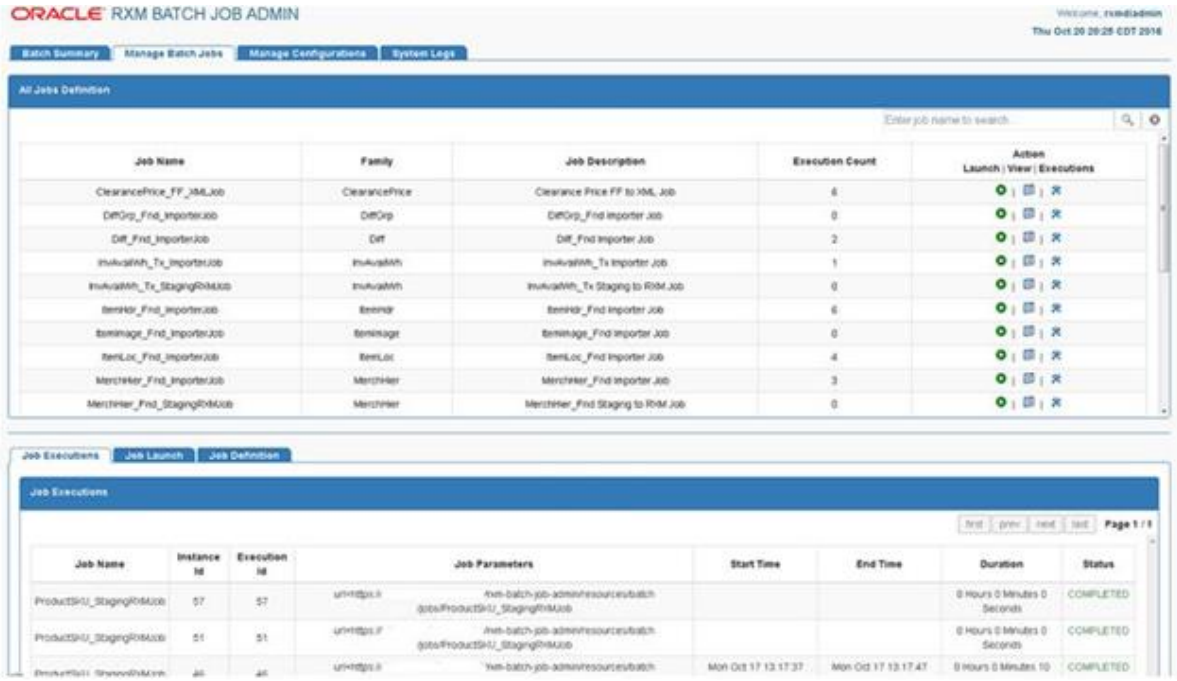
6

Java Batch

Overview

Java Batch is used to call the ODI Web Service which invoke ODI Scenarios through the ODI Java EE Agent. The Java EE Agent is installed during the installation and configuration of the ODI. ODI Scenarios are imported during ODI setup.

RXM Batch Job Admin



The RXM Batch Job Admin Console is a web-based User Interface (deployed as a WAR) that allows the launching of different kinds of jobs.

The Job Admin Console uses Java Batch. The Java Batch job xmls are used to invoke corresponding ODI scenarios which then transform the data.

The Job Admin Console houses jobs which:

- Transform data from BDI to RXMDI Staging (that is, Importer Jobs).
- Transform data from RXMDI Staging to RXM (these include Direct Load SQL as well as XML file generation).
- Transform Regular and Clearance Price data from flat files to RXM (through Flat file to DB and then XML file generation).
- Transform Promotion data from flat files to RXMDI Staging (through Flat File to DB) followed by Staging to RXM (through Direct Load SQL).

Additional jobs can be added by configuring new Job xmls, including them in the WAR, and then redeploying the WAR. The new Job xmls must have corresponding ODI scenarios which they will invoke.

Job XMLs

The following example shows a typical Job XML is written using Java Batch's Job Specification Language (JSL):

Java Batch Job XML

```
<?xml version="1.0" encoding="UTF-8"?>

<job id="Diff_Fnd_ImporterJob" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
version="1.0" >
  <!-- Set the common properties for ODI Batchlet at the Job level. These
properties
       can be overriden by setting a value at the Batchlet level -->
    <properties>
        <property name="description" value="Diff_Fnd Importer Job"/>
    </properties>
    <step id="determineDataset" next="stepDecider1">
        <batchlet ref="oracle.retail.commerce.batch.DatasetBatchlet">
            <properties>
                <property name="sourceSchemaJNDI" value="jdbc/BDIInterface" />
                <property name="stagingSchemaJNDI" value="jdbc/RXMDIStaging" />
                <property name="interfaceModule" value="Diff_Fnd" />
            </properties>
        </batchlet>
    </step>
    <decision id="stepDecider1" ref="oracle.retail.commerce.batch.StepDecider">
        <next on="COMPLETED" to="invokeODIStep"/>
        <stop on="STOPPED" exit-status="STOPPED"/>
        <fail on="FAILED" exit-status="STOPPED"/>
    </decision>
    <step id="invokeODIStep" next="stepDecider2">
        <batchlet ref="oracle.retail.commerce.batch.ODIBatchlet">
            <properties>
                <property name="providerUrl"
value="http://<hostname>:<port>/oracledіagent/OdiInvoke" />
                <property name="odiAgentCredentialKey" value="odiAgent" />
                <property name="workRep" value="WORKREP" />
                <property name="targetName" value="DIFF_BDI_STAGING" />
                <property name="targetVersion" value="001" />
                <property name="targetContext" value="PROD" />
                <property name="scenario" value="true" />
                <property name="projectCode" value="DATAIMPORT" />
            </properties>
        </batchlet>
    </step>
    <decision id="stepDecider2" ref="oracle.retail.commerce.batch.StepDecider">
        <next on="COMPLETED" to="auditStep"/>
        <next on="STOPPED" to="auditStep"/>
        <next on="FAILED" to="auditStep"/>
    </decision>
    <step id="auditStep">
        <batchlet ref="oracle.retail.commerce.batch.AuditBatchlet">
            <properties>
                <property name="stagingSchemaJNDI" value="jdbc/RXMDIStaging" />
            </properties>
        </batchlet>
        <end on="COMPLETED"/>
    </step>
</job>
```

Main Components include:

- Schemas: sourceSchemaJNDI, stagingSchemaJNDI. These must match the data sources set up in the WebLogic domain where ODI is set up and RXMDI is deployed.
- Batchlets: Depending on the Job, different Batchlets may be used in the JSL such as the DatasetBatchlet, ODIBatchlet, and AuditBatchlet.
- Decider: Depending on the complexity of a Job, a Decider is also configured such as the StepDecider.

Configurations include:

- providerURL: The Endpoint for the OdiInvoke Web Service. Typically, this is deployed to a Managed Server secured through HTTPS (TLS).
- odiAgentCredentialKey: The Credential Store Framework (CSF) key for the ODI Agent credentials.
- workRep: The ODI work repository name for the ODI Agent.
- targetName: The name of the ODI Scenario or Load Plan.
- targetVersion: The target version of the ODI Scenario or Load Plan.
- targetContext: The context for executing the ODI Scenario or Load Plan.
- Scenario: The flag for target type. True if ODI Scenario, false if ODI Load Plan.

There is also an odiInvoke.properties where the Endpoint of the ODIInvoke Web Service can be configured. The values in the Job XMLs will override the value in the odiInvoke.properties.

Batchlets and Deciders

Batchlet

A Batchlet is a type of batch step that can be used for any type of background processing.

Decider

A Decider receives control as part of a decision element in a job. It is used to direct execution flow during job processing. It returns an exit status that updates the current job execution's exit status.

RXMDI Job Admin WAR

The RXM Batch Job Admin Console is built using the BdiEdgeAppJobAdminPak16.0.2ForRxm16.0.2 tool. This tool will deploy the war automatically. The BdiEdgeAppJobAdminPak16.0.2ForRxm16.0.2 tool is available as part of the RXMDI release package. The configurable components of this WAR include the Java Batch jar containing Batchlets and Deciders and the Job Xml Batch Jobs.

Extensibility

- Batchlets and Deciders are written in Java and can be extended. New Batchlets and Deciders can also be written and the JSL can be updated to use the new Batchlets and Deciders.
- If a new ODI Scenario is created, then a new Job XML can be created with configurations to match the new ODI Scenario.
- Any new Batchlets and Deciders can be compiled and added to the Java Batch jar and added to the WAR.

- Job XMLs can be updated or new Job XMLs can be added to the WAR. The WAR can then be redeployed to reflect the new changes.

7

Oracle Data Integrator

The Oracle Data Integrator (ODI) is an Extract, Load, and Transform tool. ODI is used to transform data from Database Tables to Database Tables, Database Tables to XML Files and Flat Files to Database Tables.

Components

ODI Java EE Agent and ODI Standalone Collocated Agent

- The Java EE Agent is invoked by Java Batchlets through the ODI Web Service. The agent in turn invokes the ODI Scenarios which perform the transformation.
- The Standalone Collocated Agent is used to encrypt passwords used in Physical Topologies.

ODI Scenarios

- Scenarios are compiled artifacts that perform transformations.
- Scenarios use Logical Topologies.

Logical Topologies

- Logical Topologies are associated with Physical Topologies through a Context.

Context

- A Context not only associates Logical Topologies with Physical Topologies, but also specifies the ODI Agent to be used.

Physical Topologies

- Physical Topologies represent real resources such as DB Schema, XML File, and so on.

Java Batchlets

- Batchlets are a part of the RXM Batch Job Admin Console.
- They are used to call the ODI Web Service which invoke ODI Scenarios through the ODI Java EE Agent.

Packaging

- Scenarios, Logical Topologies, and Context are included in the RXMDI release package.
- Since Physical Topologies represent real resources, they will need to be created. Instructions for creating these are included in the RXMDI release package.

Extensibility

- The ODI Studio is the development tool to create packages, update mappings, and so on, using the ODI Designer.
- The updates made in ODI Studio can then be compiled into a new Scenario or used to update an existing Scenario.
- If a new Scenario is created, then a new Java Batch Job XML can be added to invoke the new Scenario. An update to an existing Scenario will be invoked by the corresponding Java Batch Job XML.

Transformations

Database Tables to Database Tables

ODI uses Direct SQL Load to transform data from one DB schema directly into another DB schema.

Feeds that Use this Type of Transformation

- All feeds from BDI to RXMDI Staging.
- Inventory, Store, Store Address, Warehouse, and Warehouse Address data from RXMDI Staging to RXM Production.
- Promotion data from RXMDI Staging to RXM Publishing.

Process

1. Using ODI Studio, the relevant tables are reverse engineered for both Source and Target DBs.
2. A mapping is performed between the Source and Target DB tables.
3. Any transformation required is done as part of the mapping to massage data.

Extensibility

1. For custom solutions, DB tables can once again be reverse engineered so customizations are visible to ODI.
2. Mappings/Transformations can then be updated.
3. Scenarios can be regenerated based on the changes or new Scenarios can be created with a corresponding Java Batch Job XML.

Database Tables to XML Files

ODI transforms data from DB schema into an XML file.

Feeds that use this type of transformation

- Regular and Clearance Price data is transformed from RXMDI Staging work tables into a priceList XML file that can be imported into RXM's Business Control Center (BCC) using the StartSQLImport Utility provided by Oracle Commerce from where it is published to RXM's production server.
- Merchandise Hierarchy and Product SKU data is transformed from RXMDI Staging tables into a productCatalog XML file that can be imported into BCC using the

StartSQLImport Utility provided by Oracle Commerce from where it is published to RXM's production server.

Process

1. Using ODI Studio, the relevant Source DB tables are reverse engineered as well as the XML XSDs.
2. A mapping is performed between the Source DB tables and the XSDs.
3. Anu transformation required is done as part of the mapping to massage data.

Extensibility

- For custom solutions, DB tables and XSDs can be reverse engineered so customizations are visible to ODI.
- Mappings/Transformations can be updated.
- Scenarios can be regenerated based on the changes or new Scenarios can be created with a corresponding Java Batch Job XML.

Flat Files to Database tables

ODI transforms data from Flat Files into a DB schema.

Feeds that use this type of transformation

Regular and Clearance Price data is transformed from flat files provided by RPM into RXMDI Staging work tables which are then converted into a priceList XML file as part of a different ODI Scenario. This file can be imported into BCC using StartSQLImport Utility provided by Oracle Commerce from where it is published to RXM's production server.

- Promotion data is transformed from flat files provided by RPM into RXMDI Staging work tables which are then loaded into the RXM Publishing schema as part of a different ODI Scenario. Using a timed Scheduler, they are then processed via the PromotionImportExport API provided by Oracle Commerce and imported into BCC from where it is published to RXM's production server.
-

Process

- Using ODI Studio, the relevant flat files are reverse engineered as well as the target DB tables.
- Then a mapping is performed between the flat files and the target DB tables.
- Finally, any transformation required is done as part of the mapping to massage data.

Extensibility

- For custom solutions, flat files and DB tables can be reverse engineered so customizations are visible to ODI.
- Mappings/Transformations can be updated.
- Scenarios can be regenerated based on the changes or new Scenarios can be created with a corresponding Java Batch Job XML.

8

Promotion and Pricing Integration

The RXMDI component integrates Oracle Retail Price Management (RPM) flat files into Oracle Commerce (including some RXM extensions). It is built upon the Oracle Data Integrator (ODI) application.

Promotion Integration

Pre-requisite: For promotion flat files, Location(loc_id) should exist in RXMDI_STORE_SITE table's column STORE_ID.

The RXMDI ODI scenario expects RPM flat files containing promotional information to arrive in a directory that can be read by ODI. The default location is C:/ODI/RXMDI/RPM/SOURCE.

The ODI scenario is to be scheduled to run on an interval. It will scan the directory for incoming files (*.dat).

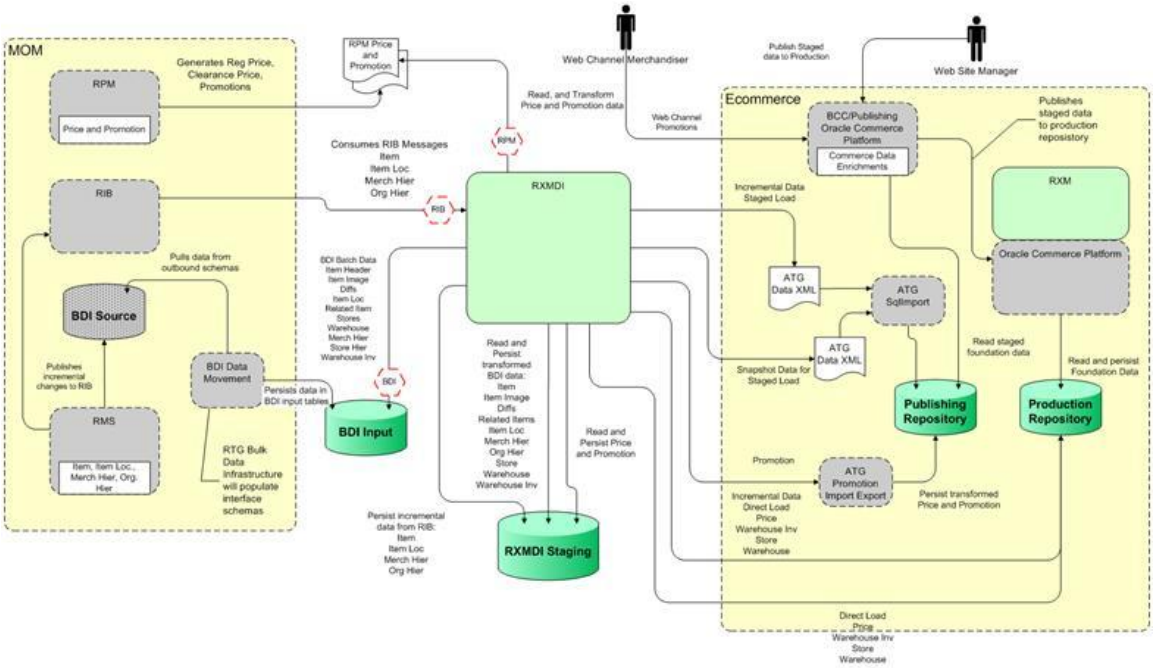
Upon finding file names that match external site IDs that have already been imported by ODI (in the format, RPMPC_[timestamp]_[site_id]_S.dat), those files will be one-at-a-time read and processed. The files are processed in timestamp order.

After processing, promotion flat files will be renamed to PRMPC_INPROGRESS_S.dat.

The files are read into the ODI working table's area and then ODI maps that data into the RXMDI_PROMO tables that are inside the Publishing schema (as defined by RXM). ODI performs some minor transformation upon insertion into Publishing.

In the Publishing stack, RXM has provided a new SingletonScheduledService component at /retail/commerce/integration/promotion/PromotionImportService. This service is configured to scan the RXMDI_PROMO tables once a day at 4am. However, this config can be changed easily. If records are found that have null in the IMPORT_SESSION_ID column, those records are read and processed.

The service uses a /retail/commerce/integration/promotion/PromotionImportService to transform the data and then feeds the objects to the Oracle Commerce PromotionImportExport API. This API is provided by Oracle Commerce to import promotions into projects for review and publishing within the Business Control Center (BCC). A business user then logs into BCC to embellish and promote projects to Production.



Promotion Commerce Enhancements

A repository is defined for ODI to put the promotions into for staging. The component is at /retail/commerce/integration/promotion/PromotionIntegrationRepository.

The PromotionImportService scans this repository for new promotions.

The promotion item descriptor is enhanced to support additional IDs, such as externalPromoId, and a deleted flag. Both support input from RPM. The IDs will match RPM IDs and when RPM sends a DEL message, the matching promotion will be marked as "enabled" = false and "deleted" = true.

> **Note:** RPM overlapping promotions are not supported by this version of RXM.

Pricing Integration

Prerequisites:

- For processing pricing flat files, Location(loc_id) should exist in RXMDI_STORE_SITE table's column STORE_ID.
- Price List ID and Sales List ID should be created in BCC and those generated IDs should be inserted into RXMDI_STORE_SITE table before processing pricing flat file.

The pricing integration works in a similar way to the promotion integration with some differences. RPM is still expected to produce flat files *.dat. The flat file name should be of the format <event type>_<date in YYYYMMDDHH24MISS format>_<loc id>_<loc type>.dat (For example: REGPC_20160810121204_2222_S.dat). After processing, flat file will be moved to SOURCE/archive folder and priceLists.xml will be moved from it's source folder to SOURCE/processed folder.

Those flat files are picked up by ODI in the same directory as promotions. ODI processes the flat files into a working area. However, instead of pushing the data into a Publishing staging area, ODI transforms the data into XML files to be later processed by Oracle Commerce's startSQLImport program manually.

Clearance prices are put into the salePriceList instead of the regular priceList.

Pricing Commerce Enhancements

The /atg/commerce/pricing/priceLists/priceLists.xml definition includes an externalPriceId to identify prices through RPM IDs and a priceType property to identify clearance prices. Additionally, support is available for clearance endDate since it comes from RPM in a separate ClearanceReset entity.

9

Job Mapping

BDI/RIB

Note: In the case of RIB, the first step is irrelevant. The data is pushed through RIB directly into the RXMDI Staging schema.

Feed Type	First Step (BDI Interface – RXMDI Staging	Second Step (RXMDI Staging – RXM	Target	Comments
Inventory	INVAvailWH_Tx IMporter Job	InvAvailWh_Tx Staging to RXM Job	RXM Production DB	
Store	▪ Store_Fnd Importer Job ▪ StoreAddr_Fnd Importer Job	Store Staging to RXM Job	RXM Production DB	

Feed Type	First Step (BDI Interface – RXMDI Staging	Second Step (RXMDI Staging – RXM	Target	Comments
Item	▪ MerchHier_Fnd Importer Job ▪ ItemHdr_Fnd Importer Job ▪ ItemLoc_Fnd Importer Job ▪ Item Image_Fnd Importer Job ▪ Related Item_Fnd Importer Job ▪ Related Item_Fnd Importer Job ▪ Diff_Fnd Importer Job ▪ DiffGrp_Fnd Importer Job ▪ PackItem_Fnd Importer Job	Warehouse	▪ Wh_Fnd Importer Job ▪ WhAddr_Fnd Importer Job	Items need to be ranged to the RMS webstore location configured in the RXMDI_STORE_SITE table. **BDI business process flow:** It is very important to run the BDI jobs for MerchHier_Fnd, ItemHdr_Fnd, and ItemLoc_Fnd prior to running the ProductSKU Staging to RXM Job. **RIB business process flow:** It is very important to run the ProductSKU Staging to RXM Job after Merchandise Hierarchy and its associated Item data is in the RXMDI/Staging database. **Note:** Once in the publishing database, the data can be pushed to the RXM Production DB through BCC.
Organization				Commerce does not support OrgHier Out of the box
Hierarchy				

Pricing and Promotions

Feed Type	Source	Flat File to XML	Destination	Additional Comments
Regular Price	Flat File provided by RPM	Regular Price FF to XML Job	Generates XML (priceList.xml) which needs to be imported into BCC using SQLImport utility.	This data can be pushed to RXM Production DB through BCC.
Clearance Price	Flat File provided by RPM	Clearance Price FF to XML Job	Generates XML (priceList.xml) which needs to be imported into BCC using SQLImport utility.	This data can be pushed to RXM Production DB through BCC.
Promotions	Flat File provided by RPM	Promotion FF to RXM Job	RXM Publishing DB.	Commerce Scheduler pushes data to RXM Production DB.

10

Packaging and Deployment

RXMDI includes the following artifacts as part of the RXMDI release package:

- RXMDI EAR:
  - RIB Application Plugin JAR
  - RXMDI WAR
  - JPA Artifacts
  - Incremental JAR
- RXM Batch Job Admin WAR:
  - Job XMLs -> ODI Scenarios
  - Java Batch JAR
- ODI:
  - Scenarios, Logical Topologies, and Context are included.
  - Since Physical Topologies represent real resources, they will need to be created. Instructions for creating these are included.
- DB:
  - RXMDI Staging Schema DDL
  - RIB Error Hospital Schema DDL