Oracle® Retail Store Inventory Management

Implementation Guide, Volume 2 – Integration with Oracle Retail Applications

Release 13.2.4.0.1

June 2012

ORACLE®

Oracle Retail Store Inventory Management Implementation Guide, Volume 2 – Integration with Oracle Retail Applications, Release 13.2.4.0.1

Primary Author: Graham Fredrickson

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

C Appendix: External Inventory Adjustments

List of Tables

List of Figures

Send Us Your Comments

Oracle Retail Store Inventory Management Implementation Guide, Volume 2 – Integration Information, Release 13.2.4.0.1.

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 new Applications Release Online Documentation CD available on My Oracle Support and www.oracle.com. 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

The *Oracle Retail Store Inventory Management Implementation Guide, Volume 2 – Integration Information* provides detailed information about the integration of Store Inventory Management (SIM) with other applications:

- SIM Integration – Functional

  This chapter includes information describes the functional role that Oracle Retail Integration Bus (RIB) messages play with regard to SIM functionality.

  This chapter also provides information about the integration between SIM and other applications:

  - From SIM to a Warehouse Management System
  - From a Warehouse Management System to SIM
  - From Oracle Retail Point-of-Service to SIM
  - From the Merchandising System to SIM
  - From SIM to the Merchandising System
  - From SIM to the Merchandising System Using the Stock Upload Module in the Merchandising System
  - From SIM to the Reporting System
  - From SIM to a Price Management System
  - From a Price Management System to SIM
- SIM Integration – Technical

  This chapter includes information describes the technical aspects of SIM integration, focusing on the following:

  - RIB-based Integration
  - RSL-based Integration
  - Web Service-based Integration
  - File-based Integration

Audience

This document is intended for the Oracle Retail Store Inventory Management application integrators and implementation staff, as well as the retailer's IT personnel.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at http://www.oracle.com/accessibility/.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/support/contact.html or visit http://www.oracle.com/accessibility/support.html if you are hearing impaired.

Related Documents

For more information, see the following documents in the Oracle Retail Store Inventory Management Release 13.2.4.0.1 documentation set:

- *Oracle Retail Store Inventory Management Release Notes*

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, 13.2) or a later patch release (for example, 13.2.4). If you are installing the base release, additional patch, and bundled hot fix releases, read the documentation for all releases that have occurred since the base release before you begin installation. Documentation for patch and bundled hot fix releases can contain critical information related to the base release, as well as information about code changes since the base release.

Oracle Retail Documentation on the Oracle Technology Network

Documentation is packaged with each Oracle Retail product release. Oracle Retail product documentation is also available on the following Web site:

http://www.oracle.com/technology/documentation/oracle_retail.html

(Data Model documents are not available through Oracle Technology Network. These documents are packaged with released code, or you can obtain them through My Oracle Support.)

Documentation should be available on this Web site within a month after a product release.

Conventions

The following text conventions are used in this document:

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

1

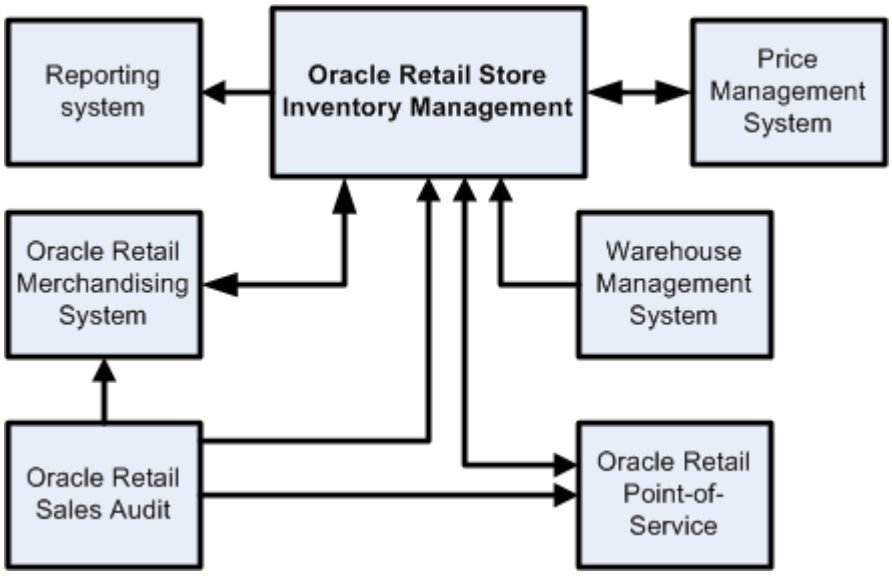
Store Inventory Management Integration Points into the Retail Enterprise

This chapter identifies integration points for Store Inventory Management (SIM) into the retail enterprise.

SIM Integration Points

Figure 1–1 is a high-level diagram that shows the overall direction of the data among systems and products across the enterprise. For a detailed description of Figure 1–1, see Chapter 2, "SIM Integration – Functional".

Figure 1–1 SIM-Related Dataflow Across the Enterprise



Integration with Oracle Retail Workspace

For information about Oracle Single Sign-On and Oracle Retail Store Inventory Management 13.2.4.0.1, please see the latest *Oracle Retail Store Inventory Management Installation Guide*.

For information on how to integrate SIM with Oracle Retail Workspace, see the latest *Oracle Retail Workspace Implementation Guid*e.

Oracle Retail Workspace is a supported configuration of Oracle WebCenter Spaces 11.1.1.5 for Oracle Retail. For the Oracle Retail 13.2.x enterprise, Oracle Retail Workspace replaces previous versions of Oracle Retail Workspace. There is no more packaged Oracle Retail Workspace code, only configuration instructions for Oracle WebCenter Spaces 11.1.1.5.

The Oracle Retail Workspace configuration utilizes the external application functionality and the application navigator taskflow of the Oracle WebCenter Framework to configure SIM in Oracle WebCenter Spaces.

Integration Compatibility Requirements

See the *Oracle Retail Store Inventory Management Installation Guide* for information about compatible Oracle Retail application versions.

2

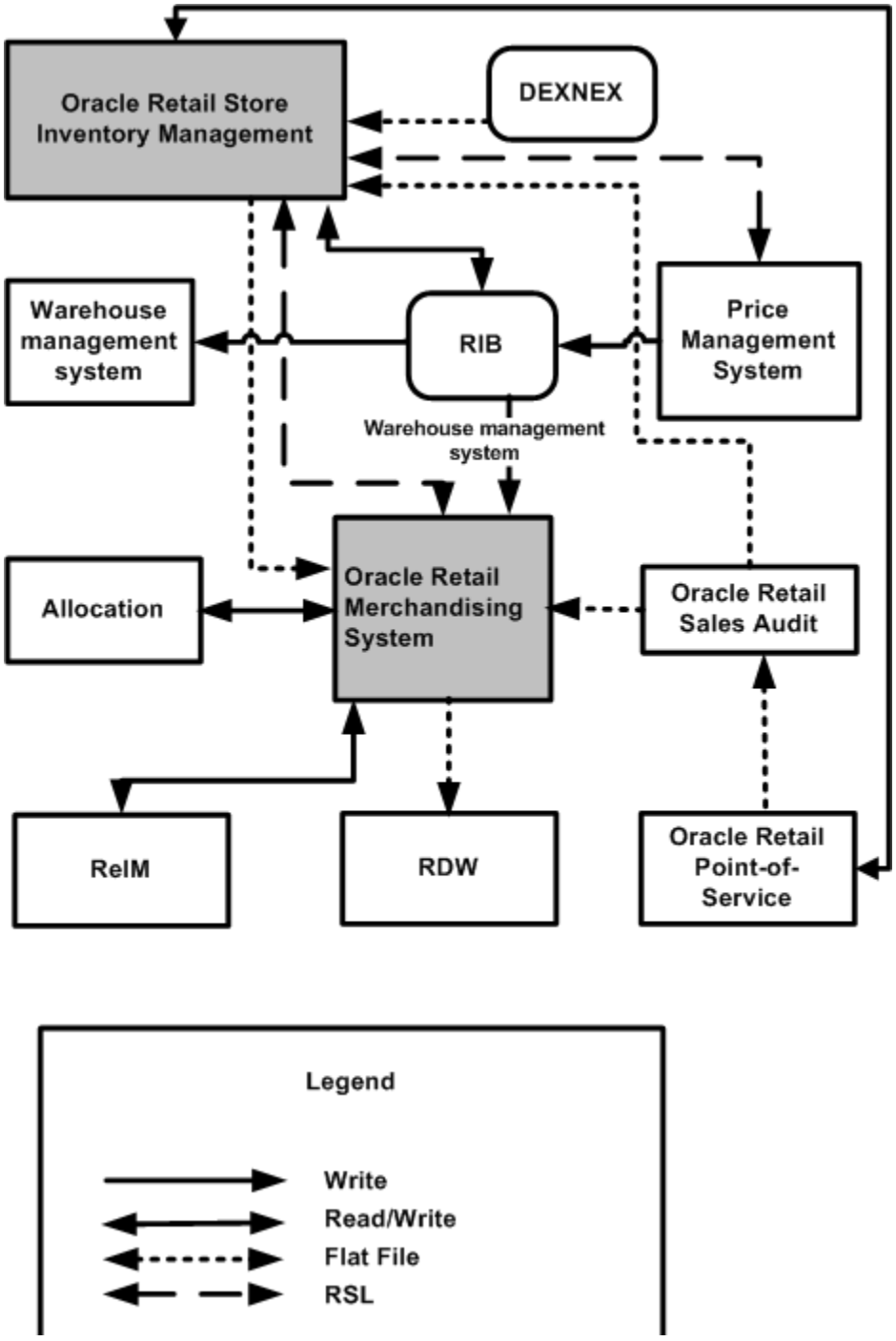
SIM Integration – Functional

This chapter provides a functional overview of how SIM integrates with other systems (including other Oracle Retail systems).

The first section in this chapter provides you with Figure 2–1 that illustrates the various Oracle Retail products and databases that SIM interfaces with as well as the overall dataflow among the products. The accompanying explanations are written from a system-to-system perspective, illustrating the movement of data.

System-to-System SIM Dataflow

Figure 2–1 SIM Functional Dataflow



For information about the technical means through which the interfaces pass data, see Chapter 3, "SIM Integration – Technical" as well as "Technical Architecture" and "Batch Processes" in the *Oracle Retail Store Inventory Management Operations Guide*.

Functional Descriptions of RIB Messages

Table 2–1 briefly describes the functional role that messages play with regard to SIM functionality. The table also illustrates whether SIM is publishing the message to the RIB or subscribing to the message from the RIB. For additional information, see the *Oracle Retail Integration Bus Operations Guide* and other RIB documentation.

Table 2–1 Functional Descriptions of RIB Messages

Functional area	Subscription/ Publication	Integration to Products	Description
ASN in	Subscription	A warehouse management system, Vendor (external)	These messages contain inbound shipment notifications from both vendors (PO shipments) and warehouses (transfer and allocation shipments).
ASN out	Publication	RMS, a warehouse management system	These messages are used by SIM to communicate store-to-warehouse transfers (returns to warehouse) to both RMS and a warehouse management system. These messages are also used to communicate store-to-store transfers to RMS.
Delivery Slot	Subscription	RMS	This message is communicated by RMS and consists of the delivery slot information, which is needed by transfers and other shipment transactions.
Diff IDs	Subscription	RMS	These messages are used to communicate differentiator IDs from RMS to SIM.
DSD receipts	Publication	RMS	These messages are used by SIM to communicate the receipt of a supplier delivery for which no RMS purchase order had previously existed.
Items	Subscription	RMS	These are messages communicated by RMS that contain all approved items records, including header information, item/supplier, and item/supp/country details, item/UDA, Item/Image, and item/ticket information. The item/supplier/manufacturer and the Item/Supplier/Dimension information also gets published to SIM by this message family as part of this release.
Item/location	Subscription	RMS	These are messages communicated by RMS that contain item/location data used for ranging of items at locations and communicating select item/location level parameters used in store orders.
Inventory adjustments	Publication	RMS	These messages are used by SIM to communicate inventory adjustments. RMS uses these messages to adjust inventory accordingly.
Inventory request	Publication	RMS	These messages are used by SIM to communicate the request for inventory of a particular item. RMS uses this data to fulfill the requested inventory through either auto-replenishment or by creating a one-off purchase order/transfer.
Merchandise Hierarchy	Subscription	RMS	These messages are communicated by RMS. These messages include department/class/subclass information.
Partner	Subscription	RMS	These messages are communicated by RMS. These messages include External Finishers.
Price change	Subscription	A price management system	These messages facilitate price changes for permanent, clearance and promotions.

Table 2–1 (Cont.) Functional Descriptions of RIB Messages

Functional area	Subscription/ Publication	Integration to Products	Description
Price Inquiry	RSL calls	A price management application	This service, provided by a price management application, allows an inquiring system to request the effective retail for an item at a specified location on a given date. a price management application provides the retail value and indicates whether the value is promotional, clearance or regular.
Purchase orders	Subscription	RMS	These messages contain approved, direct to store purchase orders. Direct Deliveries are received against the POs created in RMS.
Receipts	Publication	RMS	These messages are used by SIM to communicate the receipt of an RMS purchase order, a transfer, or an allocation.
Receiver unit adjustments	Publication	RMS	These messages are used by SIM to communicate any adjustments to the receipts of purchase orders, transfers, and allocations. These messages are part of the RECEIVING message family (receiving unit adjustments only use the RECEIPTMOD message type).
Return to vendor	Publication	RMS	These messages are used by SIM to communicate the shipment of a return to vendor from the store.
RTV request	Subscription	RMS	These are messages communicated by RMS that contain a request to return inventory to a vendor.
Seed data	Subscription	RMS	These messages communicated by RMS contain differentiator type values. The creation, modification and deletion of the various diff types in RMS flows to SIM through the seed data message.
Stock count schedules	Publication	RMS	These messages are used by SIM to communicate unit and value stock count schedules to RMS. RMS uses this schedule to take an inventory snapshot of the date of a scheduled count.
Stock order status	Publication	RMS	These messages are used by SIM to communicate the cancellation of any requested transfer quantities. For example, the merchandising system can create a transfer request for 90 units from a store. If the sending store only ships 75, a cancellation message is sent for the remaining 15 requested items.
Stock order status	Subscription	A warehouse management system	These messages are used by a warehouse management system to communicate the creation or modification of a warehouse delivery in a warehouse management system.
Stores	Subscription	RMS	These are messages communicated by RMS that contain stores set up in the system (RMS).
Store ordering	Publication	RMS	These messages are used by SIM to communicate the request for inventory of a particular item.
Transfer request	Subscription	RMS	These messages are communicated by RMS and contain a request to transfer inventory out of a store. Upon shipment of the requested transfer, SIM uses the ASN Out message to communicate what was actually shipped. In addition, SIM uses the stock order status message to cancel any requested quantity that was not shipped.

Table 2–1 (Cont.) Functional Descriptions of RIB Messages

Functional area	Subscription/ Publication	Integration to Products	Description
Vendor	Subscription	RMS, external (financial)	These are messages communicated by RMS containing vendors set up in the system (RMS or external financial system).
Warehouses	Subscription	RMS	These are messages that are communicated by RMS that contain warehouses set up in the system (RMS). SIM only gets physical warehouse records.
UDA	Subscription	RMS	These are messages that are communicated by RMS that contain UDAs. This information will be used as extra criteria to search for items as well as display the UDA information for the item.

From SIM to a Warehouse Management System

For returns to the warehouse using the RIB, SIM sends outbound ASN data to facilitate the communication of store-to-warehouse shipment data to a warehouse management system.

From a Warehouse Management System to SIM

The following warehouse management system data is published through the RIB for SIM subscription:

Outbound advance shipping notice (ASN) data converted to inbound ASN data to facilitate warehouse-to-store shipments, the warehouse management system provides SIM outbound ASN data. ASNs are associated with shipments and include information such as to and from locations, container quantities, and so on. Note that outbound ASN data is converted to inbound ASN data by the RIB for SIM's subscription purposes. The data is the same, but the format is slightly different. The conversion takes place so that ASN inbound data can be the same among applications.

SIM subscribes to the following information from a warehouse management system:

When a warehouse delivery originates in a warehouse management system, a Stock Order Status message is sent to both SIM and RMS with the creation. If the warehouse updates the quantity on the transfer prior to shipping it, SIM subscribes directly to the stock order status message from a warehouse management system and updates the transfer accordingly with an increase or decrease.

From Oracle Retail Point-of-Service to SIM

The following data is sent from Oracle Retail Point-of-Service through ReSA (optional) to SIM:

- Sales and returns data

  SIM uses the data to update the stock on hand (SOH) for store/item combinations. In other words, SIM learns about inventory movement (what is sold and what is returned).

- Customer Order data

  SIM uses this information to reserve or unreserve inventory for customer orders created or originated in Oracle Retail Point-of-Service. This affects the available and unavailable inventory buckets in SIM.

From the Merchandising System to SIM

The following merchandising system data is published through the RIB for SIM subscription:

- PO data

  SIM allows the user to receive against direct store delivery (DSD)-related PO data. DSD occurs when the supplier drops off merchandise directly in the retailer's store.

- External store orders

  SIM is able to create purchase orders directly in RMS through the SIM GUI.

- Item data (sellable and non-sellable items)

  SIM processes only transaction-level items (SKUs) and below (such as UPC), so there is no interface for parent (or style) level items. See the RMS documentation for more information about its three-level item structure. In addition to approved items records, the item data includes including header information, item/supplier, and item/supp/country details. Merchandise hierarchy data is an attribute of the item data to which SIM subscribes.

- Location data (updated store and warehouse location information)

- Item-location data

  SIM uses this data for ordering parameters (for example, allowing the user to determine whether an item is a store order type item).

- Diff data

- Supplier and supplier address data

- Transfer request data

  Corporate users can move inventory across stores using RMS transfer requests.

- Return requests

  The merchandise system sends return requests from a store to a warehouse (RTW) and/or to a vendor (RTV). The store itself ships the goods.

From SIM to the Merchandising System

The following SIM data is published using the RIB for the subscription of the merchandising system:

- Receipt data

  By sending the receipt data, SIM notifies the merchandising system of what SIM received. Types of receipt data are related to the following:

  - Transfers
  - Existing (merchandising system) POs associated with DSDs
  - New POs associated with DSDs
  - Merchandising system (such as RMS) purchase orders

- RTV and RTW data

  SIM notifies the merchandising system about returns to vendors and returns to warehouses.

- Return to warehouse data

SIM uses ASN out data to notify the merchandising system about returns to warehouses.

- Store ordering data

  SIM sends this data to communicate a request for inventory of a particular item. The merchandising system can use this data to calculate a store order replenishment type item's recommended order quantity (ROQ).

- Stock count schedule data

  The merchandising system uses this data to help maintain the synchronicity of the inventory levels in SIM and the merchandising system. Once the merchandising system has the stock count schedule data, SIM and the merchandising system perform a snapshot count at the same time. The store does a physical count and uploads the results, and the merchandising system compares the discrepancies.

- Price change request data

  A SIM user is able to request price changes, along with effective dates, from the price management system.

From SIM to the Merchandising System Using the Stock Upload Module in the Merchandising System

- Stock count results

  Once a stock count is authorized and completed, SIM creates a flat file and stages it to a directory. Using the flat file generated by SIM, the merchandising system's stock upload module retrieves and uploads the physical stock count data. The merchandising system uses this data to help maintain the synchronicity of the inventory levels in SIM and the merchandising system.

From SIM to the Reporting System

- Data for reports

  SIM has the ability to produce reports that retailers can customize to reflect the unique requirements of their business. To facilitate reporting functionality, the report tool used by SIM is Oracle BI Publisher.

From SIM to a Price Management System

- Request for approval of price change data

  Regular, clearance, and simple fixed price promotion price change data are sent to a price management system. The price management system performs a conflict check and returns a validation status (successful or not successful). If the validation was successful, the price change is returned immediately to SIM and persisted.

From a Price Management System to SIM

- Price change data

  A price management system sends price change data to SIM. This type of price change data can originate at a corporate level or at the store level.

3

SIM Integration – Technical

This chapter is divided into the following four sections that address SIM's methods of integration:

- "RSL-based Integration"
- "Web Service-based Integration"
- "File-based Integration"
- "RIB-based Integration"

Each section includes information concerning the architecture of the integration method and the data that is being passed back and forth. For additional functional descriptions of the dataflow, see Chapter 2, "SIM Integration – Functional".

For more information about message families and message type names and the XML schema documents that describe the XML messages, see "Appendix: Subscription and Publishing Designs".

RSL-based Integration

RSL handles the interface between a client application and a server application. The client application typically runs on a different host than the service. However, RSL allows for the service to be called internally in the same program or Java Virtual Machine as the client without the need for code modification. All services are defined using the same basic paradigm -- the input and output to the service, if any, is a single set of values. Errors are communicated through Java Exceptions that are thrown by the services. The normal behavior when a service throws an exception is for all database work performed in the service call being rolled back. RSL works within the J2EE framework. All services are contained within an interface offered by a Stateless Session Bean. To a client application, each service appears to be merely a method call.

- RSL is used to integrate SIM with a price management application for future retail price inquiry and price change requests. RSL for a price management application runs within a price management application.
- RSL is used to integrate SIM with RMS for store order inquiry and creation. RSL for RMS runs as a standalone service that is part of the Retail Integration application.

For more information on RSL, see the *Oracle Retail Service Layer Programmer's Guide* and *Oracle Retail Service Layer Installation Guide* that is part of Oracle Retail Integration application.

Table 3–1 **RSL services used by SIM**

Service Name	Description
PriceInquiryService	This service, provided by a price management application, allows an inquiring system to request the effective retail for an item at a specified location on a given date. A price management application provides the retail value and indicates whether the value is promotional, clearance or regular.
PriceChangeService	This service allows for the creation of a price change in a price management application for a permanent, clearance or promotion.
StoreOrderServices	SIM makes a call to RMS for the store order creation and inquiry. In addition to queries, there are requests/replies for the creation, modification, and deletion of store orders.

Table 3–2 **Payloads used in RSL services**

RSL Service	Payload
StoreOrderServices	LocPODesc
StoreOrderServices	LocPODtl
StoreOrderServices	LocPOHdrsRsp
StoreOrderServices	LocPOHdrsRspDtl
PriceInquiryService	PrcInqReq
PriceInquiryService	PrcInqReqDtl
PriceChangeService	PrcChgDesc
PriceChangeService	RegPrcChgDtl
PriceChangeService	PrmPrcChgSmp
PriceChangeService	PrmPrcChgDtl
PriceChangeService	ClrPrcChgDtl

For specific information about the request and response processing associated with the following services, see the latest Message Families and Types Report, which is part of Oracle Retail Integration documentation.

Web Service-based Integration

SIM Web service is deployed as a separate web-module within the SIM application. The document literal type (Doc-Lit) message format is used to define the messages. Security can be enabled at many levels through Oracle Application Server (OAS). For security, OAS provides a comprehensive WS-Security implementation for authentication, confidentiality with encryption, and integrity with XML Digital Signatures.

The wsdl for SIM Web service is found at:

```
http://<sim-installation-host>:<port>/sim-ws/simWebService?WSDL
```

Currently, SIM Web services are set to use OAS security model with username token digest profile, which uses base-64-encoding and a SHA-1 digest.

To change security model to use clear text password or disable security, use OAS-Enterprise manager to change the settings or manually change the <sim-application-path>/<sim-ws-module-path>/WEB-INF/oracle-webservices.xml config file and bounce the server.

Web Services How-to Note

1. Web services cannot provide new functionality by themselves, but are used only to expose existing functionality.
2. Write your own new WSDL.
3. Write your own new Web service.
4. Use existing base business objects and base services to accomplish functionality.
5. Package Web service WSDL and implementation with the SIM ear so you have access to all base EJBs just like base Web services do. This means customized build/deploy scripting & packaging.
6. In custom implementation, call `"UniversalContext.startSession()"` before accessing base-code, or you will run into issues with values not being found.

File-based Integration

Currently SIM has the following file-based integrations:

- Sales data: SIM imports sales data through flat file from Sales Audit System.
- Third Party Stock Count: SIM import third party stock count file and upload the files to RMS for future processing.
- Direct Exchange (DEX) and Network Exchange (NEX) Receiving.
- Price Bulk Processing: SIM imports pricing files from a price management system and updates the price information of the items.
- Customer orders are also interfaced through ReSA. These are layaway and other different types of orders created in Oracle Retail Point-of-Service and interfaced with SIM using the ReSA batch file.

See "Batch Processes" in the *Oracle Retail Store Inventory Management Operations Guide* for additional details on SIM file-based integrations.

SIM Web Service Application Programming Interface (API) Reference

Inventory Lookup API

Used to lookup inventory for a store (the requesting store itself or another store), transfer zone or buddy stores. Based on the request, the resulting response can contain inventory buckets for a single store or multiple stores.

Operation	Purpose
lookupInventory	Looks up store inventory buckets for a store, another store, all stores for a transfer zone of a store, all Buddy stores for a store.

lookupInventory Operation

The request to look up store inventory is lookupInventory.

lookupInventory Types

Table 3–3 LookupInventoryRequest

Element	Description	Data Type	Required
StoreID	Requesting store ID	xs:string	Yes
ItemID	The item number	xs:string	Yes
InventoryLocationID	The location for which inventory is being requested	xs:string	Yes if InventoryLocationType is STORE
InventoryLocationType		invlookup:LocationWSType{STORE,REQUESTING_LOCATION, TRANSFER_ZONE,BUDDY_STORE}	Yes

Table 3–4 LookupInventoryResponse

Element	Description	Data Type	Required
StockInfos (0..*)	List of stock infos	invlookup:StockWSInfo	

Table 3–5 StockWSInfo

Element	Description	Data Type	Required
ItemID	The item number in SIM	xs:string	
LocationID	The location ID (store ID for now)	xs:string	
LocationType	The location type	invlookup:LocationWSType{STORE,REQUESTING_LOCATION, TRANSFER_ZONE,BUDDY_STORE}	
IsRanged	Determines if item is ranged to store	xs:boolean	
CaseSize	The case size for the item.	xs:decimal	
StandardUOM	The standard unit of measure	xs:string	
PackageUOM	Package unit of measure	xs:string	
OnOrderQty	Quantity on order	xs:decimal	
AllocatedQty	Quantity allocated	xs:decimal	
UnavailableQty	Unavailable quantity	xs:decimal	
CustomerReservedQty	Quantity reserved for customers	xs:decimal	
InTransitQty	In transit quantity	xs:decimal	
StockInDeliveryBay	Quantity in delivery bay	xs:decimal	
StockInBackRoom	Quantity in back room	xs:decimal	

Table 3–5 StockWSInfo (Cont.)

Element	Description	Data Type	Required
StockOnShopFloor	Quantity in back room	xs:decimal	
StockOnHand	Quantity on hand	xs:decimal	
AvailableInventory	Available Inventory	xs:decimal	
TransferReservedQty	Quantity reserved for transfers	xs:decimal	
VendorReturnQty	Quantity reserved for vendor return	xs:decimal	
CalculatedQty	Indicates if it is a calculated quantity or actual physical quantity	xs:boolean	

Customer Order API

Used for creating or modifying the Customer Orders that update the inventory buckets through an external system, such as Oracle Retail Point-of-Service. Two separate APIs are supported to process single or multiple customer order requests from the external system.

Operation	Purpose
processCustomerOrder	Process a single customer order in asynchronous manner.
processMultipleCustomerOrders	Process multiple customer orders, in an asynchronous (batch) mode. These requests will be staged and processed later using the pollingtimer framework.

Customer Order Types

Table 3–6 ProcessCustomerOrderRequest

Element	Description	Data Type	Required
StoreID	Requesting store ID	xs:string	Yes
ID	Unique order of customer order within the external system	xs:string	Yes
ReservationType	A unique identifier that represents the type of reservation being made. When displaying Customer Orders in SIM, SIM uses the unique identifier presented and decodes it using the new ReservationType setup table.	xs:string	Yes

Table 3–6 ProcessCustomerOrderRequest (Cont.)

Element	Description	Data Type	Required
OrderComments	The location for which inventory is being requested	xs:string	No
TransactionTimeStamp	Time stamp	xs:datetime	Yes
Items (0..*)	The CO items (multiple)	invlookup: ProcessCustomerOrderItemRequest	Yes

Table 3–7 ProcessCustomerOrderItemRequest

Element	Description	Data Type	Required
ItemID	The item number	xs:string	Yes
Action	The action	customerorder: CustomerOrderActionWSType{NEW, MODIFY, CANCEL_ RESERVATION, FULFILL, CANCEL_FULFILL}	Yes
Quantity	Quantity ordered	xs:decimal	Yes
UOM	Unit of measure	xs:string	Yes
UIN	The serial number		
UINType	The serial number type	customerOrder:UINWSType{SERIAL_NUMBER, AGSN}	No, defaults to SERIAL_ NUMBER
ItemComments	CO comments	xs:string	No

Table 3–8 ProcessMultipleCustomerOrdersRequest

Element	Description	Data Type	Required
CustomerOrders(1..*)	Customer Order Request batch (multiple)	Customerorder:ProcessCustomerOrderRequest	Yes

Table 3–9 ProcessCustomerOrderResponse

Element	Description	Data Type	Required
CustomerOrderID	The customerOrderID	invlookup:StockWSInfo	

Item Basket API

This API is used to look up an Item Basket for a given store. The user has to provide the Item Basket ID and Store ID. The Order Type is an optional argument. Based on the search criteria, the response will either have the attribute of the Item Basket or the error code.

Operation	Purpose
lookupItemBasket	Look up Item Basket created with a SIM handheld.

lookupItemBasket Operation

The request to look up the ItemBasket created with a SIM handheld.

lookupItemBasket Types

Table 3–10 LookupItemBasketRequest

Element	Description	Data Type	Required
StoreID	Requesting store ID	xs:string	Yes
ItemBasketID	The Item basket ID	xs:string	Yes
OrderType	The order type (Item Basket, Customer Order)	xs:string	No

Table 3–11 LookupItemBasketResponse

Element	Description	Data Type	Required
ItemBasket	Item Basket wrapper	Itembasket:ItemBasketWS Type	

Table 3–12 ItemBasketWSType

Element	Description	Data Type	Required
ID	Item basket ID	xs:string	
OrderType	The order type (Item Basket, Customer Order)	xs:string	
LineItems(1..*)	Line items (multiple)	itembasket: ItemBasketLineItemWSTy pe	

Table 3–13 ItemBasketLineItemWSType

Element	Description	Data Type	Required
ItemID	Item ID	xs:string	
UIN	The UIN serial number if any	xs:string	
Quantity	Quantity	xs:decimal	
CaseSize	Case size	xs:decimal	
UOM	Unit of measure	xs:string	

POS Transaction API

This is the single Web service API that takes care of processing UIN creates/updates, customer order/inventory reservation and sale/return/void sale/void return. Oracle Retail Point-of-Service uses this API to update inventory in SIM.

Operation	Purpose
processMultiplePosTransactions	Process multiple Point-of-Service transactions, in an asynchronous (batch) mode. These requests will be staged and processed later using the pollingtimer framework.

POS Transaction Types

Table 3–14 ProcessMultiplePosTxnsRequest

Element	Description	Data Type	Required
ProcessPosTxnRequest	This represents one transaction. Multiple instances of this object is present for the number of transactions present in ProcessMultiplePosTxnsRequest	posTransaction:ProcessPosTxnRequest	Yes

Table 3–15 ProcessPosTxnRequest

Element	Description	Data Type	Required
TransactionID	Transaction ID which uniquely identifies the transaction	xs:string	Yes
RequestingEntity	The entity requesting the service.	xs:string	Yes
ReservationType	Type of reservation in case there is an order ID associated with the transaction	xs:decimal	Yes
OrderID	Order ID associated with the customer order/inventory reservation.	xs:string	No
OrderComments	Comments associated with the order if it is an order transaction like Customer Order	xs:string	No
TransactionTimestamp	The serial number type	xs:dateTime	Yes
SOHUpdateRequired	Indicates whether the inventory updates should take place in ORSIM or not.	xs:boolean	Yes

Table 3–16 ProcessPosTxnItemRequest

Element	Description	Data Type	Required
ItemID	ID of the item that is present in the transaction	xs:string	Yes
Action	Specifies the action taken on the item. For example, Sale, Return and so forth.	posTransaction:PosTransactionActionWSType	Yes
Quantity	Quantity of the item present in the transaction	xs:decimal	Yes
UOM	Unit of Measure	xs:string	Yes
ItemComments	Comments associated with the order if it is an order transaction like Customer Order	xs:string	No
ItemDisposition	Is to be used for Inventory reservation	xs:string	
DropShipIndicator	Defaulted to false. Reserved for future use.	xs:boolean	
UIN	Serial Number of the item	xs:string	No
UINType	Type of the UIN	xs:string	No

Table 3–17 ProcessPosTxnResponse

Element	Description	Data Type	Required
Successful	Indicates success or failure of the Point-of-Service transaction	posTransaction:ResponseWSType	

Sale Return API

This API is responsible for posting only four types of transactions to SIM. The transactions are Sale, Return, Void Sale and Void Return.

Operation	Purpose
updateSaleReturn	Processes one Sale/Return transaction and updates the stock on hand in the SIM database

Sale Return Types

Table 3–18 UpdateSaleReturnRequest

Element	Description	Data Type	Required
UpdateSaleReturnRequest	This represents one transaction posted to SIM. This consists of all the items present in the transaction.	salereturntxn:UpdateSaleReturnRequest	Yes

Table 3–19 transactionRequest

Element	Description	Data Type	Required
TransactionID	Transaction ID which uniquely identifies the transaction	xs:string	Yes
TransactionType	Type of the transaction	xs:string	Yes
RequestingEntity	The entity requesting the service.	xs:string	Yes
TransactionTimestamp	Time the transaction took place	xs:dateTime	Yes
TransactionItemRequest	Items present in the transaction	salereturntxn:SaleReturnItemWSType	Yes

Table 3–20 SaleReturnItemWSType

Element	Description	Data Type	Required
ItemNumber	ID of the item that is present in the transaction	xs:string	Yes
Action	Specifies the action taken on the item. For example, Sale, Return and so forth.	salereturntxn:SaleReturnActionWSType	Yes
Quantity	Quantity of the item present in the transaction	xs:decimal	Yes
SellingUOM	Unit of measure	xs:string	Yes
ItemType	Comments associated with the order if it is an order transaction like Customer Order	xs:string	No
ItemDisposition	Is to be used for Inventory reservation	xs:string	

Table 3–20 SaleReturnItemWSType (Cont.)

Element	Description	Data Type	Required
DropShipIndicator	Defaulted to false. Reserved for future use	xs:boolean	
UIN	Serial Number of the item	xs:string	No
UINType	Type of the UIN	xs:string	No

Table 3–21 UpdateSaleReturnResponse

Element	Description	Data Type	Required
UpdateStatus	Indicates success or failure of the transaction	xs:boolean	

Serialization API

Used to lookup, create and update Unique Identification Numbers.

Operation	Purpose
LookupUIN	Lookup a UIN/UINs for an item or even existence of a UIN
CreateUIN	Creates a UIN.
UpdateUIN	Updates a UIN
ProcessUINs	Creates/Updates a batch of UINs

Serialization Types

Table 3–22 CreateUINRequest

Element	Description	Data Type	Required
StoreID	Requesting store ID	xs:string	Yes
ItemID	Item ID	xs:string	Yes
UIN	The UIN	xs:string	Yes
UINType	The UIN Type	Uinserial:UINWSType{SERIAL_NUMBER, AGSN}	Yes
ExternalTransactionID	External transaction ID if any that needs to be associated	xs:string	No
UserID	User ID	xs:string	No
Quantity	The quantity	xs:decimal	Yes

Table 3–23 UpdateUINRequest

Element	Description	Data Type	Required
StoreID	Requesting store ID	xs:string	Yes
ItemID	Item ID	xs:string	Yes
UIN	The UIN	xs:string	Yes
UINType	The UIN Type	Uinserial:UINWSType{SERIAL_NUMER, AGSN}	Yes

Table 3–23 UpdateUINRequest (Cont.)

Element	Description	Data Type	Required
ExternalTransactionID	External transaction ID if any that needs to be associated	xs:string	No
UserID	User ID	xs:string	No
Quantity	The quantity	xs:decimal	Yes
Action	The action	Uinserial:UINActionWSType{SALE, RETURN, VOID_SALE, VOID_RETURN}	Yes

Table 3–24 UINRequest

Element	Description	Data Type	Required
ItemID	Item ID	xs:string	Yes
UIN	The UIN	xs:string	Yes
UINType	The UIN Type	Uinserial:UINWSType{SERIAL_NUMER, AGSN}	Yes
ExternalTransactionID	External transaction ID if any that needs to be associated	xs:string	No
UserID	User ID	xs:string	No
Quantity	The quantity	xs:decimal	Yes
Action	The action	Uinserial:UINActionWSType{SALE, RETURN, VOID_SALE, VOID_RETURN}	No

Table 3–25 ProcessUINRequest

Element	Description	Data Type	Required
StoreID	Requesting store ID	xs:string	Yes
UINRequests (1..*)	UIN requests	uinserial:UINRequest	Yes
StagedMode	Indicates if it is staged mode or not	xs:boolean	Yes

Table 3–26 UINInquiryRequest

Element	Description	Data Type	Required
StoreID	Requesting store ID	xs:string	Yes
ItemID	Item ID	xs:string	Yes
UIN	The UIN	xs:string	Yes
Barcode	If not known if identifier is item number or UIN	xs:string	Yes

Table 3–27 CreateUINResponse/UpdateUINResponse

Element	Description	Data Type	Required
ItemID	The item ID	xs:string	
Status	The status	uinserial:UINStatusWSType{ SOLD, SHIPPED_TO_WAREHOUSE, SHIPPED_TO_STORE, RESERVED_FOR_SHIPPING, SHIPPED_TO_VENDOR, REMOVE_FROM_INVENTORY, UNAVAILABLE, MISSING, IN_RECEIVING, CUSTOMER_RESERVED, CUSTOMER_FULFILLED, SHIPPED_TO_FINISHER}	

Table 3–28 UINResponse

Element	Description	Data Type	Required
ItemID	The item ID	xs:string	
UIN	The UIN	xs:string	
Status	The status	uinserial:UINStatusWSType{ SOLD, SHIPPED_TO_WAREHOUSE, SHIPPED_TO_STORE, RESERVED_FOR_SHIPPING, SHIPPED_TO_VENDOR, REMOVE_FROM_INVENTORY, UNAVAILABLE, MISSING, IN_RECEIVING, CUSTOMER_RESERVED, CUSTOMER_FULFILLED, SHIPPED_TO_FINISHER}	
Problem	Problem if any	uinserial:ErrorMessageWSType{ INVALID_ACTION, CANNOT_CHANGE_STATUS, INVALID_STATE, INVALID_ITEM_ID, UIN_DOES_NOT_EXIST, UIN_INVALID_REQUESTING_STORE, INVALID_REQUESTING_LOCATION, ITEM_NOT_RANGED, UIN_ALREADY_EXISTS, STORE_ITEM_NOT_CONFIGURED_FOR_UIN, NO_IDENTIFIER_PROVIDED, UIN_CREATION_FROM_EXTERNAL_SYSTEM_NOT_ALLOWED}	

Table 3–29 ProcessUINResponse

Element	Description	Data Type	Required
UINresponses (0..*)	The item ID	Uinserial:UINResponses	
Successful	When processed in staged mode	xs:string	

Table 3–30 UINInquiryResponse

Element	Description	Data Type	Required
UINInquiryResponseDetails (1..*)	Inquiry response	Uinserial:UINInquiryResponseDetail	
UIN	The UIN	xs:string	
Status	The status	uinserial:UINStatusWSType{ SOLD, SHIPPED_TO_ WAREHOUSE, SHIPPED_TO_ STORE, RESERVED_FOR_ SHIPPING, SHIPPED_TO_ VENDOR, REMOVE_FROM_ INVENTORY, UNAVAILABLE, MISSING, IN_RECEIVING, CUSTOMER_RESERVED, CUSTOMER_FULFILLED, SHIPPED_TO_FINISHER}	

Table 3–31 UINInquiryResponseDetail

Element	Description	Data Type	Required
ItemID	The item ID	xs:string	
UIN	The UIN	xs:string	
UINType	The UIN type	Uinserial:UINWSType{SERIAL_NUMER, AGSN}	
Status	The status	uinserial:UINStatusWSType{ SOLD, SHIPPED_TO_ WAREHOUSE, SHIPPED_ TO_STORE, RESERVED_ FOR_SHIPPING, SHIPPED_ TO_VENDOR, REMOVE_ FROM_INVENTORY, UNAVAILABLE, MISSING, IN_RECEIVING, CUSTOMER_RESERVED, CUSTOMER_FULFILLED, SHIPPED_TO_FINISHER}	

RIB-based Integration

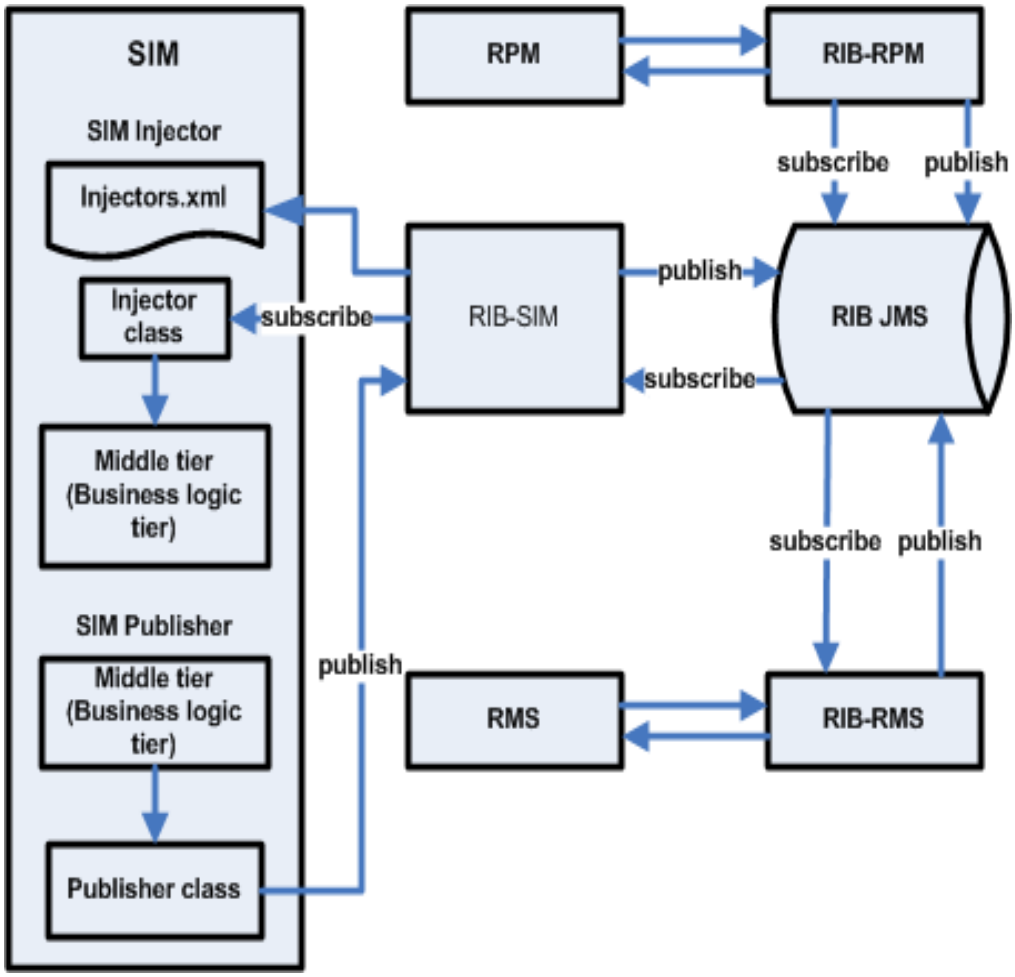
SIM can integrate with other Retail products (such as RMS, a price management system, a warehouse management system) through Oracle Retail Integration Bus (RIB). RIB utilizes publish and subscribe (pub/sub) messaging paradigm with some guarantee of delivery for a message. In a pub/sub messaging system, an adapter publishes a message to the integration bus that is then forwarded to one or more subscribers. The publishing adapter does not know, nor care, how many subscribers are waiting for the message, what types of adapters the subscribers are, what the subscribers current states are (running/down), or where the subscribers are located.

Delivering the message to all subscribing adapters is the responsibility of the integration bus.

See the *Oracle Retail Integration Bus Operations Guide* and other RIB-related documentation for additional information.

For more information about message families and message type names and the XML schema documents that describe the XML messages, see "Appendix: Subscription and Publishing Designs".

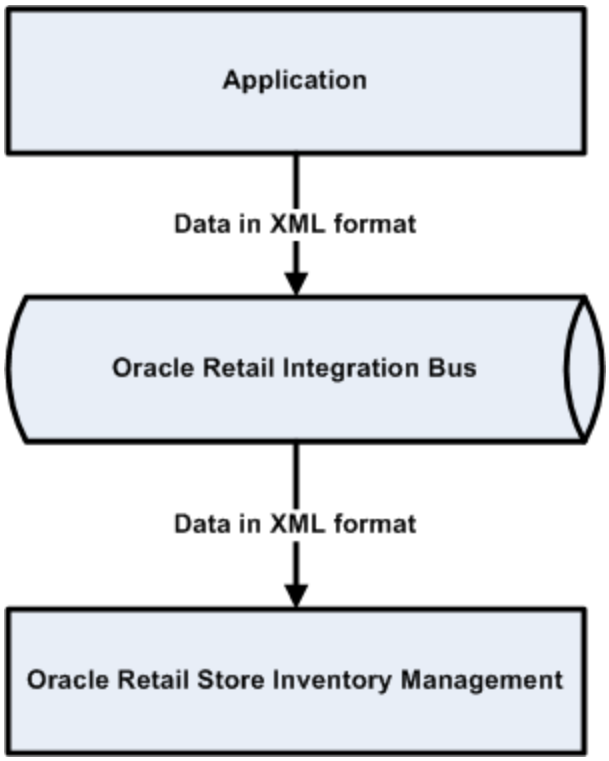
Figure 3–1 SIM/RIB Integration Diagram



The XML Message Format

The XML message format is defined by XML schema document. See "Message Family and Message Types" in the *Oracle Retail Integration Bus Implementation Guide* for additional information.

Figure 3–2 Data Across the RIB in XML Format



SIM Message Subscription Processing

The SIM application subscribes to the JMS topics published by other Oracle Retail applications published to RIB JMS. For each J2EE-based integrated Oracle Retail application (such SIM, a price management system, and so forth), RIB and its corresponding RIB-<app> component are running on the application server (for example, Oracle Application Server) to handle the publishing and subscribing messages through RIB.

On a subscribe operation, the MDB is responsible for taking the XML message from the JMS and calling the appropriate RIB binding code for processing each XML message.

The RIB Binding code is responsible for calling the Subscribing Java application, the corresponding Injector class in the subscribing J2EE application is specified in injectors.xml file. The subscribing application component applies the application specific business logic and injected into the application. If an exception is returned from the subscribing application, the transaction is rolled back and the XML message is sent to the RIB Error Hospital. RIB application utilizes a container-managed transaction and both the JMS and database resources are included in a two-phase commit XA compliant transaction.

See the *Oracle Retail Integration Bus Operations Guide* and other RIB-related documentation for additional information on message subscription process.

RIB Message Publication Processing

SIM publishes message (payload) to RIB's JMS through RIB-SIM component, and RIB Binding subsystem converts the payload object into an XML string. The object on the Binding subsystem is put into a RIB envelope called RibMessage. The data within RibMessage eventually becomes a message on the RIB. A Publisher class in the Binding subsystem is called to write the data to the RIB's JMS queue. On a regular basis, the RIB engages in polling the JMS queue, searching for the existence of a message. A publishable message that appears on the JMS queue is processed.

See the *Oracle Retail Integration Bus Operations Guide Release* and other RIB-related documentation for additional information.

RIB Hospital

The RIB Hospital is a set of Java classes and database tables located within the SIM application but owned by the RIB. The RIB Hospital is designed to segregate and trigger re-processing for messages that had some error with their initial processing. The intent is to provide a means to halt processing for messages that cause errors while allowing continued processing for the good messages. The RIB Hospital references tables within SIM (for example, RIB_MESSAGE, RIB_MESSAGE_FAILURE, RIB_MESSAGE_ROUTING_INFO). For more information about the RIB Hospital, see the latest RIB Technical Architecture Guide, RIB Operations Guide, or RIB Hospital Administration online help.

SIM Decoupled from the Oracle Retail Integration Bus (RIB)

SIM has always been designed to interoperate with a merchandising system. By default, SIM has only worked with the Oracle Retail Merchandising System (RMS), because of the SIM dependency on the Oracle Retail Integration Bus (RIB).

SIM 13.2 is decoupled from RIB. Because SIM does not require RIB, SIM can be deployed with merchandising systems of other vendors.

SIM Standalone Integration

What is meant by stand-alone integration? The message flow now looks like the following, for both inbound and outbound messages:

- Inbound:

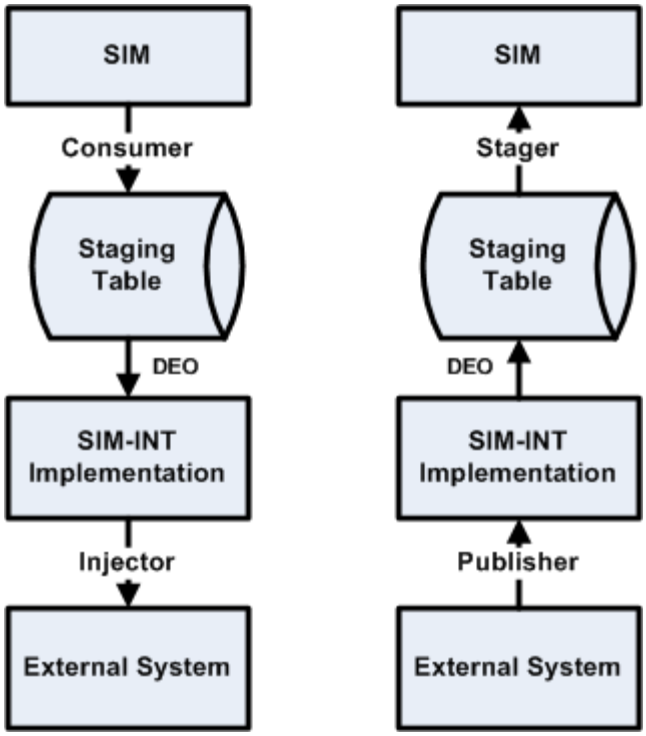
  External App -> Inject -> Consume -> SIM

- Outbound:

  External App <- Publish <- Stage <- SIM

Figure 3–3 provides more information:

Figure 3–3 SIM RIB Decoupling Framework Overview



Other attributes of standalone integration include:

Decoupled Code

The core code (sim-core) is now isolated and insulated from integration points. The integration code (sim-int) for various external systems is now modularized.

Decoupled Message Processing

- Inbound and outbound messages are staged prior to being processed.
- Asynchronous processing (formerly synchronized with an external system) to allow SIM to function even when external system is not available.

The following list defines some important terms in the standalone integration:

- **Stager** – Takes business objects from SIM and stages them to the staging table as outbound messages
- **Publisher** – Takes outbound messages from the staging table and publishes them to an external system
- **Injector** – Takes inbound messages from an external system and injects them into the staging table
- **Consumer** – Takes inbound messages from the staging table and consumes them in SIM
- **Data Exchange Object (DEO)** – Defines the message data held inside a staged message

Stager

The sim-core code can only stage messages rather than publish them.

1. SIM code looks up the appropriate SimMessageStager and calls stage():

   ```
   ASNOutReturnStager
   ```

2. SimMessageStager maps business objects to DEO:

   ```
   ASNOut > ASNOutDEO
   ```

3. SimMessageStager persists DEO to STAGED_MESSAGE table.

> **Note:** Messages will be published asynchronously by the polling timers at a later time (usually within seconds).

Publisher

The polling timers handle staged messages in an abstract way.

1. Polling timers look up the appropriate SimMessageHandler and call handleMessage()

   ```
   PublishHandler
   ```

2. PublishHandler looks up the appropriate SimMessagePublisher and calls publish()

   ```
   SimRibPublisher
   ```

> **Note:** Messages successfully processed will be deleted from the STAGED_MESSAGE table by running PurgeStagedMessage.sh at a later time (up to customer, usually daily).
>
> Performance impact if PurgeStagedMessage.sh isn't run frequently. It is also recommended to re-index after the purge script is run.

Injector

1. RIB-SIM makes a remote EJB call to SIM to inject a message:

   ```
   ApplicationMessageInjector
   ```

2. SIM looks up the appropriate Injector and calls inject():

   ```
   SimMessageRibInjector
   ```

3. Injector maps Payload to a DEO with SimMessageMapperUtil:

   ```
   ASNInDesc -> ASNInDEO)
   ```

4. Injector persists DEO to STAGED_MESSAGE table.

> **Note:** Messages will be consumed asynchronously by the polling timers at a later time (usually within seconds).

Consumer

1. Polling timers look up the appropriate SimMessageHandler and call handleMessage():

   `ConsumeHandler`

2. ConsumeHandler looks up the appropriate SimMessageConsumer and calls consume():

   `ASNInConsumer`

3. Messages are marked as PROCESSED=**Y** in the STAGED_MESSAGE table (or their MESSAGE_ERROR and RETRY_COUNT is modified).

Note: Messages successfully processed will be deleted from the STAGED_MESSAGE table by running PurgeStagedMessage.sh at a later time (usually within seconds).

Figure 3–4 Detailed Injection Flow from External System to SIM

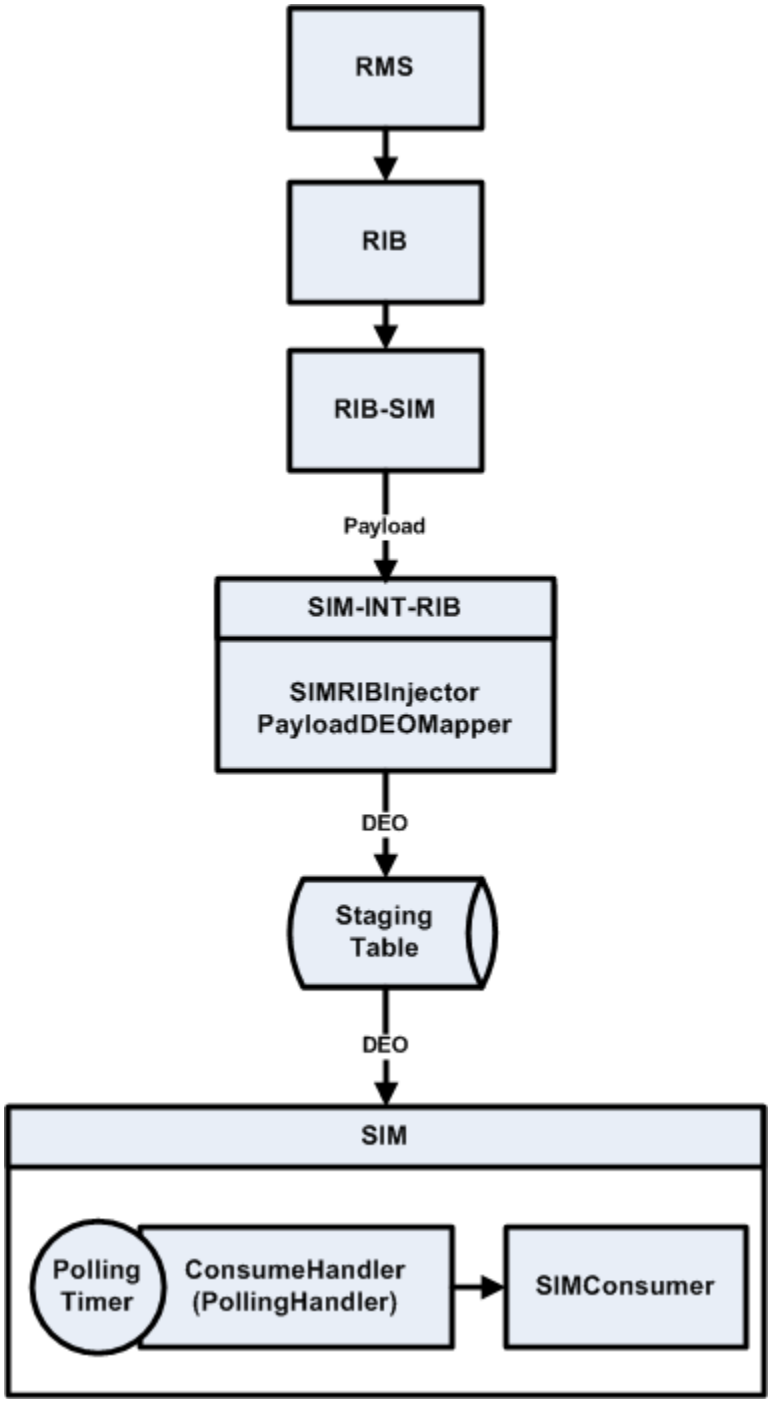
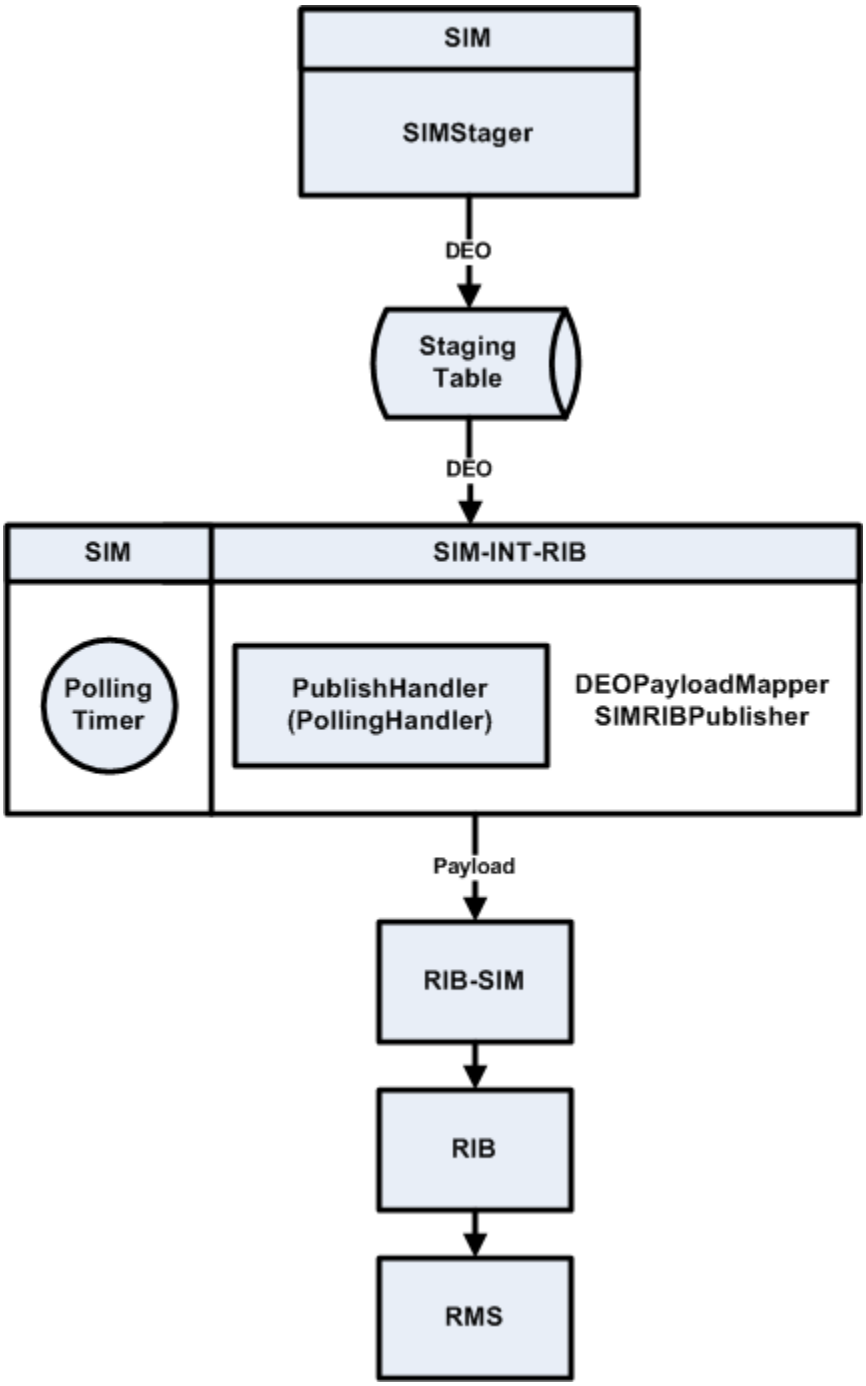


Figure 3–5 Detailed Publish Flow to External System from SIM



Staged Messages

Staged messages are processed by threads within the app server, spawned using an EJB polling timer framework.

Multiple background threads process staged messages concurrently and asynchronously. Messages are grouped and processed by message_family (and business_id if present).

Admin screens are built into the SIM client to allow management of polling timers and staged messages.

The idea of decoupling SIM from any external system relies on the idea of messages being stored in a table on the database. This way, the data becomes detached from the external system (in both directions, inbound and outbound). A new table is introduced named STAGED_MESSAGE. This table holds staged messages. The table looks like the following:

ID
STORE_ID
MESSAGE_FAMILY
MESSAGE_TYPE
MESSAGE_DIRECTION
RETRY_COUNT (number of times the record has been retried)
Documentation Corrections for Polling Timers and Staged Messages 9
CREATE_TIME
UPDATE_TIME
BUSINESS_ID
MESSAGE_DESC
PROCESSED (this flag is either Y or N to denote if this message has been processed)
DELETED (this flag is either Y or N to denote if this message has been deleted)
MESSAGE_ERROR (the reason why it failed)
MESSAGE_DATA (CLOB)

Note: A staged message can be in one of three states:

- PENDING – A staged message that has a RETRY_COUNT = 0.
- RETRY – A staged message that has a RETRY_COUNT > 0 and RETRY_COUNT < the POLLING_TIMERS_MAX_MESSAGE_ RETRIES defined in the server.cfg file.
- FAILED – A staged message that has a RETRY_COUNT >= the POLLING_TIMERS_MAX_MESSAGE_RETRIES defined in the server.cfg file.

Messages are either inbound or outbound, and are processed by multiple threads:

Inbound

An external system injects messages into the staging table that are later consumed by SIM.

Outbound

SIM stages messages into the staging table that later get published to an external system.

Outbound messages are handled in a generic way by SIM and then later are picked up and published by a specific piece of code written to integrate with some external system. Inbound messages are handled by some specific piece of code written to get messages from an external system and transformed and persisted to the staging table in a generic way.

One or more recurring threads that can run on the server side are needed to process these messages. To avoid bottlenecks with just one thread, a configurable sized thread pool is implemented.

J2EE 1.4 introduced the concept of a timer service, which enables developers to create a program that can schedule a business process to occur at a predetermined time or at a regular interval. The EJB container manages the timer service to allow EJB methods to register a call back at a scheduled time or regular interval; EJB timers provide facilities to schedule predetermined tasks and activities. Using stateless beans (PollingCoordinatorBean and PollingTimerBean), timers were created to be used in OracleAS Containers for J2EE (OC4J) that will process the staged messages. The EJB container provides the timer service, which is the infrastructure for the registration and callbacks of timers, and provides the methods for creating and canceling the timers, as well as wrapping everything in transactions.

Inbound messages are not guaranteed to be processed in the order they were injected (due inherently to multi-threaded asynchronous processing). Order can be guaranteed by an external application by populating the BUSINESS_ID field within the message, but this only holds true for messages in the same family. Messages in the same family with the same BUSINESS_ID are processed in the order they were injected. There is no way to guarantee message order between families (for example, Item, ItemLoc, and so forth). Messages in the same family with the same BUSINESS_ID can block subsequent messages if an earlier message is in RETRY or FAIL state (this is in order to guarantee ordering).

A polling timer represents a recurring unit of work and associated attributes to aid in the selection of the staged messages upon which each polling timer should act.

The POLLING_TIMER table looks like the following:

ID
MESSAGE_FAMILY
MESSAGE_DIRECTION
ENABLED

The idea behind each record in this tables is that each {“message family”, “message direction”} tuple uniquely identifies one polling timer. For example, there is a row in the POLLING_TIMER table for {ASNOut, Outbound}. That polling timer will act on outbound messages in the ASNOut family. The POLLING_TIMER table is populated during data seeding and should rarely have any INSERT/DELETE against it. There will be many updates to records in this table.

- PollingCoordinatorBean: The managing thread that has a simple job: it queries the POLLING_TIMER table every five seconds and looks for any polling timers that need to be fired. If PollingCoordinatorBean finds any, it immediately spawns or schedules a PollingTimerBean to fire. The following criteria are used to determine if a polling timer is ready to fire:
  - Check if the polling timer is enabled (turned on)
  - Check if the polling timer is expired (reached/exceeded sleep timeout)
  - Check if the polling timer is not locked (not currently running)

  If these conditions are met, the last job of the coordinator is to query for all staged messages.

Integration Transaction Boundaries

Global transactions (XA) were formerly between core SIM and the external integration point. This meant that a publish would cause a blocking call on the SIM GUI until the external system accepted delivery.

With the staging table in place, SIM can do a simple drop (stage) a message into the staged message table (for increased performance of the SIM GUI) without blocking.

Global transactions are limited to interaction between the integration point: SIM-INT module and the external system.

Application Server Settings

The Oracle Application Server needs to have the executor.concurrent.tasks setting increased to support the number of concurrent threads it can handle. This value must be enough to handle all the threads that could theoretically be firing at any given time. This can be done by setting the system property value on the command line when starting OC4J. Example:

```
-Dexecutor.concurrent.tasks=150
```

The default value for Oracle Application Server is **8**. More information on setting Java system property values can be found here:

http://java.sun.com/j2se/1.5.0/docs/tooldocs/windows/java.html

> **Note:** Transaction timeout of the application server must be set sufficiently high depending on settings for the polling_qty for each thread. For example, if you choose to process 5000 messages on one thread, the timeout must be set much higher than if you were to set 5000 messages over 25 threads since each thread will be using its own transaction and will have a smaller chunk of work to accomplish, therefore less time per thread to complete.

SIM Polling Timer Configuration

Configuration for the polling timer framework can be done in the server.cfg file. The following is an example excerpt:

```
# Enable or disable the entire polling timer sub-system (true|false)
POLLING_TIMERS_ENABLED=true
# Enable or disable the polling timer stats (true|false)
POLLING_TIMERS_STATS_ENABLED=true
# Enable or disable the polling timers from processing outbound messages
(true|false)
```

```
POLLING_TIMERS_PROCESS_OUTBOUND_MESSAGES=true
# Enable or disable the polling timers from processing inbound messages
(true|false)
POLLING_TIMERS_PROCESS_INBOUND_MESSAGES=true
# The maximum number of concurrent polling timer threads that may be running
(integer)
POLLING_TIMERS_MAX_CONCURRENT_THREADS=5
# The maximum number of messages a single polling timer thread may process at once
(integer)
POLLING_TIMERS_MAX_MESSAGES_PER_THREAD=10
# The maximum number of times a message may be retried before it is considered
failed (integer)
# Note: This number can be fairly high to allow the retry polling timers
sufficient attempts
POLLING_TIMERS_MAX_MESSAGE_RETRIES=500
# The amount of time the polling coordinator sleeps before checking on the status
of messages and threads in the system (integer)
# Note: This is a lightweight thread that can be run every 5 to 10 seconds with
little impact on server
POLLING_TIMERS_COORDINATOR_TIMEOUT_SECONDS=10
```

Staged Message Admin Screen

The polling timers process staged messages at an interval defined by POLLING_ TIMERS_COORDINATOR_TIMEOUT_SECONDS in the server.cfg file. Messages are in one of three states:

- PENDING
- RETRY
- FAILED

The staged message screen can be used to RESET or DELETE messages, but more frequently it is used to fix a FAILED message by directly manipulating the invalid data in the XML. This is done by filtering for the specific staged message you are looking for (or across all staged message) and double-clicking on any row. This opens a dialog that enables you to see the error message and edit the data in XML format.

Known Issues and Reminders

- Make sure the polling timer is turned on.
- Make sure the POLLING_TIMERS_MAX_MESSAGE_RETRIES value (defined in server.cfg) is set high enough so messages are given a chance to retry (default is 500).
- Make sure there are no staged messages that exist in RETRY or FAILED state that are holding up other associated messages.
- If the messages are outbound, verify the RIB is up and running. The sim.log shows many stack traces and error messages similar to Check that the RIB is up and running or Unable to connect to the rib.

The following background info might be helpful:

When a polling timer fires, it queries for a certain number of messages to process defined by a calculation of configuration values (POLLING_TIMERS_MAX_CONCURRENT_THREADS and POLLING_TIMERS_MAX_MESSAGES_PER_THREAD). The query only chooses PENDING messages with this exclusion:

```
Exclude all staged messages where MESSAGE_FAMILY = X and BUSINESS_ID = Y if there exists one or more messages with the same family and direction that are in RETRY or FAILED state.
```

There exists a retry timer that processes the messages that are in a RETRY state, but it too has a query to only choose RETRY messages with this exclusion:

```
Exclude all staged messages where MESSAGE_FAMILY = X and BUSINESS_ID = Y if there exists one or more messages with the same family and direction that are in FAILED state.
```

The only difference between the regular polling timer and the retry timer is that the retry timer processes messages one at a time, each having its own transaction. The regular polling timer processes messages in one transaction: if one transaction fails, all transactions fail, which equates to all messages getting an incremented retry count.

Threading of all the polling timers is done by processing messages with available worker threads (from a pool). The number of threads is defined by the POLLING_TIMERS_MAX_CONCURRENT_THREADS in the server.cfg file. This defines how many threads can be running at once, and so on. The defaults for this should be adequate for average load.

Database Considerations

Rebuilding the indexes on the STAGED_MESSAGE table each day is recommended. This must be coordinated to run after the daily purge batch script that removes already processed or deleted messages from the table.

The DBA does the following, either with an SQL script or using dynamic SQL in a PL/SQL module:

1. Reset the high-water mark for the table and rebuild the index(es)
2. ALTER TABLE staged_message ENABLE ROW MOVEMENT;
3. ALTER TABLE staged_message SHRINK SPACE CASCADE;
4. ALTER TABLE staged_message DISABLE ROW MOVEMENT;
5. Gather the table statistics
6. Call DBMS_STATS.GATHER_TABLE_STATS(tabname => staged_message, ESTIMATE_PERCENT => dbms_stats.auto_sample_size, CASCADE => 'true')

The high-water reset requires the staged_message table to exist in a tablespace with automatic segment space management (ASSM), which is already recommended for the required tablespaces (RETEK_DATA, RETEK_INDEX, and so forth). Regardless if this table is in an existing tablespace or whether it will have its own that is separate from the normal tablespaces, the tablespace needs ASSM (which is the default).

For more information about recommendations for the required tablespaces, see the *Oracle Retail Store Inventory Management Installation Guide*.

Subscribers Mapping Table

Table 3–32 lists the message family and message type name and the XML schema documents that describe the XML message. A common SimMessageRibInjector class intercepts all messages, which is responsible to stage the message into SIM. This staged message is then later consumed into SIM. For additional information, see the *Oracle Retail Integration Bus Operations Guide* and other RIB documentation.

For more information about Subscribers, see "Appendix: Subscription and Publishing Designs".

Table 3–32 **Subscribers Mapping Table**

Family	Type	Payload	Consumer
ASNIN	ASNOUTCRE	ASNInDesc	ASNInCreateConsumer
ASNIN	ASNINDEL	ASNInRef	ASNInRemoveConsumer
ASNIN	ASNINMOD	ASNInDesc	ASNInModifyConsumer
CLRPRCCHG	CLRPRCCHGCRE	ClrPrcChgDesc	ClrPrcChgCreateConsumer
CLRPRCCHG	CLRPRCCHGMOD	ClrPrcChgDesc	ClrPrcChgModifyConsumer
CLRPRCCHG	CLRPRCCHGDEL	ClrPrcChgRef	ClrPrcChgRemoveConsumer
DIFFS	DIFFCRE	DiffDesc	DifferentiatorCreateConsumer
DIFFS	DIFFDEL	DiffRef	DifferentiatorRemoveConsumer
DIFFS	DIFFMOD	DiffDesc	DifferentiatorModifyConsumer
ITEMS	ITEMBOMCRE	ItemBOMDesc	ItemBOMCreateConsumer
ITEMS	ITEMBOMDEL	ItemBOMRef	ItemBOMRemoveConsumer
ITEMS	ITEMBOMMOD	ItemBOMDesc	ItemBOMModifyConsumer
ITEMS	ITEMCRE	ItemDesc	ItemCreateConsumer
ITEMS	ITEMDEL	ItemRef	ItemRemoveConsumer
ITEMS	ITEMHDRMOD	ItemHdrDesc	ItemModifyConsumer
ITEMS	ITEMIMAGECRE	ItemImageDesc	ItemImageCreateConsumer
ITEMS	ITEMIMAGEDEL	ItemImageRef	ItemImageRemoveConsumer
ITEMS	ITEMIMAGEMOD	ItemImageDesc	ItemImageModifyConsumer
ITEMS	ITEMSUPCRE	ItemSupCtyDesc	ItemSupCreateConsumer
ITEMS	ITEMSUPCTYCRE	ItemSupCtyRef	ItemSupCtyCreateConsumer
ITEMS	ITEMSUPCTYDEL	ItemSupCtyRef	ItemSupCtyRemoveConsumer
ITEMS	ITEMSUPCTYMOD	ItemSupCtyDesc	ItemSupCtyModifyConsumer
ITEMS	ITEMSUPDEL	ItemSupRef	ItemSupRemoveConsumer
ITEMS	ITEMSUPMOD	ItemSupDesc	ItemSupModifyConsumer
ITEMS	ITEMUPCCRE	ItemUPCDesc	ItemUPCCreateConsumer
ITEMS	ITEMUPCDEL	ItemUPCRef	ItemUPCRemoveInjector
ITEMS	ITEMUPCMOD	ItemUPCDesc	ItemUPCModifyInjector
ITEMS	ISCDIMCRE	ISCDimDesc	ISCDimCreateConsumer
ITEMS	ISCDIMMOD	ISCDimDesc	ISCDimModifyConsumer
ITEMS	ISCDIMDEL	ISCDimRef	ISCDimRemoveConsumer
ITEMS	ISCMFRMOD	ItemSupCtyMfrDesc	ItemSupCtyMfrModifyConsumer
ITEMS	ISCMFRDEL	ItemSupCtyMfrRef	ItemSupCtyMfrRemoveConsumer

Table 3–32 **Subscribers Mapping Table**

Family	Type	Payload	Consumer
ITEMS	ISCMFRCRE	ItemSupCtyMfrDesc	ItemSupCtyMfrCreateConsumer
ITEMS	ITEMTCKTCRE	itemTcktDesc	ItemTcktCreateConsumer
ITEMS	ITEMTCKTDEL	ItemTcktRef	ItemTcktRemoveConsumer
ITEMS	ITEMUDADATECRE	ItemUDADateDesc	ItemUDACreateConsumer
ITEMS	ITEMUDADATEDEL	ItemUDADateRef	ItemUDARemoveConsumer
ITEMS	ITEMUDADATEMOD	ItemUDADateDesc	ItemUDAModifyConsumer
ITEMS	ITEMUDAFFCRE	ItemUDAFFDesc	ItemUDACreateConsumer
ITEMS	ITEMUDAFFDEL	ITEMUDAFFRef	ItemUDARemoveConsumer
ITEMS	ITEMUDAFFMOD	ItemUDAFFDesc	ItemUDAModifyConsumer
ITEMS	ITEMUDALOVCRE	ItemUDALOVDesc	ItemUDACreateConsumer
ITEMS	ITEMUDALOVDEL	ItemUDALOVRef	ItemUDARemoveConsumer
ITEMS	ITEMUDALOVMOD	ItemUDALOVDesc	ItemUDAModifyConsumer
ORDER	POCRE	PODesc	PurchaseOrderCreateConsumer
ORDER	PODEL	PORef	PurchaseOrderRemoveConsumer
ORDER	PODTLCRE	PODesc	PurchaseOrderDetailCreateConsumer
ORDER	PODTLDEL	PORef	PurchaseOrderDetailRemoveConsume r
ORDER	PODTLMOD	PODesc	PurchaseOrderDetailModifyConsumer
ORDER	POHDRMOD	PODesc	PurchaseOrderModifyConsumer
PARTNER	PARTNERCRE	PartnerDesc	PartnerCreateConsumer
PARTNER	PARTNERMOD	PartnerDesc	PartnerModifyConsumer
PARTNER	PARTNERDEL	PartnerRef	PartnerRemoveConsumer
PARTNER	PARTNERDTLCRE	PartnerDesc	PartnerAddressCreateConsumer
PARTNER	PARTNERDTLMOD	PartnerDesc	PartnerAddressModifyConsumer
PARTNER	PARTNERDTLDEL	PartnerRef	PartnerAddressRemoveConsumer
PRCCHGCONF	PRCCHGCONFCRE	PrcChgConfDesc	PrcChgConfCreateConsumer
PRMPRCCHG	MULTIBUYPROMOCRE	PrmPrcChgDesc	PrmPrcChgCreateConsumer
PRMPRCCHG	MULTIBUYPROMODEL	PrmPrcChgDesc	PrmPrcChgModifyConsumer
PRMPRCCHG	MULTIBUYPROMOMOD	PrmPrcChgRef	PrmPrcChgRemoveConsumer
REGPRCCHG	REGPRCCHGCRE	RegPrcChgDesc	RegPrcChgCreateConsumer
REGPRCCHG	REGPRCCHGMOD	RegPrcChgDesc	RegPrcChgModifyConsumer
REGPRCCHG	REGPRCCHGDEL	RegPrcChgRef	RegPrcChgRemoveConsumer
RCVUNITADJMOD	RCVUNITADJDTL	RcvUnitAdjDesc	RcvUnitAdjModConsumer
RTVREQ	RTVREQCRE	RTVReqDesc	RTVReqCreateConsumer
RTVREQ	RTVREQMOD	RTVReqDesc	RTVReqModifyConsumer
RTVREQ	RTVREQDEL	RTVReqRef	RTVReqRemoveConsumer
RTVREQ	RTVREQDTLCRE	RTVReqDesc	RTVReqDetailCreateConsumer
RTVREQ	RTVREQDTLDEL	RTVReqRef	RTVReqDetailRemoveConsumer
RTVREQ	RTVREQDTLMOD	RTVReqDesc	RTVReqDetailModifyConsumer
SEEDDATA	DIFFTYPECRE	DiffTypeDesc	DifferentiatorTypeCreateConsumer

Table 3–32 **Subscribers Mapping Table**

Family	Type	Payload	Consumer
SEEDDATA	DIFFTYPEDEL	DiffTypeRef	DifferentiatorTypeRemoveConsumer
SEEDDATA	DIFFTYPEMOD	DiffTypeDesc	DifferentiatorTypeModifyConsumer
SOSTATUS	SOSTATUSCRE	SOStatusDesc	StockOrderStatusConsumer
STOCKORDER	SOCRE	SODesc	StockOrderCreateConsumer
STOCKORDER	SODTLCRE	SODesc	StockOrderCreateConsumer
STOCKORDER	SODTLDEL	SORef	StockOrderRemoveConsumer
STOCKORDER	SODTLMOD	SODesc	StockOrderModifyConsumer
STOCKORDER	SOHDRDEL	SORef	StockOrderRemoveConsumer
STOCKORDER	SOHDRMOD	SODesc	StockOrderModifyConsumer
STORES	STORECRE	StoresDesc	StoreCreateConsumer
STORES	STOREDEL	StoresRef	StoreRemoveConsumer
STORES	STOREMOD	StoresDesc	StoreModifyConsumer
VENDOR	VENDORADDRCRE	VendorAddrDesc	SupplierAddrCreateConsumer
VENDOR	VENDORADDRDEL	VendorAddrRef	SupplierAddrRemoveConsumer
VENDOR	VENDORADDRMOD	VendorAddrDesc	SupplierAddrModifyConsumer
VENDOR	VENDORCRE	VendorDesc	SupplierCreateConsumer
VENDOR	VENDORDEL	VendorRef	SupplierRemoveConsumer
VENDOR	VENDORHDRMOD	VendorHdrDesc	SupplierModifyConsumer
VENDOR	VENDOROUCRE	VendorDesc	SupplierCreateConsumer
VENDOR	VENDOROUDEL	VendorDesc	SupplierRemoveConsumer
MERCHANDISE HIERARCHY	DEPTCRE	MrchHrDeptDesc	MrchDeptCreateConsumer
MERCHANDISE HIERARCHY	DEPTMOD	MrchHrDeptDesc	MrchDeptModifyConsumer
MERCHANDISE HIERARCHY	DEPTDEL	MrchHrDeptRef	MrchDeptRemoveConsumer
MERCHANDISE HIERARCHY	CLASSCRE	MrchHrClsDesc	MrchClassCreateConsumer
MERCHANDISE HIERARCHY	CLASSMOD	MrchHrClsDesc	MrchClassModifyConsumer
MERCHANDISE HIERARCHY	CLASSDEL	MrchHrClsRef	MrchClassRemoveConsumer
MERCHANDISE HIERARCHY	SUBCLASSCRE	MrchHrSclsDesc	MrchSubclassCreateConsumer
MERCHANDISE HIERARCHY	SUBCLASSMOD	MrchHrSclsDesc	MrchSubclassModifyConsumer
MERCHANDISE HIERARCHY	SUBCLASSDEL	MrchHrSclsRef	MrchSubclassRemoveConsumer
DELIVERYSLOT	DLVYSLTCRE	DeliverySlotDesc	DeliverySlotCreateConsumer
DELIVERYSLOT	DLVYSLTMOD	DeliverySlotDesc	DeliverySlotModifyConsumer
DELIVERYSLOT	DLVYSLTDEL	DeliverySloRef	DeliverySlotRemoveConsumer
UDAS	UDAHDRCRE	UDADesc	UDACreateConsumer
UDAS	UDAHDRDEL	UDARef	UDARemoveConsumer

Table 3–32 **Subscribers Mapping Table**

Family	Type	Payload	Consumer
UDAS	UDAHDRMOD	UDADesc	UDAModifyConsumer
UDAS	UDAVALCRE	UDAValDesc	UDAValueCreateConsumer
UDAS	UDAVALDEL	UDAValRef	UDAValueRemoveConsumer
UDAS	UDAVALMOD	UDAValDesc	UDAValueModifyConsumer
WH	WHCRE	WHDesc	WareHouseCreateConsumer
WH	WHDEL	WHRef	WareHouseRemoveConsumer
WH	WHMOD	WHDesc	WareHouseModifyConsumer

Publishers Mapping Table

Table 3–33 illustrates the relationship among the message family, message type and the DTD/payload object that the application creates. For additional information, see the *Oracle Retail Integration Bus Operations Guide* and other RIB documentation.

For more information about Publishers, see "Appendix: Subscription and Publishing Designs".

Table 3–33 **Publishers Mapping Table**

Family	Type	Payload
ASNOUT	ASNOUTCRE	ASNOutDesc
DSDRECEIPT	DSDRECEIPTCRE	DSDReceiptDesc
INVADJUST	INVADJUSTCRE	InvAdjustDesc
INVREQ	INVREQCRE	InvReqDesc
PRCCHGREQ	PRCCHGREQCRE	PrcChgReqDesc
RECEIVING	RECEIPTCRE	ReceiptDesc
RECEIVING	RECEIPTMOD	ReceiptDesc
RTV	RTVCRE	RTVDesc
SOSTATUS	SOSTATUSCRE	SOStatusDesc
STKCOUNTSCH	STKCOUNTSCHCRE	StkCountSchDesc
STKCOUNTSCH	STKCOUNTSCHDEL	StkCountSchRef
STKCOUNTSCH	STKCOUNTSCHMOD	StkCountSchDesc

A

Appendix: Oracle Retail Point-of-Service to Oracle Retail Store Inventory Management

Oracle Retail Point-of-Service to Oracle Retail Store Inventory Management Overview

Oracle Retail Store Inventory Management enables store personnel to quickly and easily perform an array of in-store operations using a high-speed internet connection and portable, handheld wireless devices to receive merchandise, manage physical inventories, conduct stock counts, order stock, or transfer stock.

This integration enables an operator at Oracle Retail Point-of-Service to perform an inventory inquiry to Oracle Retail Store Inventory Management.

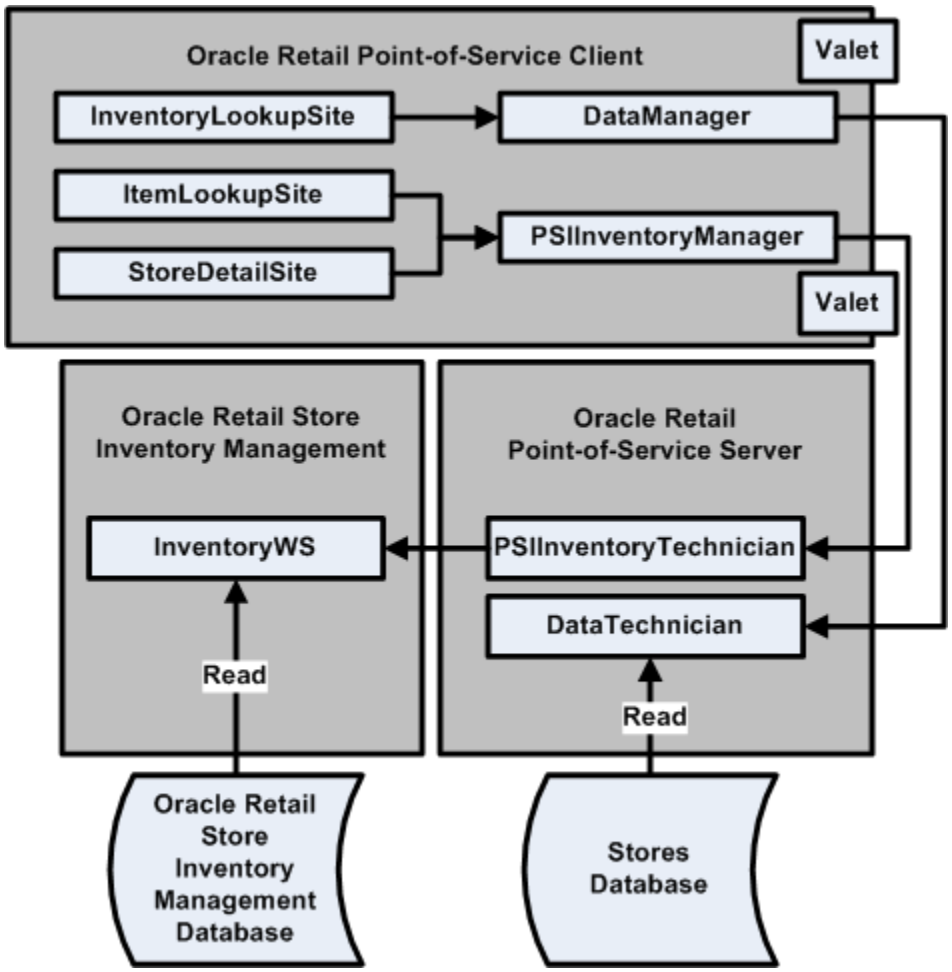
Oracle Retail Point-of-Service can request inventory information for a single store or for a group of stores. The operator can request inventory numbers of an item in the home store, stores within the related buddy stores (buddy store functionality enables the retailer to set up a group of stores within a transfer zone in Store Inventory Management to which the retailer often transfers items), stores within the related transfer zones (a set of locations where transfers are allowed) or for a specific store. Item inquiry can search on one item at a time. You can perform an item inquiry during a transaction, as well as outside a transaction.

The reply from Oracle Retail Store Inventory Management contains item, location and inventory information.

The default topology for Oracle Retail Store Inventory Management is centralized multi-store.

Figure A–1 depicts the interaction of the Oracle Retail Point-of-Service Client and Server with Oracle Retail Store Inventory Management.

Figure A–1 High-Level Model for Oracle Retail Point-of-Service to Oracle Retail Store Inventory Management Integration



ItemLookupSite and StoreDetailSite use existing DataManager/DataTechnician components for item lookup and Store information respectively. InventoryLookupSite uses PSIInventoryManager to get item inventory information. The item information is passed to PSIInventoryTechnician through Valet. PSIInventoryTechnician gets item inventory information from InventoryWS Web service deployed in Oracle Retail Store Inventory Management.

Note: Communication to Oracle Retail Store Inventory Management over HTTPS is not supported.

Communication is unidirectional. Oracle Retail Store Inventory Management sends inventory information only. Oracle Retail Point-of-Service does not send transaction information to Oracle Retail Store Inventory Management.

Oracle Retail Point-of-Service to Oracle Retail Store Inventory Management Architecture

The Oracle Retail Point-of-Service to Oracle Retail Store Inventory Management integration is intended to provide integration for Point-of-Service to interact with Oracle Retail Store Inventory Management application for inventory information.

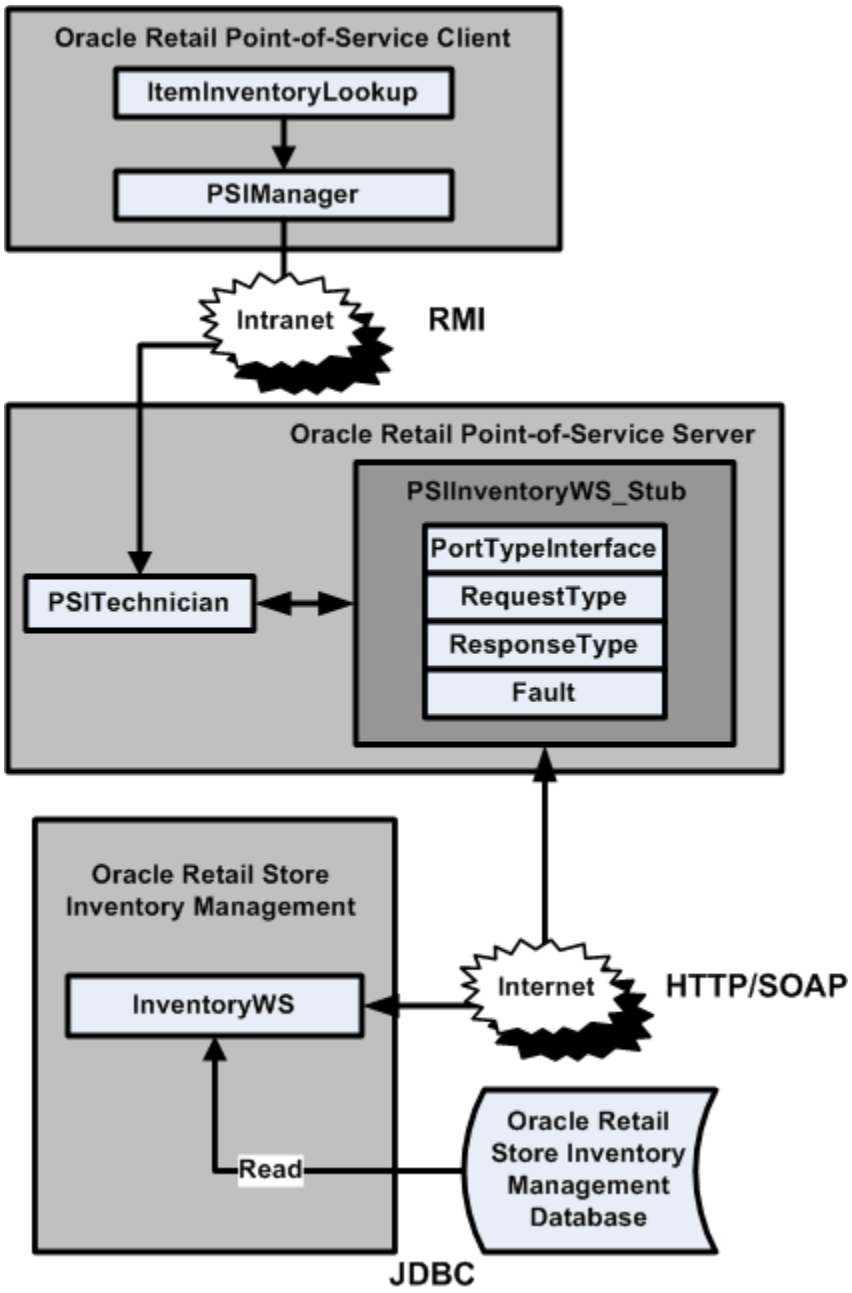
This feature is provided to enable Oracle Retail Point-of-Service to check the item inventory in Home Store, Buddy Store, Specific Store and Transfer zone. The Item Inventory feature is available to Oracle Retail Point-of-Service Client only when the Point-of-Service Client is in the ONLINE mode.

There is a need for operators at Oracle Retail Point-of-Service to check the item inventory from the Oracle Retail Point-of-Service Client. The item inventory information is not available in stores server database; therefore, Oracle Retail Point-of-Service is unable to get item inventory information. The Oracle Retail Point-of-Service to Oracle Retail Store Inventory Management integration is intended to integrate Oracle Retail Point-of-Service with the Oracle Retail Store Inventory Management to get item inventory information. Oracle Retail Point-of-Service gets the item information and stores information from the stores database and interacts with Oracle Retail Store Inventory Management for item inventory information.

The following outlines the Oracle Retail Point-of-Service to Oracle Retail Store Inventory Management integration approach:

1. Expose the item inventory information from Oracle Retail Store Inventory Management in the form of Web service.
2. Provide new tour for item inventory inquiry in Oracle Retail Point-of-Service.
3. Provide pluggable inventory lookup interface to integrate Oracle Retail Point-of-Service to Oracle Retail Store Inventory Management.
4. Oracle Retail Point-of-Service Client interacts with Oracle Retail Point-of-Service Server over RMI as in the existing Point-of-Service architecture. Oracle Retail Point-of-Service Server interacts with inventory lookup interface to interact with Oracle Retail Store Inventory Management.

Figure A–2 Oracle Retail Point-of-Service to Oracle Retail Store Inventory Management Architecture



The Oracle Retail Point-of-Service to Oracle Retail Store Inventory Management integration system is broken into four main sub-systems:

Oracle Retail Point-of-Service Client
Changes to the Oracle Retail Point-of-Service Client to incorporate new inventory lookup tour and new component PSIManager for interaction with PSITechnician for Item inventory lookup.

Oracle Retail Point-of-Service Server

New Component PSITechnician is added in Oracle Retail Point-of-Service Server. PSITechnician interacts with PSIInventoryWS_Stub to call InventoryWS over intranet using HTTP/SOAP protocol.

ORSIM Server

InventoryWS component deployed in Oracle Retail Store Inventory Management Server provides item inventory information. InventoryWS gets item inventory information from ORSIM DB.

ORSIM DB

Oracle Retail Store Inventory Management Inventory Database.

Error Handling

Error handling is limited to logging errors during the inventory lookup. The exceptions such as IOException and invalidItem that occur during WSService communication are re-thrown as WSException, as well as logged for error tracking and resolution.

Logging

Oracle Retail Point-of-Service to Oracle Retail Store Inventory Management uses Log4J for logging. The following logging levels can be used:

- Info: For logging information messages.
- Debug: For logging all the debug messages.
- Error: For logging application errors.

However, the logging level can be configured with log4J.xml.

B

Appendix: Subscription and Publishing Designs

Messages are either inbound or outbound:

- Inbound: An external system injects messages into the staging table that later get consumed by SIM.
- Outbound: SIM stages messages into the staging table that later get published to an external system.

The messages that are considered outbound are handled in a generic way by SIM and then later are picked up and published by a specific piece of code written to integrate with some external system. Similarly, inbound messages are handled by some specific piece of code written to get messages from an external system and transformed and persisted to the staging table in a generic way.

- Data Exchange Object (DEO): An object that encapsulates outbound and inbound data internally to SIM. This object is passed between sim and sim-int-* (for example, sim-int-rib13)
- Stager: A class which takes the SIM outbound data (for example, business objects) and wraps them in the specific DEO and then persists the information to the Staging table (as a StagedMessage object).
- Publisher: A class which takes the outbound staging records (DEO) then converts and publishes them to an external system (for example, RIB). Any given implementation of this object would reside in the respective sim-int-* implementation.
- Injector: A class which takes inbound data (such as Payload) and injects it into the staging table by mapping the data from the Payload into a DEO and then finally persisting it as a StagedMessage
- Consumer: A class which processes SIM inbound staging records by consuming staging records and persisting into SIM transactional tables.

Note: SIM DataExchangeObject (DEO) versus RIB Payload

SIM DEO is similar to a RIB payload in that it is a wrapper for a representation of data

RIB payloads are meant to be externally shared between RIB and other systems that integrate with RIB

SIM DEO is an internal representation of a functional chunk of data that only SIM needs and is only changed internally from the core component to SIM integration components.

For more information, see Chapter 3, "SIM Integration – Technical".

Subscribers

Oracle Retail Stores Inventory Management subscribes to the JMS topics published by other Oracle Retail applications published to RIB JMS. For each J2EE-based integrated Oracle Retail application (such as SIM), RIB and its corresponding RIB-<app> component are running on the application server (for example, Oracle Application Server) to handle the publishing and subscribing messages through RIB.

A common SimMessageRibInjector class intercepts all messages, which is responsible to convert external message payload (for example, RIB Payload) into SIM Data Exchange Object (DEO) and stage the message into SIM. This staged message is then later consumed into SIM. For additional information, see the *Oracle Retail Integration Bus Operations Guide* and other RIB documentation.

Note: Mapping between message types and SIM injectors occurs in injectors.xml.

Subscription API Message Family: Asnin

Business Overview

These messages contain inbound shipment notifications from both vendors (PO shipments) and warehouses (transfer and allocation shipments).

Integration to Products

A warehouse management system, supplier (external).

Message Type: Asnincre – Create Shipment

Primary APIs

- Payload: ASNInDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ASNInCreateConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: ASNInDEO

Notes

Not applicable.

Message Type: Asnindel – Deletes Shipment

Primary APIs

- Payload: ASNInRef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ASNInRemoveConsumer.consume (DataExchangeObject)
- SIM Data Exchange Object: ASNInRefDEO

Notes

Not applicable.

Message Type: Asninmod – Updates Shipment

Primary APIs

- Payload: ASNInDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ASNInModifyConsumer.consume (DataExchangeObject)
- SIM Data Exchange Object: ASNInDEO

Notes

Not applicable.

Primary Tables Involved

Table
STAGED_MESSAGE
RK_SHIPMENTS
RK_SHIPMENT_CARTON
RK_SHIPMENT_ITEM
RK_SHIPMENT_LINE_ITEM_UIN
UIN_DETAIL
PA_SPR
PARTNER
RSS_MINI_LOCATION (view)

Subscription API Message Family: ClearancePriceChange (ClrPrcChg)

Business Overview

These messages are used by a price management system to communicate the approval of a clearance price change or a clearance price change reset within the application. This message is published at a transaction item/location level, and is used by SIM for visibility to the new clearance retail on the effective date for the clearance price change through the effective date for the clearance price change reset.

Integration to Products

A price management system

Message Type: ClrPrcChgCre

Primary APIs

- Payload: ClrPrcChgDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ClrPrcChgCreateConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: ClrPrcChgDEO

Notes

Not applicable.

Message Type: ClrPrcChgMod

Primary APIs

- Payload: ClrPrcChgDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ClrPrcChgModifyConsumer.consume (DataExchangeObject)
- SIM Data Exchange Object: ClrPrcChgDEO

Notes

Not applicable.

Message Type: ClrPrcChgDel

Primary APIs

- Payload: ClrPrcChgRef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ClrPrcChgRemoveConsumer. consume (DataExchangeObject)
- SIM Data Exchange Object: ClrPrcChgRefDEO

Notes

Not applicable.

Primary Tables Involved

Table
AS_ITM_RTL_STR
LE_HST_ITM_SLS_PRC

Subscription API Message Family: Differentiator ID (Diffs)

Business Overview

These messages are used to communicate differentiator IDs from RMS to SIM.

Integration to Products

RMS

Message Type: DiffCre – Creates Differentiator

Primary APIs

- Payload: DiffDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: DifferentiatorCreateConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: DiffDEO

Notes

Not applicable.

Message Type: DiffDel – Deletes Differentiator

Primary APIs

- Payload: DiffRef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: DifferentiatorRemoveConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: DiffRefDEO

Notes

Not applicable.

Message Type: DiffMod – Modify Differentiator

Primary APIs

- Payload: DiffDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: DifferentiatorModifyConsumer.consume(DataExchangeDEO)
- SIM Data Exchange Object: DiffDEO

Notes

Not applicable.

Primary Tables Involved

Table
RK_DIFF_DESC

Subscription API Message Family: Item

Business Overview

These are messages communicated by RMS that contain all approved items records, including header information, item/supplier, and item/supplier/country details, and item/ticket information. The item/supplier/manufacturer and the Item/Supplier/Dimension information also gets published to SIM by this message family as part of this release.

Integration to Products

RMS

Message Type: ItemMod – Updates item

Primary APIs

- Payload: ItemDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ItemModifyConsumer.consume (DataExchangeObject)
- SIM Data Exchange Object: ItemDEO

Notes

Populates item information in the SIM database. This information is all the master information related to items.

Primary Tables Involved

Table
AS_ITM

Message Type: ItemDel – Deletes item

Primary APIs

- Payload: ItemRef
- Injector: SimMessageRibInjector.consume (messageType, businessId, Payload)
- Consumer: ItemRemoveConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: ItemRefDEO

Notes

Marks the item as deleted in AS_ITM table.

Primary Tables Involved

Table
AS_ITM

Message Type: ItemBomCre – Create item bill of materials

Primary APIs

- Payload: ItemBomDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ItemBomCreateConsumer.consume (DataExchangeObject)
- SIM Data Exchange Object: ItemBOMDEO

Notes

Populates information such as item_id, pack_no and quantity, and so forth.

Primary Tables Involved

Table
CO_CLN_ITM

Message Type: ItemBomDel – Delete item bill of materials

Primary APIs

- Payload: ItemBomRefDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ItemBomRemoveConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: ItemBOMRefDEO

Notes

Populates information such as item_id, pack_no and quantity, and so forth.

Primary Tables Involved

Table
AS_ITM

Message Type: ItemBomMod – Updates item bill of materials

Primary APIs

- Payload: ItemBomDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ItemBomModifyConsumer (DataExchangeObject)
- SIM Data Exchange Object: ItemBomDEO

Notes

Populates information such as item_id, pack_no and quantity, and so forth.

Primary Tables Involved

Table
CO_CLN_ITM

Message Type: ItemCre – Creates item

Primary APIs

- Payload: ItemDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ItemCreateConsumer.consume (DataExchangeObject)
- SIM Data Exchange Object: ItemDEO

Notes

Populates item information in the SIM database. This information is all the master information related to items.

Primary Tables Involved

Table
AS_ITM
RK_ITEM_SUPPLIER
RK_ITEM_SUPP_COUNTRY
ITEM_SUPP_MANUFACTURE
ITEM_SUPP_COUNTRY_DIM

Message Type: ItemHdrMod – Updates header info for Item

Primary APIs

- Payload: ItemHdrDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ItemModifyConsumer.consume (DataExchangeObject)
- SIM Data Exchange Object: ItemHdrDEO

Notes

Marks the item as deleted in AS_ITM table

Primary Tables Involved

Table
AS_ITM

Message Type: ItemSupCre – Create item-supplier

Primary APIs

- Payload: ItemSupDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ItemSupCreateConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: ItemSupDEO

Notes

Not applicable.

Primary Tables Involved

Table
RK_ITEM_SUPPLIER

Message Type: ItemSupCtyCreate – Creates Item Supplier Country

Primary APIs

- Payload: ItemSupCtyDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ItemSuppCtyCreateConsumer.consume (DataExchangeObject)
- SIM Data Exchange Object: ItemSupCtyDEO

Notes

Puts data into the RK_ITEM_SUPP_COUNTRY as well as the AS_ITM table. It updates the case size into the AS_ITM table after data is persisted in RK_ITEM_SUPP_ COUNTRY.

Primary Tables Involved

Table
RK_ITEM_SUPP_COUNTRY
AS_ITM

Message Type: ItemSupCtyDel – Removes Item Supplier Country

Primary APIs

- Payload: ItemSupCtyRef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ItemSupCtyRemoveConsumer.consume (DataExchangeObject)
- DataExchangeObject: ItemSupCtyRefDEO

Notes

Deletes data from RK_ITEM_SUPP_COUNTRY

Primary Tables Involved

Table
RK_ITEM_SUPP_COUNTRY

Message Type: ItemSupCtyMod – Updates Item Supplier Country

Primary APIs

- Payload: ItemSupCtyDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ItemSupCtyModifyConsumer.consume (DataExchangeObject)
- SIM Data Exchange Object: ItemSupCtyDEO

Notes

Deletes data from RK_ITEM_SUPP_COUNTRY.

Primary Tables Involved

Table
RK_ITEM_SUPP_COUNTRY
AS_ITM

Message Type: ItemSupDel – Deletes item-supplier

Primary APIs

- Payload: ItemSupRef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ItemSupRemoveConsumer.consume (DataExchangeObject)
- SIM Data Exchange Object: ItemSupRefDEO

Notes

Not applicable.

Primary Tables Involved

Table
RK_ITEM_SUPPLIER

Message Type: ItemSupMod – Updates item supplier

Primary APIs

- Payload: ItemSupDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ItemSupModifyConsumer (DataExchangeObject)
- SIM Data Exchange Object: ItemSupDEO

Notes

Updates the primary supplier indicator and vpn attributes.

Primary Tables Involved

Table
RK_ITEM_SUPPLIER

Message Type: ItemSupCre – Creates Item Supplier

Primary APIs

- Payload: ItemSupDesc
- Injector: SimMessageRibInjector.inject (messageType, ItemSupDesc)
- Consumer: ItemSupCreateConsumer.consume (DataExchangeObject)
- SIM Data Exchange Object: ItemSupDEO

Notes

Not applicable.

Primary Tables Involved

Table
RK_ITEM_SUPPLIER

Message Type: ItemUpcCre – Creates Item UPC

Primary APIs

- Payload: ItemUPCDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ItemUPCCreateConsumer.consume (DataExchangeObject)
- SIM Data Exchange Object: ItemUPCDEO

Notes

Not applicable.

Primary Tables Involved

Table
AS_ITM

Message Type: ItemUpcDel – Removes Item UPC

Primary APIs

- Payload: ItemUPCRef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ItemUPCRemoveConsumer.consume (DataExchangeObject)
- SIM Data Exchange Object: ItemUPCRefDEO

Notes

Not applicable.

Primary Tables Involved

Table
AS_ITM

Message Type: ItemUpcMod – Updates Item UPC

Primary APIs

- Payload: ItemUPCDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ItemUPCModifyConsumer.consume (DataExchangeObject)
- SIM Data Exchange Object: ItemUPCDEO

Notes

Updates barcode information like barcode format and barcode prefix, item level and so forth.

Primary Tables Involved

Table
AS_ITM

Message Type: ItemMfrCre – Creates Item Supplier Country of Manufacture

Primary APIs

- Payload: ItemSupCtyMfrDesc
- Injector: SimMessageRibInjector.inject(messageType, businessId, Payload)
- Consumer: ItemSupCtyMfrCreateConsumer.consume (DataExchangeObject)
- SIM DataExchangeObject: ItemSupCtyMfrDEO

Notes

Not applicable.

Primary Tables Involved

Table
ITEM_SUPP_MANUFACTURE

Message Type: ItemMfrDel – Removes Item Supplier Country of Manufacture

Primary APIs

- Payload: ItemSupCtyMfrRef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ItemSupCtyMfrRemoveConsumer.consume (DataExchangeObject)
- SIM Data Exchange Object: ItemSupCtyMfrRefDEO

Notes

Not applicable.

Primary Tables Involved

Table
ITEM_SUPP_MANUFACTURE

Message Type: ItemMfrMod – Updates Item Supplier Country of Manufacture

Primary APIs

- Payload: ItemSupCtyMfrDesc
- Injector: SimMessageRibInjector.inject(messageType, businessId, Payload)
- Consumer: ItemSupCtyMfrModifyConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: ItemSupCtyMfrDEO

Notes

Not applicable.

Primary Tables Involved

Table
ITEM_SUPP_MANUFACTURE

Message Type: ItemDimCre – Creates Item Dimension

Primary APIs

- Payload: ISCDimDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: IscDimCreateConsumer.consume (DataExchangeObject)
- SIM Data Exchange Object: ISCDimDEO

Notes

Not applicable.

Primary Tables Involved

Table
ITEM_SUPP_COUNTRY_DIM

Message Type: ItemDimDel – Removes Item Dimension

Primary APIs

- Payload: ISCDimRef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: IscDimRemoveConsumer.consume (DataExchangeObject)
- SIM Data Exchange Object: ISCDimRefDEO

Notes

Not applicable.

Primary Tables Involved

Table
ITEM_SUPP_COUNTRY_DIM

Message Type: ItemDimMod – Updates Item Dimension

Primary APIs

- Payload: ISCDimDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: IscDimModifyConsumer.consume (DataExchangeObject)
- SIM Data Exchange Object: ISCDimDEO

Notes

Not applicable.

Primary Tables Involved

Table
ITEM_SUPP_COUNTRY_DIM

Message Type: ItemSupCtyMfrMod – Updates item supplier country of mfg

Primary APIs

- Payload: ItemSupCtyMfrDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ItemSupCtyMfrModifyConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: ItemSupCtyMfDEO

Notes

Not applicable.

Message Type: ItemSupCtyMfrDel – Deletes item supplier country of mfg

Primary APIs

- Payload: ItemSupCtyMfrRef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ItemSupCtyMfrRemoveConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: ItemSupCtyMfrDEO

Notes

Not applicable.

Message Type: ItemSupCtyMfrCre – Creates item supplier country of mfg

Primary APIs

- Payload: ItemSupCtyMfrDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ItemSupCtyMfrCreateConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: ItemSupCtyMfrDEO

Notes

Not applicable.

Primary Tables Involved

Table
ITEM_SUPP_MANUFACTURE

Message Type: ItemTcktCre – Creates item ticket

Primary APIs

- Payload: ItemTcktDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ItemTcktCreateConsumer.consume (DataExchangeObject)
- SIM Data Exchange Object: ItemTcktDEO

Notes

Updates the suggested ticket type code in the AS_ITM table.

Primary Tables Involved

Table
AS_ITM

Message Type: ItemTcktDel – Deletes item ticket

Primary APIs

- Payload: ItemTcktRef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ItemTcktRemoveConsumer.consume (DataExchangeObject)
- SIM Data Exchange Object: ItemTcktRefDEO

Notes

Updates the suggested ticket type code in the AS_ITM table.

Primary Tables Involved

Table
AS_ITM

Message Type: ItemImageCre – Creates item Image

Primary APIs

- Payload: ItemImageDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ItemImageCreateConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: ItemImageDEO

Notes

Not applicable.

Primary Tables Involved

Table
ITEM_IMAGE

Message Type: ItemImageMod – Updates item Image

Primary APIs

- Payload: ItemImageDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ItemImageModifyConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: ItemImageDEO

Notes

Not applicable.

Primary Tables Involved

Table
ITEM_IMAGE

Message Type: ItemImageDel – Deletes item Image

Primary APIs

- Payload: ItemImageRef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ItemImageRemoveConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: ItemImageDEO

Notes

Not applicable.

Primary Tables Involved

Table
ITEM_IMAGE

Message Type: ItemUDADateCre – Creates item UDA Date

Primary APIs

- Payload: ItemUDADateDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ItemUDACreateConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: ItemUDADEO

Notes

Not applicable.

Primary Tables Involved

Table
Item_UDA

Message Type: ItemUDADateDel – Deletes item UDA Date

Primary APIs

- Payload: ItemUDADateRef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ItemUDARemoveConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: ItemUDARefDEO

Notes

Not applicable.

Primary Tables Involved

Table
Item_UDA

Message Type: ItemUDADateMod – Updates item UDA Date

Primary APIs

- Payload: ItemUDADateDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ItemUDACreateConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: ItemUDADEO

Notes

Not applicable.

Primary Tables Involved

Table
Item_UDA

Message Type: ItemUDAFFDel – Deletes item UDA with display type of FF (Free-Form)

Primary APIs

- Payload: ItemUDAFFRef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ItemUDACreateConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: ItemUDADEO

Notes

Not applicable.

Primary Tables Involved

Table
Item_UDA

Message Type: ItemUDAFFMod –Updates item UDA with display type of FF (Free-Form)

Primary APIs

- Payload: ItemUDAFFDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ItemUDAModifyConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: ItemUDADEO

Notes

Not applicable.

Primary Tables Involved

Table
Item_UDA

Message Type: ItemUDALOVCre – Insert item UDA LOV (List Of Value)

Primary APIs

- Payload: ItemUDALOVDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ItemUDACreateConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: ItemUDADEO

Notes

Not applicable.

Primary Tables Involved

Table
Item_UDA

Message Type: ItemUDALOVDel – Deletes item UDA LOV (List Of Value)

Primary APIs

- Payload: ItemUDALOVRef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ItemUDARemoveConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: ItemUDARefDEO

Notes

Not applicable.

Primary Tables Involved

Table
Item_UDA

Message Type: ItemUDALOVMod – Updates item UDA LOV (List Of Value)

Primary APIs

- Payload: ItemUDALOVDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: ItemUDAModifyConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: ItemUDADEO

Notes

Not applicable.

Primary Tables Involved

Table
Item_UDA

Subscription API Message Family: UDA (User Defined Attributes)

Business Overview

These messages contain the User Defined Attributes information.

Integration to Products

RMS

Message Type: UDAHdrCre – Creates UDA

Primary APIs

- Payload: UDADesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: UDACreateConsumer.consume (DataExchangeObject)
- SIM Data Exchange Object: UDADEO

Notes

Not applicable.

Primary Tables Involved

Table
UDA

Message Type: UDAHdrMod –Updates UDA

Primary APIs

- Payload: UDADesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: UDAModifyConsumer.consume (DataExchangeObject)
- SIM Data Exchange Object: UDADEO

Notes

Not applicable.

Primary Tables Involved

Table
UDA

Message Type: UDAHdrDel – Deletes UDA

Primary APIs

- Payload: UDARef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: UDARemoveConsumer.consume (DataExchangeObject)
- SIM Data Exchange Object: UDARefDEO

Notes

Not applicable.

Primary Tables Involved

Table
UDA

Message Type: UDAVALCre – Creates UDA Value

Primary APIs

- Payload: UDAValDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: UDAValueCreateConsumer.consume (DataExchangeObject)
- SIM Data Exchange Object: UDAValueDEO

Notes

Not applicable.

Primary Tables Involved

Table
UDA_VALUE

Message Type: UDAValMod –Updates UDA Value

Primary APIs

- Payload: UDAValDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: UDAValueModifyConsumer.consume (DataExchangeObject)
- SIM Data Exchange Object: UDAValueDEO

Notes

Not applicable.

Primary Tables Involved

Table
UDA_VALUE

Message Type: UDAValDel – Deletes UDA Value

Primary APIs

- Payload: UDAValRef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: UDAValueRemoveConsumer.consume (DataExchangeObject)
- SIM Data Exchange Object: UDAValueRefDEO

Notes

Not applicable.

Primary Tables Involved

Table
UDA_VALUE

Subscription API Message Family: Partner

Business Overview

These messages contain the Partner Information.

Integration to Products

RMS

Message Type: PartnerCre – Creates Partner

Primary APIs

- Payload: PartnerDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: PartnerCreateConsumer.consume (DataExchangeObject)
- SIM Data Exchange Object: PartnerDEO

Notes

Not applicable.

Primary Tables Involved

Table
RK_PARTNER_ORG_UNIT
PARTNER

Message Type: PartnerMod– Updates Partner

Primary APIs

- Payload: PartnerDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: PartnerModifyConsumer.consume (DataExchangeObject)
- SIM Data Exchange Object: PartnerDEO

Notes

Not applicable.

Primary Tables Involved

Table
RK_PARTNER_ORG_UNIT
PARTNER

Message Type: PartnerDel– Deletes Partner

Primary APIs

- Payload: PartnerRef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: PartnerREmoveConsumer.consume (DataExchangeObject)
- SIM Data Exchange Object: PartnerRefDEO

Notes

Not applicable.

Primary Tables Involved

Table
RK_PARTNER_ORG_UNIT
PARTNER

Message Type: PartnerDtlCre– Creates Partner Detail

Primary APIs

- Payload: PartnerDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: PartnerAddressCreateConsumer.consume (DataExchangeObject)
- SIM Data Exchange Object: PartnerDEO

Notes

Not applicable.

Primary Tables Involved

Table
ADDRESS_DETAIL

Message Type: PartnerDtlDel– Deletes Partner Detail

Primary APIs

- Payload: PartnerRef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: PartnerAddressRemoveConsumer.consume (DataExchangeObject)
- SIM Data Exchange Object: PartnerRefDEO

Notes

Not applicable.

Primary Tables Involved

Table
ADDRESS_DETAIL

Message Type: PartnerDtlMod– Updates Partner Detail

Primary APIs

- Payload: PartnerDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: PartnerAddressModifyConsumer.consume (DataExchangeObject)
- SIM Data Exchange Object:PartnerDEO

Notes

Not applicable.

Primary Tables Involved

Table
ADDRESS_DETAIL

Subscription API Message Family: Order

Business Overview

These messages contain approved, direct to store purchase orders. Direct Deliveries are received against the POs created in RMS.

Integration to Products

RMS

Message Type: POCre – Creates Purchase Order

Primary APIs

- Payload: PODesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: PurchaseOrderCreateConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: PODEO

Notes

Not applicable.

Message Type: PODel – Deletes Purchase Order

Primary APIs

- Payload: PORef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: PurchaseOrderRemoveConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: PORefDEO

Notes

Not applicable.

Message Type: PODtlCre – Creates Purchase Order Details

Primary APIs

- Payload: PODesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: PurchaseOrderDetailCreateConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: PODEO

Notes

Not applicable.

Message Type: PODtlDel – Deletes Purchase Order Details

Primary APIs

- Payload: PORef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: PurchaseOrderDetailRemoveConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: PORefDEO

Notes

Not applicable.

Message Type: PODtlMod – Updates Purchase Order Details

Primary APIs

- Payload: PODesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: PurchaseOrderDetailModifyConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: PODEO

Notes

Not applicable.

Message Type: POHdrMod – Updates Purchase Order Header

Primary APIs

- Payload: PODesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: PurchaseOrderModifyConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: PODEO

Notes

Not applicable.

Primary Tables Involved

Table
RK_ORDERS
RK_ORDER_ITM

Subscription API Message Family: PromotionPriceChange (PrmPrcChg)

Business Overview

This message is used by a price management system to communicate the modification of a new retail on an already approved promotion. This message is at a transaction item/location level. It is used by SIM for visibility to the modified promotional retail on the effective date for the promotion.

Integration to Products

A price management system

Message Type: MultiBuyPromoCre

Primary APIs

- Payload: PrmPrcChgDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: PrmPrcChgCreateConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: PromotionDEO

Notes

Not applicable.

Message Type: MultiBuyPromoMod

Primary APIs

- Payload: PrmPrcChgDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: PrmPrcChgModifyConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: PromotionDEO

Notes

Not applicable.

Message Type: MultiBuyPromoDel

Primary APIs

- Payload: PrmPrcChgRef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: PrmPrcChgRemoveConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: PromotionRefDEO

Notes

Not applicable.

Subscription API Message Family: RegularPriceChange (RegPrcChg)

Business Overview

This message is used by a price management system to communicate the approval of a price change within the application. This message is published at a transaction item/location level.

Integration to Products

RMS, a price management system

Message Type: RegPrcChgCre

Primary APIs

- Payload: RegPrcChgDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: RegPrcChgCreateConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: RegPrcChgDEO

Notes

Not applicable.

Message Type: RegPrcChgMod

Primary APIs

- Payload: RegPrcChgDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: RegPrcChgModifyConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: RegPrcChgDEO

Notes

Not applicable.

Message Type: RegPrcChgDel

Primary APIs

- Payload: RegPrcChgRef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: RegPrcChgRemoveConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: RegPrcChgRefDEO

Notes

Not applicable.

Primary Tables Involved

Table
AS_ITM_RTL_STR
LE_HST_ITM_SLS_PRC

Subscription API Message Family: Receiver Unit Adjustment (RcvUnitAdjMod)

Business Overview

These messages are used by SIM to communicate any adjustments to the receipts of purchase orders, transfers, and allocations. These messages are part of the RECEIVING message family (receiving unit adjustments only use the RECEIPTMOD message type).

Integration to Products

RMS

Message Type: RcvUnitAdjDtl

Primary APIs

- Payload: RcvUnitAdjDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: RcvUnitAdjModConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: RcvUnitAdjDEO

Notes

Not applicable.

Primary Tables Involved

Table
RK_ORDERS
RK_SHIPMENT_ITEM
RK_SHIPMENT_CARTON

Subscription API Message Family: RTV Request (RtvReq)

Business Overview

These are messages communicated by RMS that contain a request to return inventory to a vendor.

Integration to Products

RMS

Message Type: RtvReqCre

Primary APIs

- Payload: RTVReqDesc
- Injector: SimMessageRibInjector.inject (messageType, Payload)
- Consumer: RTVReqCreateConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: RTVReqDEO

Notes

Not applicable.

Message Type: RtvReqMod

Primary APIs

- Payload: RTVReqDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: RTVReqModifyConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: RTVReqDEO

Notes

Not applicable.

Message Type: RtvReqDel

Primary APIs

- Payload: RTVReqRef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: RTVReqRemoveConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: RTVReqRefDEO

Notes

Not applicable.

Message Type: RtvReqDtlCre

Primary APIs

- Payload: RTVReqDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: RTVReqDetailCreateConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: RTVReqDEO

Notes

Not applicable.

Message Type: RtvReqDtlDel

Primary APIs

- Payload: RTVReqRef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: RTVReqDetailRemoveConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: RTVReqRefDEO

Notes

Not applicable.

Message Type: RtvReqDtlMod

Primary APIs

- Payload: RTVReqDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: RTVReqDetailModifyConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: RTVReqDEO

Notes

Not applicable.

Primary Tables Involved

Table
RETURN
RETURN_LINE_ITEM

Subscription API Message Family: Seed Data (SeedData)

Business Overview

These messages communicated by RMS contain differentiator type values. The creation, modification and deletion of the various diff types in RMS flows to SIM through the seed data message.

Integration to Products

RMS

Message Type: DiffTypeCre

Primary APIs

- Payload: DiffTypeDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: DifferentiatorTypeCreateConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: DiffTypeDEO

Notes

Not applicable.

Message Type: DiffTypeDel

Primary APIs

- Payload: DiffTypeRef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: DifferentiatorTypeRemoveConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: DiffTypeRefDEO

Notes

Not applicable.

Message Type: DiffTypeMod

Primary APIs

- Payload: DiffTypeDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: DifferentiatorTypeModifyConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: DiffTypeDEO

Notes

Not applicable.

Primary Tables Involved

Table
RK_DIFF_TYPE

Subscription API Message Family: Stock Order Status (SOStatus)

Business Overview

These messages are used by a warehouse management system to communicate the creation or modification of a warehouse delivery in a warehouse management system.

Integration to Products

A warehouse management system.

Message Type: SOStatusCre

Primary APIs

- Payload: SOStatusDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: StockOrderStatusConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: SOStatusDEO

Notes

Not applicable.

Subscription API Message Family: StockOrder

Business Overview

These messages are used by a warehouse management system to communicate the creation or modification of a warehouse delivery in a warehouse management system.

Integration to Products

A warehouse management system.

Message Type: SOCre – Creates Stock order

Primary APIs

- Payload: SODesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: StockOrderCreateConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: SODEO

Notes

Not applicable.

Message Type: SODtlCre – Creates stock order detail

Primary APIs

- Payload: SODesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: StockOrderCreateConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: SODEO

Notes

Not applicable.

Message Type: SODtlDel – Deletes stock order detail

Primary APIs

- Payload: SORef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: StockOrderRemoveConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: SODEO

Notes

Not applicable.

Message Type: SODtlMod – Updates Stock order details

Primary APIs

- Payload: SODesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: StockOrderModifyConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: SODEO

Notes

Not applicable.

Message Type: SOHdrDel – Deletes Stock order header

Primary APIs

- Payload: SORef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: StockOrderRemoveConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: SODEO

Notes

Not applicable.

Message Type: SOHdrMod – Updates stock order header

Primary APIs

- Payload: SODesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: StockOrderModifyConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: SODEO

Notes

Not applicable.

Primary Tables Involved

Table
RK_ALLOCATIONS
TRANSFER
TRANSFER_LINE_ITEM
RETURN
RETURN_LINE_ITEM

Subscription API Message Family: Stores

Business Overview

These are messages communicated by RMS that contain stores set up in the system (RMS).

Integration to Products

RMS

Message Type: StoreCre – Creates Store

Primary APIs

- Payload: StoresDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: StoreCreateConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: StoreDEO

Notes

Not applicable.

Message Type: StoreDel – Deletes Store

Primary APIs

- Payload: StoresRef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: StoreRemoveConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: StoreDEO

Notes

Not applicable.

Message Type: StoreMod – Updates Store

Primary APIs

- Payload: StoresDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: StoreModifyConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: StoreDEO

Notes

Not applicable.

Primary Tables Involved

Table
PA_STR_RTL
PA_PRTY
RK_TRANSFER_ZONES
PA_PRTY
PA_RO_PRTY
PA_ORGN
ADDRESS_DETAIL

Subscription API Message Family: Vendor

Business Overview

These are messages communicated by RMS containing vendors set up in the system (RMS or external financial system).

Integration to Products

RMS, external (financial)

Message Type: VendorAddrCre

Primary APIs

- Payload: VendorAddrDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: SupplierAddrCreateConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: VendorDEO

Notes

Not applicable.

Message Type: VendorAddrDel

Primary APIs

- Payload: VendorAddrRef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: SupplierAddrRemoveConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: VendorDEO

Notes

Not applicable.

Message Type: VendorAddrMod

Primary APIs

- Payload: VendorAddrDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: SupplierAddrModifyConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: VendorDEO

Notes

Not applicable.

Message Type: VendorCre

Primary APIs

- Payload: VendorDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: SupplierCreateConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: VendorDEO

Notes

Not applicable.

Message Type: VendorDel

Primary APIs

- Payload: VendorRef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: SupplierRemoveConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: VendorDEO

Notes

Not applicable.

Message Type: VendorHdrMod

Primary APIs

- Payload: VendorHdrDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: SupplierModifyConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: VendorHdrDEO

Notes

Not applicable.

Message Type: VendorOUCre

Primary APIs

- Payload: VendorDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: SupplierCreateConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: VendorDEO

Notes

Not applicable.

Message Type: VendorOUDel

Primary APIs

- Payload: VendorDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: SupplierRemoveConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: VendorDEO

Notes

Not applicable.

Primary Tables Involved

Table
ADDRESS_DETAIL
PA_PRTY
PA_RO_PRTY
PA_ORGN
RK_PARTNER_ORG_UNIT
PA_SPR

Subscription API Message Family: Merchandise Hierarchy

Business Overview

These messages are communicated by RMS. These messages include department/class/subclass information.

Integration to Products

RMS

Message Type: DeptCre

Primary APIs

- Payload: MrchHrDeptDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: MrchDeptCreateConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: MrchHrDeptDEO

Notes

Not applicable.

Message Type: DeptMod

Primary APIs

- Payload: MrchHrDeptDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: MrchDeptModifyConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: MrchHrDeptDEO

Notes

Updates department name in RK_HIERARCHY table

Message Type: DeptDel

Primary APIs

- Payload: MrchHrDeptRef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: MrchDeptRemoveConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: MrchHrDeptRefDEO

Notes

Not applicable.

Message Type: ClassCre

Primary APIs

- Payload: MrchHrClsDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: MrchClassCreateConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: MrchHrClsDEO

Notes

Not applicable.

Message Type: ClassMod

Primary APIs

- Payload: MrchHrClsDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: MrchClassModifyConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: MrchHrClsDEO

Notes

Not applicable.

Message Type: ClassDel

Primary APIs

- Payload: MrchHrClsRef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: MrchClassRemoveConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: MrchHrClsRefDEO

Notes

Not applicable.

Message Type: SubClassCre

Primary APIs

- Payload: MrchHrSclsDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: MrchSubclassCreateConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: MrchHrSclsDEO

Notes

Not applicable.

Message Type: SubClassMod

Primary APIs

- Payload: MrchHrSclsDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: MrchSubclassModifyConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: MrchHrSclsDEO

Notes

Not applicable.

Message Type: SubClassDel

Primary APIs

- Payload: MrchHrSclsRef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: MrchSubclassRemoveConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: MrchHrSclsRefDEO

Notes

Not applicable.

Primary Tables Involved

Table
RK_HIERARCHY

Subscription API Message Family: Delivery Slot (DeliverySlot)

Business Overview

This message is communicated by RMS and consists of the delivery slot information, which is needed by transfers and other shipment transactions.

Integration to Products

RMS

Message Type: DlvySltCre

Primary APIs

- Payload: DeliverySlotDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: DeliverySlotCreateConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: DeliverySlotDEO

Notes

Not applicable.

Message Type: DlvySltMod

Primary APIs

- Payload: DeliverySlotDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: DeliverySlotModifyConsumer
- SIM Data Exchange Object: DeliverySlotDEO

Notes

Not applicable.

Message Type: DlvySltDel

Primary APIs

- Payload: DeliverySloRef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: DeliverySlotRemoveConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: DeliverySlotDEO

Notes

Not applicable.

Primary Tables Involved

Table
DELIVERY_SLOT

Subscription API Message Family: Warehouse (WH)

Business Overview

These are messages that are communicated by RMS that contain warehouses set up in the system (RMS). SIM only gets physical warehouse records.

Integration to Products

RMS

Message Type: WHCre

Primary APIs

- Payload: WHDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: WareHouseCreateConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: WHDEO

Notes

Not applicable.

Message Type: WHDel

Primary APIs

- Payload: WHRef
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: WareHouseRemoveConsumer.consume(DataExchangeObject)
- SIM Data Exchange Object: WHDEO

Notes

Not applicable.

Message Type: WHMod

Primary APIs

- Payload: WHDesc
- Injector: SimMessageRibInjector.inject (messageType, businessId, Payload)
- Consumer: WareHouseModifyConsumer
- SIM Data Exchange Object: WHDEO

Notes

Not applicable.

Primary Tables Involved

Table
PA_STR_RTL
PA_PRTY
PA_RO_PRTY
PA_ORGN
ADDRESS_DETAIL

Publishers

Oracle Retail Stores Inventory Management publishes messages (payload) to RIB JMS through the RIB-SIM component, and RIB Binding subsystem converts the payload object into an XML string. The object on the Binding subsystem is put into a RIB envelope called RibMessage. The data within RibMessage eventually becomes a message on the RIB. A Publisher class in the Binding subsystem is called to write the data to the RIB's JMS queue. On a regular basis, the RIB engages in polling the JMS queue, searching for the existence of a message. A publishable message that appears on the JMS queue is processed.

For additional information, see the *Oracle Retail Integration Bus Operations Guide* and other RIB documentation.

Publication API Message Family: ASNOut

Business Overview

These messages are used by SIM to communicate store-to-warehouse transfers (returns to warehouse) to both RMS and a warehouse management system. These messages are also used to communicate store-to-store transfers to RMS.

Integration to Products

RMS, a warehouse management system

Message Type: ASNOutCre

Primary APIs

- Payload: ASNOutDesc
- Publisher: SimRib13Publisher.publish(RibMessageVO)

> **Note:** Payload is encapsulated in RIBMessageVO wrapper to be published to external system.

- SIM Data Exchange Object: ASNOutDEO
- Stager:
  - ASNOutTransferStager.stage()
  - ASNOutReturnStager.stage()

Notes

Not applicable.

Primary Tables Involved

Table
STAGED_MESSAGE

Publication API Message Family: DSDReceipt

Business Overview

These messages are used by SIM to communicate the receipt of a supplier delivery for which no RMS purchase order had previously existed.

Integration to Products

RMS

Message Type: DSDReceiptCre

Primary APIs

- Payload: DSDReceiptDesc
- Publisher: SimRib13Publisher.publish(RibMessageVO)
- SIM Data Exchange Object: DSDReceiptDEO
- Stager: DSDReceiptStager.stage()

Message Type: DSDReceiptMod

Primary APIs

- Payload: DSDReceiptDesc
- Publisher: SimRib13Publisher.publish(RibMessageVO)
- SIM Data Exchange Object: DSDReceiptDEO
- Stager: DSDReceiptStager.stage()

Notes

Not applicable.

Primary Tables Involved

Table
STAGED_MESSAGE

Publication API Message Family: InvAdjust

Business Overview

These messages are used by SIM to communicate inventory adjustments. RMS uses these messages to adjust inventory accordingly.

Integration to Products

RMS

Message Type: InvAdjustCre

Primary APIs

- Payload: InvAdjustDesc
- Publisher: SimRib13Publisher.publish(RibMessageVO)
- SIM Data Exchange Object: InventoryAdjustmentDEO
- Stager:
  - InventoryAdjustmentStager
  - InventoryAdjustmentVOStager

Notes

Not applicable.

Primary Tables Involved

Table
STAGED_MESSAGE

Publication API Message Family: InvReq

Business Overview

These messages are used by SIM to communicate the request for inventory of a particular item. RMS uses this data to fulfill the requested inventory through either auto-replenishment or by creating a one-off purchase order/transfer.

Integration to Products

RMS

Message Type: InvReqCre

Primary APIs

- Payload: InvReqDesc
- Publisher: SimRib13Publisher.publish(RibMessageVO)
- SIM Data Exchange Object: ItemRequestDEO
- Stager: ItemRequestStager

Notes

Not applicable.

Primary Tables Involved

Table
STAGED_MESSAGE

Publication API Message Family: Receiving

Business Overview

These messages are used by SIM to communicate the receipt of an RMS purchase order, a transfer, or an allocation.

Integration to Products

RMS

Message Type: ReceiptCre

Primary APIs

- Payload: ReceiptDesc
- Publisher: SimRib13Publisher.publish(RibMessageVO)
- SIM Data Exchange Object: ReceiptDEO
- Stager:
  - DirectDeliveryReceiptStager
  - TransferInReceiptStager
  - WarehouseDeliveryReceiptStager

Notes

Not applicable.

Primary Tables Involved

Table
STAGED_MESSAGE

Publication API Message Family: SOStatus

Business Overview

These messages are used by SIM to communicate the cancellation of any requested transfer quantities. For example, the merchandising system can create a transfer request for 90 units from a store. If the sending store only ships 75, a cancellation message is sent for the remaining 15 requested items.

Integration to Products

RMS

Message Type: SOStatusCre

Primary APIs

- Payload: SOStatusDesc
- Publisher: SimRib13Publisher.publish(RibMessageVO)
- SIM Data Exchange Object: SOStatusDEO
- Stager: SOStatusTransferStager

Notes

Not applicable.

Primary Tables Involved

Table
STAGED_MESSAGE

Publication API Message Family: StkCountSch

Business Overview

These messages are used by SIM to communicate unit and value stock count schedules to RMS. RMS uses this schedule to take an inventory snapshot of the date of a scheduled count.

Integration to Products

RMS

Message Type: StkCountSchCre

Primary APIs

- Payload: StkCountSchDesc
- Publisher: SimRib13Publisher.publish(RibMessageVO)
- SIM Data Exchange Object: StockCountScheduleDEO
- Stager: StockCountScheduleCreateStager

Notes

Not applicable.

Message Type: StkCountSchDel

Primary APIs

- Payload: StkCountSchRef
- Publisher: SimRib13Publisher.publish(RibMessageVO)
- SIM Data Exchange Object: StockCountScheduleDEO
- Stager: StockCountScheduleModifyStager

Notes

Not applicable.

Message Type: StkCountSchMod

Primary APIs

- Payload: StkCountSchDesc
- Publisher: SimRib13Publisher.publish(RibMessageVO)
- SIM Data Exchange Object: StockCountScheduleDEO
- Stager: StockCountScheduleDeleteStager

Notes

Not applicable.

Primary Tables Involved

Table
STAGED_MESSAGE

C

Appendix: External Inventory Adjustments

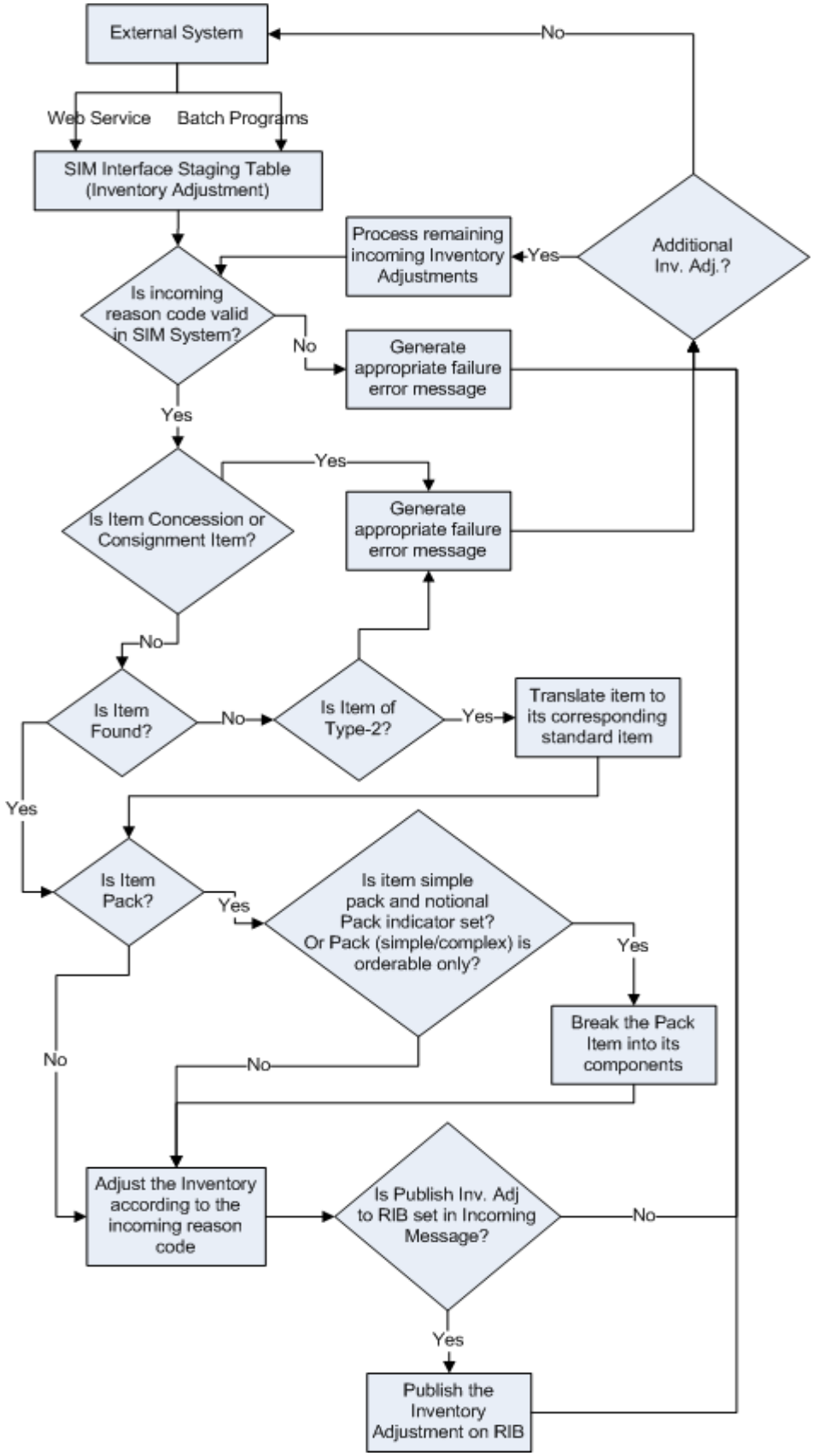
This feature enables processing of the Inventory Adjustments sent by an external system through Web Services into the staging interface tables. The SIM system is responsible for the following operations:

- Set up appropriate reason codes in the SIM system.
- Pick up the inventory adjustments from the staging interface tables for processing.
- Determine whether the Item is valid for inventory adjustment. If the item is a concession or consignment item, or the item does not exist for the store, or the reason code is invalid, the message fails with an appropriate error message.
- If the item is a simple pack item and the notional pack indicator is set or it is an orderable-only simple pack, the pack is split into its components and the components are updated.
- SIM to process Type-2 items by translating to a standard item and updating the SOH by the external inventory adjustment quantity and not the embedded quantity of the Type-2 item.
- An inventory adjustment message is published on RIB based on the parameter sent by the external system.

The following information is contained in the incoming inventory adjustment message:

- Item – The external system passes in the Item ID for which the inventory is to be adjusted.
- Store – The Store for which the adjustment is to be performed
- Quantity – The quantity by which the inventory is to be adjusted.
- Reason code ID – This determines what direction and bucket the stock is moving. The inventory is either incremented or decremented based on the disposition.
- RIB Processing – This is an indicator, which specifies if the processing of the external inventory adjustment should publish an inventory adjustment on the RIB.
- External Entity – This parameter helps identify the external system. The entity can be a user ID or a system identifier.
- UIN – one or more UINs that should be equal to the quantity if the item is a UIN item that is tracked in the store.

Figure C–1 External Inventory Adjustment Process Flow



Item

For Concession and Consignment Item and non-inventory items, the inventory is not tracked. If the incoming item is one of these, generate a failure with an appropriate error message and continue processing the remaining external inventory adjustments if any.

If the item ID is invalid for the store, generate a failure with an appropriate error message and continue processing the remaining external inventory adjustments, if any.

If the item is an orderable only simple pack item or a simple pack item with the notional pack indicator set, break down the pack into its components and adjust the inventory at the component level.

If item is an orderable complex pack, the pack will be broken down to its components.

In case the item is a Type –2 Item, translate it to a standard item and adjust the inventory of the standard item by the quantity from the external message and not by the embedded quantity of the Type-2 Item.

Inventory Adjustment Reason Codes

The SIM system applies reason codes with appropriate disposition to handle the external inventory management. The dispositions to be supported are:

- **+ Stock On Hand**: Indicates an increase in stock on hand. (DIST (Out) –> ATS (Available to Sell))
- **- Stock On Hand**: Indicates decrease in stock on hand. (ATS –> DIST)
- **+ Unavailable**: Indicates an increase in unavailable quantity. (ATS –> TRBL (Unavailable))
- **- Unavailable**: Indicates decrease in unavailable quantity. (TRBL –> ATS)
- **- Stock on Hand and - Unavailable**: Indicates decrease in stock on hand and the unavailable quantity. (TRBL –> DIST)
- **- Stock on Hand and - Unavailable**: Indicates decrease in stock on hand and the unavailable quantity. (TRBL –> DIST)
- **+ Customer Order Reserve**: Indicates increase customer reserved inventory (ATS –> COR)
- **- Customer Order Reserve**: Indicates decrease customer reserved inventory (COR –> ATS)

UIN Processing

This section only applies to Store receiving capture time UINs. Any other capture time type ignores the UIN field and acts as a regular item.

- All dispositions with exception of customer order (COR and + Stock on Hand):

  SIM will require UINs to be attached to the adjustment equal to the quantity for the item, if the UIN is an AGSN or Serial number item.

  After this validation check, validation needs to be performed that the item is in the correct status to move to the to disposition.

- Customer Order Reserved dispositions (to and from COR):

  At this time, these are not allowed from external systems. Those systems need to use the customer order Web services.

- In Stock movement (DIST –> ATS):
  - Serial UIN type items:

    SIM will require UINs to be attached to the adjustment equal to the quantity for the item.

    Validate that the item is in the correct status.

  - AGSN item:

    Validate that no UINs are attached.

    Generate new AGSNs for the quantity of the inventory adjustment.

  - Special rules when moving UINs from one store to another if the UIN is In Stock at another store and disposition is from DIST to ATS:

    Normal process is to prompt the user, since this is a backend system, the move from the other store is allowed if the Allow Unexpected UINs system admin parameter is set to **Enabled**.

    If this indicator is not set, the system will automatically fail the item.

UIN Status Management

Inventory adjustments are only allowed to move the status of a UIN to the following states. All other disposition movements are not allowed.

- **In Stock**–This means that the item is in stock.
- **Removed From Inventory**–A UIN is removed from inventory.
- **Unavailable**–A UIN is made unavailable.

The following dispositions allow UIN states to move from one state to another regardless of how the Allow Unexpected UINs system admin parameter is set.

Note: The assumption is that for inventory in stock states (in stock, unavailable), the store associated to the UIN is the store the inventory adjustment is created for. Those states (missing, sold, and so forth) that indicate a UIN has left SIM, can move inventory into any store regardless of where that item originally went missing, sold, and so forth.

Table C–1 UIN Status Dispositions

From UIN status	To UIN Status	From Disposition	To Disposition
In Stock	Unavailable	ATS	TRBL
Unavailable	In Stock	TRBL	ATS
Unavailable	Removed from Inventory	TRBL	DIST
In Stock	Removed from Inventory	ATS	DIST
Missing	In Stock	DIST	ATS

Table C–1 (Cont.) UIN Status Dispositions

From UIN status	To UIN Status	From Disposition	To Disposition
Removed from Inventory	In Stock	DIST	ATS
Shipped to vendor	In Stock	DIST	ATS
Shipped to Warehouse	In Stock	DIST	ATS
Customer Order Fulfilled	In Stock	DIST	ATS
Unconfirmed	In Stock	DIST	ATS
<no UIN exists>	In Stock	DIST	ATS
Sold	In Stock	DIST	ATS

Note: Customer Order disposition cannot move UINs.

In addition to Figure C–1, the following move is also allowed from one state to another if the Allow Unexpected UINs system admin parameter is set to **Enabled** and the in-stock UIN is associated to another store.

Table C–2 UIN Status Dispositions

From UIN status	To UIN Status	From Disposition	To Disposition
In Stock (at other store)	In Stock (store creating inventory adjustment)	DIST	ATS

This adjustment triggers an exception process in the other store.

Pending Inventory Adjustments

Making a UIN unavailable will attach that UIN to a pending inventory adjustment.

Moving the UIN out of the unavailable status removes it from the pending inventory adjustment. There are two ways to do this:

- Directly access the pending adjustment.
- Create a new inventory transaction and remove the UIN from unavailable to DIST or ATS.

AGSNs

Inventory adjustments might need to generate a ticket. When an inventory adjustment reason is selected that has a disposition movement of DIST to ATS, SIM auto-generates the UIN for the quantity adjusted minus the scanned UINs (same as stock counts).

- SIM associates the new UINs to the item
- SIM automatically prints a label for each UIN using the Ticket Type setup for the item.

Note: This is on the PC only.

Moving Items From One Store to Another

If a UIN is scanned that exists at a different location, and the UIN status is **In Stock** and the disposition used is **DIST to ATS**, the UIN can be automatically moved over to the current store if the Allow Unexpected UINs system admin parameter is set to **Enabled**:

- The item will appear on Problem Line count for the other store to flag the item as an issue. This logic should be added to the existing Problem Line batch.
- The user should be prompted that the UIN is associated with a different store and if it is ok to move the UIN to this store.
- Upon completing the inventory adjustment, an email notification will be sent to the store the UIN existed at prior to appearing in the current store. The email will include the following information:

```
Subject:

Resolve UIN Discrepancy

Body:

This is a generated message.  Please do not reply.

UIN <01233456> has a status of <status> in your store <x> but is physically
located in Store <y>.  Please reconcile the discrepancy.

Item: 1234 - Cell phone
UIN: 0123456
```

RIB Processing

Based on the value of the RIB Processing indicator for each external inventory adjustment message, publish the Inventory Adjustment on the RIB.

The value for this indicator can be **Y** or **N**. Publish the inventory adjustment on RIB only if the value is set to **Y**.

Continue processing the remaining external inventory adjustments, if any.

The SIM system continues processing the remaining adjustments in case one of the adjustments fails. The records are processed in a batch mode.