Oracle® Retail Store Inventory Management

Implementation Guide – Volume 2 – Integration Information

Release 13.1

June 2009

ORACLE®

Oracle Retail Store Inventory Management Implementation Guide – Volume 2 – Integration Information, Release 13.1

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 software component known as **ACUMATE** developed and licensed by Lucent Technologies Inc. of Murray Hill, New Jersey, to Oracle and imbedded in the Oracle Retail Predictive Application Server - Enterprise Engine, Oracle Retail Category Management, Oracle Retail Item Planning, Oracle Retail Merchandise Financial Planning, Oracle Retail Advanced Inventory Planning, Oracle Retail Demand Forecasting, Oracle Retail Regular Price Optimization, Oracle Retail Size Profile Optimization, Oracle Retail Replenishment Optimization applications.

(ii) 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.

(iii) the **SeeBeyond** component developed and licensed by Sun MicroSystems, Inc. (Sun) of Santa Clara, California, to Oracle and imbedded in the Oracle Retail Integration Bus application.

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

(v) the software component known as **Crystal Enterprise Professional and/or Crystal Reports Professional** licensed by SAP and imbedded in Oracle Retail Store Inventory Management.

(vi) 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.

(vii) 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.

(viii) the software component known as **Style Report™** developed and licensed by InetSoft Technology Corp. of Piscataway, New Jersey, to Oracle and imbedded in the Oracle Retail Value Chain Collaboration application.

(ix) the software component known as **DataBeacon™** developed and licensed by Cognos Incorporated of Ottawa, Ontario, Canada, to Oracle and imbedded in the Oracle Retail Value Chain Collaboration 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

List of Figures

List of Tables

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 a Point of Sale System 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 (such as RPM)
  - From a Price Management System (such as RPM) 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
  - Integration with Oracle Retail Workspace

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.

Related Documents

For more information, see the following documents in the Oracle Retail Store Inventory Management Release 13.1 documentation set:

- *Oracle Retail Store Inventory Management Release Notes*
- *Oracle Retail Store Inventory Management Installation Guide*
- *Oracle Retail Store Inventory Management User Guide*
- *Oracle Retail Store Inventory Management Online Help*
- *Oracle Retail Store Inventory Management Operations Guide*
- *Oracle Retail Store Inventory Management Data Model*
- *Oracle Retail Store Inventory Management Implementation Guide – Volume 1*
- *Oracle Retail Store Inventory Management Implementation Guide – Volume 3 – Mobile Store Inventory Management*
- *Oracle Retail Store Inventory Licensing Information*
- Oracle Retail Service Layer documentation
- Oracle Retail Integration Bus documentation

Customer Support

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

- https://metalink.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 recreate
- Exact error message received
- Screen shots of each step you take

Review Patch Documentation

If you are installing the application for the first time, you install either a base release (for example, 13.0) or a later patch release (for example, 13.0.2). If you are installing a software version other than the base release, be sure to read the documentation for each patch release (since the base release) before you begin installation. Patch documentation can contain critical information related to the base release and code changes that have been made since the base release.

Oracle Retail Documentation on the Oracle Technology Network

In addition to being packaged with each product release (on the base or patch level), all Oracle Retail documentation is available on the following Web site (with the exception of the Data Model which is only available with the release packaged code):

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

Documentation should be available on this Web site within a month after a product release. Note that documentation is always available with the packaged code on the release date.

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.

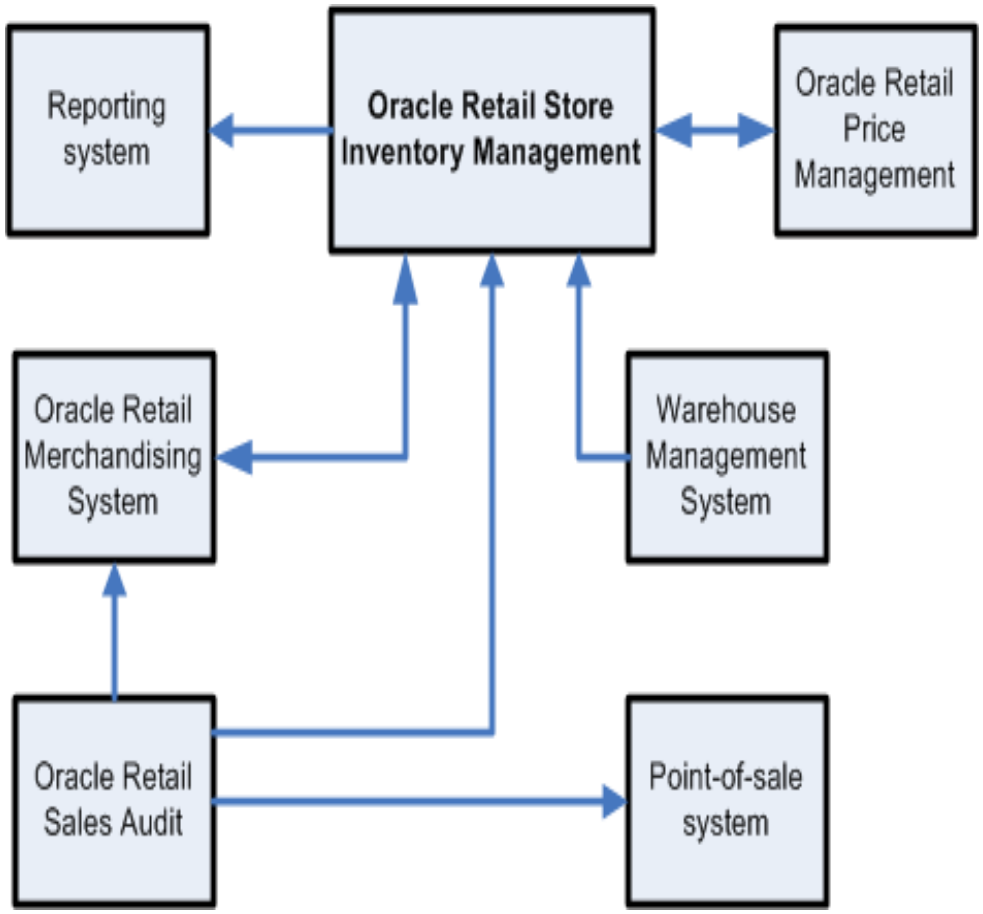
1

SIM Integration Points into the Retail Enterprise

SIM Integration Points

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

Figure 1–1 SIM-Related Dataflow Across the Enterprise



2

SIM Integration – Functional

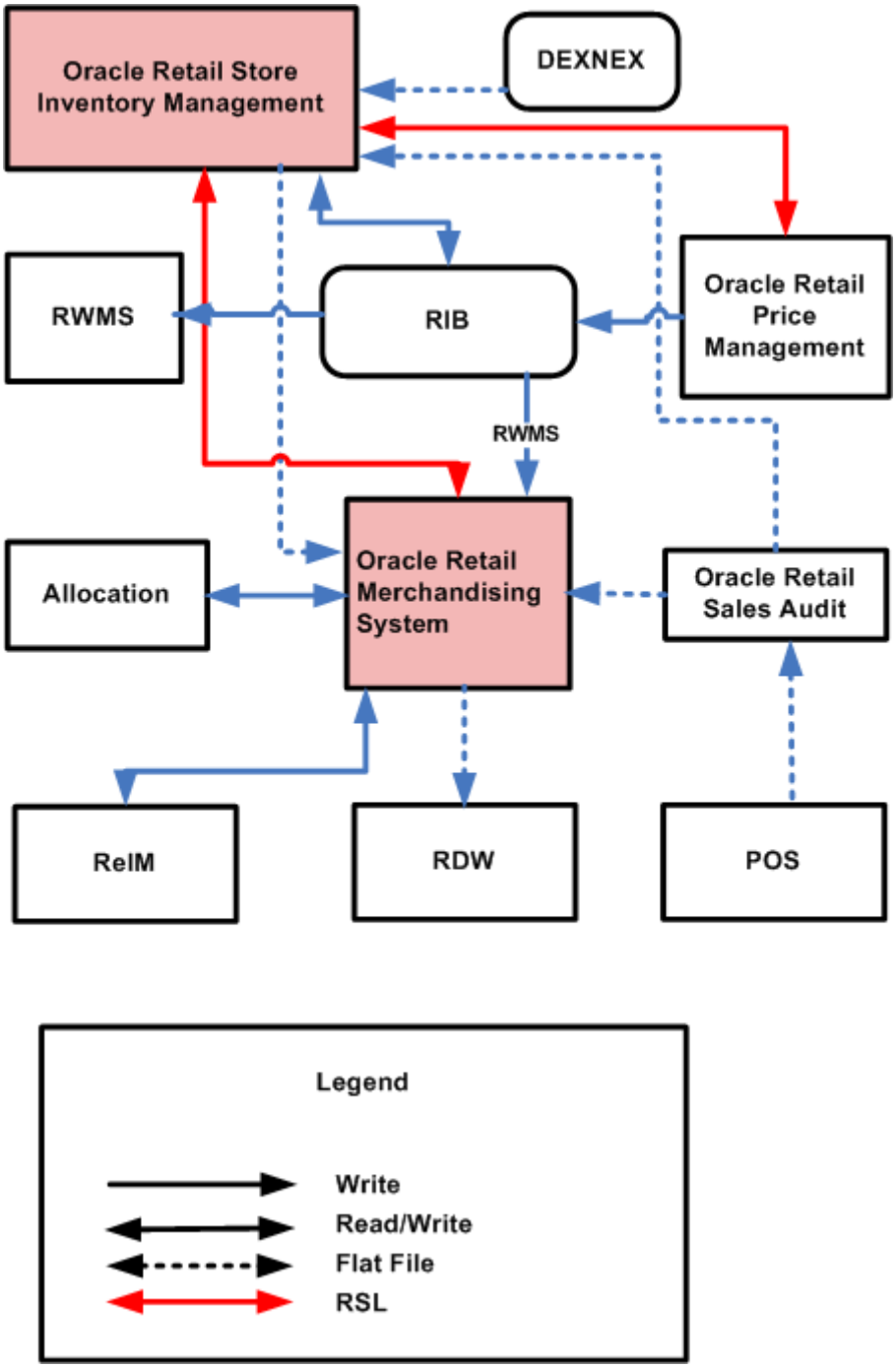
This chapter provides a functional overview of how SIM integrates with other systems (including other Oracle Retail systems).

Overview

The first section in this chapter provides you with a diagram illustrating 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

The following table 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, 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.
Price change	Subscription	RPM	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	RPM	This service, provided by RPM, allows an inquiring system to request the effective retail for an item at a specified location on a given date. RPM 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.

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

Functional area	Subscription/ publication	Integration to Products	Description
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.
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.

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 via 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 a Point of Sale System to SIM

The following data is sent from a point of sale (POS) system 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 a point of sale system. This affects the available and unavailable inventory buckets in SIM.

From the Merchandising System to SIM

The following merchandising system data is published via 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 (such as RPM)

- Request for approval of price change data

  Regular, clearance, and simple fixed price promotion price change data are sent to RPM. RPM 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 (such as RPM) to SIM

- Price change data

  RPM 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:

- Oracle Retail Integration Bus (RIB)-based integration
- Oracle Retail Service Layer (RSL)-based integration
- Web Service-based integration
- File-based integration

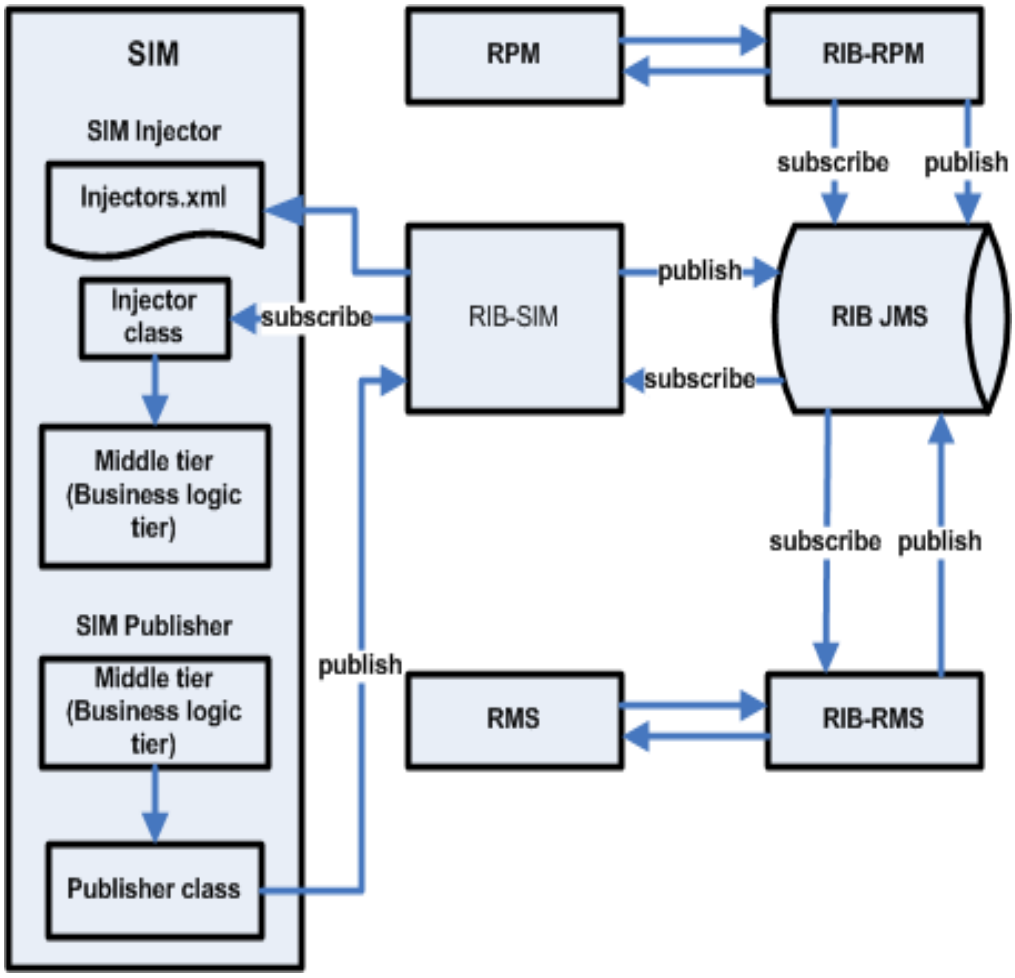
Each section includes information concerning the architecture of the integration method and the data that is being passed back and forth. For additional functional descriptions of the dataflow, see Chapter 2, "SIM Integration – Functional".

RIB-based Integration

SIM can integrate with other Retail products (such as RMS, RPM, 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.

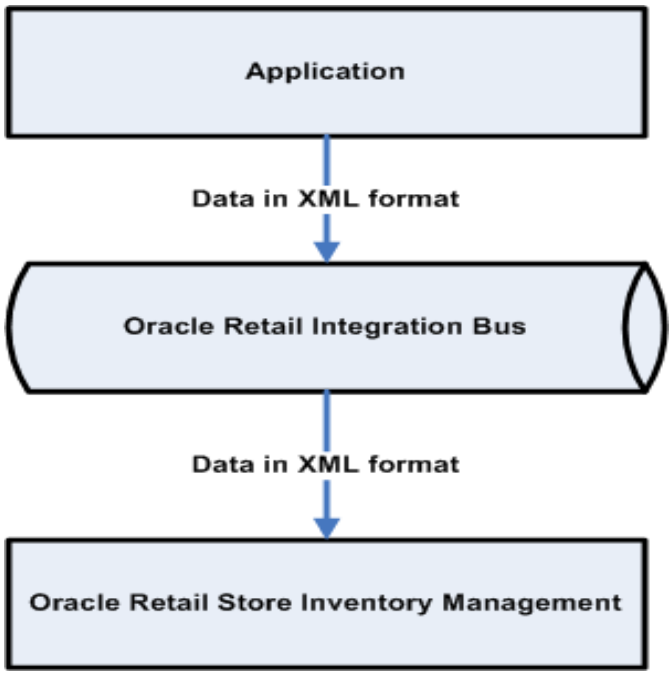
Figure 3–1 SIM/RIB Integration Diagram



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, RPM, 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 RIB Decoupling Framework

The SIM RIB Decoupling framework:

- Enables SIM to support multiple versions of RMS without affecting core SIM application logic. (Supporting different versions of RMS, needs functional analysis and implementation work, this framework would make it technically easier to go in that direction.)
- Enables SIM to integrate with RMS or any other system through different means other than the RIB.
- Isolates integration code.

Note: By default, SIM 13.1 only works with the Enterprise 13.1 release. (RIB/RMS/RPM)

Figure 3–3 SIM RIB Decoupling Framework Overview

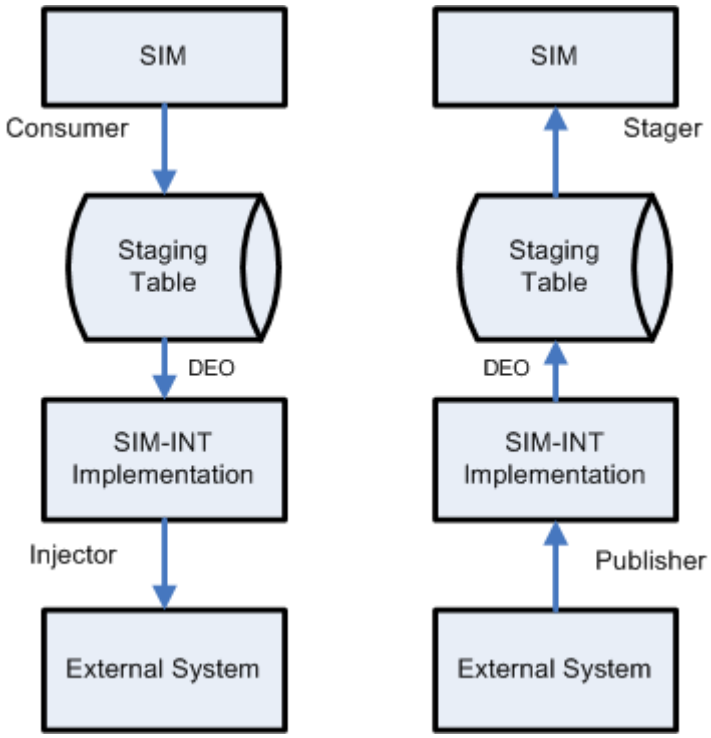


Figure 3–4 Detailed Injection Flow from External System to SIM

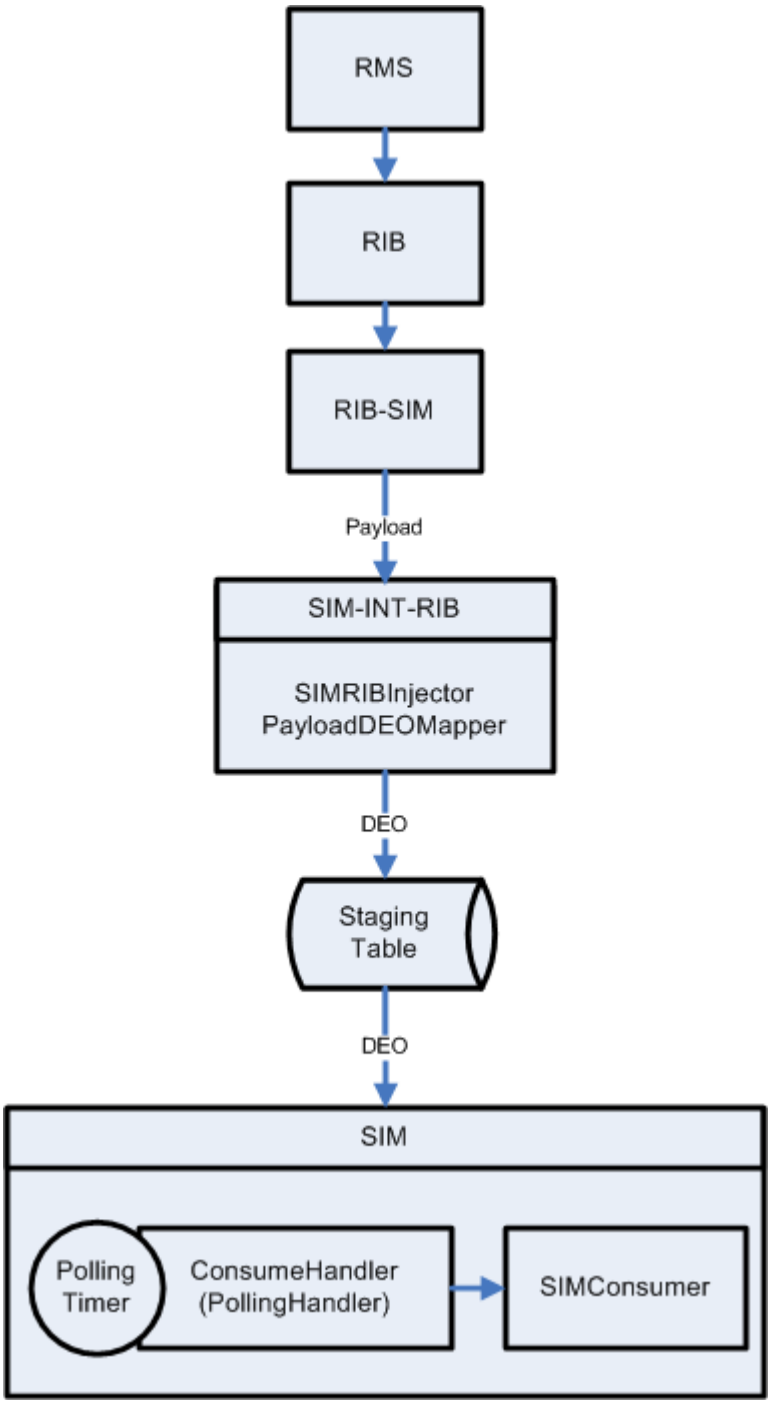
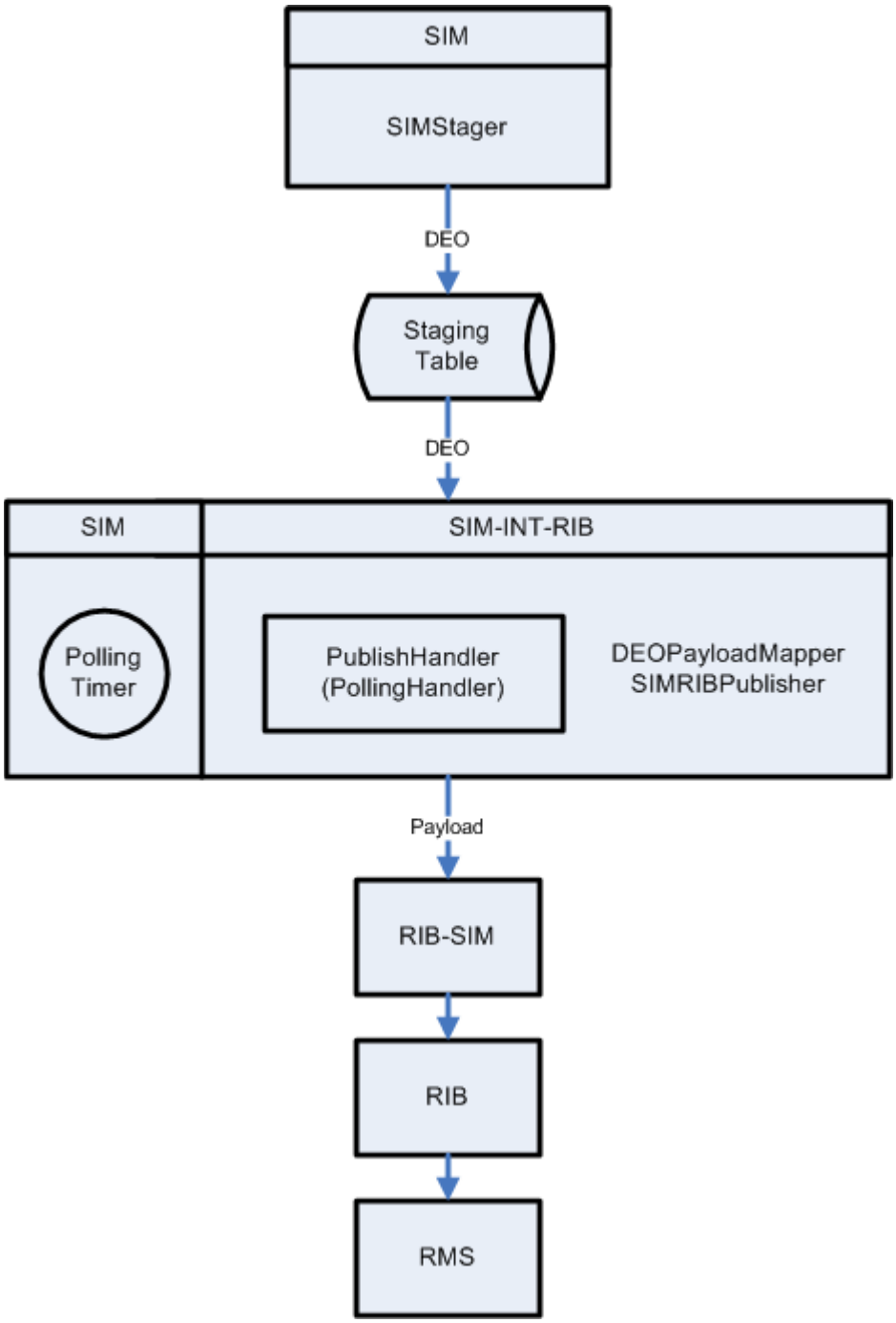


Figure 3–5 Detailed Publish Flow to External System from SIM



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 INTEGRATION_STAGING. 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)
CREATE_TIME
UPDATE_TIME
BUSINESS_ID
BUSINESS_TYPE
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)
ERROR_MESSAGE (the reason why it failed)
SERIALIZED_DEO (BLOB)

Note: A staged message can only be in three theoretical 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 MAX_RETRY defined in the POLLING_TIMER table that matches that message's {MESSAGE_FAMILY, MESSAGE_DIRECTION} tuple.

- FAILED

  A staged message that has a RETRY_COUNT >= the MAX_RETRY defined in the POLLING_TIMER table that matches that message's {MESSAGE_FAMILY, MESSAGE_DIRECTION} tuple.

Messages are either inbound or outbound:

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.

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.

A new table is introduced named POLLING_TIMER. The table looks like the following:

ID
TIMEOUT_SEC
MESSAGE_FAMILY
MESSAGE_DIRECTION
MAX_RETRIES
ENABLED
RECORDS_TO_PROCESS
LAST_RUN_TIMESTAMP
LOCK_TIMESTAMP
LOCK_COUNT

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.

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.

Polling Timer Admin Screen

POLLING_QTY

Defines the quantity of staged messages that are polled to process each time this polling timer fires. This is needed to limit the amount of messages returned as opposed to allowing it to process all available staged messages for this family (which could be a very large number).

MAX_RETRIES

The number times a staged message can be retried for this family before it is deemed in a FAIL state.

Note: Intermediate state is RETRY, and new messages are in a PENDING state.

TIMEOUT

Controls how often the polling timer should fire, in seconds. It will process staged messages that have been piling up between firings.

SIM guarantees ordering of messages processed within the same family with the same business_id.

Normal processing will be that a certain number of messages, not exceeding the POLLING_QTY, are queried and processed and spread over the number of threads defined in sim.cfg. Each thread will be in its own transaction, and following transactionality, if one thread fails, all threads fail. Therefore, if your POLLING_QTY is set to 500 and one message in that group fails during processing, all 500 messages are marked with a retry count of **1**. But they will immediately be processed by the retry timer which will be slower since it has to guarantee ordering, but this only happens in exceptional cases.

Messages with the following queried criteria are not able to be processed by the normal polling timer:

- In the staging table within the same family
- Have the same business_id
- Have one or more messages with a retry count >= 1

If this scenario is encountered, the retry timer will handle these situations in another thread by processing those messages one at a time. If a staged message's RETRY_ COUNT reaches the MAX_RETRIES value, that message and all associated messages with the same message family and same business_id are no longer processed by the polling timers.

SIM Polling Timer Configuration

There are 2 config entries in sim.cfg that also define how the polling timer threading works:

```
# Polling timer thread configuration
MAX_CONCURRENT_POLLING_TIMER_THREADS = 100;
MAX_PERCENT_OF_AVAILABLE_THREADS_PER_FAMILY = 25;
```

The first setting defines how many polling timer threads can be concurrently running on the system. The second setting determines what percentage of the available threads should be given to each message family.

Staged Message Admin Screen

The polling timers process staged messages at an interval defined by TIMEOUT. These messages are performed in batch chunks according to the POLLING_QTY setting of each polling timer. Messages are in one of three states: PENDING, RETRY, or 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 will open a dialog that enables you to see the error message and edit the data in XML format.

Customizations

Integration with external systems has been decoupled from SIM. This integration code now resides in a sim-int-<*> project, where, <*> refers to the external system. Currently, only sim-int-rib13 is fully implemented.

A PollingCoordinatorBean thread runs every five seconds, searching for polling timers that need to fire. When found, a PollingTimerBean thread is spawned with a List of StagedMessage objects. These PollingTimerBean threads process staged messages generically by calling the PollingTimerCommand.processStagedMessages(List<StagedMessage>). Once inside the PollingTimerCommand method, a SimMessageHandler is looked up and a handleMessages(List<StagedMessages) is called.

Each polling timer found in the POLLING_TIMER table corresponds to a SimMessageHandler defined by either the polling timer's message family or the message direction, or both. This mapping is done in the class that implements the SimMessageHandlerFactoryInterface defined in the sim.cfg in the key MESSAGE_HANDLER_FACTORY_IMPL.

The PollingTimerCommand is called by each PollingTimerBean to process message. To process the message, the SimMessageHandlerFactory is called to get the SimMessageHandler to use.

After messages are in the INTEGRATION_STAGING_TABLE, polling timers act on these messages. POLLING_TIMERS are defined in the POLLING_TIMER table.

Creating a custom consumer Do the following to create a custom consumer class:

1. Extend SimMessageConsumer.
2. Implement the consume(List<DataExchangeObject> deos) method.
3. Update SimMessageConsumerFactoryImpl to map the desired SimMessageType to your consumer.

> **Note:** The message family to which the message type belongs will define which polling timer controls how many messages are processed, and how often those messages are processed.

Creating a custom stager Do the following to create a custom stager:

1. Create a custom stager class by extending SimMessageStager.
2. Implement the SimMessageType getMesageType() method.
3. Implement the List<DataExchangeObject> getDataExchangeObjects() method.
4. Implement the String getStoreId() method.
5. Update SimMessageStagerFactoryImpl to return your custom Stager in the method for the desired business object.

> **Note:** The message family to which the message type belongs will define which polling timer controls how many messages are processed, and how often those messages are processed.

Known Issues and Reminders

- Make sure the polling timer is turned on.
- Make sure the timeout value is set low enough to avoid extended sleep time between firing (default is 60 seconds).
- Make sure the max retry value for the polling timer is set high enough so messages are given a chance to retry (default is **5**).

- 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**.

> **Note:** If the RIB is down the staged message retry count will not be incremented. This is because a failed publish, due to lack of connectivity, is deemed a non-attempt rather than a failed attempt, and the system silently retries until the RIB comes back online.

The following background info might be helpful:

When a polling timer fires, it queries for a certain number of messages defined by the RECORDS_TO_PROCESS field in the POLLING_TIMER table. 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 ore 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 grouping all similar staged messages into buckets. The buckets are then distributed over a number of threads. The number of threads is defined by the MAX_CONCURRENT_POLLING_TIMER_THREADS and the MAX_PERCENT_OF_AVAILABLE_THREADS_PER_FAMILY config values in the sim.cfg file. This defines how many threads can be running at once and so on. The defaults for this should be adequate.

Database Considerations

Rebuilding the indexes on the INTEGRATION_STAGING 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 integration_staging ENABLE ROW MOVEMENT;
3. ALTER TABLE integration_staging SHRINK SPACE CASCADE;
4. ALTER TABLE integration_staging DISABLE ROW MOVEMENT;
5. Gather the table statistics

6. Call DBMS_STATS.GATHER_TABLE_STATS(tabname => 'integration_staging', ESTIMATE_PERCENT => dbms_stats.auto_sample_size, CASCADE => 'true')

The high-water reset requires the integration_staging 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

The following table 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.

Table 3–1 **Subscribers Mapping Table**

Family	Type	Payload	Subscribing class (injector)
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	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

Table 3–1 **Subscribers Mapping Table**

Family	Type	Payload	Subscribing class (injector)
ITEMS	ISCDIMCRE	ISCDimDesc	ISCDimCreateConsumer
ITEMS	ISCDIMMOD	ISCDimDesc	ISCDimModifyConsumer
ITEMS	ISCDIMDEL	ISCDimRef	ISCDimRemoveConsumer
ITEMS	ISCMFRMOD	ItemSupCtyMfrDesc	ItemSupCtyMfrModifyConsumer
ITEMS	ISCMFRDEL	ItemSupCtyMfrRef	ItemSupCtyMfrRemoveConsumer
ITEMS	ISCMFRCRE	ItemSupCtyMfrDesc	ItemSupCtyMfrCreateConsumer
ITEMS	ITEMTCKTCRE	itemTcktDesc	ItemTcktCreateConsumer
ITEMS	ITEMTCKTDEL	ItemTcktRef	ItemTcktRemoveConsumer
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
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
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

Table 3–1 **Subscribers Mapping Table**

Family	Type	Payload	Subscribing class (injector)
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
WH	WHCRE	WHDesc	WareHouseCreateConsumer
WH	WHDEL	WHRef	WareHouseRemoveConsumer
WH	WHMOD	WHDesc	WareHouseModifyConsumer

Publishers Mapping Table

This table 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.

Table 3–2 **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

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 via 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 RPM for future retail price inquiry and price change requests. RSL for RPM runs within the RPM 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 3–3 **RSL services used by SIM**

Service name	Description
PriceInquiryService	This service, provided by RPM, allows an inquiring system to request the effective retail for an item at a specified location on a given date. RPM 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 RPM 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 3–4 **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 services below, 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
```

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 RPM 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 POS 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.

Integration with Oracle Retail Workspace

The Oracle Retail Workspace installer prompts you to enter the URL for your supported Oracle Retail applications. However, if a client installs a new application after Oracle Retail Workspace is installed, the retail-workspace-page-config.xml file needs to be edited to reflect the new application.

The file as supplied comes with all appropriate products configured, but the configurations of non-installed products have been turned off. Therefore, when turning on a product, locate the appropriate entry, set rendered to **true**, and enter the correct URL and parameters for the new application.

The entry consists of the main URL string plus one parameter named **template**. The installer inserts the value of the template parameter. Somewhere in the installer property files there is a value for the properties **deploy.retail.product.rms.url** and **deploy.retail.product.rms. template**.

For example, suppose RMS was installed on mycomputer.mycompany.com, port 7777, using a standard install and RMS configured with the application name of **rms121sedevhpsso**. If you were to access RMS directly from your browser, you would enter:

```
http://mycomputer.mycompany.com:7777/forms/frmservlet?template=rms121sedevhpsso
```

The entry in the retail-workspace-page-config.xml after installation would resemble the following:

```
<url>http://mycomputer.mycompany.com:7777/forms/frmservlet</url>
<parameters>
<parameter name=" template ">
<value>rms121sedevhpsso</value>
</parameter>
</parameters>
```

For more information about Oracle Single Sign-on, see "Oracle Single Sign-on Overview" in the *Oracle Retail Store Inventory Management Implementation Guide – Volume 1*.

SIM Web Service 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
lookupStoreInventory	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.

lookupStoreInventory Operation

LookupStoreInventory is your request to lookup store inventory.

lookupStoreInventory Types

Figure 3–6 Namespace: invlookup

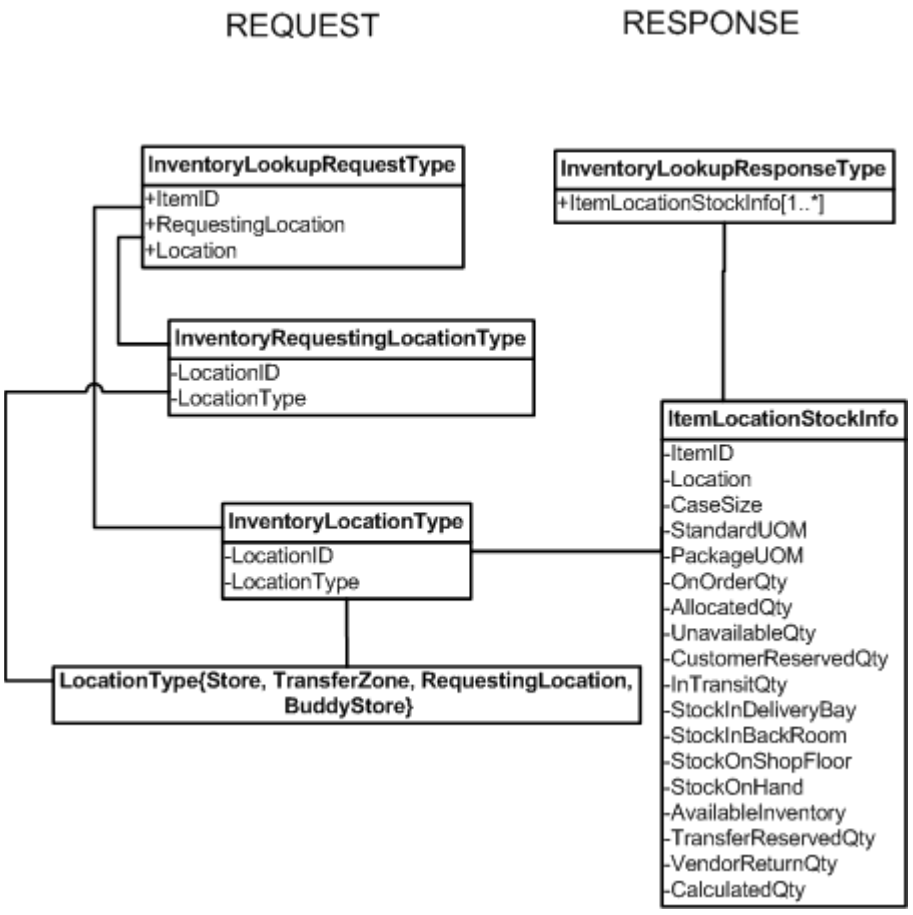


Table 3–5 InventoryLookupRequestType

Element	Description	Data Type	Required
ItemID	The item number in SIM.	xs:string	Yes
RequestingLocation	The requesting location	invlookup:InventoryRequestingLocationType	Yes
Location	The location for which inventory is being requested.	invlookup:InventoryLocationType	Yes Note: If RequestingLocation.LocationType is **RequestingLocation**, then this field is not required.

Table 3–6 InventoryRequestingLocationType

Element	Description	Data Type	Required
LocationID	The location ID.	xs:string	Yes
LocationType	The location type. Only **Store** is allowed.	invlookup:LocationType{Store, RequestingLocation, TransferZone, BuddyStore}	Yes

Table 3–7 InventoryLocationType

Element	Description	Data Type	Required
LocationID	The location ID.	xs:string	Yes
LocationType	The location type.	invlookup:LocationType{Store, RequestingLocation, TransferZone, BuddyStore}	Yes

Table 3–8 InventoryLookupResponseType

Element	Description	Data Type
ItemLocationStockInfo (0..*)	Stock information for item and location (multiple)	Invlookup:ItemLocationStockInfo

Table 3–9 ItemLocationStockInfo

Element	Description	Data Type
ItemID	The item number in SIM.	xs:string
Location	The location for which inventory is being requested.	invlookup:InventoryLocationType
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

Table 3–9 ItemLocationStockInfo

Element	Description	Data Type
StockInDeliveryBay	Quantity in delivery bay	xs:decimal
StockInBackRoom	Quantity in back room	xs:decimal
StockOnShopFloor	Quantity on shop floor	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 a point-of-sale system. 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 a synchronous manner.
processMultipleCustomerOrder	Process multiple customer orders, in an asynchronous (batch) mode. These requests will be staged and processed later using the polling timer framework.

processCustomerOrder Operation

This API allows an external system to create or update a single customer order request in a synchronous manner. This API performs the following tasks:

- Validates the customer order request for proper inputs.
- Performs the UOM conversion, if necessary.
- Creates or updates the customer order.
- Creates the inventory adjustment to update customer reserve quantity and publish the same to RMS.
- Inserts the request details to audit tables.

processCustomerOrder Types

Figure 3–7 NameSpace: customerOrder

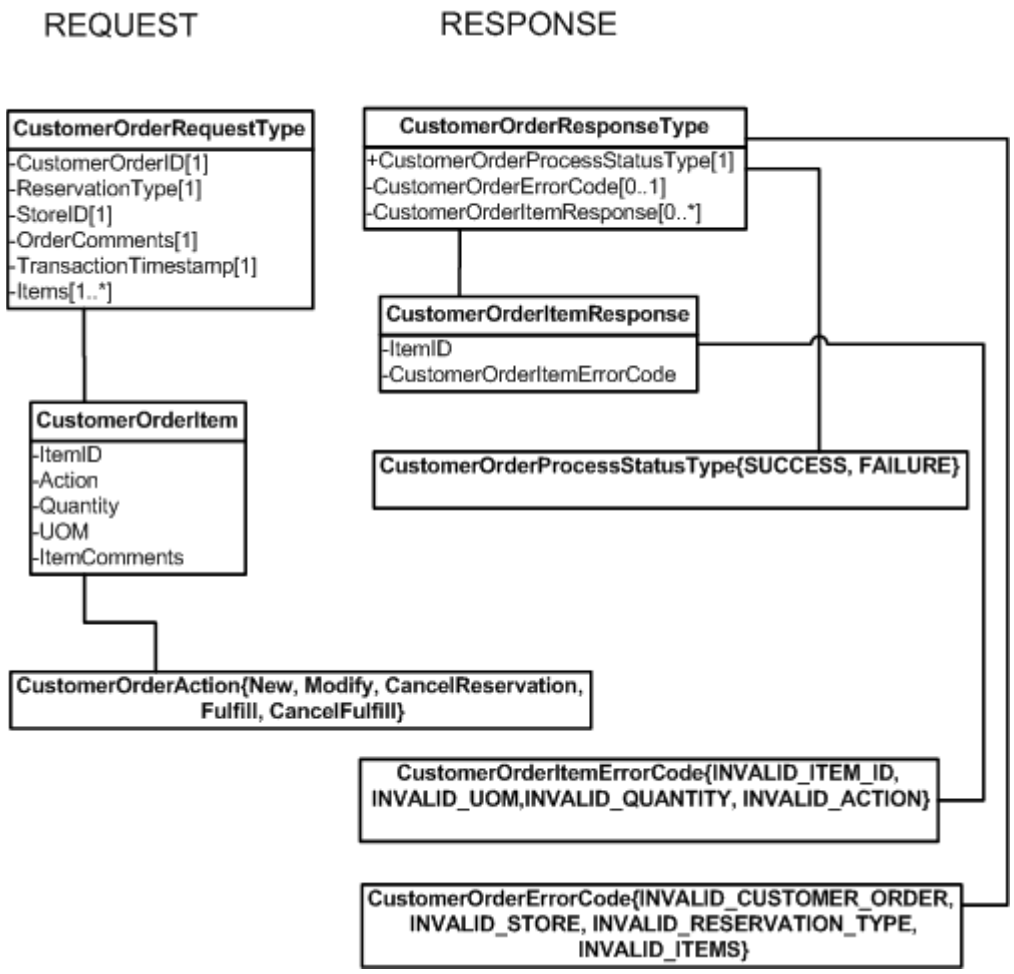


Table 3–10 CustomerOrderRequestType

Element	Description	Data Type	Required
CustomerOrderID	The unique identifier of the customer order as defined by 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 will use the unique identifier presented and decode it using the new Reservation Type setup table.	xs:string	Yes
StoreID	The unique identifier of the Store that is reserving the inventory.	xs:string	Yes
OrderComments	CO comments	xs:string	No
TransactionTimeStamp	TS	xs:datetime	Yes
Items (0..*)	The CO items (multiple)	customerorder:CustomerOrderItem	Yes

Table 3–11 CustomerOrderItem

Element	Description	Data Type	Required
ItemID	The item number whose inventory needs to be reserved in SIM.	xs:string	Yes
Action	Indicates what type of action the system should take with the Customer Order item record.	customerorder:CustomerOrder Action {New, Modify, CancelReservation, Fulfill, CancelFulfill}	Yes
Quantity	The quantity of the item being reserved. With the combination of the action field, this value will always be a positive number. Actions of New, Modify and CancelFulfill will increase the customer reserved quantity and actions of CancelReservation or Fulfill will decrease the customer reserve quantity.	xs:decimal	Yes
UOM	The UOM of the quantity being reserved. SIM performs UOM conversion if necessary.	xs:string	Yes
ItemComments	Any comments passed in from the external systems that pertain to the specific item.	xs:string	No

Table 3–12 CustomerOrderResponseType

Element	Description	Data Type
CustomerOrderProcessStatusType	The process status	customerorder:CustomerOrderProcessStatusType {SUCCESS, FAILURE}
CustomerOrderErrorCode	Error code if any.	customerorder:CustomerOrderErrorCode{INVALID_CUSTOMER_ORDER, INVALID_STORE, INVALID_RESERVATION_TYPE, INVALID_ITEMS}
CustomerOrderItemResponse (0..*)	Response per item (multiple)	customerorder:CustomerOrderItemResponse

Table 3–13 CustomerOrderItemResponse

Element	Description	Data Type
ItemID	The item ID present in the request.	xs:string
CustomerOrderItemErrorCode	Error code if any at item level.	customerorder:CustomerOrderItemErrorCode{INVALID_ITEM_ID, INVALID_UOM,INVALID_QUANTITY, INVALID_ACTION}

processMultipleCustomerOrder Operation

This API allows external system to create or update multiple customer orders, in an asynchronous (batch) mode. These requests are staged and processed at a later time using the polling timer framework. The multiple order requests are written to staging tables and the response is sent back to the calling program. These requests are processed some time later and the inventory buckets (customer reserve quantity) are updated accordingly.

The API performs following tasks:

- Write the request to staging tables (staged messages) and send the SUCCESS response back to the calling program.
- These staged messages are later picked up one by one for processing, using polling timer framework.

The following processing is done on each message (request):

- Validate the customer order request for proper inputs.
- Perform the UOM conversion, if necessary.
- Create or update the customer order.
- Create the inventory adjustment to update customer reserve quantity and publish the same to RMS.
- Insert the request details to audit tables.

processMultipleCustomerOrder Types

Figure 3–8 NameSpace: customerOrder

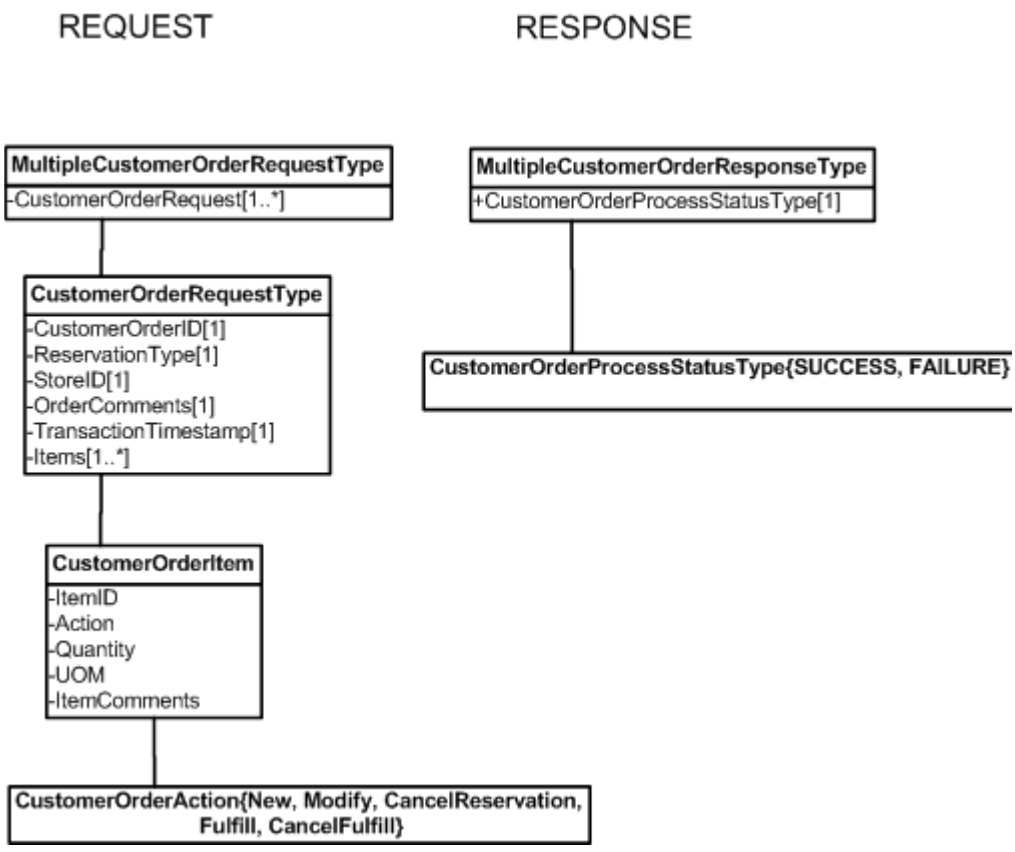


Table 3–14 MultipleCustomerOrderRequestType

Element	Description	Data Type	Required
CustomerOrderRequest (1..*)	Customer Order Request batch (multiple)	customerOrder:CustomerOrderRequestType	Yes
OrderComments	CO comments	xs:string	No

Table 3–15 CustomerOrderRequestType

Element	Description	Data Type	Required
CustomerOrderID	The unique identifier of the customer order as defined by 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 will use the unique identifier presented and decode it using the new Reservation Type setup table.	xs:string	Yes
StoreID	The unique identifier of the Store that is reserving the inventory.	xs:string	Yes
OrderComments	CustomerOrder comments	xs:string	No
TransactionTimeStamp	TS	xs:datetime	Yes
Items (0..*)	The CustomerOrder items (multiple)	customerOrder:CustomerOrderItem	Yes

Table 3–16 CustomerOrderItem

Element	Description	Data Type	Required
ItemID	The item number whose inventory needs to be reserved in SIM.	xs:string	Yes
Action	Indicates what type of action the system should take with the Customer Order item record.	customerOrder:CustomerOrderAction	Yes
Quantity	The quantity of the item being reserved. With the combination of the action field, this value will always be a positive number. Actions of New, Modify and CancelFulfill will increase the customer reserved quantity and actions of CancelReservation or Fulfill will decrease the customer reserve quantity.	xs:decimal	Yes
UOM	The UOM of the quantity being reserved. SIM performs UOM conversion if necessary.	xs:string	Yes
ItemComments	CO comments	xs:string	No

Table 3–17 MultipleCustomerOrderResponseType

Element	Description	Data Type
CustomerOrderProcessStatusType	The process status	customerOrder:CustomerOrderProcessStatusType {SUCCESS, FAILURE}

Item Basket API

This 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 Item Basket 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

Figure 3–9 NameSpace: itembasket

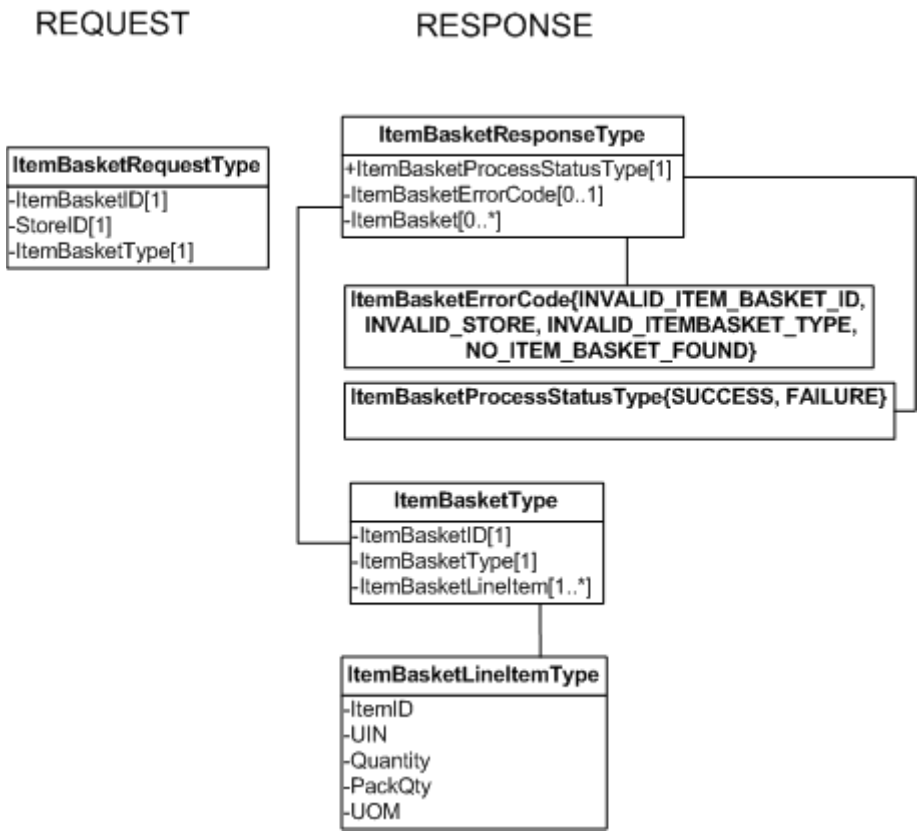


Table 3–18 ItemBasketRequestType

Element	Description	Data Type	Required
ItemBasketID	The Item Basket ID	xs:string	Yes
StoreID	The store ID	xs:string	Yes
ItemBasketType	The item basket type	xs:string	Yes

Table 3–19 ItemBasketResponseType

Element	Description	Data Type
ItemBasketProcessStatusType	The process status, its value can be **SUCCESS** or **FAILURE**	itembasket:ItemBasketProcessStatusType
ItemBasketErrorCode	Error code if any. Currently there are the following error codes: ■ INVALID_ITEMBASKET_ID ■ INVALID_ITEMBASKET_TYPE ■ INVALID_STORE	itembasket:ItemBasketErrorCode
ItemBasket	This contains all the attributes of Item Basket.	Itembasket:ItemBasketType

Table 3–20 ItemBasketType

Element	Description	Data Type
ItemBasketID	This is the unique identifier for Item Basket.	xs:string
ItemBasketType	The item basket type, its value can be **Item Basket** or **Customer Order**.	xs:string
ItemBasketLineItem (1..*)	This is a list of attributes of items contained in Item Basket.	itembasket:ItemBasketLineItemType

Table 3–21 ItemBasketLineItemType

Element	Description	Data Type
ItemID	Item number.	xs:string
UIN	UIN Serial Number, unimplemented.	xs:string
Quantity	This value is the number of items contained in Item Basket.	xs:decimal
PackQty	This value is currently defaulted to **1**.	xs:decimal
UOM	Selling Unit of Measure of the Item present in Item Basket.	xs:string