Oracle® Retail Store Inventory Management

Operations Guide
Release 13.0

April 2008

ORACLE

Oracle® Store Inventory Management Operations Guide, Release 13.0

Primary Author: Graham Fredrickson

Value-Added Reseller (VAR) Language

(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 and Oracle Retail Demand Forecasting 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 Store Inventory Management.

(v) the software component known as **Crystal Enterprise Professional and/or Crystal Reports Professional** licensed by Business Objects Software Limited ("Business Objects") 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 **WebLogic™** developed and licensed by BEA Systems, Inc. of San Jose, California, to Oracle and imbedded in the Oracle Retail Value Chain Collaboration application.

(x) 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.

Contents

Preface

Oracle Retail Operations Guides are designed so that you can view and understand the application's 'behind-the-scenes' processing, including such information as the following:

- Key system administration configuration settings
- Technical architecture
- Functional integration dataflow across the enterprise

Audience

Anyone who has an interest in better understanding the inner workings of the Oracle Retail Store Inventory Management (SIM) system can find valuable information in this guide. There are three audiences in general for whom this guide is written:

- System analysts and system operation personnel:
    - who are looking for information about SIM's processes internally or in relation to the systems across the enterprise.
    - who operate SIM on a regular basis.
- Integrators and implementation staff who have the overall responsibility for implementing SIM into their enterprise.
- Business analysts who are looking for information about processes and interfaces to validate the support for business scenarios within SIM and other systems across the enterprise.

Related Documents

For more information, see the following documents in the Oracle Retail Store Inventory Management Release 13.0 documentation set:

- Oracle Retail Store Inventory Management Data Model
- Oracle Retail Store Inventory Management Handheld Terminal Quick Reference Guide
- Oracle Retail Store Inventory Management Implementation Guide
- Oracle Retail Store Inventory Management Installation Guide
- Oracle Retail Store Inventory Management Licensing Information
- Oracle Retail Store Inventory Management Online Help
- Oracle Retail Store Inventory Management Release Notes
- Oracle Retail Store Inventory Management User Guide

Customer Support

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 re-create
- Exact error message received
- Screen shots of each step you take

Review Patch Documentation

For a base release (".0" release, such as 13.0), Oracle Retail strongly recommends that you read all patch documentation before you begin installation procedures. Patch documentation can contain critical information related to the base release, based on new information 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:

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

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

> **Note:** This is a note. It is used to call out information that is important, but not necessarily part of the procedure.

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

A hyperlink appears like this.

1
Introduction

This operations guide serves as an Oracle Retail Store Inventory Management (SIM) reference to explain backend processes. SIM is designed as a standalone application that can be customized to work with any merchandising system.

Overview

SIM empowers store personnel to sell, service, and personalize customer interactions by providing users the ability to perform typical back office functionality on the store sales floor. The results are greatly enhanced customer conversion rates, improved customer service, lower inventory carrying costs, and fewer markdowns. SIM delivers the information and flexible capabilities that store employees need to maintain optimal inventory levels and to convert shoppers into buyers.

The SIM solution performs the following:

- Improves perpetual inventory levels by enabling floor-based inventory management through handheld devices and store PCs.
- Minimizes the time to process receipt and check-in of incoming merchandise.
- Receives, tracks, and transfers merchandise accurately, efficiently, and easily.
- Reduces technology costs by centralizing hardware requirements.
- Guides users through required transactions.
- Allows customizations to the product through an extensible technology platform. The retailer's modifications are isolated during product upgrades, lowering the total cost of ownership.

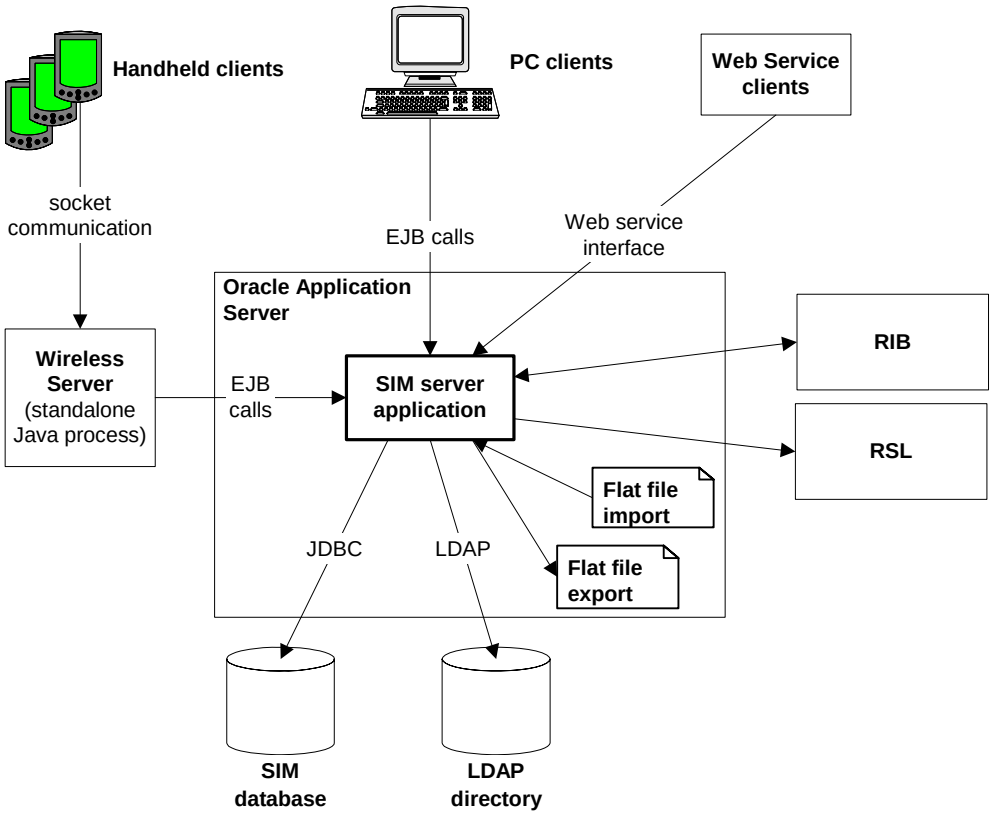
Technical Architecture Overview

SIM's robust distributed computing platform enables enhanced performance and allows for scalability.

SIM has a client tier, a server tier, and a data tier. The n-tier architecture of SIM allows for the encapsulation of business logic, shielding the client from the complexity of the backend system. The separation of presentation, business logic, and data makes the software cleaner, more maintainable, and easier to modify. Any given tier need not be concerned with the internal functional tasks of any other tier.

One of SIM's most significant advantages is its flexible distributed topology. SIM offers complete location transparency because the location of data and/or services is based upon the retailer's business requirements, not upon technical limitations. The server is not deployed within the store. The application's clients talk to the server across the wire in almost real time.

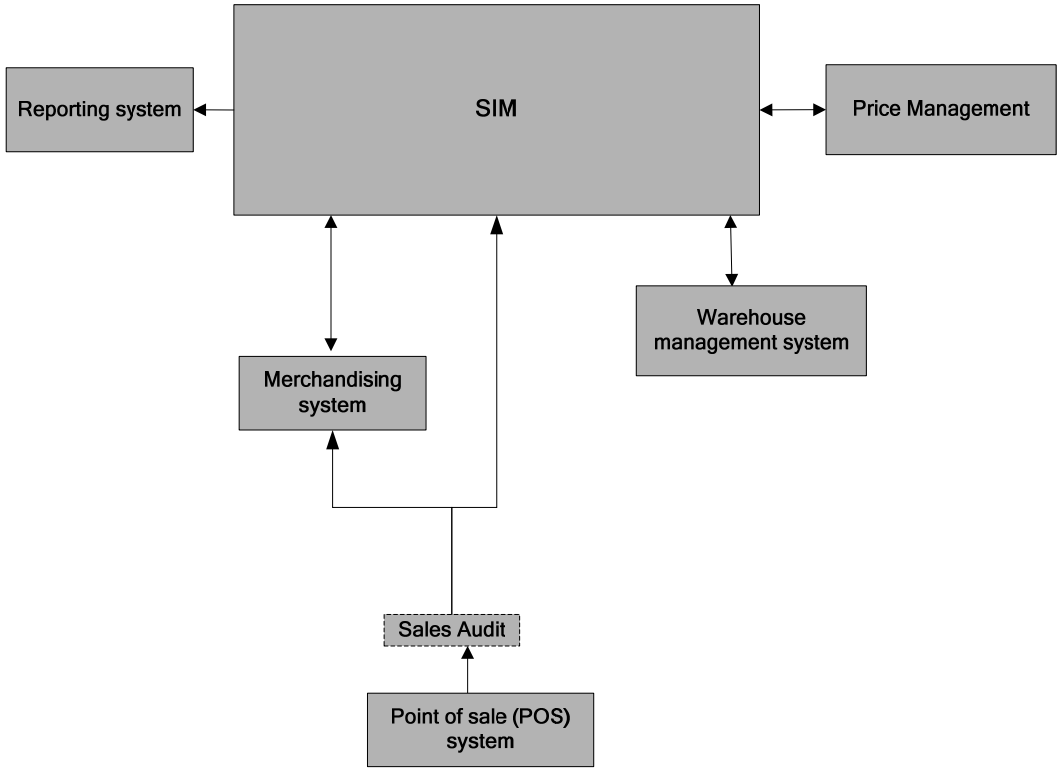
The following diagram offers a high-level conceptual view of the main components and integration points of the SIM architecture. For a detailed description of this diagram, see "Technical Architecture".



SIM's technical architecture

SIM's Integration Points into the Retail Enterprise

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 "SIM Integration – Functional".



SIM-related dataflow across the enterprise

2

Backend System Configuration

This chapter of the operations guide is intended for administrators who provide support and monitor the running system.

The content in this chapter is not procedural, but is meant to provide descriptive overviews of key system parameters, logging settings, and exception handling.

Configuring SIM Across Time Zones

For many SIM retailers, a corporate server is located in a different time zone than the stores connected to that corporate server. When a transaction is processed at these respective locations, there is timestamp information associated with these transactions. SIM has the ability to reconcile these time zone differences.

System administration options enable you to specify the time zone to use when timestamps are published to or received from the Oracle Retail Integration Bus (RIB). The system options are called 'Enable GMT for…', with options for Inventory Adjustments, Price Changes, Store Orders, Store Transfers, Warehouse Transfers, Receiving, Direct Deliveries, Vendor ASN, RTV, Item Requests, Sales Data, Foundation Data, Dex/Nex, Stock Counts, and Third Party Stock Counts.

- If Enable GMT is set to 'yes,' timestamps are published to the RIB in GMT, and incoming timestamps in RIB messages will be read as GMT.
- If Enable GMT is set to 'no,' timestamps are published to the RIB in the store time zone, and incoming timestamps in RIB messages will be read as the store time zone.

The PA_RTL_STR table contains the field RK_TIMEZONE, which holds the time zones for each store. An administrator (or DBA) should determine the correct time zone, and enter this information into the table. As stated above, once retailers have specified the local (store) time, they can specify which time zone, GMT or store, to use for timestamp publication to the RIB.

> **Note:** A complete list of time zones has been compiled and is packaged with the release of this version of SIM, and can also be found in the SIM database view TIME_ZONE_NAMES_V.

Supported Oracle Retail Products/Environments

SIM is compatible with Oracle Retail Merchandising System (RMS) and Oracle Retail Price Management (RPM). This functionality is described in greater detail in the integration chapters.

For information about requirements for SIM's client, server(s), and database, see the *SIM Installation Guide*.

Configuration Files

Key client-defined configurations for SIM are described in this section. The system parameters contained in these files are also detailed. Many parameters have been omitted from this section because retailers should *not* have to change them.

Note that within these files (and thus in some of the examples from those files below), a # sign that precedes a value in the file signifies that what follows is a comment and is not being utilized as a setting.

Some settings in the files are configurable. Thus, when retailers install SIM into an environment, they must update these values to their specific settings.

batch_db.cfg – Database connection info for batch programs

This file is no longer used.

dao.cfg – Data access object implementations

This file defines the implementation classes for all data access objects in SIM. Each value is the fully qualified class name of the implementation class for that key. If a retailer customizes SIM, they may need to change some of the class names in this file.

date.cfg – Date format configuration

This file contains java format pattern strings for several different types of dates defined in the system. These pattern strings follow the rules defined in java for SimpleDateFormat. The key for the date is defined as language and country followed by the pattern key. For example, enAU.entryDate is the entry format for dates in English for Australia.

The pattern keys are entryDate (used for date entry in calendar editor), shortDate (format for short length date - this is the most commonly used), mediumDate (format for medium length date), longDate (nearly complete date format), fullDate (fully written out date format), monthPattern (formats only month and day), wirelessInput (defines entry for wireless device), wirelessOutput (defines the format of dates on the wireless device) and wirelessDisplay (defines the exact text string to display to the user at the entry location).

The prefix before the pattern key is the ISO standard two-character code for language and then two-character code for country. Both language and country must be present. Additional language/country combinations may be added as desired (for example, frUS is french in United States).

integration.cfg – Integration (RIB and RSL) settings

This file contains settings related to SIM integration via Oracle Retail Integration Bus (RIB) and Oracle Retail Service Layer (RSL). This file contains the following keys:

- ribMessagePublishEnabled – if set to "true", SIM will actually publish messages to the RIB during processing. If set to "false", SIM will not publish messages to the RIB, but will instead log the messages to the SIM server log file. This is intended to be used only for troubleshooting purposes. For an integrated production environment, the value should be "true".
- rslCallsEnabled – if set to "true", SIM will actually make RSL calls during processing. If set to "false", SIM will not allow the user to access areas of the application that call RSL. This is intended to be used only for troubleshooting purposes. For an integrated production environment, the value should be "true".

- *_PUB – the various keys that end in "_PUB" are the class names of classes that implement interfaces to publish messages to the RIB. If a retailer customizes SIM, they may need to change some of the class names in this file.
- RMS_VERSION – this key is no longer used.

jndi.cfg – JNDI settings

This file contains JNDI configuration settings. In the SIM server, the only key used is:

- DB_JNDI_NAME – the name of the datasource SIM will use to get database connections

However, java processes that are clients to the SIM server (the wireless server and the java batch programs), the other keys are used to determine the JNDI information for looking up the SIM server's EJBs:

- INITIAL_CONTEXT_FACTORY – the name of the factory used to get an initial JNDI context. This should not be changed.
- OBJECT_FACTORY_PACKAGES – the java packages containing object factories. This should not be changed.
- NAMING_SERVER_URL – the JNDI URL for the naming server. This should be configured to point at the SIM server's JNDI URL. The SIM installer should have set this.
- SECURITY_PRINCIPAL – the user name to connect to the Oracle Application Server's JNDI context. The SIM installer should have set this.
- SECURITY_CREDENTIALS – the password to connect to the Oracle Application Server's JNDI context. The SIM installer should have set this.

ldap.cfg – Configuration for connecting to an LDAP server

This file contains various configuration parameters for connecting to an LDAP server. The SIM installer should have set all values.

log4j.xml

This contains configuration about what information gets logged and where it gets logged. See "Logging Information" for more information.

reporting.cfg – Configuration for printing reports

See the Reporting chapter in the *Oracle Retail Store Inventory Management Implementation Guide* for more information about this file.

services.cfg – Service implementation classes

This file contains entries for every service in SIM that define the client-side, downtime, and server-side implementations of a given service interface. If a retailer customizes SIM, they may need to modify this file.

sim.cfg – SIM server configuration

This file contains various parameters used by the SIM server. This file contains the following keys:

- DB_LOCK_WAIT_TIME – the number of seconds that SIM should wait when trying to acquire a database lock.
- DB_BATCH_MAX_PARAM – the maximum number of parameters allowed in a batch JDBC statement. When the server arranges a large number of JDBC statements into groups to execute them as a JDBC batch, this will be the maximum group size.
- BO_FACTORY_IMPL – the fully qualified class name of the class implementing the BOFactory interface. This implementation is responsible for instantiating new Business Objects in the SIM code. A retailer might need to change this value if customizing SIM.
- SERVER_INITIALIZE_CLASSNAMES – a comma-delimited list of classes that implement oracle.retail.sim.closed.common.Initializer. These classes are run when the SIM server is started.
- CURRENCY_DEFAULT_TYPE – the currency code for the default currency. If none is given, the base currency defaults to **USD**. This currency code will only be used when currency information is not available for something in SIM, which is a rare situation.
- STOCK_COUNT_MAX_AUTH_LINES – when breaking a stock count into groups of line items to authorize, the groups will be a maximum of this size

Parameters For User Information For Connecting To Oracle Retail Price Management (RPM)

This information does not need to correspond to actual RPM users; it is used only for logging.

- RPM_USER_NAME – the user name with which to connect to RPM.
- RPM_USER_FIRST_NAME – the first name of the user connecting to RPM.
- RPM_USER_LAST_NAME – the last name of the user connecting to RPM.

Parameters Related To Server-Side Cache Refresh Rates

All settings are given in milliseconds. When a client needs to read information that can be cached, it will first look in the cache. If the cache is empty or expired, the client will call the server to find the current data. Otherwise the cached settings are used.

- REFRESH_RATE_CONFIG – timeout for cache of system configuration parameters
- REFRESH_RATE_STORE – timeout for store-specific configuration parameters
- REFRESH_RATE_TRANSLATION – timeout for holding translations on the server. Translations displayed on the wireless client are held in this cache. Translations displayed on the PC client are not held in this cache.
- REFRESH_RATE_WIRELESS_ITEM – timeout for storing differentiators for items on stock counts displayed in the wireless client.
- REFRESH_RATE_MERCH_HIERARCHY – timeout for holding merchandise hierarchy information

Parameters Related To Batch Processing Operation

- BATCH_NUM_THREADS_IN_POOL
- DEXNEX_INPUT_DIR

- DEXNEX_ERROR_DIR
- RESA_FILE_DIR
- RESA_FILE_ORIG_DIR
- RESA_TRANSACTION_SIZE

See "Batch Processes" for more details about these parameters.

telephone.cfg – Telephone format configuration

This file contains various formatting styles for phone numbers in the United States, Germany, and the United Kingdom. There are many different formats included for each country. A retailer could modify this file to add their own format if desired. See the file itself for more detailed documentation.

wireless_client_master.cfg – Wireless Server Configuration

This file contains configuration used by the Wireless Server. The only key used is:

- INITIALIZE – a comma-delimited list of classes that implement oracle.retail.sim.closed.common.Initializer. These classes are run when the Wireless server is started.

wireless_services.cfg

This file is no longer used.

retek/jndi_providers.xml - JNDI Configuration File

SIM uses this file as part of its RSL-based integration with the Oracle Retail Price Management (RPM) and Oracle Retail Merchandising System (RMS) applications, and for connecting to the Retail Integration Bus (RIB). For more information about this integration, see the integration chapters of this document. The jndi providers file contains JNDI naming URL information for the other Oracle Retail applications to which SIM makes remote calls.

retek/jndi_providers_ribclient.xml - JNDI Configuration File for RIB Communication

SIM uses this XML file to provide the location of the JNDI directory used to communicate with the RIB. The following properties must be defined:

- java.naming.factory.initial – the JNDI factory class. Should not need to be changed.
- java.naming.provider.url – the JNDI URL of the RIB J2EE application. Set by the installer.
- java.naming.security.principal – the JNDI username to connect to the RIB. Set by the installer.
- java.naming.security.credentials – the JNDI password to connect to the RIB. Set by the installer.

retek/rules_sim.xml – Business Rules configuration

This file defines business rules that are run in SIM. If a retailer customizes SIM, this file may need to be modified.

retek/rib/injectors.xml – RIB subscriber configuration

This file defines the classes that are used in SIM to handle messages coming in over the RIB. A class is defined for each family type/message type combination that is supported by SIM. If a retailer customizes SIM, this file may need to be modified. For more information, see the Integration chapters of this document, and the RIB documentation.

Port Configuration

The SIM PC and handheld clients require a number of ports to be open on the SIM server in order to communicate. That means these ports will have to be opened on any firewalls between the SIM clients and the SIM server.

The following types of ports are required to be open by SIM:

- OAS HTTP port (to download the SIM client)
- OAS RMI ports (to make RMI calls from the SIM client to the SIM server)
- Wavelink server port (for the handheld devices to communicate with the Wavelink server)

The Wavelink port is defined in wavelink-startup.sh and wireless_services.cfg. See the "Wireless Server Port in wavelink-startup.sh and wireless_services.cfg" section of the "SIM Configuration Files" appendix of the *SIM Installation Guide* for more information.

The Oracle Application Server controls the HTTP and RMI ports. The HTTP port is a single port, but the RMI ports are defined as a range of ports. These port numbers can be changed if necessary. Refer to the following documentation for descriptions and instructions on how to change the ports:

Oracle® Application Server Administrator's Guide

Section D.2 Port Numbers (Sorted by Port Number) - *Shows the port ranges assigned by default*

(http://download.oracle.com/docs/cd/B25221_04/core.1013/b25209/portnums.htm#i688124)

Section 4.3.1 Changing OC4J Ports - Instructions for how to change port ranges

(http://download.oracle.com/docs/cd/B25221_04/core.1013/b25209/ports.htm#i1038852)

Configuring the Transaction Timeout for SIM

The default transaction timeout in an OC4J instance is 30 seconds. This is not sufficient for some of SIM's processes, especially batch processes. The recommended transaction timeout for SIM is 7200 seconds. Refer to the following documentation for instructions on how to set the transaction timeout:

Oracle® Containers for J2EE Services Guide

Section 5.3 "Configuring the OC4J Transaction Manager"

(http://download.oracle.com/docs/cd/B25221_04/web.1013/b14427/jta.htm#sthref520)

Logging Information

One of the first places to look for information concerning a problem in SIM is in the log files. Stack traces and debugging information can be found within the log files.

The log files are configured to roll over once they reach a certain size (currently 10 MB). Once a log file reaches the configured size, it will be renamed (for example, sim.log will be renamed to sim.log.1) and new log messages will be written to a new file (for example, sim.log). If there are already rolled-over logs, they will be also be renamed for example, sim.log.1 becomes sim.log.2, sim.log.2 becomes sim.log.3, etc). Only ten files are kept – if ten files already exist and the current file rolls over, the oldest log file is deleted.

For information about logging related to the DexnexFileParser batch process, see "Batch Processes".

Default Location of Log Files

Server Log Files

The log file for the server is located at:

`<sim-oc4j-instance>/sim-home/log/sim.log`

It can be changed by changing the value of the "File" param in the "sim.appender" appender in sim-home/files/prod/config/log4j.xml.

The log file for the java batch programs is located at:

`<sim-oc4j-instance>/sim-home/log/sim-batch.log`

It can be changed by changing the value of the "File" param in the "sim.appender" appender in sim-home/batch-config/log4j.xml.

Client Log Files

Client-side log files are put in a directory called "log", which is put wherever "user.dir" is defined in your system. For example, if you launched the web start client with Firefox, "user.dir" is the directory where Firefox is installed. This means (depending on where you have Firefox installed) your logs could be in: C:\Program Files\Mozilla Firefox\log\sim.log.

To find the location of "user.dir", double-click on the status bar at the bottom of the SIM PC client to bring up the "Client Information" dialog. Click on the "Version" tab; one of the entries in the table is for the System Property "user.dir". The value in the "Version" column shows the location of "user.dir" on the current client's system.

Changing Logging Levels

Sometimes it is useful to change the amount of information that the SIM server logs. There are two ways to change logging levels – editing the log4j.xml file, or using the Oracle Enterprise Manager Application Server Control user interface.

Editing log4j.xml

log4j.xml is in the SIM OC4J instance, in sim-home/files/prod/config/log.xml. It is possible to change the level of any logger in the file. It is also possible to add new loggers if you want a certain SIM class to log more information. For more detail about loggers and logging levels, see Log4J documentation at http://logging.apache.org/log4j/docs/documentation.html.

> **Note:** After changing a log level in log4j.xml the SIM server must be bounced before the change will take effect.

Using Oracle Enterprise Manager Application Server Control

Sometimes it is useful to change a logging level without bouncing the SIM server. This can be done by using the Oracle Enterprise Manager UI. There is an MBean defined in the SIM application that lists all currently defined loggers and allows you to type in a new value for those loggers. This MBean also allows you to create new loggers.

Do the following to find this Mbean:

1. Launch the Oracle Enterprise Manager Application Server Control and log in. The list of OC4J instances on this server should be displayed.
2. Click on the OC4J instance for SIM.
3. Click on the Applications tab. This should show you the SIM and SIM-CLIENT applications.
4. Click on the "Application Defined MBeans" icon for the SIM application. This will display the Application MBeans defined by SIM.
5. Click on the "LogLevelMBean" in the left frame.

> **Note:** After changing a log level with Enterprise Manager Application Server Control, the change will be lost if the server is bounced. Whenever the server is started, it takes on the log levels defined in log4j.xml.

3

Technical Architecture

This chapter describes the overall software architecture for SIM, offering a high-level discussion of the general structure of the system, including the various layers of Java code. This information is valuable when the retailer wishes to take advantage of SIM's extensible capabilities and write its own code to fit into the SIM system.

Overview

SIM Technology Stack

SIM has an n-tier architecture consisting of a client tier, a server tier, and a data tier. The client tier contains a PC client (a Java desktop application) and handheld devices. The server tier contains the SIM server (deployed as a J2EE application inside the Oracle Application Server) and the Wavelink server (a standalone server for the handheld devices). The data tier consists of an Oracle 10g database and an LDAP directory.

Advantages of the Architecture

SIM's robust distributed computing platform enables enhanced performance and allows for scalability.

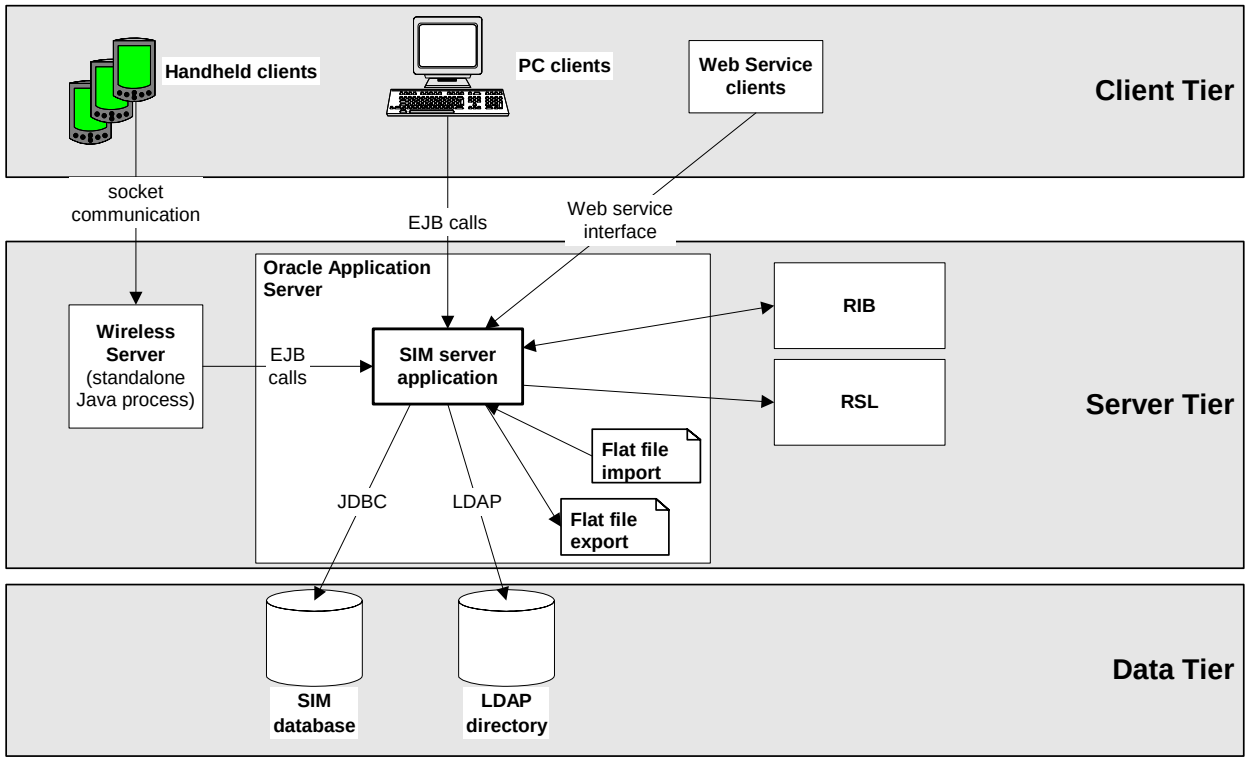
The n-tier architecture of SIM allows for the encapsulation of business logic, shielding the client from the complexity of the backend system. Any given tier need not be concerned with the internal functional tasks of any other tier.

The following list is a summary of the advantages that accompany SIM's use of an n-tier architectural design.

- Scalability: Hardware and software can be added to meet retailer requirements for each of the tiers.
- Maintainability: The separation of presentation, business logic, and data makes the software cleaner, more maintainable, and easier to modify.
- Platform independence: The code is written once but can run anywhere that Java can run.
- Cost effectiveness: Open source market-proven technology is utilized, while object-oriented design increases reusability for faster development and deployment.
- Ease of integration: The reuse of business objects and function allows for faster integration to enterprise subsystems. N-tier architecture has become an industry standard.
- High availability: Middleware is designed to run in a clustered environment or on a low-cost blade server.
- Endurance: Multi-tiered physically distributed architecture extends the life of the system.
- Flexibility: The system allocates resources dynamically based on the workload.

SIM Technical Architecture Diagrams and Description

This section provides a high-level overview of SIM's technical architecture. The diagrams below illustrate the major pieces of the typical three-tiered SIM implementation. Descriptions follow both diagrams.



SIM's technical architecture

Client Tier

SIM can be deployed on a wide variety of clients, including a desktop computer, a hand-held wireless device, and so on. The graphical user interface (GUI) is responsible for presenting data to the user and for receiving data directly from the user through the 'front end'. The presentation tier only interacts with the middle tier (as opposed to the database tier). To optimize performance, the SIM PC front end facilitates robust client-side processing.

The PC side of SIM is built upon a fat client architecture, which was developed using Swing, a toolkit for creating rich GUIs in Java applications.

The handheld communication infrastructure piece, known as the Oracle Retail Wireless Foundation Server, enables the handheld devices to communicate with the SIM server. The handheld devices 'talk' to the Oracle Retail Wireless Foundation Server, which in turn makes calls as a client to the SIM server.

Middle (Server) Tier

By providing the link between the SIM client and the database, the middle tier handles virtually all of the business logic processing that occurs within SIM's multi-tiered architecture. The middle tier is comprised of services, most of which are related to business functionality. For example, an item service gets items, and so on. Within SIM,

business objects are beans (that is, Java classes that have one or more attributes and corresponding set / get methods) that represent a functional entity. Most business objects have very few operations; in other words, business objects can be thought of as data containers, which have almost no business functionality.

Although the PC client and the handheld client use the middle tier's functionality differently, the middle tier is the same for both clients. For example, the handheld device, used 'on the fly', performs frequent commits to the database, while the PC performs more infrequent commits. The application is flexible in that it accommodates the different styles of client-driven processing.

The middle tier is designed to operate in a 'stateless' manner, meaning it receives whatever instruction it needs to access the database from the client and does not retain any information between client calls. Further, SIM has failover abilities; if a specific middle tier server fails, processing can roll over to another SIM server for continued processing.

If the workload warrants, SIM can be vertically scaled by adding additional application servers. Because SIM servers are running on multiple application servers in a stateless system, work can be seamlessly distributed among the servers. The result of this feature is that SIM clients do not need to know that additional application servers have been added to help with the workload. SIM application servers can contain multiple containers, each of which is related to a unique Java Virtual Machine (JVM). Each container corresponds to a specific SIM instance. Introducing multiple instances of a container allows SIM retailers to more effectively distribute the processing among several containers and thereby horizontally scale the platform. As the request load for a service increases, additional instances of the service are automatically created to handle the increased workload.

The middle tier consists of the following core components, which allow it to make efficient and reliable calls to the SIM database:

- Server services contain the pertinent business logic.
- DAO objects handle database interaction.
- Databeans contain the SQL necessary to retrieve data from and save data to the database.

> **Note:** There is at least one databean for every table and view in the database, but there may be more, used for different specific purposes.

Data Access Objects (DAO)

DAOs are classes that contain the logic necessary to find and persist data. They are used by services when database interaction is required.

Java Database Connectivity (JDBC)

DAOs communicate with the database via the industry standard Java database connectivity (JDBC) protocol. In order for the SIM client to retrieve the desired data from the database, a JDBC connection must exist between the middle tier and the database. JDBC facilitates the communication between a Java application and a relational database. In essence, JDBC is a set of application programming interfaces (API)s that offer a database-independent means of extracting and/or inserting data to or from a database. To perform those insertions and extractions, SQL code also resides in this tier facilitating create, read, update, and delete actions.

Database Tier

Note: The SIM data model includes some tables and columns that are SIM-specific and some that derive their names from the Association for Retail Technology Standards (ARTS) data model. Note, though, that SIM uses but does not fully conform to the ARTS standard.

The database tier is the application's storage platform, containing the physical data used throughout the application. The database houses data in tables and views; the data is used by the SIM server and then passed to the client. The database also houses stored procedures to do data manipulation in the database itself.

Distributed Topology

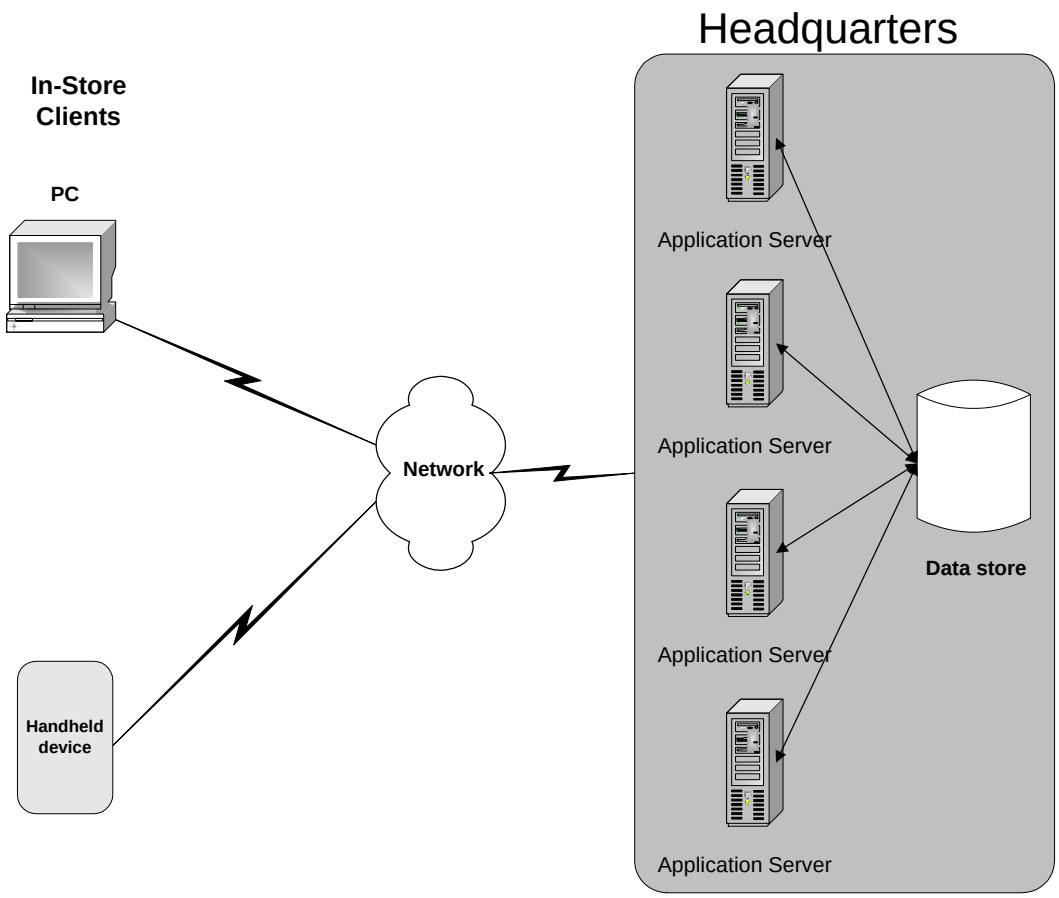
One of SIM's most significant advantages is its flexible distributed topology. SIM offers complete location transparency because the location of data and/or services is based upon the retailer's business requirements, not upon technical limitations. SIM's client server communication is an EJB call (which uses RMI). Because the server does not have to be in the same store as the in-store clients, the clients log onto the server 'over the wire'.

SIM's client code makes use of helper and framework classes that contain the logic to look up remote references to EJBs on the server and make calls to them. These helper and framework contain no business logic but contain only enough code to communicate with the server.

For example, if a helper class is called by the client to perform the method 'update shipment', the helper class appears to have that capability, though in reality it only behaves as a passage to the EJB remote reference, which is looked up from the server. The EJB remote reference communicates across the network with the server to complete the business-logic driven processing. The server performs the actual 'update shipment' business logic and returns any return values or errors to the client.

Connectivity between the SIM client and the middle tier is achieved via the Java Naming and Directory Interface (JNDI), which the SIM client accesses with the necessary IP address and port. JNDI contains the means for the client to look up services available on the application server.

The following diagram illustrates SIM's deployment.



SIM's deployment

A Word About Activity Locking

Activity locking has been designed to be controlled from within SIM. The following example illustrates the logic of activity locking.

A user becomes involved with a warehouse delivery that includes containers with multiple items in containers; that is, a significant amount of back and forth processing between screen and server is occurring. From the GUI, a call is made to the activity lock that instructs the system that the user is working with the warehouse delivery. If some other user has the lock, the system asks the user whether he or she wishes to break it and take over. A 'yes' response to the prompt implies that former owner of the lock left the lock dangling without a good reason (left to get lunch and so on). A 'no' response to the prompt implies that the former owner of the lock continues to legitimately need it in place in order to finish processing.

4

SIM Integration – Technical

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

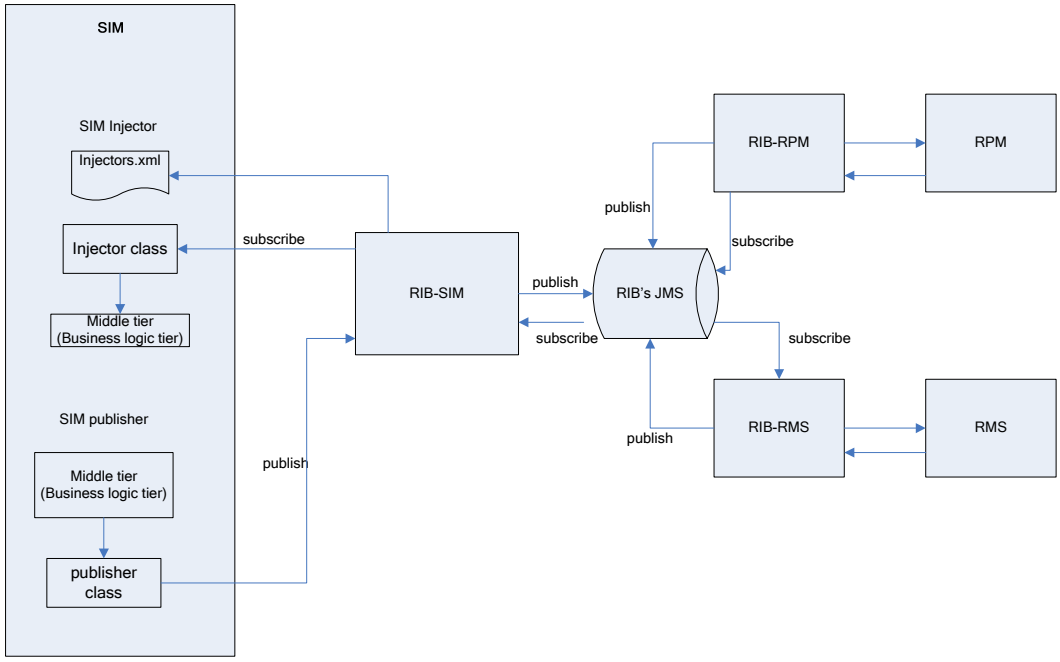
- Oracle Retail Integration Bus (RIB)-based integration
- Oracle Retail Service Layer (RSL)-based integration
- Web Service-based integration
- File-based integration

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 "SIM Integration – Functional".

RIB-based Integration

SIM can integrate with Other Retail products (such as RMS, RWMS) 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.

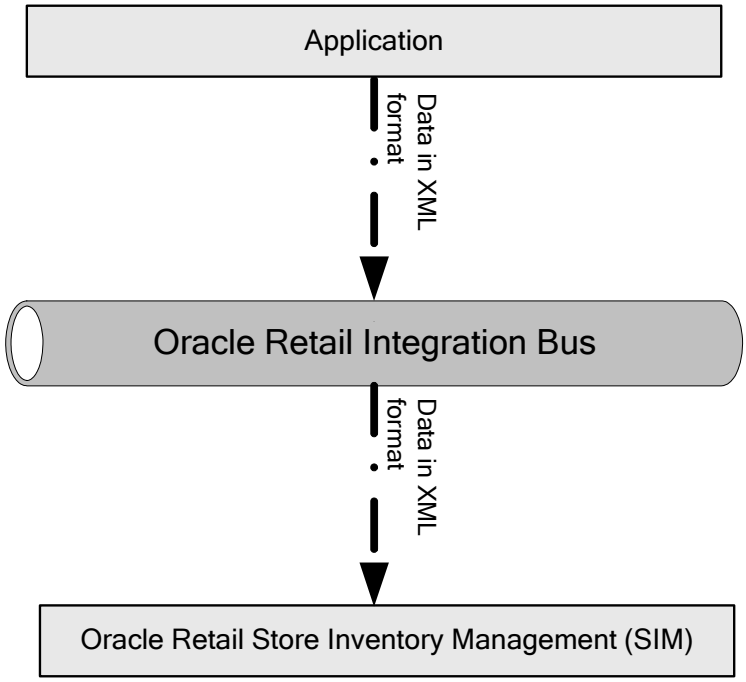
See the latest *Oracle Retail Integration Bus Operations Guide* and other RIB-related documentation for additional information.



SIM/RIB Integration Diagram

The XML Message Format

As shown by the diagram below, the messages to which SIM subscribes are in an XML format and have their data structure defined by document type definitions (DTDs) or XML schema documents.



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, etc), RIB and its corresponding RIB-<app> component are running on the application server (e.g. 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 utilize container manages the transaction and both the JMS and database resources are included in a two-phase commit XA compliant transaction.

See the latest *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, 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 latest *Oracle Retail Integration Bus Operations Guide* 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*.

Subscribers Mapping Table

The following table lists the message family and message type name, the document type definition (DTD) that describes the XML message, and the subscribing classes that facilitate the data's entry into the application's middle tier. These classes are described in the code as 'injectors'. For additional information, see the latest *Oracle Retail Integration Bus Operations Guide* and other RIB documentation.

Family	Type	Payload	Subscribing class ('injector')
ASNIN	ASNOUTCRE	ASNInDesc	ASNInCreateInjector
ASNIN	ASNINDEL	ASNInRef	ASNInRemoveInjector
ASNIN	ASNINMOD	ASNInDesc	ASNInModifyInjector
ASNOUT	ASNOUTCRE	ASNOutDesc	ASNOutCreateInjector
CLRPRCCHG	CLRPRCCHGCRE	ClrPrcChgDesc	ClrPrcChgCreateInjector
CLRPRCCHG	CLRPRCCHGMOD	ClrPrcChgDesc	ClrPrcChgModifyInjector
CLRPRCCHG	CLRPRCCHGDEL	ClrPrcChgRef	ClrPrcChgRemoveInjector
DIFFS	DIFFCRE	DiffDesc	DifferentiatorCreateInjector
DIFFS	DIFFDEL	DiffRef	DifferentiatorRemoveInjector
DIFFS	DIFFMOD	DiffDesc	DifferentiatorModifyInjector
ITEMS	ITEMBOMCRE	ItemBOMDesc	ItemBOMCreateInjector
ITEMS	ITEMBOMDEL	ItemBOMRef	ItemBOMRemoveInjector
ITEMS	ITEMBOMMOD	ItemBOMDesc	ItemBOMModifyInjector
ITEMS	ITEMCRE	ItemDesc	ItemCreateInjector
ITEMS	ITEMDEL	ItemRef	ItemRemoveInjector
ITEMS	ITEMHDRMOD	ItemHdrDesc	ItemModifyInjector
ITEMS	ITEMSUPCRE	ItemSupCtyDesc	ItemSupCreateInjector
ITEMS	ITEMSUPCTYCRE	ItemSupCtyRef	ItemSupCtyCreateInjector
ITEMS	ITEMSUPCTYDEL	ItemSupCtyRef	ItemSupCtyRemoveInjector
ITEMS	ITEMSUPCTYMOD	ItemSupCtyDesc	ItemSupCtyModifyInjector
ITEMS	ITEMSUPDEL	ItemSupRef	ItemSupRemoveInjector
ITEMS	ITEMSUPMOD	ItemSupDesc	ItemSupModifyInjector
ITEMS	ITEMUPCCRE	ItemUPCDesc	ItemUPCCreateInjector
ITEMS	ITEMUPCDEL	ItemUPCRef	ItemUPCRemoveInjector
ITEMS	ITEMUPCMOD	ItemUPCDesc	ItemUPCModifyInjector
ORDER	POCRE	PODesc	PurchaseOrderCreateInjector
ORDER	PODEL	PORef	PurchaseOrderRemoveInjector
ORDER	PODTLCRE	PODesc	PurchaseOrderDetailCreateInjector
ORDER	PODTLDEL	PORef	PurchaseOrderDetailRemoveInjector
ORDER	PODTLMOD	PODesc	PurchaseOrderDetailModifyInjector
ORDER	POHDRMOD	PODesc	PurchaseOrderModifyInjector
PRCCHGCONF	PRCCHGCONFCRE	PrcChgConfDesc	PrcChgConfCreateInjector
PRMPRCCHG	PRMPRCCHGCRE	PrmPrcChgDesc	PrmPrcChgCreateInjector
PRMPRCCHG	PRMPRCCHGMOD	PrmPrcChgDesc	PrmPrcChgModifyInjector
PRMPRCCHG	PRMPRCCHGDEL	PrmPrcChgRef	PrmPrcChgRemoveInjector
REGPRCCHG	REGPRCCHGCRE	RegPrcChgDesc	RegPrcChgCreateInjector

Family	Type	Payload	Subscribing class ('injector')
REGPRCCHG	REGPRCCHGMOD	RegPrcChgDesc	RegPrcChgModifyInjector
REGPRCCHG	REGPRCCHGDEL	RegPrcChgRef	RegPrcChgRemoveInjector
RCVUNITADJMOD	RCVUNITADJDTL	RcvUnitAdjDesc	RcvUnitAdjModInjector
RTVREQ	RTVREQCRE	RTVReqDesc	RTVReqCreateInjector
RTVREQ	RTVREQMOD	RTVReqDesc	RTVReqModifyInjector
RTVREQ	RTVREQDEL	RTVReqRef	RTVReqRemoveInjector
RTVREQ	RTVREQDTLCRE	RTVReqDesc	RTVReqDetailCreateInjector
RTVREQ	RTVREQDTLDEL	RTVReqRef	RTVReqDetailRemoveInjector
RTVREQ	RTVREQDTLMOD	RTVReqDesc	RTVReqDetailModifyInjector
SEEDDATA	DIFFTYPECRE	DiffTypeDesc	DifferentiatorTypeCreateInjector
SEEDDATA	DIFFTYPEDEL	DiffTypeRef	DifferentiatorTypeRemoveInjector
SEEDDATA	DIFFTYPEMOD	DiffTypeDesc	DifferentiatorTypeModifyInjector
STOCKORDER	SOCRE	SODesc	StockOrderCreateInjector
STOCKORDER	SODTLCRE	SODesc	StockOrderCreateInjector
STOCKORDER	SODTLDEL	SORef	StockOrderRemoveInjector
STOCKORDER	SODTLMOD	SODesc	StockOrderModifyInjector
STOCKORDER	SOHDRDEL	SORef	StockOrderRemoveInjector
STOCKORDER	SOHDRMOD	SODesc	StockOrderModifyInjector
STORES	STORECRE	StoresDesc	StoreCreateInjector
STORES	STOREDEL	StoresRef	StoreRemoveInjector
STORES	STOREMOD	StoresDesc	StoreModifyInjector
VENDOR	VENDORADDRCRE	VendorAddrDesc	SupplierAddrCreateInjector
VENDOR	VENDORADDRDEL	VendorAddrRef	SupplierAddrRemoveInjector
VENDOR	VENDORADDRMOD	VendorAddrDesc	SupplierAddrModifyInjector
VENDOR	VENDORCRE	VendorDesc	SupplierCreateInjector
VENDOR	VENDORDEL	VendorRef	SupplierRemoveInjector
VENDOR	VENDORHDRMOD	VendorHdrDesc	SupplierModifyInjector
WH	WHCRE	WHDesc	WareHouseCreateInjector
WH	WHDEL	WHRef	WareHouseRemoveInjector
WH	WHMOD	WHDesc	WareHouseModifyInjector

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 latest *Oracle Retail Integration Bus Operations Guide* and other RIB documentation.

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 *Service Layer Programmer's Guide* and *Service Layer Installation Guide* that is part of Oracle Retail Integration application.

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.

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

RSL Service	Payload
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. The SIM web service provides the external application exchange information with SIM. Currently SIM web service only provides one operation; Store Inventory Lookup.

File-based Integration

Currently SIM has three 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

See Chapter 6 – Batch Processes 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 type in:

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*.

5

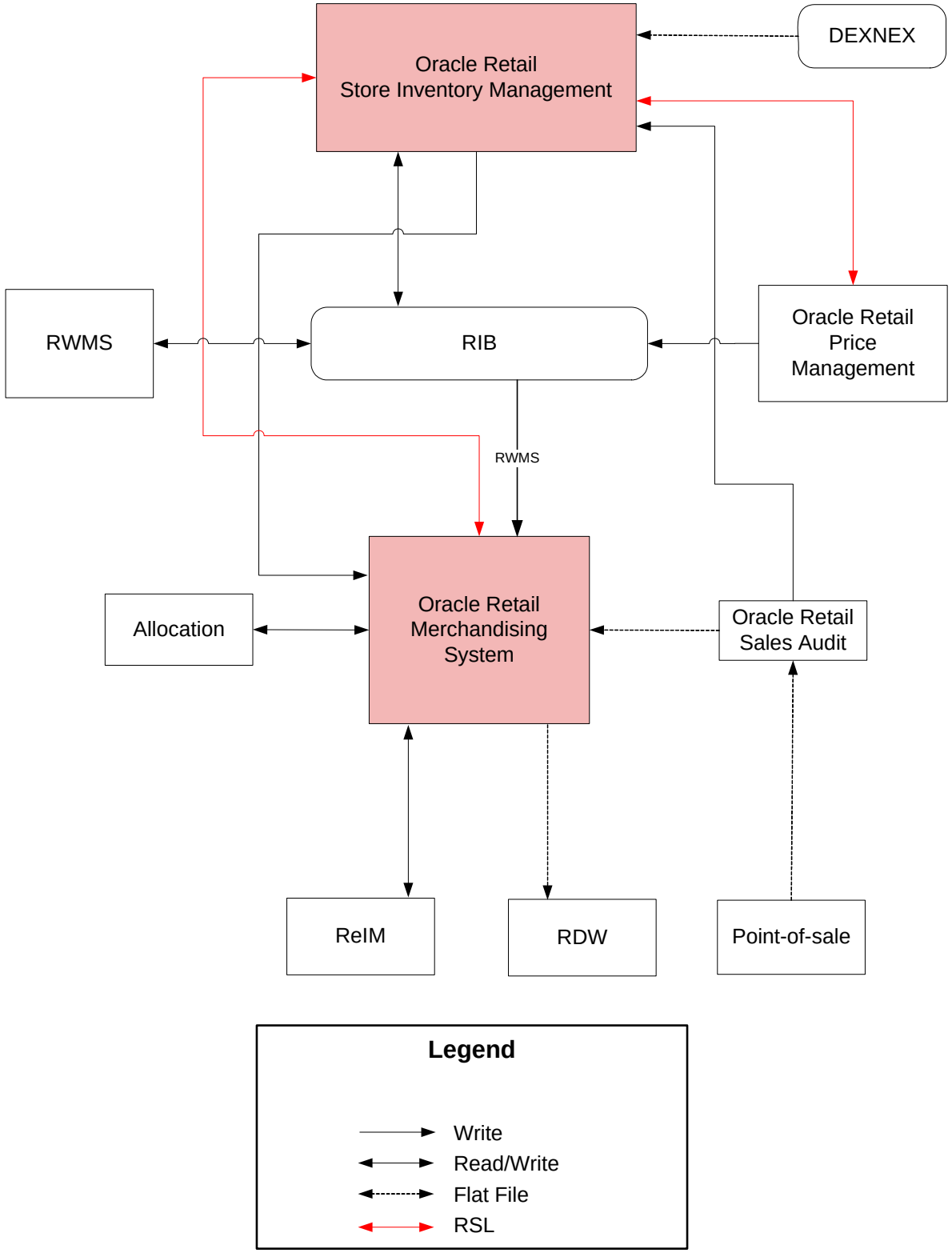
SIM Integration – Functional

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



For information about the technical means through which the interfaces pass data, see "Technical Architecture", "SIM Integration – Technical" and "Batch Processes".

Functional Descriptions of RIB Messages

The table below 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 latest *Oracle Retail Integration Bus Operations Guide* and other RIB documentation.

Functional area	Subscription/ publication	Integration to Products	Description
ASN in	Subscription	RWMS, Vendor (external)	These messages contain inbound shipment notifications from both vendors (PO shipments) and warehouses (transfer and allocation shipments).
ASN out	Publication	RMS, RWMS	These messages are used by SIM to communicate store-to-warehouse transfers (returns to warehouse) to both RMS and RWMS. These messages are also used to communicate store-to-store transfers to RMS.
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.
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.

Functional area	Subscription/ publication	Integration to Products	Description
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.
Price change	Subscription	RPM	These messages facilitate price changes for permanent, clearance and promotions.
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. SIM uses these to receive direct deliveries against.
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.

Functional area	Subscription/ publication	Integration to Products	Description
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.
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.
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.

Functional area	Subscription/ publication	Integration to Products	Description
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.
Vendor	Subscription	RMS, external (financial)	These are messages communicated by RMS containing vendors set up in the system (RMS or external financial system).

From SIM to the Warehouse Management System (WMS)

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

From the WMS to SIM

The following WMS 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 WMS 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.

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).

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 via 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 via 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 via 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 which 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.

6

Batch Processes

This chapter provides the following:

- An overview of SIM's batch processing
- A description of how to run batch processes, along with key parameters
- A functional summary of each batch process, along with its dependencies
- A description of some of the features of the batch processes (batch return values, restart and recovery)

Batch Processing Overview

SIM batches are executed as java batch processes. Most of the java batch processes engage in some primary processing of their own. However, the majority of work is done by services running on the SIM server; the java batch processes make remote calls to the server to access these services.

Note the following characteristics of SIM's Java batch processes:

- They are not accessible through a graphical user interface (GUI).
- They are scheduled by the retailer.
- They are designed to process large volumes of data, depending upon the circumstances and process.

Running a Batch Process

SIM batch programs are installed under $ORACLE_HOME/j2ee/<sim-oc4j-instance>/sim-home/bin, SIM batch processes are run from this location through executable shell scripts (.sh) files.

Oracle Retail provides the shell scripts (.sh files). They perform the following internally:

- Set up the Java runtime environment before the Java process is run.
- Start the Java batch process.

For more information about batch usage, see the batch design and usage sections in this chapter.

Summary of Executable Shell Scripts, Batch Files, Java Packages

The following table describes the executable shell scripts, batch files, and Java packages.

Executable shell script	Batch program executed
ActivatePriceChanges.sh	oracle.retail.sim.closed.batchjob.ActivatePriceChangeJob
CleanupPickList.sh	oracle.retail.sim.closed.batchjob.CleanupPickListJob
CloseProdGroupSchedule.sh	oracle.retail.sim.closed.batchjob.ProductGroupScheduleCleanupJob
DexnexParser.sh	oracle.retail.sim.closed.batchjob.DexnexFileParserJob
ExtractStockCount.sh	oracle.retail.sim.closed.batchjob.GenerateUnitCountJob oracle.retail.sim.closed.batchjob.GenerateUnitAmountCountJob
ItemRequest.sh	oracle.retail.sim.closed.batchjob.GenerateItemRequestJob
LateSalesInventoryAdjustmentPublishJob.sh	oracle.retail.sim.closed.batchjob.InventoryAdjustmentPublishJob
ProblemLineStockCount.sh	oracle.retail.sim.closed.batchjob.GenerateproblemLineCountJob
PurgeAdHocStockCount.sh	oracle.retail.sim.closed.batchjob.PurgeAdhocStockCountJob
PurgeAll.sh	oracle.retail.sim.closed.batchjob.PurgeAllJob
PurgeAudits.sh	oracle.retail.sim.closed.batchjob.PurgeAuditsJob
PurgeDSDReceivings.sh	oracle.retail.sim.closed.batchjob.PurgeDSDReceivingsJob
PurgeInventoryAdjustments.sh	oracle.retail.sim.closed.batchjob.PurgeInventoryAdjustmentsJob
PurgeItemRequests.sh	oracle.retail.sim.closed.batchjob.PurgeItemRequestsJob
PurgeItemTickets.sh	oracle.retail.sim.closed.batchjob.PurgeItemTicketsJob
PurgeLockings.sh	oracle.retail.sim.closed.batchjob.PurgeLockingsJob
PurgePickList.sh	oracle.retail.sim.closed.batchjob.PurgePickListsJob
PurgePriceChanges.sh	oracle.retail.sim.closed.batchjob.PurgePriceChangesJob
PurgePriceHistories.sh	oracle.retail.sim.closed.batchjob.PurgePriceHistoriesJob
PurgeReceivedTransfers.sh	oracle.retail.sim.closed.batchjob.PurgeReceivedTransfersJob
PurgeStockCounts.sh	oracle.retail.sim.closed.batchjob.PurgeStockCountsJob
PurgeStockReturns.sh	oracle.retail.sim.closed.batchjob.PurgeStockReturnsJob
PurgeWHDReceivings.sh	oracle.retail.sim.closed.batchjob.PurgeWHDReceivingsJob
ResaFileParser.sh	oracle.retail.sim.closed.batchjob.ResaFileParserJob
ResaOpenStkCnt	oracle.retail.sim.closed.batchjob.ResaOpenStockCountJob
ReturnNotAfterDateAlert.sh	oracle.retail.sim.closed.batchjob.ReturnNotAfterDateAlertJob
StockCountAuthorizeRecovery.sh	oracle.retail.sim.closed.batchjob.StockCountAuthorizeRecoveryJob
ThirdPartyStockCountParser.sh	oracle.retail.sim.closed.batchjob.ThirdPartyStockCountParserJob
WastageInventoryAdjustments.sh	oracle.retail.sim.closed.batchjob.GenerateInventoryWastageJob
WastageInventoryAdjustmentPublishJob.sh	oracle.retail.sim.closed.batchjob.InventoryAdjustmentPublishJob

Scheduler and the Command Line

If the retailer uses a scheduler, arguments are placed into the scheduler.

If the retailer does not use a scheduler, arguments must be passed in at the command line.

Return Value Batch Standards

The following guidelines describe the function return values and the program return values that SIM's batch processes utilize:

- 0 - The function completed without error, and processing should continue normally.
- 1 - A non-fatal error occurred (such as validation of an input record failed), and the calling function should either pass this error up another level or handle the exception.

Batch logging

Relevant progress messages are logged with regard to batch program runtime information. The location of sim batch log and logging levels can be configured in log4j.xml file which is located in sim-home/batch-config.

For more information, see "Logging Information".

Note: Some batch programs evoke Oracle stored procedure which runs on the Oracle database server, the log generated by the Oracle process may exist in different location which can be accessed by the Oracle database process. The log location is specified in batch detail section if it is different from the default batch log location.

Functional Descriptions and Dependencies

The following table summarizes SIM's batch processes and includes both a description of each batch process's business functionality and its batch dependencies.

Batch process	Description	Batch dependencies
ActivatePriceChanges	This batch process activates price changes which are effective today or on the user specified date.	No dependencies
CleanupPickList	The end of day batch process runs at the end of each day to reset the delivery bay and close any open pending pick lists.	No dependencies
CloseProdGroupSchedule	This batch process closes the product group schedule.	No dependencies
DexnexFileParser	This batch imports the direct delivery shipment records (PO, shipment and receipt) from dex/nex files in the DEX/NEX directory into SIM, the process creates a 'DEX/NEX direct delivery' in SIM.	No dependencies
ExtractStockCount	The Extract Stock Count Batch program generates Unit stock counts or Unit and Amount stock counts.	No dependencies
ItemRequest	The batch process generates item requests in pending or worksheet status for 'item request' product group schedule which was scheduled for current date.	No dependencies
LateSalesInventoryAdjustmentPublishJob	LateSalesInventoryAdjustmentPublishJob process publishes the late sale inventory adjustments records to Retail Merchandise System (RMS) through the Retail Integration Bus (RIB).	This batch program must be run in the sequence below: 1) ResaFileParser 2) ResaOpenStkCnt 3) LateSalesInventoryAdjustmentPublishJob
ProblemLineStockCount	The problem line batch process goes through the list of items in the problem line group, determining which fall within the user specified parameters (negative SOH, negative available, etc …). The system automatically creates a stock count from those items that do fall within the parameters.	No dependencies

Batch process	Description	Batch dependencies
PurgeAdHocStockCount	This batch process deletes ad hoc stock counts with a status of "in progress".	No dependencies
PurgeAll	This process deletes records from the SIM application that meet certain business criteria.	No dependencies
PurgeAudits	This batch process deletes audits.	No dependencies
PurgeDSDReceivings	This batch process deletes the Direct Store Delivery receiving.	No dependencies
PurgeInventoryAdjustments	This batch process deletes inventory adjustments.	No dependencies
PurgeItemRequests	This batch process deletes item requests.	No dependencies
PurgeItemTickets	This batch process deletes item tickets.	No dependencies
PurgeLockings	This batch process deletes lockings.	No dependencies
PurgePickList	This batch process deletes pick lists.	No dependencies
PurgePriceChanges	This batch process deletes price changes.	No dependencies
PurgePriceHistories	This batch process deletes price histories.	No dependencies
PurgeReceivedTransfers	This batch process deletes received transfers.	No dependencies
PurgeStockCounts	This batch process deletes stock counts.	No dependencies
PurgeStockReturns	This batch process deletes stock returns.	No dependencies
PurgeWHDReceivings	This batch process deletes the Warehouse delivery receivings.	No dependencies
ResaFileParser	This batch process imports sales and returns data that originates in a point of sale (POS) system. SIM uses the data to update the SOH for the store/items combinations in each file.	This batch program must be run in the sequence below: 1) ResaFileParser 2) ResaOpenStkCnt 3) LateSalesInventoryAdjustmentPublishJob
ResaOpenStkCnt	ResaOpenStkCnt batch processes the ReSA (Retail Sales Audit) open stock count items which are generated by the ResaFileParser process. This batch updates the snapshot or stock on hand quantities.	This batch program must be run in the sequence below: 1) ResaFileParser 2) ResaOpenStkCnt 3) LateSalesInventoryAdjustmentPublishJob

Batch process	Description	Batch dependencies
ReturnNotAfterDateAlert	This batch process warns users 'x' number of days in advance that the RTV/RTW is about to reach the Not After Date and must be dispatched. Note that the 'x' value is configurable via the system's administration GUI screens.	No dependencies
StockCountAuthorizeRecovery	This batch re-processes any stock counts that failed to complete their authorization process. It should be executed any time a severe server error occurs during the final authorization or export of a stock count.	No dependencies
ThirdPartyStockCountParser	This batch process imports stock count file from a third-party counting system (such as RGIS), the stock on hand quantities are updated for the existing unit and amount stock count records in SIM.	No dependencies
WastageInventoryAdjustments	This batch process looks for wastage product groups that are scheduled for today and creates an inventory adjustment for each item in the product group.	No dependencies
WastageInventoryAdjustmentPublishJob	The batch process picks up all items that were flagged for publishing to the merchandising system. After an item is published, the flag is reset.	No dependencies

Batch Process Scheduling

Before setting up an SIM batch process schedule, familiarize yourself with the scheduling dependencies:

Job Sequence	Batch Name
1	ResaFileParser
2	ResaOpenStkCnt
3	LateSalesInventoryAdjustmentPublishJob

For more details on the batch, see the Batch Details section in this chapter.

Batch Details

The following section summarizes SIM's batch processes and includes both an overview of each batch process' business functionality, assumptions, and scheduling notes for each batch.

Activate PriceChanges Batch

Overview

This batch process scans the price changes with pending or ticket list status. If the price change effective date matches the user specified batch date, the process activates the price changes (the price change status is changed to active) or marks the price change as completed.

Usage

The following command runs the ActivatePriceChanges batch job:

```
ActivatePriceChanges.sh <activate_date>
```

Where the activate_date is optional, date format must be in dd/mm/yyyy if the date is specified.

If the user does not specify the date, the current server date in GMT time will be used to find the matching price changes.

If the user passes a date string, then the batch process uses that date as the store local time to find the matching price changes for each store.

CleanupPickList

Overview

The end of day batch process runs at the end of each day to reset the delivery bay and close any open pending pick lists. The system takes the entire inventory from the delivery bay and moves it to the back room. Any pending or in progress pick lists are changed to a cancelled state. Users who are actioning a pick list are 'kicked out' of the system. That is, the system takes over their database lock, so they cannot make a save. After the batch process is run, all pick lists are either completed or cancelled, and the delivery bay has zero inventory.

Usage

The following command runs the CleanupPickList batch job:

```
CleanupPickList.sh
```

CloseProdGroupSchedule Batch

Overview

This batch program searches for all open product group schedules that have ended date before today (or user specified date), and change the product group schedule status to closed.

Usage

The following command runs the CloseProdGroupSchedule batch:

```
CloseProdGroupSchedule.sh <close_date>
```

Where the close_date is optional and a date is not entered, then the server date is used.

DexnexFileParser Batch

Overview

This batch imports the direct delivery shipment records (PO, shipment and receipt) from dex/nex files in the DEX/NEX directory into SIM.

With the uploaded data, SIM processing creates a 'DEX/NEX direct delivery', allowing the store user to view, edit, and confirm the information contained in the DEX/NEX file before approving it so that it can become an 'in progress' direct delivery.

Usage

The following command runs the DexnexFileParser batch:

```
DexnexFileParser.sh file_name
```

Where file_name is the DEX/NEXT file name that resides at the location specified in sim_batch.cfg file under DEXNEX_INPUT_DIR, errors are written to the location specified by DEXNEX_ERRORS_DIR in the same sim_batch.cfg file.

ExtractStockCount Batch

Overview

The Extract Stock Count Batch program generates Unit stock counts or Unit and Amount stock counts.

On a daily basis, the batch process creates the stock counts that are scheduled for the current day or future date which matches the next scheduled date. The system looks at all the scheduled stock count records and determines whether any are scheduled for today or the user specified future date. The process creates the stock counts for each individual store. If a scheduled count includes a list of 5 stores, 5 separate stock count records are created.

For Unit stock counts, if the system is configured to use unguided stock counts, the batch process does not generate multiple counts even if the item is located at multiple locations within the store.

For unit and amount stock counts, if an all location stock count is being run, the batch processing generates individual counts for every macro sequence location.

The date parameter is optional when running the Extract Stock Counts batch. If no date is provided, today's date is used. The date format is dd/mm/yyyy.

Usage

The following command runs the ExtractStockCount batch:

```
ExtractStockCount.sh <extract_date>
```

Where the extract_date is optional, if specified, it must be in format of dd/mm/yyyy.

> **Note:** If date is not passed in when the batch is run, today's date on the server is used.

ItemRequest

Overview

The batch process looks for those product groups that are set up as 'item request type' that are scheduled for the current date. It generates the item request (with items and quantities) in a pending or worksheet status. The user (for example, a manager) can then add items, delete items, change quantities, and so on before submitting the data to the merchandising system. The merchandising system can generate PO(s) or warehouse to store transfer(s) as applicable.

Usage

The following command runs the ItemRequest batch:

```
ItemRequest.sh
```

LateSalesInventoryAdjustmentPublishJob

Overview

LateSalesInventoryAdjustmentPublishJob process publishes the late sale inventory adjustments records to Retail Merchandise System (RMS) through the Retail Integration Bus (RIB). Late sale inventory adjustment could be the result of processing late sale records in ReSA sale data file by ResaFileParser batch or ResaOpenStockCnt batch.

Operationally, the LateSalesInventoryAdjustmentPublishJob should be run every time ResaFileParser batch and ResaOpenStk complete.

Usage

The following command runs the LateSalesInventoryAdjustmentPublishJob.sh

```
LateSalesInventoryAdjustmentPublishJob.sh
```

Assumptions and Scheduling Notes

This batch must run after following batch programs:

- ResaFileParser
- ResaOpenStkCnt

The following batches must be run in the sequence as below:

ResaFileParser

ResaOpenStkCnt

LateSalesInventoryAdjustmentPublishJob

ProblemLineStockCount Batch

Overview

Before the batch process runs, the retailer establishes a group of items and item hierarchies (by associating them to the problem line group type) and selects applicable parameters (negative SOH, negative available, and so on). The problem line batch process goes through the list of items in the group, determining which fall within the parameters. The system automatically creates a stock count from those items that do fall within the parameters.

If an item is a problem line item (negative inventory for example) on a stock count, and the user does not get the chance to perform the stock count on it that day, the next day the item may no longer be a problem line (positive inventory). However, the system continues to create a stock count for that item because a problem existed at one time.

Usage

The following command runs ProblemLineStockCount batch:

```
problemLineStockCount.sh
```

ResaFileParser Batch

Overview

This batch program imports sales that originated in a point of sale system. SIM uses the sales data to update the stock on hand for the store/items combinations in the sale file. In other words, from the batch program, SIM learns about inventory movement (what is sold and what is returned). Once SIM attains the data, it assumes that sales should be taken from the store's shelf-related inventory bucket. This assumption is important to SIM's shelf replenishment processing. Similarly, SIM assumes that returns should go first to the backroom bucket; the system's logic is that returns must be inspected.

In addition to handling the regular sales items, the ResaFileParser batch process handles non-ranged items, REF items, late sales, and open stock count items.

For item type ITM (the item type in ReSA file is marked as 'ITM'):

- If an item in the ReSA file has an item level below the transaction level (e.g. item level =3, transaction level = 2) and no stock on hand record, then it is an invalid item, and will be written to the rerun file.
- If an item in the ReSA file has an item level equal to the transaction level and no stock on hand record, then a new ranged item record is created for the item/store, and stock on hand is updated.

For item type REF (the item type in ReSA file is marked as 'REF'):

- If an item in the ReSA file has an item level below the transaction level (e.g. item level =3, transaction level = 2), then the parent item for this ref item is looked up.
- If the parent item's item level equals the transaction level, and it is ranged, then the stock on hand of the parent item is updated.
- If the parent item is a transaction level item, but is not ranged for the store, then a new ranged item is created for that store, and the stock on hand for the parent item is updated.

Note: For any item in the ReSA file that has an item level ABOVE the transaction level (for example, item level = 2, transaction level = 3), that item is invalid and is written to the rerun file. In the merchandise hierarchy, level-2 is "above" level-3 and level-1 is "above" level-2 and so on.

For late sale items:

- A late sale is a sales transaction that took place before a stock count was completed and the sale data file is processed after the count has started.
- A late sale is identified according to the 'Timestamp Processing' or 'Daily Sales Processing' stock count sales processing system parameters. For daily sales processing stock count, the 'Before Store Open' or 'After Store Close' stock count time frame parameters are used.
- Timestamp Processing indicates that sale data in the Sales Audit upload file has the timestamp for the transaction date. The sales data transaction timestamp is compared against the timestamps taken during the stock count to decide if the transaction is a late sale.
- Daily Sales Processing indicates sale data in the ReSA upload file does not have the timestamp for the transaction date. For daily sales processing, the 'Before Store Open' or 'After Store Close' stock count time frame parameters are used to determine whether the stock count occurred before or after business hours so that SIM knows how to handle late sales. Only the date is used to determine if a sale is late or not.

  'Before Store Open,' indicates the stock count will be performed before the opening of the store.

  'After Store Close,' indicates the stock count will be performed after the close of the store.
- Late sales should only be performed if the stock count was done after the store closed and the sales transaction was for the same day when the stock count was performed.
- For the late sale record, the late sale process decrements or increments the stock on hand depending on the sales transaction. In addition, a stock count inventory adjustment transaction within SIM is recorded to offset the sales transaction. The stock count inventory adjustment is published to RMS by running the LateSalesInventoryAdjustmentPublishJob batch if the count is a unit or an ad hoc count. The inventory adjustment is not sent to RMS if it is a unit and amount count since RMS has its own late sales process for unit and amount counts.

For open stock count items:

- An open stock count item is the in-progress stock count item
- If an open stock count exists, ResaFileParser updates the stock on hand and writes the record into the rk_resa_open_stk_cnt_item table. The ReSA open stock count item records are processed by ResaOpenStkCnt batch after the ResaFileParser batch.

Usage

The ReSA File Parser batch processes ReSA data files through the Oracle database stored procedure. The stored procedure locates the file location through database directory objects: RESA_DIR and RESA_ORIGINAL_DIR which are created during installation of SIM. The read and write privileges on these directory objects should be granted to the schema owner. The ReSA data file needs to reside on the database server or locations that can be accessed by oracle database process. The oracle process should have full access to the directories specified by RESA_DIR and RESA_ORIGINAL_DIR, and the ReSA data

file permissions need to be changed to allow the oracle process to read and write (remove) the file.

The corresponding operating system directories for the file storage must be created. The system or database administrator must ensure that the operation system directory had the correct read and write permissions for the Oracle database processes.

The following command runs the ResaFileParser batch:

```
ResaFileParser.sh  <file_name> <starting_line_num> <block_snapshot_ind>
```

> **Note:** ReSA data file needs to be put at the location specified by the RESA_DIR Oracle directory object on the database server (or the location specified in sim_batch.cfg file for key RESA_DIR if Oracle directory object is not used, and the location must be accessible by Oracle database process). The Oracle database process must have full access to the ReSA data file. Use chmod 777 to change the ReSA data file before starting the batch. The actual file location on the database server can be found by executing the following queries: select directory_name, directory_path from dba_directories where directory_name in ('RESA_DIR', 'RESA_ORIGINAL_DIR');

Where:

- file_name (required) is the name of the ReSA file containing the sales data from one store.
- starting_line_num (required) is the line number at which to start within the POSU file; starts with line 1 to start process a new data file. If the ReSA parser process terminates due to failure, and some of the records have already been processed, start the ReSA process with a line number from the failing point.
- block_snapshot_ind (required) is the flag indicating if a snapshot is allowed during the ReSA process. Valid values are: 'Y' and 'N'. 'Y' does not allow the snapshot to be taken until ResaOpenStkCnt.sh completes, and should always be used to ensure accurate SOH.

The ReSA batch process controls transactions at pre-defined transaction blocks size. Changing the parameter RESA_TRANS_SIZE in the sim_batch.cfg file can change this value. The default value is 100.

The batch process deletes the data file if it completes successfully. If the batch program encounters bad record(s) or a failure occurs during the parsing process, the batch process creates a rerun file in the same directory as the file being processed and the original ReSA data file is moved to resaOriginal directory.

> **Note:** The re-run file contains the bad records (or all uncommitted records within the transaction block on the event of fatal errors, the batch process terminates on event of the fatal error).

Assumptions and Scheduling Notes

Following batches must be run after this batch process:

- ResaOpenStkCnt
- LateSalesInventoryAdjustmentPublishJob

The following batches must be run in the sequence as below:

ResaFileParser

ResaOpenStkCnt

LateSalesInventoryAdjustmentPublishJob

ResaOpenStkCnt Batch

Overview

ResaOpenStkCnt batch processes the ReSA open stock count items which are generated by the ResaFileParser batch. It updates the snapshot or stock on hand records as appropriate based on the current stock count. Operationally, ResaOpenStkCnt should be run every time ResaFilePaser batch completes.

ResaOpenStkcCnt batch processes open stock count items as follow:

- Each open stock count item in the RESA open stock count item table. The process checks if the item is still an open stock count item by looking at the open stock count flag in rk_store_item_soh table.
- If an open stock count still exists and there is no timestamp taken for the physical count, then the snapshot is updated with the sales quantity (the SOH is updated for the open stock count item in ResaFileParser process).
- For items whose stock count has been confirmed, the process decides if this sale is a late sale by comparing the timestamp on the sales data and the timestamp of the item's physical count.
- If the timestamp of the sale is before the physical count and the count has not been completed yet, then the snapshot is updated (the SOH is updated for the open stock count item in the ResaFileParser process).
- If the sales timestamp is before the Authorization timestamp, late sales processing takes place. (For late sale details, see the ResaFileParser late sale section.)
- If the sales timestamp is after the physical count but before the confirmation of an authorized quantity that was saved, then late sales is also processed for that item. (For late sale details, see the ResaFileParser late sale section).

> **Note:** In case stock count processing is set to daily stock count processing, the above rules still apply, but instead of comparing the timestamp, the batch program will compare the sales date with the stock count date. In addition, it will determine a possible adjustment based on when the stock count was taken before store open or after store close.

Usage

```
ResaOpenStkCnt.sh <store_id>
```

Where store_id (optional) only processes records for a given store. If store_id is not passed in, ResaOpenStkCnt processes records for all stores.

Assumptions and Scheduling Notes

The following batch must run before this batch process:

- ResaFileParser

This batch must run before this batch process:

- LateSalesInventoryAdjustmentPublishJob

Following batches must be run in the sequence as below:

ResaFileParser

ResaOpenStkCnt

LateSalesInventoryAdjustmentPublishJob

ReturnNotAfterDateAlert Batch

Overview

This batch process warns users 'x' number of days in advance that the RTV/RTW is about to reach the 'Not after date' and must be dispatched. Note that the 'x' value is configurable via the system's administration GUI screens.

Usage

Following command runs the ReturnNotAfterDateAlert batch:

```
ReturnNotAfterDateAlert.sh
```

StockCountAuthorizeRecovery Batch

Overview

This batch process looks for stock counts that are stuck in authorization in progress state. This is a unique state that appears as 'Completed' visually, but in which the updating of stock on hand and creation of adjustments has not yet occurred. The batch attempts to authorize the stock count. Errors in the process are logged to the server error logs. Successfully authorized stock counts will move to authorized completed state.

Usage

The following command runs the StockCountAuthorizeRecovery batch:

```
StockCountAuthorizeRecovery.sh.
```

This batch job takes no command line input.

ThirdPartyStockCountParser Batch

Overview

This batch process imports stock count file from a third-party counting system (such as RGIS), the stock on hand quantities are updated for the existing unit and amount stock count records in SIM.

If the auto authorize admin flag is set to 'no', the following is true:

- The import file contains item and quantity counted information. SIM populates the count quantity on the stock count records and sets the authorize quantity equal to the count quantity. Once the file has been imported from the RGIS system, the stock count records type is set to 'authorize' and the status is set to 'in progress'.

- If any items are sent from RGIS that were not already ranged to the store, SIM adds the item to the appropriate stock count record (based on department), and sets the snapshot SOH amount to 0.
- During the import process from RGIS to SIM, any 'unknown' item data is written to the Not On File table.

If the auto authorize admin flag is set to 'yes', the following is true:

- The import file contains item and quantity counted information. SIM populates the count quantity on the stock count records, and sets the authorize quantity equal to the count quantity. Once the file has been imported from the RGIS system, the stock count records type is set to 'authorize' and the status is set to 'completed'.
- If any items are sent from RGIS that were not already ranged to the store, SIM adds the item to the appropriate stock count record (based on department), and sets the snapshot SOH amount to 0.
- During the import process from RGIS to SIM, any 'unknown' item data is written to the Not On File table.
- Once the import process is complete, SIM automatically authorizes the unit and amount stock counts and exports the stock count data to RMS. Under normal operating circumstances, this manual process is triggered by a SIM user through the front end. If the store admin flag for auto authorizing a third-party stock count is set to 'y', this process occurs as part of the import of the 3rd party file. Note that in this case, any items that are considered 'Not On File' are not assigned to an existing item. This business process assumes the retailer has resolved all discrepancies and data conflicts prior to exporting the count data from the third-party system. An assumption is also made that no data will be reviewed or changed using SIM. This process merely updates SIM with the stock count data. SIM, in turn, updates RMS with the same stock count data. No user intervention is required within SIM for this process to occur.

ThirdPartyStockCount Integration Assumptions

- RMS provides an 'item export' file to RGIS prior to the count in order for RGIS to validate the items that are scanned.
- The items coming from RGIS are identified based on an RMS item number (for example, an RIN, UPC, or other number set up in RMS).
- All quantities passed back from RGIS are assumed to be in the item's standard unit of measure (UOM) as established by RMS (for example, units, KG, and so on).
- The RGIS file sends back the total quantity counted for each item, regardless of whether the item was counted in several areas of the store (rolled up total by item).
- For items that exist in the SIM stock count records but do not have a counted quantity sent back from the RGIS system, SIM assumes a count quantity of '0', and enters this value on the stock count record.
- For items that have a SOH quantity in SIM but have a RGIS count of 0, the discrepancy check uses the variance units (not the variance %) value to determine whether the item is discrepant and should be displayed through the front end.

Usage

The ThirdPartyStockCountParser batch processes stock count import files through the Oracle database stored procedure. The stored procedure locates the file location through database directory objects: STOCK_COUNT_DIR and STOCK_COUNT_UPLOAD_DIR, the read and write privileges on these directory objects should be granted to the schema owner. The stock count import data file needs to reside on the database server or locations that can be accessed by Oracle database process. The Oracle process should have full access to the directories specified by STOCK_COUNT_DIR and STOCK_COUNT_UPLOAD_DIR, and the stock count import data file permissions need to be changed to allow the oracle process to read and write (remove) the file.

The corresponding operating system directories for the file storage must be created. The system or database administrator must ensure that the operation system directory add the correct read and write permissions for the Oracle database processes.

> **Note:** The Oracle database directory objects STOCK_COUNT_DIR and STOCK_COUNT_UPLOAD_DIR are created when the SIM application is installed.

Following command runs ThirdPartyStockCountParser batch:

```
ThirdPartyStockCountParser.sh <file_name>
```

Where the file_name is the import file data from one store; the stock count import data file needs to be put at the location specified by the STOCK_COUNT_DIR Oracle directory. The upload file is in STOCK_COUNT_UPLOAD_DIR. This upload file is an export file to RMS. The Oracle database process must have full access to the stock count data file. Use chmod 777 to change the stock count import data file before start the batch.

WastageInventoryAdjustments Batch

Overview

This batch process looks for wastage product groups that are scheduled for today and creates an inventory adjustment for each item in the product group. The batch process uses amounts based on percentage/units. Note that if both a percentage and unit exist, the batch process applies the least amount of the two. For example, consider an item with a stock on hand value of 100. If the two values are 10% and 5 units, the batch process would create an inventory adjustment of 5 units for the item.

The batch process creates a completed inventory adjustment record using the adjustment reason of 'Shrinkage' (code = 1) for each item that is published to the merchandising system.

Usage

Following command runs the WastageInventoryAdjustments batch:

```
WastageInventoryAdjustments.sh
```

After the batch process complete, the retailer needs to run another batch WastageInventoryAdjustmentPublishJob.sh to publish the inventory adjustment generated by the above batch to the merchandising system.

WastageInventoryAdjustmentPublishJob

Overview

The batch process picks up all items that were flagged for publishing to the merchandising system. After an item is published, the flag is reset.

Usage

Following command runs the WastageInventoryAdjustmentPublishJob batch:

```
WastageInventoryAdjustmentPublishJob.sh
```

SIM Purge Batch ProcessOverview

Transactional and historial records in SIM can be purged as below:

- PurgeAll batch: trigger all pre-defined purge batch processes and delete records which match the purging criteria.
- Run each individual batch to purge particular data.

For details on how to run the purge batch, see the batch program overview and usage section listed below.

PurgeAll Batch

Overview

This process deletes records from the SIM application that meet certain business criteria (for example, records that are marked for deletion by the application user, records that linger in the system beyond certain number of days, and so on).

Following is the list of transactions whose records get purged by the PurgeAll.sh batch

- Received transfers
- Stock Counts
- Inventory Adjustments
- Warehouse Receivings
- DSD/DSDASN Receivings
- Stock Returns
- Price Changes
- Price Histories
- Pick Lists
- Item Requests
- Item Tickets
- Audits
- Lockings
- Adhoc Stock Counts

Usage

```
PurgeAll.sh <purge_date>
```

Where purge_date is optional, date format must be in dd/mm/yyyy if purge_date is specified.

PurgeAdHocStockCount Batch

Overview

This batch program deletes ad hoc stock counts with a status of "in progress". Any ad hoc stock count with a creation date/time stamp older than the 'Days to Hold In Progress Ad Hoc Counts' parameter value will be deleted. For example, the default value is 1. If the batch program is run with the default value, the batch program would delete all in progress counts more than 24 hours old.

Usage

```
PurgeAdHocStockCount.sh
```

PurgeAudits

Overview

This batch process deletes audit records. Any audit record with a create date/timestamp older than the 'Days To Hold Audit Records' parameter value is deleted. For example, if the default value is 30 and the batch program is run with the default value, the batch program would delete all the audit records that are more than 30 days old.

Usage

```
PurgeAudits.sh <purge_date>
```

Where purge_date is optional and the date format must be in dd/mm/yyyy if purge_date is specified.

PurgeDSDreceivings Batch

Overview

This batch process deletes the Direct Store Delivery receivings.

Any DSD record which is in 'Closed'/'Cancelled' status and which has a complete date older than 'Days to Hold Received Shipments' is an eligible record for purge.

However, before a DSD record is purged, checks are made to ensure that the purchase order associated with a particular DSD is also completed and is older than 'Days to Hold Purchase Orders'.

Another check is made to identify the DSDASN's associated with a DSD record. If the DSDASN is cancelled/completed and is older than 'Days to Hold Received Shipments', only then it can get purged.

In effect a DSD record can be purged only if its associated PO and DSDASN records can be purged.

Usage

```
PurgeDSDReceivings.sh <purge_date>
```

Where purge_date is optional and the date format must be in dd/mm/yyyy if purge_date is specified.

PurgeInventoryAdjustments Batch

Overview

This batch process deletes inventory adjustments. Any inventory adjustment record with a create date/timestamp older than 'Days To Hold Completed Inventory Adjustments' parameter value will be deleted. For example, the default value is 30. If the batch program is run with the default value, the batch program would delete all the inventory adjustment records, which are more than 30 days old.

Usage

```
PurgeInventoryAdjustments.sh <purge_date>
```

Where purge_date is optional and the date format must be in dd/mm/yyyy if purge_date is specified.

PurgeItemRequests Batch

Overview

This batch process deletes item requests which are in 'Cancelled'/ 'Completed' status. Any item request record with a process date/timestamp older than 'Days To Hold Item Requests' parameter value will be deleted. For example, the default value is 30. If the batch program is run with the default value, the batch program would delete all the item request records, which are more than 30 days old.

Usage

```
PurgeItemRequests.sh <purge_date>
```

Where purge_date is optional and the date format must be in dd/mm/yyyy if purge_date is specified.

PurgeItemTickets Batch

Overview

This batch process deletes item tickets which are in 'Printed'/ 'Completed' status. Any item tickets record with a status date/timestamp older than 'Days To Hold Item Tickets' parameter value will be deleted. For example, the default value is 30. If the batch program is run with the default value, the batch program would delete all the item ticket records, which are more than 30 days old.

Usage

```
PurgeItemTickets.sh <purge_date>
```

Where purge_date is optional and the date format must be in dd/mm/yyyy if purge_date is specified.

PurgeLocking Batch

Overview

This batch process deletes lockings records from RK_LOCK_RECORD table. Any lock record with a lock date/timestamp older than 'Days To Hold Locking Records' parameter value will be deleted. For example, the default value is 30. If the batch program is run with the default value, the batch program would delete all the lock records, which are more than 30 days old.

Usage

PurgeLockings.sh <purge_date>

Where purge_date is optional and the date format must be in dd/mm/yyyy if purge_date is specified.

PurgePickList Batch

Overview

This batch process deletes pick lists which are in 'Completed'/ 'Cancelled' state. Any pick list record with a post date/timestamp older than 'Days To Hold Pick Lists' parameter value will be deleted. For example, the default value is 30. If the batch program is run with the default value, the batch program would delete all the pick list records, which are more than 30 days old.

Usage

PurgePickList.sh <purge_date>

Where purge_date is optional and the date format must be in dd/mm/yyyy if purge_date is specified.

PurgePriceChanges Batch

Overview

This batch process deletes price changes which are in 'Approved'/ 'Rejected'/ 'Completed' status. Any price change record with an effective date/timestamp older than 'Days To Hold Price Changes' parameter value will be deleted. For example, the default value is 30. If the batch program is run with the default value, the batch program would delete all the price change records, which are more than 30 days old

Usage

PurgePriceChanges.sh <*batch_date*>

Where purge_date is optional and the date format must be in dd/mm/yyyy if purge_date is specified.

PurgePriceHistories Batch

Overview

This batch process deletes price histories. At least a minimum of 4 historical prices are maintained for an item/store. 'Days To Hold Price History' will determine the number of days that price histories can be kept in the database.

Usage

PurgePriceHistories.sh <*batch_date*>

Where purge_date is optional and the date format must be in dd/mm/yyyy if purge_date is specified.

PurgeReceivedTransfers Batch

Overview

This batch process deletes received transfers. The transfer in and transfer out transactions will be purged from the database. The transfer out transactions which are in 'Received'/ 'Auto Received'/ 'Complete Approved'/ 'Complete Reject'/ 'Cancelled' / 'Cancelled Request' will be purged if the records are older than 'Days To Hold Received Transfer Records' parameter. Also, the 'Purge Received Transfers' parameter must be set to 'Yes' in the admin screen to enable purging of the received transfers.

Usage

```
PurgeReceivedTransfers.sh <purge_date>
```

Where purge_date is optional and the date format must be in dd/mm/yyyy if purge_date is specified.

PurgeStockCounts Batch

Overview

This batch process deletes stock counts which are in 'Completed'/ 'Cancelled' status. Any stock count with a schedule date/timestamp older than 'Days To Hold Completed Stock Counts' parameter value will get deleted. For example, the default value is 30.If the batch program is run with the default value, the batch program would delete all the stock return records, which are more than 30 days old

Usage

```
PurgeStockCounts.sh <purge_date>
```

Where purge_date is optional and the date format must be in dd/mm/yyyy if purge_date is specified.

PurgeStockReturns Batch

Overview

This batch process deletes stock returns which are in 'Dispatched/ 'Cancelled' status. Any stock return record with a completed date/timestamp older than 'Days To Hold Returns' parameter value will be deleted. For example, the default value is 30.If the batch program is run with the default value, the batch program would delete all the stock return records, which are more than 30 days old

Usage

```
PurgeStockReturns.sh <purge_date>
```

Where purge_date is optional and the date format must be in dd/mm/yyyy if purge_date is specified.

PurgeWHDReceivings Batch

Overview

This batch process deletes the Warehouse delivery receivings which are in 'completed' / 'cancelled' status. The warehouse receivings records which are older than the 'Days To Hold Received Shipments' will get purged, based on the value set for this parameter.

Usage

```
PurgeWHDReceivings.sh <purge_date>
```

Where purge_date is optional and the date format must be in dd/mm/yyyy if purge_date is specified.

A Note About Multi-Threading and Multiple Processes

SIM's batch processes are generally not set up to be multi-threaded or to undergo multi-processing. However, for data file batch processing, if performance is a concern, then the file can be break into smaller parts, each process can then consume one file and run parallel with as many other files as there are resources to support this processing. The recommended ratio is approximately 1-1.5 processes per available CPU.

Some batch programs do create multiple threads to call the server in order to do work more efficiently. Those batch programs are listed below. They generally work in the following pattern:

Query the server to find a set of data that needs to be processed.

Break the set of data into units of work that can be worked on independently in separate threads.

Create threads to work concurrently on the units of work.

Wait for all threads to finish.

Report any errors and return.

The number of threads that will be created to work on the units of work is determined by the configuration parameter **NUM_THREADS_IN_POOL** in **sim_batch.cfg** (located at sim-home/files/prod/retek/sim_batch.cfg). If no value is specified, a default value of 4 is used.

Batch Programs that Create Threads

- WastageInventoryAdjustments
- ItemRequest
- ProblemLineStockCount
- ExtractStockCount

A

Appendix: Stock Count File Layout Specification

rmsupload.cfg Configuration File

The configuration file, rmsupload.cfg, specifies the location of the unit/amount stock count output file that is to be uploaded into RMS. The default directory is the following:

`\retek\sim\files\prod\upload\`

This directory does not exist in the packaging but is automatically created upon the first completed unit/amount stock count. The directory can also be created manually.

Stock Count Results Flat File Specification

Once a stock count is authorized and completed, the SIM server creates a flat file during runtime and stages it to a directory that is configured during installation. Using the flat file generated by SIM, the merchandising system's stock upload module retrieves and uploads the physical stock count data. The file is formatted as follows:

Record name	Field name	Field type	Description
File Header	file type record descriptor	Char(5)	hardcode 'FHEAD'
	file line identifier	Number(10)	Id of current line being processed., hardcode '000000001'
	file type	Char(4)	hardcode 'STKU'
	file create date	Date(14) YYYYMMDD HHMISS	date written by convert program
	stocktake_date	Date(14) YYYYMMDD HHMISS	stake_head.stocktake_date
	cycle count	Number(8)	stake_head.cycle_count
	loc_type	Char(1)	hardcode 'W' or 'S'
	location	Number(10)	stake_location.wh or stake_location.store
Transaction record	file type record descriptor	Char(5)	hardcode 'FDETL'
	file line identifier	Number(10)	Id of current line being processed, internally incremented
	item type	Char(3)	hardcode 'ITM'
	item value	Char(25)	item id
	inventory quantity	Number(12,4)	total units or total weight

Record name	Field name	Field type	Description
	location description	Char(30)	Where in the location the item exists. Ex: Back Stockroom or Front Window Display
File trailer	file type record descriptor	Char(5)	hardcode 'FTAIL'
	file line identifier	Number(10)	Id of current line being processed, internally incremented
	file record count	Number(10)	Number of detail records.

B

Appendix: Batch File Layout Specifications

Flat File Used in the ResaFileParser Batch Process

This batch program imports sales that originate in a point of sale (POS) system. SIM uses the sales data to update the stock on hand for the store/items combinations in the POS file. For more information on the POS file format, see the POS Upload [posupld] section of the *Oracle Retail Merchandising System Operations Guide – Volume 1*.

Flat File Used in the DexnexFileParser Batch Process

File Structure – 894 Delivery

DEX/NEX uses the EDI Standard 894 Transaction Set to communicate with the direct delivery receiving system. The basic format for the file is as follows:

Header	
ST = Transaction Set Header	
G82 = Delivery/Return Base Record	
N9 = Reference Identification	
	Detail (repeating…)
	LS = Loop Header
	G83 = Line Item Detail DSD
	G72 = Allowance or Charge at Detail Level
	LE = Loop Trailer
Summary	
G84 = Delivery/Return Record Totals	
G86 = Signature	
G85 = Record Integrity Check	
SE = Transaction Set Trailer	

ST – Contains the transaction set number (for example, 894) and a control number.

G82 – Contains the type of delivery (Delivery or Return), supplier information, and delivery date.

N9 – Contains additional supplier information (Canada only).

LS – Contains an ID for the details loops to follow.

G83 – Contains the item #, quantity, UOM, unit cost, and item description.

G72 – Contains allowance (e.g. 10% off) or charge (e.g. environmental levy) information.

LE – Contains the loop trailer.

G84 – Contains the total quantity and cost of the delivery.

G86 – Contains the suppliers UCC signature.

G85 – Contains an authentication identifier.

SE – Contains the number of transactions in the transmission.

File details:

Segment	Sub-Segment	Name	Req?	SIM value
ST		Transaction Set Header	Yes	
ST	ST01	Transaction Set ID Code	Yes	894 - identifies the EDI file type, use to validate.
ST	ST02	Transaction Set Control #	Yes	Ignore
G82		Delivery/Return Base Record	Yes	
G82	G8201	Credit/Debit Flag Code	Yes	D=Delivery, C=Return.
G82	G8202	Supplier's Delivery/Return Number	Yes	Use as supplier's purchase order number.
G82	G8203	DUNS Number	Yes	Ignore
G82	G8204	Receiver's Location Number	Yes	Contains the Store #
G82	G8205	DUNS Number	Yes	Supplier's DUNS Number - use to determine supplier
G82	G8206	Supplier's Location Number	Yes	Supplier's DUNS Location - use with DUNS Number to determine supplier
G82	G8207	Delivery/Return Date	Yes	Delivery Date
N9		Reference Identification	No	
N9	N901	Reference Identifier Qualifier	Yes	Ignore
N9	N902	Reference Number	Yes	Use as SIM invoice number
N9	N903	Free-Form Description	No	Ignore
LS	LS01	Loop Header	Yes	Provides an ID for the loop to follow in the file
G83		Line Item Detail	Yes	
G83	G8301	DSD Number	Yes	Ignore
G83	G8302	Quantity	Yes	Unit Quantity
G83	G8303	Unit of Measure Code	Yes	CA = Case, EA = Each
G83	G8304	UPC		Item Number
G83	G8305	Product ID Qualifier		
G83	G8306	Product ID Number		
G83	G8307	UPC Case Code	No	Pack Number

Segment	Sub-Segment	Name	Req?	SIM value
G83	G8308	Item List Cost	No	Unit Cost
G83	G8309	Pack	No	
G83	G8310	Cash Register Description	No	Ignore
G72		Allowance or Charge at Detail Level	No	Ignore
G72	G7201	Allowance or Charge Code		Ignore
G72	G7202	Allowance/Charge Handling Code		Ignore
G72	G7203	Allowance or Charge Number		Ignore
G72	G7205	Allowance/Charge Rate		Ignore
G72	G7206	Allowance/Charge Quantity		Ignore
G72	G7207	Unit of Measure Code		Ignore
G72	G7208	Allowance/Charge Total Amount		Ignore
G72	G7209	Allowance/Charge Percent		Ignore
G72	G7210	Dollar Basis for Allow/Charge %		Ignore
LE	LE01	Loop Identifier		Loop Trailer, will contain same ID as loop header
G84		Delivery/Return Record Totals	Yes	
G84	G8401	Quantity	Yes	Sum of all G8302 values
G84	G8402	Total Invoice Amount	Yes	Total Cost, inclusive of charges and net of allowances.
G86	G8601	Signature	Yes	Ignore
G85	G8501	Integrity Check Value	Yes	Ignore
SE	SE01	Number of Included Segments	Yes	Total # of segments between ST and SE, used for validation
SE	SE02	Transaction Set Control #	Yes	Same as ST02, used for validation
GE	GE01	Number of transaction sets included	Yes	# of sets in functional group, used for validation
GE	GE02	Group Control Number	Yes	Same as GS06, used for validation

Flat File Used in the ThirdPartyStockCountParser Batch Process

RGIS File Layout Definition

- Number of Fields: 9
- Record Length: 80

Data name	Field Description	Dec Length	Position from	Position to	Field type
DLSSTR	STORE NUMBER	6	1	6	Character
DLSDAT	DATE MMDDYY	6	7	12	Character
DLSRAN	RGIS AREA NUMBER	10	13	22	Character
DSLF12	12 CHARACTER FILLER	12	23	34	Character
DSLF13	13 CHARACTER FILLER	13	35	47	Character
DLSUPC	UPC CODE	13	48	60	Character
DLSF12	12 ZERO FILLER	12	61	72	Character
DLSQTY	COUNT QUANTITY	7	73	79	Character
DLSF01	CONSTANT OF A "+"	1	80	80	Character

RGIS Sample File Data

```
00030105010212 068853600204 0000000000000000025
00030105010212 024000010265 0000000000000000007
00030105010212 027000422380 0000000000000000019
00030105010212 024000010265 0000000000000000004
00030105010212 755566004718 0000000000000000027
00030105010212 074027062006 0000000000000000017
00030105010212 074027062006 0000000000000000005
00030105010212 074027062006 0000000000000000003
00030105010212 035549874270 0000000000000000012
00030105010212 074027075464 0000000000000000003
00030105010212 042600065492 0000000000000000006
00030105010212 070320801199 0000000000000000014
00030105010212 067703680038 0000000000000000005
00030105010212 030267300667 0000000000000000009
00030105010212 045700155001 0000000000000000001
00030105010212 755566004718 0000000000000000018
```

C

Appendix: Price Bulk Processing

Overview

This appendix provides the details for executing Price Bulk Processing (PBP) batch programs in SIM. Price Bulk Processing imports the price changes from RPM related to Regular, Promotion and Clearance in bulk. Price Bulk Processing performs the following:

- Imports the price change information from the flat files to the staging tables in SIM database.
- Identifies bad records and discarded records, and logs them in a separate file.
- Processes each of the price change records and updates price change information in SIM.
- Logs processing information required as part of processing.
- Cleans up the staging tables after the completion of processing.
- Archives the price change input files.

Running A Script

Retailers are required to be aware of the following before running the script:

- The input price change file has to be placed in `BPP_INPUT_DIR` before starting the batch process.
- The minimum required permissions for the input file should be given. Minimal required permissions for the any price change input file are: `rw_rw_r__`

  Note: Minimal permissions of `RW` to the group users are required to move the file to archive directory.

- The number of threads for multi-threading of the batch process should be set based on the machine configuration.
- After the completion of batch processes, check for the .bad file and .dsc file to see if there are any bad or discarded records.

The following table defines the main price change files and their descriptions:

File name	Description
PromotionPrice Change.sh	This is the shell script for importing the promotion price changes to SIM. Do the following to run the script: 1. Place the promotion price change input files from RPM in `BPP_INPUT_DIR` directory. 2. Run the PromotionPriceChange.sh at the command prompt with price change input files as arguments separated by space. For example: `PromotionPriceChange.sh <<filename1>> <<filename2>>` *filename1* and *filename2* are separated by a space. The order for processing the files for price changes is from left to right. **Note:** If no filename is passed when the batch is run, the batch processes any unprocessed records from the previous execution. ▪ Shell script sets the appropriate java environment by calling the javaenv.sh shell script. ▪ The javaenv.sh shell script sets the classpath with all the jars and classpath variables required to call the bulkPricePromotionJob java class in SIM server. ▪ BulkPricePromotionJob calls the promotion_file_parser stored procedure and loads the data from the price change input file into the staging tables in SIM. BulkPricePromotionJob moves the input file to the archive directory, `BPP_ARCHIVE_DIR`. ▪ BulkPricePromotionJob sets the thread IDs to a logical unit of promotion information based on the number of threads configured for this promotion batch processing. **Note:** The PBP batch process supports multi-threading execution of the price change imports for faster processing. The number of parallel threads to execute the batch process can be configured. To configure, update the NUM_THREADS column in RK_RESTART_CONTROL table. ▪ BulkPricePromotionJob calls the process_promotion procedure to process the price change information and update into SIM. After the completion of processing, BulkPricePromotionJob deletes the promotion price change information from the staging tables.

File name	Description
ClearancePrice Change.sh	This is the shell script for importing the clearance price changes to SIM. Do the following to run the script: 1. Place the clearance price change input files from RPM in `BPP_INPUT_DIR` directory. 2. Run the ClearancePriceChange.sh at the command prompt with price change input files as arguments separated by space. For example: `ClearancePriceChange.sh <<filename1>> <<filename2>>` *filename1* and *filename2* are separated by a space. The order for processing the files for price changes is from left to right. **Note:** If no filename is passed when the batch is run, the batch processes any unprocessed records from the previous execution. ▪ Shell script sets the appropriate java environment by calling the javaenv.sh shell script. ▪ The javaenv.sh shell script sets the classpath with all the jars and classpath variables required to call the BulkPriceClearanceJob java class in SIM server. ▪ BulkPriceClearanceJob calls clearance_file_parser stored procedure and loads the data from the price change input file into the staging tables in SIM. BulkPriceClearanceJob moves the input file to the archive directory, `BPP_ARCHIVE_DIR`. ▪ BulkPriceClearanceJob sets the thread IDs to a logical unit of clearance information based on the number of threads configured for this clearance batch processing. **Note:** The PBP batch process supports multi-threading execution of the price change imports for faster processing. The number of parallel threads to execute the batch process can be configured. To configure, update the NUM_THREADS column in RK_RESTART_CONTROL table. ▪ BulkPriceClearanceJob calls the process_clearance procedure to process the price change information and update into SIM. After the completion of processing, BulkPriceClearanceJob deletes the clearance price change information from the staging tables.

File name	Description
RegularPrice Change.sh	This is the shell script for importing the regular or permanent price changes to SIM. Do the following to run the script: 1. Place the regular price change input files from RPM in `BPP_INPUT_DIR` directory. 2. Run the RegularPriceChange.sh at the command prompt with price change input files as arguments separated by space. For example: `RegularPriceChange.sh <<filename1>> <<filename2>>` *filename1* and *filename2* are separated by a space. The order of processing the files for price changes is from left to right **Note:** If no filename is passed when the batch is run, the batch processes any unprocessed records from the previous execution. ▪ Shell script sets the appropriate java environment by calling the javaenv.sh shell script. ▪ The javaenv.sh shell script sets the classpath with all the jars and classpath variables required to call the BulkPriceRegularJob java class in SIM server. ▪ BulkPriceRegularJob calls regular_price_file_parser stored procedure and loads the data from the price change input file into the staging tables in SIM. BulkPriceRegularJob moves the input file to the archive directory, `BPP_ARCHIVE_DIR`. ▪ BulkPriceRegularJob sets the thread IDs to a logical unit of regular price information based on the number of threads configured for this regular batch processing. **Note:** The PBP batch process supports multi-threading execution of the price change imports for faster processing. The number of parallel threads to execute the batch process can be configured. To configure, update the NUM_THREADS column in RK_RESTART_CONTROL table. ▪ BulkPriceRegularJob calls the process_regular_price_change procedure to process the price change information and update into SIM. After the completion of processing, BulkPriceRegularJob deletes the regular price change information from the staging tables.

File name	Description
▪ <<PRMPC>> TimeStamp.log (Promotion price change Log) ▪ <<CLRPC>> TimeStamp.log (Clearance price change Log) ▪ <<REGPC>> TimeStamp.log (Regular price change Log)	▪ Log files for each execution of batch process is generated and placed in BPP_LOG_DIR. ▪ Each log file is prefixed with the code of the batch process, PRMPC, CLRPC, REGPC and appended with the timestamp of execution to make it unique per execution.

D

Appendix: Creating An Auto-Authorized Third-Party Stock Count

The following steps indicate a generic path to perform a unit and amount stock count without reviewing the data in SIM.

1. Set Store Admin parameters.
   a. Set `Auto Authorize 3rd Party Stock Counts` parameter to **Yes**.

   Note: If this parameter is set to **Yes**, SIM automatically authorizes the count information coming from the third-party system and does not require the user to review the discrepancy data in SIM. This is the recommended method.

   If this parameter is set to **No**, SIM updates the counts and moves them to an Authorize state. In this case, the user uses the PC to view and authorize the counts manually.

   b. Set `3rd Party Stock Counts - Unit & Amount` parameter to Yes.

   Note: By default, this parameter is non-editable on the PC client. This can be changed in the database RK_STORE_CONFIG table.

   c. Select **Done** to save the changes.
2. Create a new product group.
   a. From the Main Menu, go to **Admin → Product Groups → Create a new Product Group**.
   b. Select **Type of Unit** and **Amount**.
   c. Enter a description for the count.
   d. You must enter at least one of the variance parameters (all three variance fields are enabled).

   Note: For items that have an SOH Quantity in SIM, but have a third-party count of 0, the discrepancy checks use the Variance Units (not Variance %) value to determine if the item is discrepant.

   e. SIM defaults to the store the user is logged into; if the user has SuperUser privileges, the user can select the All Stores radio button to make the group available to all stores the user has access to.
   f. Verify the **All Departments** radio button is selected (all other radio buttons on the screen are disabled when admin parameter is set to 3rd Party).
   g. Select **Add to Group**.
   h. Select **Done** to create the group.

3. Create a new product group schedule.
    a. From the Main Menu, go to **Admin → Product Group Scheduler → Create Product Group Schedule**.
    b. Select **Type of Unit** and **Amount**.
    c. In the Product Group dropdown box, select the third party product group you just created.
    d. Enter a description for the schedule.
    e. Enter a start date. The start date must be greater than or equal to today-plus-one and is controlled by the parameter value for Stock Count Lockout Days.
    f. Select the desired locations from the Available Locations box and move over to the Selected Locations box.
    g. Select **Done**.
4. Run the batch to generate the stock count.

    On the day of the scheduled stock count, run the `ExtractStockCount.sh` batch program to generate the stock counts.

    Note: After the batch has completed, from the Main Menu go to **Inv Mgmt → Stock Counts → Stock Count list** screen and notice that a separate stock count record has been created for each department. The batch creates stock count groups for all items for all departments for the store (this is including the items with SOH = 0) grouped by department.

    For each department record the Stock Count Type and Status from the stock count list screen will be **Type = Stock Count** and **Status = New**.

5. Take a snapshot of the SOH.
    a. From the Main Menu, go to **Inv Mgmt → Stock Counts → Stock Count List → Select Take Snapshot**.
    b. Select **Take Snapshot**. You must take the snapshot before uploading the third-party flat file.
    c. You must select each department record to take the snapshot for that department (**Take Snapshot** button allows multi-selection).

    Note: Selecting **Take Snapshot** takes a snapshot of the current SIM SOH figure, and assigns this to every item in the stock count records. The snapshot button is displayed only if there is an extracted 3rd Party Stock Count or Unit and Amount stock count on the Stock Count List screen. You must first select at least one record from the 3rd Party Stock Count in order for the snapshot to be taken.

    Status of the stock count will change to **In Progress**. This will indicate that the snapshot has occurred.

    The user will not be able to access the stock count records until the file has been uploaded. If the user double clicks one of the department stock counts on the list screen, SIM will prompt with the message

The stock count will not be accessible until the import process has completed.

You will not be able to drill into the detail screen if the third-party file has not yet been imported into SIM. Select **OK** to close the message, and the application remains on the Stock Count List screen.

6. Upload third-party count file to SIM.
   - Once counting is complete, the third-party input file must be placed in the STOCK_COUNT_DIR directory, which was specified during the SIM database installation. Default is /projects/SIM_DIR/sim/batch/stockCount/
   - Run batch ThirdPartyStockCountParser.sh batch file, passing in the name of the input file.

Note: The batch process generates and imports the stock count quantity from the flat file into the SIM stock count.

If the count contains items in SIM that were not ranged for the store, SIM will temporarily range the item.

If the count contains items that do not exist in SIM, they will go to the Not on File table. These unknown items can be assigned a valid SIM ID through the Not on File screen if the Auto Authorize parameter is set to **No**.

Stock Count moves to **Type = Authorize** and **Status = Completed**.

Inventory adjustment is written internally for SIM only. Inventory Adjustment is not sent to RMS for Unit and Amount stock count since the export file will send the stock count result to RMS. The same batch process will also generate an export file to import into RMS with all the valid counted quantities. . The output file will be generated in the `STOCK_COUNT_UPLOAD_DIR` directory, which was specified during the SIM database installation. Default is `/projects/sIM_DIR/sim/batch/stockCount_upload/directory`.